Metamodeling: Representing and Modeling System Knowledge in Design Processes

Massimo Cossentino[1] and Valeria Seidita[2,1]

[1] Istituto di Reti e Calcolo ad Alte Prestazioni
Consiglio Nazionale delle Ricerche, Palermo Italy
cossentino@pa.icar.cnr.it
[2] Dipartimento di Ingegneria Chimica Gestionale Informatica Meccanica
Università degli Studi di Palermo, Italy
valeria.seidita@unipa.it

Abstract. This paper reports the results of the experiences made in the representation and documentation of design processes. The aim is to explore and establish ways for managing the knowledge about the system solution developed within the processes itself. The work principally grounds on the concept of metamodel; raising the abstraction level in modeling languages during the development of software systems enables to tackle the problems complexity in a more efficient way. We propose a metamodeling layered architecture for representing software system metamodels and the rules for instantiating each layer starting from the top level one (MOF) downwards the bottom one (the software system). Besides the rules for relating the system metamodel constructs with the design process artifacts are also discussed in order to enable a detailed description of the knowledge the designer produces and manages about the new software system.

1 Introduction and Motivation

Traditionally, in software engineering, the idea, or the concept, of "model" addresses an artifact describing a software system. A model can be a set of diagram types or a single one. Each diagram can be drawn using specific modeling language, for instance UML or others.

To date it is recognized that models may be used in place of the software systems (from now on we will refer to system or to software as synonyms of software system) they represent. Models are less costly to develop than the whole system. It is furthermore recognized and accepted that constructing a model requires abstractions.

Let us, now, look at Figure 1 and consider the right hand part of it; it represents a portion of the *model* of a system that can be described through one use case diagram and one class diagram. Each element in the two diagrams is an instance of one element represented in the metamodel of the system shown in the left hand part of the figure. For instance the *Requirement*, in the metamodel, is realized by *Course Subscritpion* use case and the *Class* by the *Subscription_CTRL* class. Besides, elements of the use case diagram identified during, for instance, the analysis phase of the hypothetical design process used for developing the system, can be transformed in classes of the class diagram resulting from the design phase. The transition from one phase to another in the design process, and the related relationships among elements in different diagrams, is indicated by the relationship among elements in the metamodel.

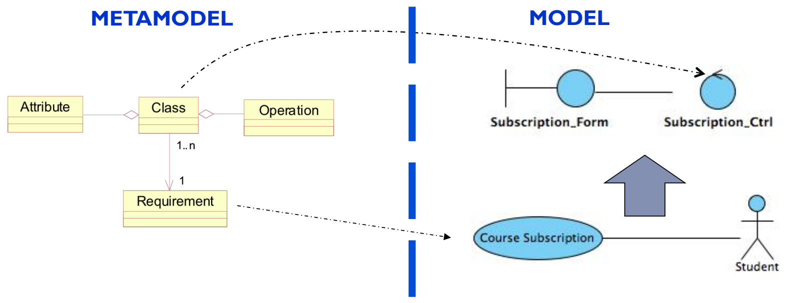

Fig. 1. Instantiation Process for System Model

Indeed, the system metamodel presents all the elements that have to be managed during the design process used for developing systems. In other words, such a metamodel describes the knowledge about the system the designer needs for performing her/his design work. The artefacts composing the model (or the models) of the system are drawn by using a specific modeling language (or a notation) and they are the result of the work done by one (or more) stakeholder(s) during a portion, phase or activity, of a design process. It is worth noting that starting from the same metamodel of the system, the artifacts can be drawn in different notations but the representation the model offers of the system is the same and it represents one instance of the metamodel.

"Metamodel is a model of models"

We base our work on this definition (by OMG [10]) and, for now, we leave apart all the problems and debates concerning multi-level metamodeling architecture like replication of concepts [1] or the difference among ontological and linguistic metamodel [3][9][7], and for a good understanding of our hypothesis we only consider what is called the *loose metamodeling* cited by [2]. *Loose metamodeling* implies that every model is an instance_of another model in the same way the traditional modeling infrastructure proposed by OMG does.

