MAS Meta-models on Test: UML vs. OPM in the SODA Case Study

Ambra Molesini[1], Enrico Denti[1], and Andrea Omicini[2]

[1] DEIS, Alma Mater Studiorum, Università di Bologna,
Viale Risorgimento 2, 40136 Bologna, Italy
amolesini@deis.unibo.it, enrico.denti@unibo.it
[2] DEIS, Alma Mater Studiorum, Università di Bologna a Cesena,
Via Venezia 52, 47023 Cesena, Italy
andrea.omicini@unibo.it

Abstract. In the AOSE (Agent-Oriented Software Engineering) area, several research efforts are underway to develop appropriate meta-models for agent-oriented methodologies. Meta-models are meant to check and verify the completeness and expressiveness of methodologies.

In this paper, we put to test the well-established standard Unified Modelling Language (UML), and the emergent Object Process Methodology (OPM), and compare their meta-modelling power. Both UML and OPM are used to express the meta-model of SODA, an agent-oriented methodology which stresses interaction and social aspects of MASs (multi-agent systems). Meta-modelling SODA allows us to evaluate the effectiveness of the two approaches over both the structural and dynamics parts. Furthermore, this allow us to find out some desirable features that any effective approach to meta-modelling MAS methodologies should exhibit.

1 Meta-models for MAS

The definition of a methodology is an interactive process, in which a core is defined and then extended to include all the needed concepts. Meta-modelling enables checking and verifying the completeness and expressiveness of a methodology by understanding its deep semantics, as well as the relationships among concepts in different languages or methods [1]. According to [2],

> *the process of designing a system (object or agent-oriented) consists of instantiating the system meta-model that the designers have in their mind in order to fulfil the specific problem requirements. In the agent world this means that the meta-model is the critical element(...) because of the variety of methodology meta-models.*

In the context of MASs, a meta-model should be a structural representation of the elements (agents, roles, behaviour, ontology, ...) that constitute the actual system, along with their composing relationships. Several meta-models of AOSE methodologies can be found in the literature—for instance, GAIA [2], PASSI [2], ADELFE [2], Tropos [3], MESSAGE [4], IGENIAS [5]. Although a number of

M. Pěchouček, P. Petta, and L.Z. Varga (Eds.): CEEMAS 2005, LNAI 3690, pp. 163–172, 2005.

these (PASSI, MESSAGE, ADELFE) adopt some kind of UML extensions to express system models, while others (GAIA, TROPOS, IGENIAS) adopt some ad-hoc symbology for the same purpose, the meta-models of all such methodologies are still expressed in UML.

1.1 Why UML for Meta-models

The Unified Modeling Language (UML)[6] is the industry-standard language for specifying, visualising, constructing, and documenting the artifacts of software systems. Like other methods, UML is based on the *decomposition principle*, here in the form of *aspect decomposition*. A system is then expressed as a multiplicity of different models, each representing a specific system aspect: actually, UML defines 12 types of diagrams, whose 4 represent the static application structure, 5 are devoted to capture the system's dynamic behaviour, and 3 are related to the organisation and management of application modules. Altogether, all these models are expected to convey a complete system specification.

However, although the availability of so many models constitutes a richness from the expressiveness viewpoint, each model introduces its own set of symbols and concepts, thus leading to an unnatural complexity in terms of vocabulary, model multiplicity and model integration [7]. This is a problem both for maintaining consistency among the different system models and views, and for the mental integration of such views, since integrating several models within one's mind an is a very difficult process. That is why the need to concurrently refer to different models in order to understand a system and the way it operates and changes over time is a critical issue, known as the *multiplicity problem* [8]. Despite this issue, however, the general adoption of UML as a world standard for system modelling makes it the first natural choice for representing meta-models.

Adopting UML to express *meta-models of methodologies* endorses some specific issues, since representing a methodology is inherently different from representing a system at the object level. In particular, when meta-modelling methodologies, UML leads to emphasise objects and object relations, leaving aside the procedural aspects, which can be revealed only indirectly, by object operations and message exchanges. Moreover, the five behavioural diagrams provided by UML to capture the dynamic behaviour of a system at the object level become of little use at the meta-level, as they were defined to express which and how interaction occurs, rather than what interaction is and what role it plays—which is what is needed when representing a methodology. So, UML-based meta-models usually exploit only package diagrams, class diagrams, and associations.

