Tackling Traceability Challenges through Modeling Principles in Methodologies Underpinned by Metamodels

Angelina Espinoza and Juan Garbajosa

Universidad Politecnica de Madrid - Technical University of Madrid,
System and Software Technology Group (SYST),
E.U. Informatica. Ctra. de Valencia Km. 7, E-28031, Spain,
http://syst.eui.upm.es; aespinoza -at- syst.eui.upm.es, jgs -at- eui.upm.es

Abstract. Traceability is recognized to be essential for supporting software development. However, a number of traceability issues are still open, such as link semantics formalization or traceability process models. Traceability methodologies underpinned by metamodels are a promising approach. However current metamodels still have serious limitations. Concerning methodologies in general, three hierarchical layered levels have been identified: metamodel, methodology and project. Metamodels do not often properly support this architecture, and that results in semantic problems at the time of specifying the methodology. Another reason is that they provide extensive predefined sets of types for describing project attributes, while these project attributes are domain specific and, sometimes, even project specific. This paper introduces two complementary modeling principles to overcome these limitations, i.e. the metamodeling three layer hierarchy, and power-type patterns modeling principles. Mechanisms to extend and refine traceability models are inherent to them. The paper shows that, when methodologies are developed from metamodels based on these two principles, the result is a methodology well fitted to project features. Links semantics is also improved.

Keywords: Traceability methodology, traceability metamodel, clabject, power-type pattern, metamodeling hierarchy, mechanisms for extension of traceability metamodels, metamodel extensibility.

1 Introduction

Traceability is considered fundamental to perform processes and tasks such as V&V, change management, and impact analysis. There is a comprehensive study about how traceability impacts directly these areas in [1] p.105-109. The motivation to start this work on traceability came from the need to provide with some traceability capabilities a document centric system/software engineering environment (SEE) designed to support the development of embedded systems [2]. Initially the work was oriented toward identifying a traceability schema, with a wide applicability spectrum, that could be integrated into the existing SEE document infrastructure. However it was found that, in line with [3], every project

needs its own tailored methodology. Among the characteristics that may influence the required attributes are team size, geographic separation, project criticality, and project priorities. The conclusion, in line with many other researchers as explained in section 2, was that it is not feasible to provide all the possible attributes, as predefined types, and the required level of granularity; and, also, the importance of link semantics, and how could that achieved, was highlighted [4].

Section 2 explains that, according to literature, traceability methodologies is one of the best approaches to cope with the various characteristics that different projects have. One of the key issues is to be able to support the reuse of traceability practices. The traceability methodology can be developed from a metamodel. As it will be discussed in 2 current metamodels also present some serious limitations. These limitations prevent developed methodologies to reach the required domain and project specificity and granularity level. Most of the approaches, such as [5–7] provide conceptual models instead of formal metamodels. Another issue is that [8] establishes three levels or layers: metamodel, methodology and project. Following this idea, it would be beneficial that there is a clear separation between these layers from a modeling point of view. Conversely, as explained in section 2, important problems are derived from not considering this issue, what is the usual situation. All these topics are regarded as open issues in a comprehensive report published by *The Center of Excellence for Traceability* [9]. This document also describes agreed traceability functionalities. Specifically, report [9] highlights the need of a formalism to represent link semantics, which is signaled as may be a non-trivial task and that link information may be domain-specific. Furthermore, we can signal that knowing and establishing the granularity of the elements being linked is relevant from a consistency perspective, and strongly influences a cost-benefit tradeoff of the traceability effort. As well, the level of granularity is influenced by the organization, domain, project, application, or even the mood of the stakeholder. Therefore, the associated challenges comprise the definition of a meta-model to represent semantic information of traceability links and providing some examples of instantiation to specific domains (D-GC1), with a granularity model. Concerning model evolution a challenge is to develop techniques for reusing traceability work products (C-GC4). Precisely, this paper introduces our proposal as a solution approach to the (D-GC1) and (C-GC4) challenges. It will be showed how these problems, that are inter-related, can be approached using two modeling principles in the metamodel definition. These principles are metamodeling layer hierarchy [10] and power-type metamodeling [11]. As it will be discussed in sect. 3.1, these metamodeling principles provide the platform to model the different project contexts, that is to develop a methodology tailored to project characteristics. Domain and project specificity is supported and contributes to enhance link semantics.