Figure 1 sketches a small example of the portion of metamodel adopted in a generic object oriented design process; any design process one wants to use he can count on a metamodel containing elements to be managed, and for which design actions can be identified for producing the artefacts composing the model.

We believe that the metamodel of the system relates the kind of systems to be modeled with the adopted design process and we agree with Rolland et al. [16] on the situatedness of design processes in specific domains.

The presented work deals with developing a meta-modeling layered architecture for representing system metamodels within design processes. The proposed architecture is complemented by the definition of the relationships among the different layers, the definition of the constructs needed to properly represent a system metamodel, the rules for representing the different situations that may occur, and a specific diagram completed by a proper notation.

To date a lot of work has been done in the context of metamodeling and several problems and still open issues have been identified. UML [20] itself suffers of problems like for instance "ambiguous classification" and "replication of concepts" well discussed in [1], and still a lot has to be done for identifying and fixing complete modeling techniques and rules useful in whatever domain context.

We claim the system metamodel is one of the most important elements of the design process and its description should receive a corresponding attention in the documentation of the design process itself. Moreover, because of the tight relationships of the system metamodel with the other design process elements (activity, role and work product [13] [6]), it constitutes a valid mean for aiding the designer in applying the design process. Besides, such a metamodel is a valid mean for reasoning about Method Engineering [4][15] implications on developing techniques for creating design process matching specific design contexts [5][19].

More in details, the contribution presented in this work consists in establishing a metamodeling layered architecture suggested for representing the system metamodel. We also illustrated all the *instance-of* relationships of the metamodel constructs with the MOF metamodel that we assume as a basis in accordance with OMG's prescriptions. In order to support the applications of this approach to the representation of several existing processes and related system metamodels[3], we developed (and here we report) a set of rules useful for managing all the possible instance_of relationships among layers. The rule are also useful for creating the diagram representing the relationships among every design artifact and the system metamodel constructs it cointains; such a diagram also reports the particular design actions performed by the designer when introducing the metamodel construct in the artifact.

The paper is organized as follows: the next section introduces some definitions about metamodeling and layers according to the approach standardized by OMG; section three reports and extends with novel details some concepts (design actions and workproduct content diagram) useful for representing system metamodels; section four presents the proposed metamodeling layered architecture, the instance_of relationships and the way for creating the diagram that

[3] See http://www.pa.icar.cnr.it/passi/FragmentRepository/fragmentsIndex.html for a repository of process fragments, see http://www.pa.icar.cnr.it/cossentino/fipa-dpdf-wg/docs.htm for the documents describing entire design processes

relates artifact with system metamodel constructs and finally section five provides concluding remarks and some future works.

2 Basic Concepts and Terminology

Before going on in describing the proposed system metamodel layered architecture and in order to avoid confusion about the used terminology, it may be useful to briefly introduce some premises on metamodeling layering in the OMG fashion and which is the definition, or the concept, we use for design process.

2.1 Metamodel Definition and Structure

What does it mean "metamodel"? A lot of definitions have been provided, from the simplest "metamodel is a model of models" to the more complex and completed one [1][3][7][12] also including concepts like ontological metamodeling vs linguistic metamodeling and so on. What is important for us is the word *meta* before the word *model*, meaning that we have to apply twice the rules used for modeling. Therefore, like the model, the metamodel is composed of elements and relationships both sometimes addressed as constructs. Metamodel elements and relationships provide rules for creating the model of the system in the same way elements and relationships of the model do for systems thus establishing an instance_of layering structure that will be better illustrated in the following.

2.2 Metamodeling Layers

Figure 2 shows the traditional Object Management Group metamodeling infrastructure. It is made by four layers each of which, except the top one, is related by an instance_of relationship with the above one.

The bottom level is the level M0, it contains the user data and is called the instance model. This represents the system solving a specific problem that runs on a specific platform. Therefore M0 represents all the elements that exist as the system runs on the real-world platform and manages the user data. The user data are instance_of the user concepts (level M1), so the level M1 represents the model of the system/software (as Figure 1 shows in its right hand part;) in the same way M2 contains information for instantiating the M1 concepts and for this purpose it is called metamodel layer (as the left hand side of Figure 1). M2 is here called UML concepts for the reason that Figure 2 deals with OMG metamodeling layers. Finally the level M3 contains information for creating metamodels; hence the meta-metamodels that is usually reported as the Meta-Object Facility (MOF) [11]. MOF is a very diffused language for describing metamodels.