1.2 Why OPM for Meta-models

In order to better address the issues of representing the dynamics at the meta-level, and possibly reduce the risk of inconsistency related to the multiplicity problem, it is natural to "look outside" the UML world, looking for some alternative approach. The *Object Process Methodology* (OPM henceforth) [9] is an integrated approach to the study and development of systems in general, and

of software systems in particular. OPM is also a reflective methodology, i.e. a methodology that can model itself without requiring any auxiliary means or external tools. OPM unifies the system's life-cycle stages (specification, design and implementation) within one single frame of reference, using a single diagramming tool—Object-Process Diagrams (OPDs)—and a corresponding subset of English, called Object-Process Language (OPL).

The basic assumption of OPM is that not only objects, but *objects and processes* constitute two equally-important classes of things, which together describe the functioning, structure and behaviour of a system in a single framework (i.e., without multiplying diagrams) in virtually any domain. Indeed, OPM's basic principle is that structure and behaviour in a system are so intertwined that effectively separating them is extremely harmful, if not impossible. Therefore, unlike the object-oriented approach, behaviour in OPM is not necessarily encapsulated within a particular object class construct: using stand-alone processes, one can model a behaviour that involves several object classes and is integrated into the system structure. Processes can be connected to the involved object classes through *procedural links*, which are divided, according to their functionality, into three groups: *enabling* links, *transformation* links, and *control* links.

Opposite to UML's aspect-based decomposition, which intrinsically violates the OPM's goal of a single all-describing model, OPM adopts *detail decomposition*: rather then decomposing a system according to its various aspects, decomposition proceeds by exploring the system's *abstraction levels*. This is done via three *refinement/abstraction mechanisms*: *unfolding/folding*, which refines/abstracts from the structural parts of a thing (mainly an object), *in-zooming/out-zooming*, which exposes/hides the inner details of a thing (mainly a process) within its enclosing frame, and state *expressing/suppressing*, which exposes/hides the states of an object.

1.3 Why Meta-modelling SODA

Interaction is a major source of complexity in software systems. This is particular true in multi-agent systems, where interaction can take different forms: for instance, *social interaction* is concerned with agents interacting with each other, while *environmental interaction* regards the agents' interaction with their environment. Although most methodologies still focus on *intra-agent* issues, more recently, methodologies like GAIA [10] and Hermes [11] have begun emphasising the role of interaction, shifting their focus toward social interaction.

So, since our purpose here is to exploit an agent-oriented methodology as a reference for stressing the pros and cons of different meta-modelling approaches, a methodology addressing only intra-agent issues would not fit: we need a methodology that widely deals with *inter-agent* issues, so that the social aspects of multi-agent systems are in the front line. SODA [12] is a methodology which explicitly focuses on suitably modelling the social aspects of a MAS. As such, it assumes interaction to be the key aspect of its modelling process: a system entity appears in a SODA model only in that it is involved in some interactions. So, designing a multi-agent system in SODA amounts to defining agents in terms

of their required observable behaviour, i.e., of the interactions in which agents are involved, and of the agents' roles in the MAS. In addition, taking interaction into account implies to consider relevant coordination issues, addressed by SODA in the design phase. Therefore, in the following we first define the SODA meta-model in UML (Section 2.1) and in OPM (Section 2.2), then comparatively discuss the pros and cons of such meta-models and, by doing so, of the two approaches in general (Section 3).

2 SODA Meta-models

SODA (Societies in Open and Distributed Agent spaces) [12] is an agent-oriented methodology for the analysis and design of agent-based systems. SODA focuses on inter-agent issues, like the engineering of societies and infrastructures for multi-agent systems. Since this conceptually covers all the interaction within an agent system, the design phase deeply relies on the notion of *coordination model* [13]. In particular, coordination models and languages are taken as a source of the abstractions and mechanisms required to engineer agent societies: social rules are designed as coordination laws and embedded into coordination artifacts, and the social infrastructure is built upon coordination system.

The analysis phase is characterised by three models: the *role model*, the *resource model* and the *interaction model*. The *design* phase is based on three strictly-related models, deriving from the models defined in the analysis phase; in particular, the analysis' role model maps on the design's *agent model* and *society model*, while the analysis' resource model maps on the design's *environment model*. The analysis' interaction model, in its turn, generates the interaction protocols and coordination rules referenced by the design's models (see [12] for more details).