The paper is organized as follows: section 2 explains limitations of published methodologies underpinned by metamodels and justifies why the proposed modeling principles may be useful. Section 3 introduces the metamodeling hierarchy and the powertype metamodeling principles, explaining how the use of these principles in the metamodeling definition solve the problems outlined in Intro-

duction and discussed in section 2. Section 4 describes how these principles can be applied to define some parts of a metamodel called TmM. Finally sect. 5 presents the conclusion and future work.

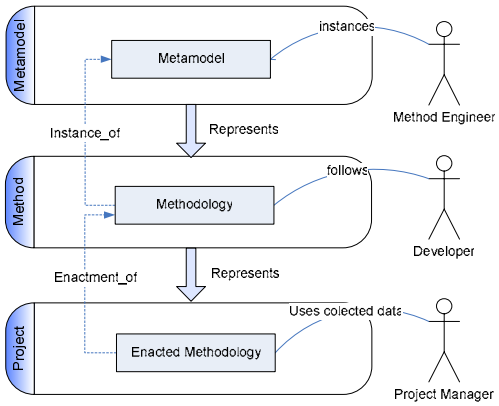

Fig. 1. The metamodeling three-layer architecture ([10] and [11]).

2 Background

The reuse of traceability practices has been recognized as a sound approach to cope with the problem of producing traceability schemas adapted to the needs of a specific project. These practices would be part of a methodology. This approach can be found in [5–7, 12–20]. Some proposals, such as [5–7, 19], suggest developing a traceability methodology from a model in an upper abstraction level; that is to say, from a traceability metamodel. For this research work, a metamodel is a model of a methodology comprising both process aspects and product aspects [11]. The role of the metamodel is to be the baseline for the development of traceability methodologies that would be fitted to project characteristics. This methodology would incorporate repeated traceability practices or patterns. Following [10], a methodology is defined as "a collection of specifications of the tasks and products that software engineers/developers are concerned with, in order to achieve a working final product delivery on time and budget". In the case of a traceability methodology, this should include a collection of specifications of the traceability tasks and products to be generated, that the software developer is concerned with. Thus, a metamodel provides a collection of specifications which represents potential elements in the methodology [10][1].

[1] This paper uses the terms "software engineer" and "software developer" as synonyms

The scenario is as follows: The method engineer defines a methodology instantiating a metamodel; the software engineer/developer follows the methodology and produces data out of his work; finally the project manager will use collected data, see figure 1. These proposals, that introduce the metamodel concept as the starting point, present a serious limitation since they cannot indicate which are the tasks, producers and work products for the stage of the methodology definition, and which for the stages of methodology usage. This limitation is closely related to the inability to support a three layer hierarchical model, encountered when working with metamodel based methodologies, and described in [8]. These layers are the metamodel that defines the methodology concepts, the methodology adapted to a project needs, and the instantiated methodology for a given project. A definition of these three layers can be found in [8]. In current approaches, these three layers are defined as a single layer. This is the case of [5,6]. For instance, concerning the traceability type concept, this is represented at the metamodel layer as the class *TraceabilityType*, from this the *satisfies* type is defined in the methodology layer. Then, in the project layer several links can be created from the satisfies type, when is necessary to indicate that an artifact satisfies or realizes another one. Concerning this limitation, i.e. providing a single layer instead of three modeling layers, we encounter some exceptions. That is the case of Kelleher in [7] and Pohl et al. in [19], which do separate the metamodeling levels. Kelleher's approach [7] provides a metamodel, namely TRAM, based on reused traceability practices, and an interesting four-layer traceability process architecture, to describe the traceability process. However, even though the traceability process enactment is included, the TRAM metamodel, as presented in [7], models exclusively those items needed to define the traceability methodology, but not those other items corresponding to the enactment of the resulting methodology. Pohl et al. in [19] provides a metamodel to support the method-driven capture of the defined trace information and project-specific, presenting a three-layer traceability process model. This approach mainly tackles the acquisition process of traces. It also introduces traceability items involved in that process, such as dependency links. This approach is sound in the sense that it describes not only the traceability steps but also, their interrelations to the development process steps. However, the metamodel does not accomplish totally with our traceability requirements. The metamodel describes exclusively the dependency links between the items generated by the process steps, without modeling which items should be used to define the traceability methodology, at the method layer. This will make more difficult the methodology customization for each project.