The same layered architecture may be used for representing development processes, it is worth noting that with this term we mean both the design process and its result. In so doing the M0 and M1 layers respectively contain the system model and the system metamodel elements seen in Figure 1. Layer M3 is the

MOF level and what we need is to define the M2 layer by instantiating it from MOF in order to create the system meta-metamodel.

By using a multi-level modeling structure in the same OMG way we can exploit all the advantages of the MOF standard and the MDD technologies and theories. Above all it is very important the fact that by instantiating M2 from the MOF there is the possibility of using/developing MOF based tools in order to manage new modeling infrastructures and/or standards. Besides we can use the same instance_of rules for creating domain specific system metamodels.

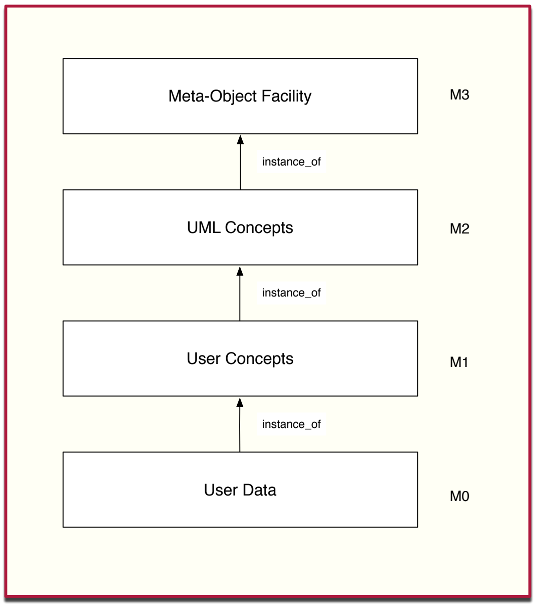

Fig. 2. The OMG Modeling Architecture

2.3 Design Process Definiton

Almost all the work done by the authors in the latest years in the field of agent oriented software engineering is centered on the following definition, by Fuggetta in [6], of the software development design process

> *"the coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy and mantain (evolve) a software product"*

This definition was also addressed during the work done within the IEEE FIPA standardization committee[4] that resulted in the standard definition [8] where it is argued that the main composing concepts of design process are: activity, process role and work product, hence enacting a design process implies a set of activities performed by process roles (the designer) for obtaining work products (artefact); moreover an important extension has been introduced by considering the metamodel as the fourth main composing element of design process.

It is our belief that the development of software following one specific design process means that during each activity one (or more) process role refer to the metamodel in order to produce work products where instances of a set of metamodel elements are managed (in order to understand this discussion keep in mind the example provided in Figure 1).

Depending on the activity the process role is performing, he manages the metamodel in different ways, namely while he is drawing a work product normally he instantiates system metamodel elements and in doing this he should need to consider or analyze another element as input that then he could report in the work product or not. All the possible actions a process role can do on a metamodel element are detailed in the following section.

3 Managing and Representing System Metamodel

In this section we highlight how we manage the knowledge about the system metamodel constructs in the process description by means of the design actions to be done for producing the process work product. For representing that, a novel kind of diagram has been created, the Workproduct content (WP content) diagram, an initial version of it has been already presented in [17] and it is now refined.

3.1 Design Actions

While composing a work product three different kinds of action can be made on each metamodel construct. Let us refer again to Figure 1 and suppose that process roles are using a design process composing of only two activities respectively aiming at producing one UML use case diagram and one UML class diagram.

During the first activity the *Requirement* element of the metamodel (the left part) is instantiated in the use case Course_Subscription hence the designer, by analyzing the problem context, is able to define this specific requirement and to draw it in the form of a use case in the diagram. Suppose that a list of actors has been already provided, then the designer is able to report on the diagram the actor Student by simply quoting this element coming from another workproduct and finally he may define a communication between the two, hence he defines a relationship between two instances of two metamodel elements.