2.1 SODA Meta-model in UML

The UML meta-model of SODA (Figure 1) reflects the SODA distinction between the analysis phase (top) and the design phase (bottom). In the analysis phase, the *boundaries* between the resource model, the interaction model, and the role model are well defined; in the design phase, instead, no such boundaries are shown, because the entities of the analysis sub-models do not map one-to-one onto analogous entities of the design model. It is worth noting that this UML model clearly emphasise the *centrality of interaction* which is typical of the SODA approach: in fact, if the interaction model were deleted, along with the corresponding classes in the design phase, concepts such as roles and resources would turn out to be separate and unrelated from one another.

Although this model captures the SODA concepts and associations as far as UML's (large yet somehow limited) graphical vocabulary makes it possible, the result is not completely satisfactory, for several reasons. First, UML provides basically a unique type of concept/symbol (the class) to represent entities which are conceptually distinct in the meta-model. More precisely, while using the

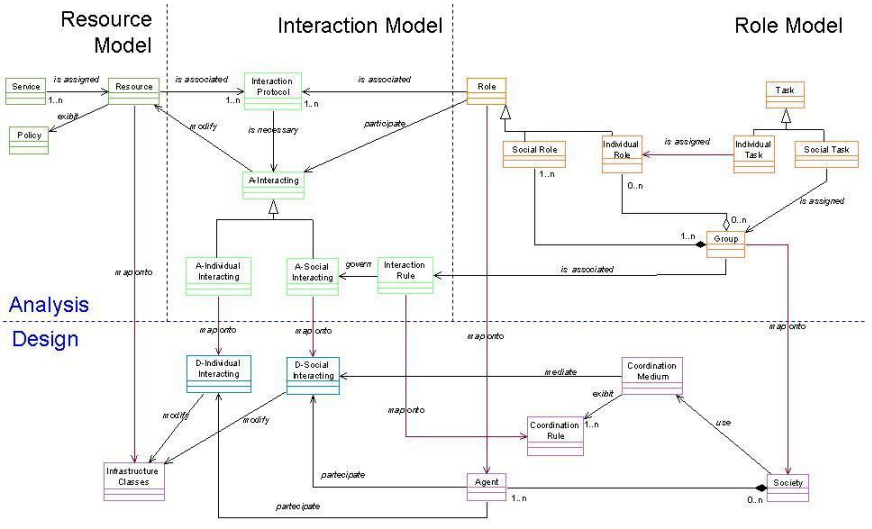

Fig. 1. SODA Meta-model in UML

UML class notion to capture the SODA *organisational structure*—i.e., entities such as roles, tasks, groups, society, agents, resources, infrastructure classes—leads to a satisfactory representation of these aspects, the same cannot be said for *interaction*, whose classes are qualitatively different from the others (both in the analysis and in the design phase), as they try to model an intrinsically dynamic dimension by means of an intrinsically static abstraction.

The model entities are connected to each other by different relations—inheritance, composition, aggregation, and generic association. In particular, the relations between Group and (respectively) Individual role / Social role emphasise that a Social role may either coincide with an already defined Individual role (aggregation), or be defined *ex-novo* (composition). Moreover, the relations between the structural entities and the "interaction entities" are critical from the modelling viewpoint, since such entities are qualitatively different; this is why they are expressed by a generic (tagged) association.

Another key aspect concerns the connections from the analysis phase to the design phase. The label "map onto" is somehow vague, yet underlines the intrinsic difficulty in expressing the complex mapping from the analysis to the design phase via a single association link. For instance, when mapping Role onto Agent, the association itself is unable to express that Agent *inherits* task, permissions and interaction protocols from Role: so, a suitable label is the only (yet unsatisfactory) way to express this fact.