Other limitation founded in current related literature, is that the semantics for the traceability metamodel classes is defined shallowly. This has an impact on the link semantics. Classes are provided with a very small number of attributes and no extension mechanisms are provided either, as in the case before. A concept that is introduced within this paper is the project *context*, similar, but in a higher layer of abstraction, to the enacting context [21,22]. The project context is defined in this research as the set of attributes that define a project

scenario: people involved, system artifacts types used by the people involved, development activities to perform by the people involved, and time constraints. For instance the project context in which a methodology is defined must exist prior to the project context to perform that defined methodology. Frequently current approaches such as [5–7, 19, 23] provide not just a very basic set of core artifacts, *metaconcepts*, on which the methodology model could be built, but a predefined set of concepts such as a given product lifecycle, or roles or meeting, either informal or formal. This is not flexible enough. Actually the metamodel should not preclude the methodology concepts. The methodology should have the possibility to define its own concepts, starting from a set of "core artifacts". The limitations described above can be addressed as long as metamodels are defined using the principles presented in section 3. The principles described are close related to each other in the same way that limitations are also related. At the same time, they do not provide any mechanism to extend these metamodels.

3 Modeling principles

This section briefly introduces the modeling principles on which our traceability metamodel is based on: the metamodeling layers hierarchy, and the clabject and power-type metamodeling principles. How these principles are useful for specifying traceability metamodels without the limitations, described in section 2, will be explained.

3.1 Three-level Metamodeling Hierarchy

A problem when a methodology is modeled is to include in the same model, support to define the resulting methodology's items and at the same time gives guidance about how to enact those items. That implies that each modeled concept requires attributes to support its definition as part of a generated methodology, and at the same time attributes which give guidance to enact it. An approach to tackle this is the three-level metamodeling hierarchy, as in [10]. Figure1 provides a simple modeling abstraction levels separation based on the three-level metamodeling hierarchy. In the figure, the method engineer defines a methodology instantiating a metamodel (top layer), and the developer follows or enacts that defined methodology (middle layer). Then, the project layer can use the collected data of the enacted methodology, to make projects and business decisions. This layer hierarchy provides a natural platform to model the separation between the method engineer work, from that of the software engineer, who enacts such methodology. Formally, in a three-level metamodeling hierarchy such that presented in [10], the *metamodel layer* comprises a metamodel and all the specifications contained in it. A methodology including all the specifications contained in it, is defined as the *method layer*. The instances of such methodology, corresponding to specific projects or endeavors, are defined as the *project layer*. The recently approved standard [8] can be considered an evolution from it. Alternative approaches such as [24–26] were also considered but they seemed

to be less adequate for the kind of problems to be addressed. Therefore, our requirement about the stakeholder's tasks separation, is accomplished by just creating the traceability metamodel, following the three-level hierarchy guidelines as stated in [10]. The three level hierarchy will be applied in combination with the power typing metamodeling technique in subsections 3.2 and 3.2, to model the abstraction levels of our traceability methodology.