[4] IEEE FIPA is a standardization committee of the IEEE Standards Society. See http://www.fipa.org

During the second activity, in order to produce the class diagram, the designer uses, as an input for his reasoning, the Course_Subscription use case and (s)he defines the Subscription_Form entity class that is an instance of the Class metamodel element then (s)he defines the control class and the relationship between the two classes. Besides, after having defined the classes, the designer can also refine them by adding attributes and operation; this action results in the definition of the attribute and operation as a special kind of element.

Therefore the possible actions the designer may operate on metamodel constructs are:

1. instantiate an element so (s)he defines that element,
2. instantiate a relationship so (s)he relates two elements,
3. use an already defined construct so he quotes an element, a relationship, an attribute or an operation,
4. instantiate an attribute or an operation so he refines an already defined element or relationship obviously refining an element includes also the quotation of that element.

3.2 The Workproduct Content Diagram

Among the others, an important use of the metamodel is with the *Workproduct Content Diagram*. We created it because it is our belief that it is an important part of the design process description and serves for representing the relationships between each work product produced during the design process and all the elements of the metamodel that are here drawn.

An example is given in Figure 3; the notation of this diagram implies classes for representing metamodel elements, arrowed lines for representing relationships among elements and labels for representing the kinds of action made on each element. Moreover packages with an icon on the left uppermost corner points out the work product and its kind (see [18] for the complete list and an explanation on work product kinds).

The aim of this diagram is to collect all the metamodel elements that are managed during the design process enactment and are also reported in the work product (as said before there can be elements of the metamodel that are not reported in the work product but used only as inputs for the work to be done).

Labels are: D for indicating the defined actions, R for defining a relationship, Q, QR, QA and QO for quoting respectively elements, relationships, attributes and operations.

From the Figure it can be seen that the work product represented by this content diagram aims at defining, the *Actor*, the *Functional Requirements* and the *Non Functional Requirements* that are related through one relationship.

This representation is useful for having an immediate panoramic on the actions to be done (and on system metamodel constructs) and it is a valid complement of the textual guidelines for producing the work product. Besides if needed it can be easily processed by a tool.

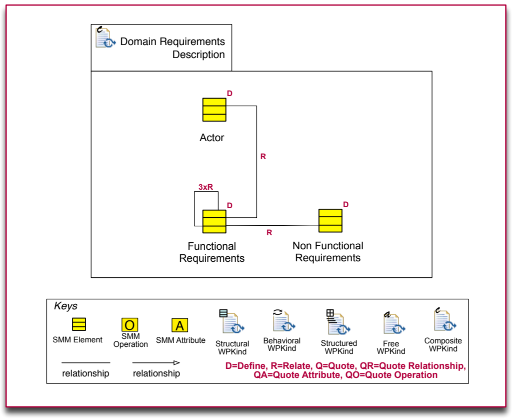

Fig. 3. An Example of the Workproduct Content Diagram

4 The Proposed System Metamodel Layered Architecture

In this section we illustrate the M2 level we defined together with all the possible constructs there can be and how each of them is an instance_of MOF.

Some definition may help in the comprehension of our approach: in the domain of design processes, we consider **System Metamodel** the set of **constructs** (and their definitions) used by designers for creating system models. During our experience in metamodeling and process definition we identified four kinds of construct: *elements*, *relationships*, *attributes* and *operations*.

– A System Metamodel Element (**SMME**) is the construct of the metamodel that can be instantiated into elements of the system model. Refer to the example given by Figure 1.
– A System Metamodel Relationship (**SMMR**) is the construct used for representing the existence of a relationship between two (or more) instances of SMMEs. For instance, the aggregation relationship among two instances of the SMME class is an instance of the SMMR association.
– A System Metamodel Attribute (**SMMA**) is a particular kind of element used for adding properties to SMMEs. The attribute's type is a SMME
– An operation (**SMMO**) is a particular kind of SMME using for describing the SMME.