2.2 SODA Meta-model in OPM

Figure 2 shows the SODA meta-model in OPM. Of course, many aspects discussed above—the distinction between the two phases, the analysis sub-models, the

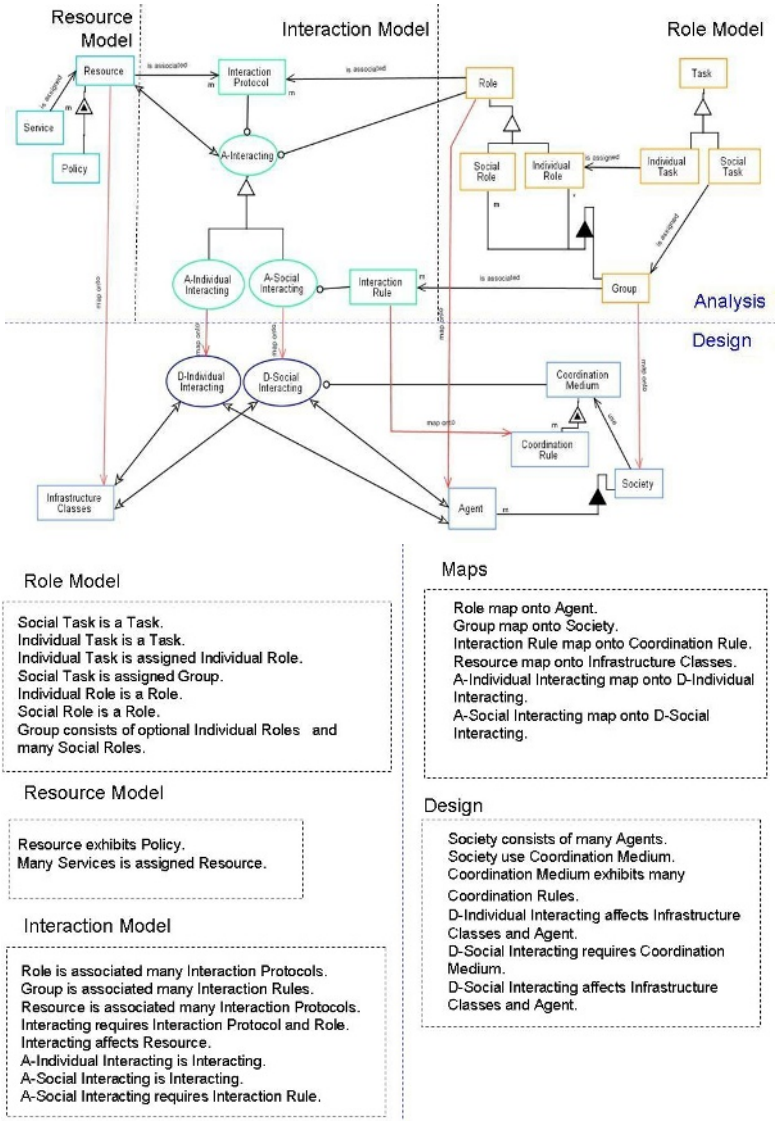

Fig. 2. SODA Metamodel in OPM: OPD (top) and OPL (down)

centrality of interaction, the association "map onto",—still hold: so, the overall model structure is basically the same as in Figure 1.

However, the richer expressiveness of OPM's graphical vocabulary with respect to UML makes it possible to model the key aspect of interaction as an OPM *process*, rather than as a class, thus expressing the dynamic aspects that the (static) class notion alone could not capture. By doing so, the OPM metamodel of SODA captures the transient nature of interaction in much a better way

than its UML counterpart. Furthermore, the richness of the OPM graphical vocabulary offers a better alternative to replace UML generic (tagged) associations with a new, semantically-clear symbology. For instance, the relation between Resource and Policy (and between Coordination Medium and Coordination Rule) now adopts a specific symbol to express that Policy not only has a structural relation with Resource, but is also an attribute of Resource.

On the other hand, since OPM introduces just one symbol (the solid black triangle) to represent both composition and aggregation, distinguishing between different relations (e.g Group/Individual Role, Group/Social Role) now requires a careful reading of the *participation constraint* of the relation (where * means "optional", *m* means "many", etc.). However, this aspect can be easily faced by using OPM's textual counterpart, OPL, that provides a human-readable description of the Object Process Diagram; the OPL of SODA meta-model is shown in Figure 2 (bottom). Despite the richness of OPM's vocabulary, some meta-modelling relations are still difficult to express: this is particularly true for the relations between structural entities and "*X*-Interacting" processes, that even the (several) object/process link types provided by OPM are unable to capture at a semantically-satisfactory level (more details in Section 3.2).