3.2 Powertype and Clabject Metamodeling

Powertype patterns and clabjects are a powerful mechanism to define project-specific methodology elements, through a three-layer modeling hierarchy. A powertype pattern is an entity of a metamodel composed by two classes [11]. One class is the *powertype* class that is used at the methodology layer to define methodology items. The other class is the partitioned class, used at the project layer to create "real" objects, e.g. test cases or traceability links. Figure 2 shows a pattern in the top square: the *TestCase*. Patterns once instantiated generate a clabject, this modeling principle is used to express a project concept with two facets: a class and an object, but both describing the same concept, and with the same name. This mechanism of two classes is to ensure the inheritance from the metamodel layer, up to the project layer, without disregarding the strict metamodeling formalisms. That is, by using the class facet it is possible to extend the concept up to the project layer. More details about compliance with strict metamodeling in [10, 27]. Figure 2 shows a clabject in the middle square.

The instantiation process of a power-type pattern is strongly related to the three-layer metamodeling hierarchy. That is the pattern belongs to the metamodel layer, the pattern instantiation into a clabject belongs to the method layer, and its enactment to the project layer. To explain how can be created a customized methodology's item from a pattern, consider the *TestCase* powertype pattern example in figure 2.

In the figure 2, at the metamodel layer, the method engineer use the *TestCase* pattern to define all the customized test cases types, according to the given project. The *TestCaseKind* class of the pattern support this definition. For instance, the middle square (methodology layer) on the right shows a the *UnitTestCase* instance, which represents test cases of the unit type. The middle square on the left show the *UnitTestCase* class, which is used to create specific unit tests cases, such as *Unit1_TC1* and *Unit2_TC2* at the project layer. Together *UnitTestCase* class and object compose a clabject. The *TestCase* class of the pattern (top square) provides attributes to keep the results of these unit test cases execution. At the project layer, the project manager use the collected results (bottom square). The two classes of a powertype pattern, provides an excellent mechanism for metamodel extensibility. The class in the methodology layer, has some attributes for method engineer to support the methodology definition. Whereas class attributes in the project layer support the use of patterns during the methodology enactment, when the methodology is used by developers. This clear model of a project concept and its use contexts, strongly facilitates

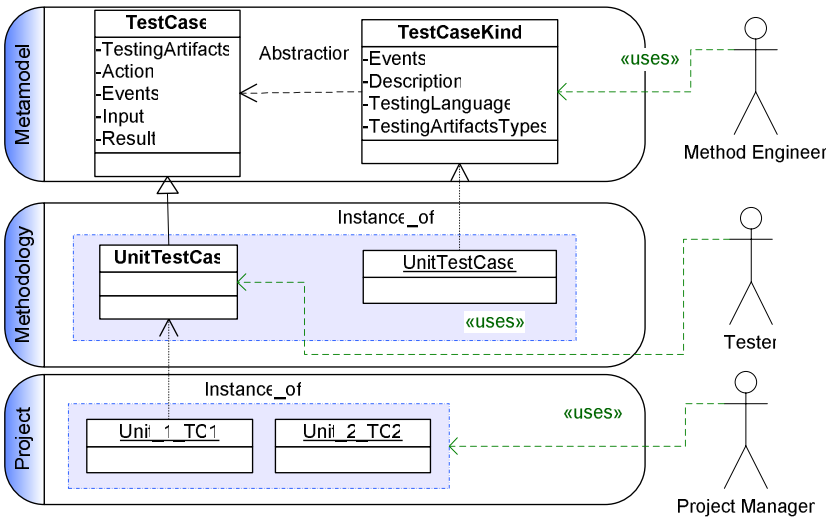

Fig. 2. *TestCase* powertype pattern instantiation process (based on [10] fig.7).

the traceability metamodel extension. More traceability issues can be easily included in the metamodel since the already modeled items are well-defined. Thus, the maintenance process of the metamodel is facilitated as well.