They are instance_of the more general M2 layer (the System meta-metamodel) constructs, shown in Figure 4, that we identified together with all the relationships with MOF elements.

In addition to the four main elements of the previous description we identified one important construct to be managed during design process enactment, the

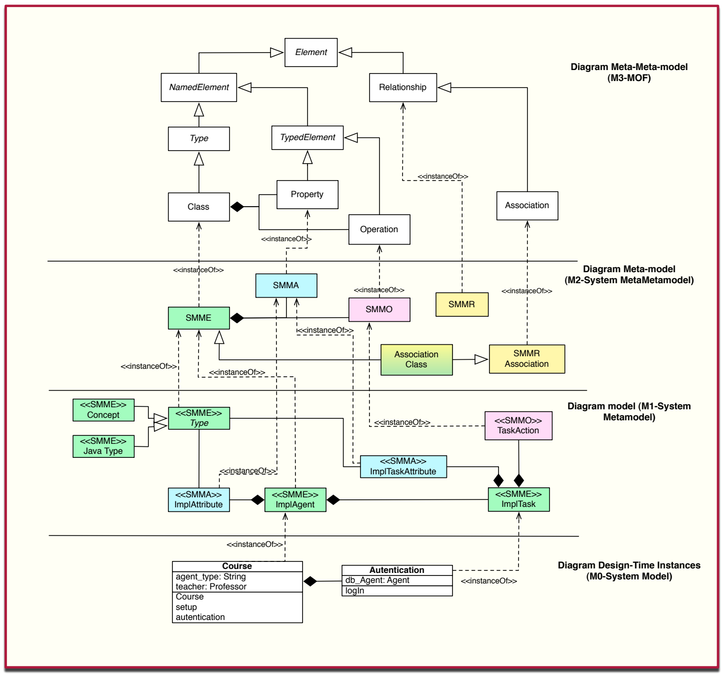

Fig. 4. The System Metamodeling Layered Architecture

Association Class. Association Class is at the same time an element (SMME) and a relationship association (**SMMR Association**), the presence of this construct comes from analyzing a lot of design processes and it was realized by extending the concept of UML association class [20]:

> *"...an association class. It will be both an association, connecting a set of classifiers and a class, and as such have features and be included in other associations. The semantics of an association class is a combination of the semantics of an ordinary association and of a class. An association class is both a kind of association and kind of a class... ".*

Some examples will be provided in the following subsection where all the possible instance_of relationships will be explored.

Besides the identification of design actions, defined in the previous section, led us to the conclusion that three different kinds of metamodels exist:

- Complete System Metamodel: it includes all the system metamodel constructs that are managed by the designer in using a specific design process. This also includes all the constructs that are accepted as external inputs of the overall process.

- Definable System Metamodel: it includes all the system metamodel constructs that are instantiated in the design process. This is a subset of the complete system metamodel.
- Workproduct System Metamodel: it includes all the system metamodel constructs that are reported in the design process work products. It is different from the Definable system metamodel because the Workproduct system metamodel may also include the quotable elements (like some inputs of the process)

Each of them supplies a specific view on the design process by specifying what and how elements are involved in the production of the system models, from now on in this paper we will use and simply refer to the Complete System Metamodel as the System Metamodel. For instance, referring to what we said in subsection 3.1, it is worth to note that only classes are drawn in the diagram whereas the requirement is not reported, hence the Workproduct System metamodel of the second activity does not contain the Requirement element and in this small case study the Workproduct System metamodel is equal to the Definable System metamodel. Of course the Complete System metamodel is larger since it also includes the input of this activity, the Course_Subscription use case.

Basing on our experience the constructs we identified are sufficient for defining the model of whatever kind of system model.

4.1 Instance_of Relationships among Layers

In this subsection we discuss all the possible instance_of relationships between M2 and M1 layers.

The notation used in the M0 level of the examples is the one defined in [14] but the way in which the system meta-metamodels has been defined guarantees to use every kind of notation one wants.