3 Discussion

In this Section, we discuss and compare the SODA meta-models in UML and OPM, outlining the respective pros and cons. Generally speaking, both meta-models fall short in modelling the SODA concept of interaction and the relations between the structural parts and dynamic parts; in particular, this applies to the relation of "participation", as we outline below.

3.1 Pros and Cons of SODA Meta-model in UML

The structural parts of the SODA methodology are well modelled. Due to its graphical vocabulary, UML is forced to model the SODA concept of interaction via its class notion, thus giving a static view of interaction, as if it were always present in the system—which is obviously misleading, since interaction has intrinsically a transient nature; indeed, capturing the transient aspects through a class diagram can be difficult.

On the other hand, UML enables the concept of "participation" to interaction to be expressed better than in OPM, thanks to the a generic tagged association: interestingly, this is possible just because interaction is represented as a class. However, distinguishing the semantic peculiarities of such associations based just on the label is not easy. For instance, although we used the same generic association for modelling the participation both in the analysis and in the design phases, in the first case the semantics is that Role participates to Interaction, while in the latter we mean that not only Agent plays an active part in interaction, but its internal state is changed by interaction, too.

3.2 Pros and Cons of SODA Meta-model in OPM

As mentioned above, the main advantage of OPM with respect to UML concerns interaction modelling, which exploits OPM's notion of process to represent the dynamic aspects. During the construction of the meta-model, however, we perceived the lack of a sort of *"tagged instrument link"* to connect objects and processes: currently, OPM's instrument link is only untagged. Such a link would have been appreciable, for instance, to express that Role *participates* to the A-Interacting process—not just that it is necessary, as expressed by the standard instrument link. In fact, necessity is a static concept, while playing an active part in interaction, as Role does, implies dynamics. A similar problem emerged in the relation between the A-Social Interacting process and the Interaction Rule object, where we could not express that Interaction Rule *governs* the social interaction—again, a more specific concept than just "being necessary".

Analogously, in the design phase, we used an *Effect link* to represent the relations between the Agent object and the X-Interacting processes; this is semantically correct because the internal state of Agent is modified by interaction, but does not express the crucial fact that Agent takes an active part to interaction, while the Effect link just expresses that its internal state is modified as a consequence of interaction. As a last issue, in the relation between the D-Social Interacting process and the Coordination Medium object, we could not express that the Coordination Medium *mediates* the social interaction by enacting the Coordination Rule—which, again, is more than just a mere "necessity".

3.3 Summing Up

Both UML and OPM proved expressive enough to capture in their meta-model the structural parts of the SODA methodology: so, for instance, the role model and the resource model are expressed in a clear way, with a specific semantics. At the same time, as partially mentioned above, both approaches present some problems, the main being that they fall short when asked to appropriately model the concept of interaction. In particular, the relation of participation, even though existing in both approaches, seems unable to capture the general concept of "participating to interaction" in a satisfactory way. This seems to indicate that while both UML and OPM methodologies are suitable to model the dynamic behaviour of *systems*, this ability is not conserved if they are used to build *meta-models*—actually, quite a different usage—although OPM expressiveness under this viewpoint is a little better than UML's.

So, we feel that neither OPM nor UML are fully adequate to capture the real essence of MAS methodologies, where interaction, in all its nuances—from a simple message exchange to mediated interaction via coordination media—is a key issue. In fact, suitably meta-modelling MAS methodologies seems to call for a specific approach, which is able to model both the structural and the dynamic parts of a methodology, and to explicitly express the idea of participating to interaction.

4 Conclusions

Several research efforts are being devoted to developing meta-models for MAS methodologies, however standardisations of methodologies for develop meta-models are not going still along way off. Although UML is often used for that purpose, meta-modelling methodologies (and in particular agent-oriented methodologies) presents several peculiarities. In this paper, we put to the test two meta-modelling approaches—UML and OPM—in order to check their expressiveness and suitability to the meta-modelling of MAS methodologies. While UML was an obvious reference for its widespread adoption, OPM was selected because even though it is an emergent methodology, it is stable and exhibits several interesting features—in particular, the explicit notion of process, and the capability of describing in a single framework all the crucial aspects of a system, instead of spreading them onto separate diagrams. Among MAS methodologies, we took SODA as our reference because it is explicitly focused on the MAS social aspects, thus putting interaction in clear evidence: this made it possible to evaluate the effectiveness of the UML and OPM approaches with respect to the issue of suitably representing interaction and, more generally, the dynamic aspects, other than the structural part.