4 The Modeling Approach in TmM

This section introduces how the principles described in 3 are applied in building some parts of a traceability metamodel named TmM (Traceability metaModel for methodology definition) [28]. From TmM it will be possible to develop traceability methodologies, according to specific project features. The *Software Engineering Metamodel for Development Methodologies* (*SEMDM*) of the ISO 24744 standard [8], was considered as the formal basis to specify TmM. The reason is that, ISO/IEC 24744 provides the needed three-layer hierarchy, and it is also based on the power-type and clabject modeling principles. The advantage of using ISO/IEC 24744 as the baseline to develop metamodels, is that those resulting metamodels have a common understanding of the metamodeling language and notation. Therefore, TmM has been extended from SEMDM following the extension rules provided in ISO/IEC 24744. TmM keeps the same metamodeling hierarchy than SEMDM and the same graphical conventions that SEMDM diagrams. Top classes connected to bottom classes through dashed arrows with the "abstraction" label, are the TmM powertype patterns. Following the SEMDM hierarchy, powertype classes are in the method layer (top classes), and partition classes (bottom classes) are in the endeavor layer. The instantiation process

which describes how to use top and bottom classes through the layers, to generate traceability methodologies, is detailed in sect. 3.2. TmM has been enriched with some features taken from [5, 29–31].

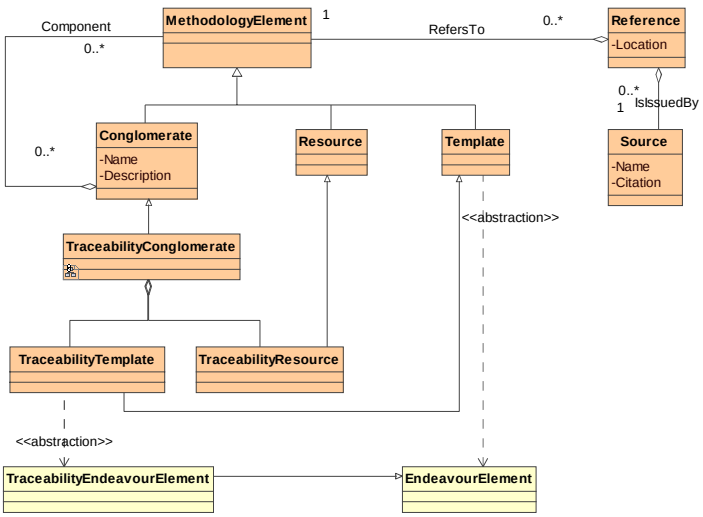

Fig. 3. TmM root class extended from the SEMDM's Conglomerate class.

4.1 Traceability metaModel Guidelines

TmM provides the baseline for the systematic and formal definition of a traceability methodology. TmM includes three aspects: the process to follow, the intermediate and final products, and the roles involved in the process. TmM is the metamodel from which the customized traceability methodology is instantiated. TmM provides as well, the model of the traceability methodology enactment. The TmM instantiation, into the resulting traceability methodology, is according to the software development methodology, life cycle model, application domain and the project business needs. TmM covers the whole process lifecycle, and does not prescribe the process model. The TmM root class, according to the SEMDM extension rules, is the *Conglomerate* class which "is a collection of related methodology elements that can be reused in different methodological contexts" [8] p.p. 19. A traceability conglomerate will be derived from this class. That traceability conglomerate is composed by the traceability methodology items, which are classified into *traceability resources* and *traceability templates*. These elements are represented in the metamodel, by the *TraceabilityTemplate* pattern and by the *TraceabilityResource* class. This specialization is shown in