Case 1: one relationship connects two different elements. Figure 5 shows the case in which at level M1 two instances of SMMEs, *Agent* and *Role*, are related by one instance of an SMMR (*Play*); this configuration of model allows to represent the system as illustrated in the level M0. The result in the M0 level depends on the notation one decides to use, in this case the *Agent* is realized by mean of a package, the *Role* by the oval and the relation *Play* by the mutual position of the ovals and the packages.

Case 2: one relationship connects two instances of the same element. Figure 6 shows the second case in which, at level M1, two *ImplemAgent* are related through the relation *Association*, the cardinality is two to one, hence in the model two instances of the same SMME are related by one instance of the SMMR. An example of diagram at level M0, regardless of the notation, implies two instance of Implementation Agent (the TLPlanner and the EngController).

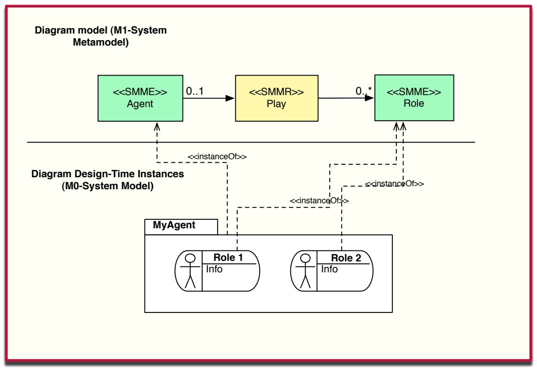

Fig. 5. Instance_of Relationships - Case 1

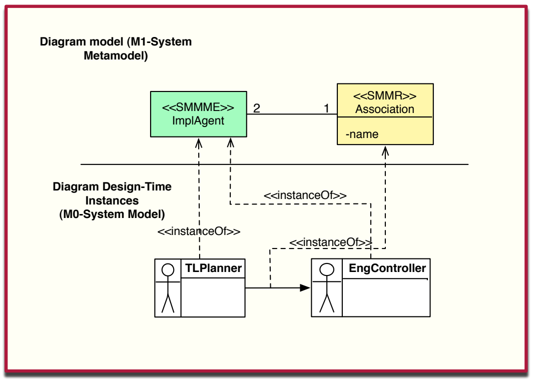

Fig. 6. Instance_of Relationships - Case 2

Case 3: one relationship among two elements determines the definition of another element of the metamodel. This is a special case of the previous one. Two instances of the same SMME are related by one instance of an SMMR, but the definition of the relationship at M1 level implies the definition of another SMME.

In order to understand when this case occurs, let us go from the bottom and consider the level M0 shown in the Figure 7: suppose to be designing a system where agents provide services to other agents depending on the role they are playing. In this case each role is dependent from the other through the service to be provided. Hence the model of the system (level M1) includes the SMMEs *Service* and *Role* and the SMMR *Service Dependency* but while the *Role* and *Service Dependency* have their own notational element (level M0) the service dosn't need that because its definition is implicit in the definition of the relationship and it would not be otherwise because if we represented the

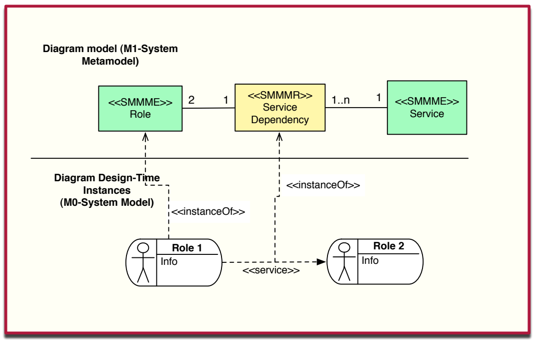
Diagram model (M1-System Metamodel)
<<SMMME>> Role
2
1
<<SMMMR>> Service Dependency
1..n
1
<<SMMME>> Service
Diagram Design-Time Instances (M0-System Model)
<<instanceOf>>
<<instanceOf>>
Role 1
Info
<<service>>
Role 2
Info

Fig. 7. Instance_of Relationships - Case 3

instance of *Service* through a specific icon we should connect it to the *Role 1* and the *Role 2* falling again, in this way, into the previous case and thus changing the semantic of the level M1 because the element cannot exist with that relation. The instantiation name of the *Service* SMME is in the name of the *ServiceDependency* instantiation.