The results of the mutual comparison between the two SODA meta-models indicates that neither approach is actually able to capture all the desired aspects in a satisfactory way. In particular, while the structural part is reasonably well modelled in both cases, the dynamic part is captured only partially—probably because both UML and OPM were introduced to model object-oriented systems, rather than systems in general; so as a consequence, they are not particularly suited to meta-modelling AOSE methodologies, especially because of their limited expressive power in capturing agent-oriented abstractions.

It should be noted that research on meta-models is also active in other field of computer science. Some papers (e.g. [14] and [15]) present meta-models for the construction of methodologies in general: meta-models are used there to "instantiate" a new methodology with the desired characteristics. Instead, our approach to meta-models moves from an existing methodology (SODA) and aims at creating a meta-model that could well capture the methodology concepts and their mutual relationship as well.

Therefore, future work will be mainly devoted to explore how to overcome such modelling weaknesses, and to devise out some meta-modelling approach to AOSE methodologies that couldfully capture the core interaction aspects.

References

1. van Hillegersberg, J., Kumar, K., Welke, R.J.: Using metamodeling to analyze the fit of object-oriented methods to languages. In: 31st Hawaii International Conference on System Sciences (HICSS 1998). Volume 5:Modeling Technologies and Intelligent Systems., Kohala Coast, HI, USA, IEEE Computer Society (1998) 323–332

2. Bernon, C., Cossentino, M., Gleizes, M.P., Turci, P., Zambonelli, F.: A study of some multi-agent meta-models. In Odell, J., Giorgini, P., Müller, J.P., eds.: Agent-Oriented Software Engineering V. Volume 3382 of LNCS., Springer (2004) 62–77 5th International Workshop, AOSE 2004, New York, NY, USA, July 19, 2004, Revised Selected Papers.
3. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: Tropos: An agent-oriented software development methodology. Autonomous Agent and Multi-Agent Systems (8) **3** (2004) 203–236
4. Gòmez-Sanz, J.J., Pavòn, J., Garijo, F.: Meta-models for building multi-agent systems. In: 2002 ACM Symposium on Applied Computing (SAC 2002), New York, NY, USA, ACM Press (2002) 37–41
5. IGENIAS: Home page. (`http://grasia.fdi.ucm.es/ingenias/metamodel/`)
6. UML: Home page. (`http://www.uml.org/`)
7. Dori, D., Reinhartz-Berger, I.: An OPM-based metamodel of system development process. In Song, I.Y., Liddle, S.W., Ling, T.W., Scheuermann, P., eds.: ER. Volume 2813 of LNCS., Springer (2003) 105–117
8. Peleg, M., Dori, D.: The model multiplicity problem: Experimenting with real-time specification methods. IEEE Transactions on Software Engineering **26** (2000) 742–759
9. Dori, D.: Object-Process Methodology: A Holistic System Paradigm. Springer (2002)
10. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The Gaia methodology. ACM Transactions on Software Engineering and Methodology (TOSEM) **12** (2003) 317–370
11. Cheong, C., Winikoff, M.: Hermes: A methodology for goal-oriented agent interactions. (2005) 4th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS05). Poster.
12. Omicini, A.: SODA: Societies and infrastructures in the analysis and design of agent-based systems. In Ciancarini, P., Wooldridge, M.J., eds.: Agent-Oriented Software Engineering. Volume 1957 of LNCS., Springer (2001) 185–193 1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000. Revised Papers.
13. Ciancarini, P., Omicini, A., Zambonelli, F.: Multiagent system engineering: The coordination viewpoint. In Jennings, N.R., Lespérance, Y., eds.: Intelligent Agents VI. Agent Theories, Architectures, and Languages. Volume 1757 of LNAI., Springer (2000) 250–259 6th International Workshop (ATAL'99), Orlando, FL, USA, 15–17 July 1999. Proceedings.
14. Henderson-Sellers, B., Gonzalez-Perez, C.: A comparison of four process metamodels and the creation of a new generic standard. Information & Software Technology **47** (2005) 49–65
15. Gonzalez-Perez, C., McBride, T., Henderson-Sellers, B.: A metamodel for assessable software development methodologies. Software Quality Journal **13** (2005) 195–214