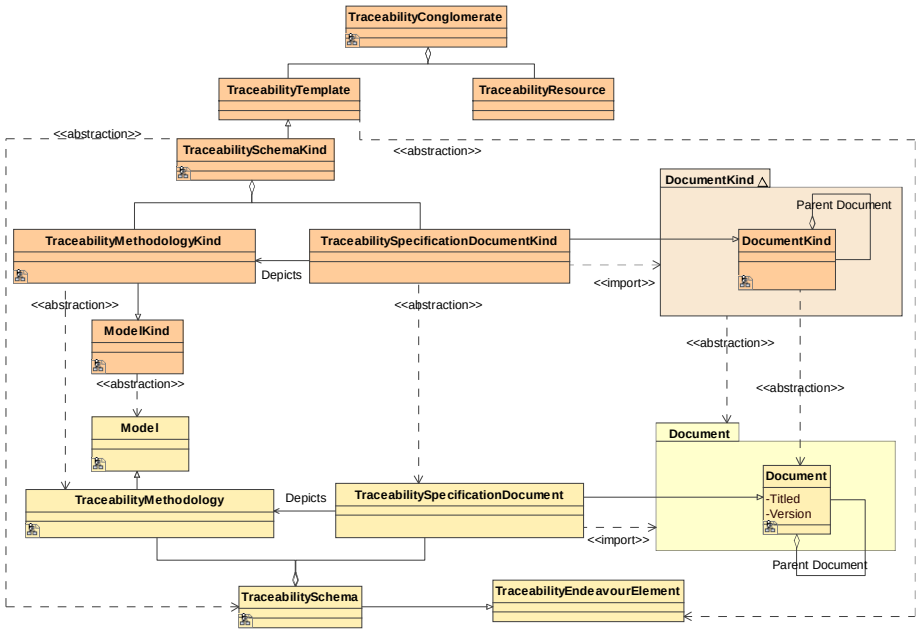

Fig. 4. TmM TraceabilityMethodology and TraceabilitySpecificationDocument power-type patterns.

figure 3. The TraceabilityTemplate pattern represents the traceability methodology elements that are used by instantiation at the endeavor level (i.e. project's templates). The TraceabilityResource class represents the traceability methodology elements that are defined at the method layer to later be used "as is" at the endeavor level (i.e. project's resources).

4.2 TmM Fundamental Parts and Core classes

The traceability schema concept is modeled in TmM, as the *TraceabilitySchema* pattern. Figure 4 shows this class as an aggregation of the patterns: *TraceabilityMethodology* and *TraceabilitySpecificationDocument*. Here, it is presented how power-type patterns supports us to model the traceability schema concept, at both layers, the methodology definition and its enactment. That is, the *TraceabilitySchemaKind* class represents, at the methodology layer, all potential traceability schemas than can be created for a specific project. This class is used by the method engineer to define the traceability schema. Whereas at the endeavour layer, the *TraceabilitySchema* class represents a specific traceability schema for that project. This class is used by the software engineer to tailor the defined traceability schema in the project. Similarly, the *TraceabilityMethodology* pattern allows us to model in the *TraceabilityMethodologyKind* class, all possible