Case 4: one metamodel construct is at the same time an SMME and an SMMR. In the fourth case two instances of the same element are related through an instance of one SMMR but this time the relationship is also an SMME and it can exist without the relationship (see Figure 8), it shares features from both SMME and SMMR hence it is an instance of a *SMMR Association Class*.

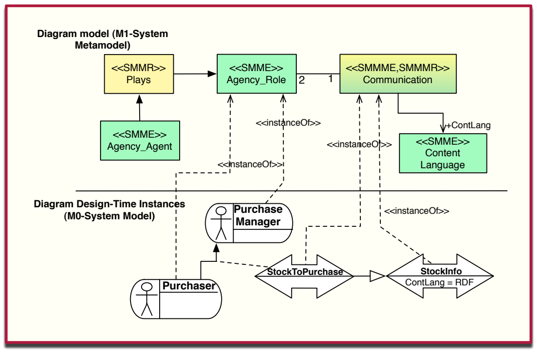

Fig. 8. Instance_of Relationships - Case 4

Case 5: one metamodel construct generalizes other metamodel constructs. This is the case when the system metamodel reports an abstract construct and all its specializations, each of them can be (separately) instantiated in different workproducts and different design actions can be made on different constructs, hence each specialized construct has to be treated as it were a single construct thus being united to all the previous cases.

4.2 Rules for Representing Metamodel Constructs in the WP Content Diagram

Each case before illustrated follows a different rule for the representation of metamodel constructs in the workproduct content diagram.

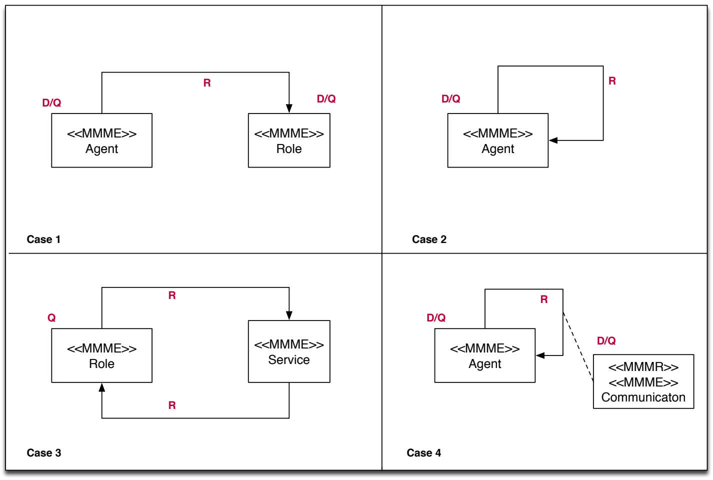

Fig. 9. Rules for WP Content Diagram

Figure 9 shows the rules for the first four cases whereas for the fifth case we have to consider that the abstract element has not to be labelled because the design action is done on one of its specialization whereas as regard the relationships we have to take into account that each construct related to the abstract one is really related to the corresponding number of specializations.

5 Conclusions

In this work we defined a metamodeling layered architecture meant to fully support the creation of system metamodel for managing the knowledge of the system in design process representation and documentation. The metamodeling layered architecture is based on the OMG modeling infrastructure and a set of instance_of relationships with MOF has been identified. The M2 (system meta-metamodel) level, we identified contains all the constructs useful for defining whatever system metamodel providing rules for modeling class of systems. In so doing we have a mean for defining system metamodels in a common way that, as our experience highlights, covers a very important role in design process. Together with the meta-metalevel we illustrated a set of rules for managing all the different design situations that can occur.

The key idea of this approach is that during the enactment of design process the designer needs a good formalization of the constructs he can use to represent the problems he is working on and all the constructs he can instantiate in the solution system model. Therefore having the rules for defining the system metamodel allows to describe and document the process, hence the knowledge on the process, in a quite uniform and consistent way. This are the contributions proposed in this paper. We are widely using this approach in our work, it has been shared by a large community of researchers with the result that the way of describing a design process is now a standard [8].

This approach provides a common base for sharing knowledge about different design processes in terms of the system metamodel constructs. For instance, by only using the knowledge provided by the system metamodel, and in some cases the work product content diagram, we can apply method engineer techniques for extracting or assembling portions of work; we can establish and apply algorithms for measuring specific features of the design process in an unbiased fashion; furthermore we can establish if a specific design process fits the needs of the designer for solving specific problems and, what is very important, avoiding the presence of skilled persons that perfectly know that design process.

References

1. C. Atkinson and T. Kuhne. The essence of multilevel metamodeling. *Uml 2001: The Unified Modeling Language: Modeling Languages, Concepts, and Tools: 4th International Conference, Toronto, Canada, October 1-5, 2001: Proceedings*, 2001.
2. C. Atkinson and T. Kuhne. Processes and products in a multi-level metamodeling architecture. *International Journal of Software Engineering and Knowledge Engineering.*, 11(6):761–783, 2001.
3. C. Atkinson and T. Kuhne. Model-driven development: A metamodeling foundation. *IEEE Software*, 20(5):36–41, September/October 2003.
4. S. Brinkkemper. Method engineering: Engineering the information systems development methods and tools. *Information and Software Technology*, 38(4):275–280, 1996.
5. M. Cossentino, S. Gaglio, A. Garro, and V. Seidita. Method fragments for agent design methodologies: from standardisation to research. *International Journal of Agent-Oriented Software Engineering (IJAOSE)*, 1(1):91–121, 2007.
6. A. Fuggetta. Software process: a roadmap. In *In Proceedings of the Conference on the Future of Software Engineering.ACM Press, New York (USA)*, pages 25–34, Limerick (Ireland), June 4-11 2000.
7. Wolfgang Hesse. More matters on (meta-)modelling: remarks on thomas kuhnes matters. *Software and Systems Modeling (SoSyM)*, 5(4):387–394, December 2006.

8. IEEE Foundation for Intelligent Physical Agents. *Design Process Documentation Template, Document number XC00097A-Experimental*, 2011.
9. Thomas Kuhne. Matters of (meta-) modeling. *Journal on Software and Systems Modeling*, 5(4):369–385, December 2006.
10. Jishnu Mukerji and Joaquin Miller. MDA guide version 1.0.1. Technical Report omg/2003-06-01, Object Management Group, 2003.
11. Object Management Group. *Meta Object Facility (MOF) Specification. http://doc.omg.org/formal/02-04-03*, 2003.
12. J. Odell. Power types. *J. Object-Oriented Programming*, 7(2):8–12, 1994.
13. OMG. Object Management Group. Software & Software Process Engineering Metamodel. version 2.0. Document number: formal/2008-04-01. 2008, 2008.
14. L. Padgham, M. Winikoff, S. DeLoach, and M. Cossentino. A unified graphical notation for aose. *Agent-Oriented Software Engineering IX*, pages 116–130, 2009.
15. J. Ralyté. Towards situational methods for information systems development: engineering reusable method chunks. *Procs. 13th Int. Conf. on Information Systems Development. Advances in Theory, Practice and Education*, pages 271–282, 2004.
16. C. Rolland, N. Prakash, and A. Benjamen. A multi-model view of process modelling. *Requirements Engineering*, 4:169–187, 1999. 10.1007/s007660050018.
17. V. Seidita, M. Cossentino, and A. Chella. *A Proposal of Process Fragment Definition and Documentation.* M. Cossentino, K. Tuyls, M. Kaisers, and G. Weiss (Editors). Springer-Verlag, in printing.
18. V. Seidita, M. Cossentino, and S. Gaglio. A repository of fragments for agent systems design. *Proc. Of the Workshop on Objects and Agents (WOA06)*, 2006.
19. V. Seidita, M. Cossentino, V. Hilaire, N. Gaud, S. Galland, A. Koukam, and S. Gaglio. The metamodel: a starting point for design processes construction. *International Journal of Software Engineering and Knowledge Engineering.*, 20(4):575–608, 2010.
20. UML. Object Management Group. OMG UML Specification v. 2.3, 05-05- 2010.