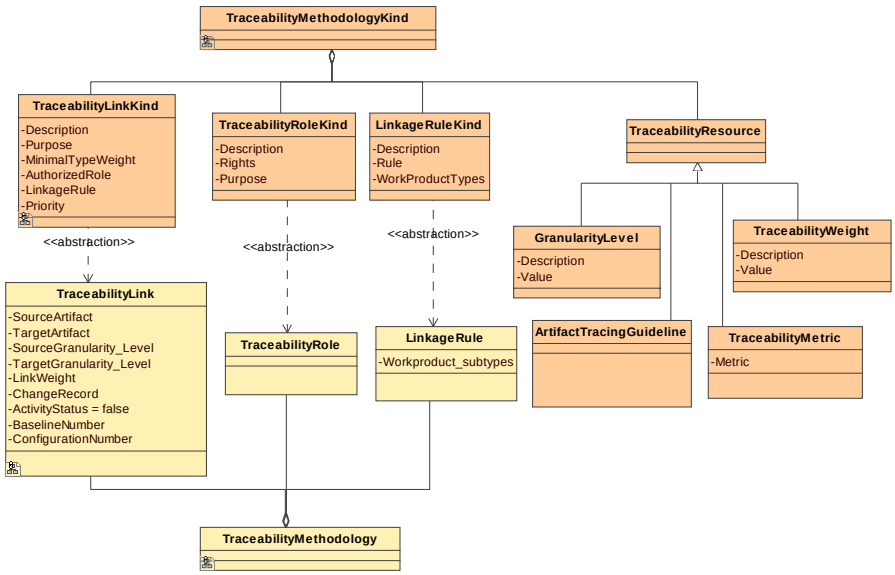

Fig. 5. TmM Core patterns and classes.

methodologies that can be generated using TmM, for a specific project. *TraceabilityMethodology* class represent the specific traceability methodology to be followed in the project. The *TraceabilitySpecificationDocument* pattern models, through the *TraceabilitySpecificationDocumentKind* class, all possible traceability documents that can be created which specify the traceability guidelines to be followed during the methodology enactment. Whereas, the *TraceabilitySpecificationDocument* class represent only one document, which keeps that fundamental data of the tailored traceability methodology.

TmM includes a set of core classes and patterns, which once instantiated according to the project features, constitute the resulting traceability methodology. These core classes were determined fundamental to compose a traceability methodology, however the rationale which justifies these decisions are out of scope of this paper. This paper will just present the core set which potentially can conform a traceability methodology, justifying how the chosen modeling principles improve the design of this metamodel respect to other current approaches. Therefore, the classes are divided into templates patterns and resources classes. The traceability templates patterns describe, those traceability elements of a methodology that will be instantiated in the project level (at Endeavor layer). The template patterns are: *TraceabilityLink*, *TraceabilityRole*, *LinkageRule*. Figure 5 shows the metamodel diagram for the core classes.

The *TraceabilityLinkKind* class of the pattern *TraceabilityLink*, represents all potential traceability link types, that might be defined in the traceability

methodology to create traces. Types such as task, resource or goal dependency between two system objects, as well as, evolution, satisfaction or rationale types, to mention some. The traceability type's definition will be according to the project characteristics, for instance trustworthy systems need to make emphasis on different variations of the dependency and rationale types, to maintain a high level of system quality assurance. The *TraceabilityLink* class of the same pattern, represents the traceability types defined to that project, from which, all traceability links will be created during the project development. Note that both pattern's classes have different attributes, which will be used in the methodology definition and in the methodology enactment. The *TraceabilityRoleKind* class of the *TraceabilityRole* pattern, expresses all the roles that can be applicable to control the traceability information accesses. From this class will be defined the traceability roles applicable to a project, during the methodology definition. The *TraceabilityRole* class expresses just a particular traceability role. These two classes will be used by the method engineer to generate all the traceability roles. Similarly, the *LinkageRuleKind* class of the *LinkageRule* pattern, expresses all possible linkage rules that might be defined to automate the links acquisition process. The *LinkageRule* class expresses a customized linkage rule, with which the traceability links will be created. These two classes will be used by the method engineer to generate all linkage rules for a particular project.

The traceability resources classes describe those traceability artifacts used "as is" in the project. That means, they are used during the entire project layer, exactly as they were configured at the methodology definition layer. Resource classes are: *GranurarityLevel*, *ArtifactTracingGuideline*, *TraceabilityWeight* and *TraceabilityMetric*. Thus, the traceability resource classes are not instantiated at the project layer, just in the methodology layer. Figure 6 shows a brief example of the *LinkageRule* pattern instantiation, process described in sect. 3.2. This specific linkage rule would be part of the traceability methodology for a given project, see method layer. The rule has been designed according to project requirements, about to improve the automation of the link acquisition between functional requirements and model which realizes them. Then this rule can be implemented by a traceability type, specialized it to create links between functional requirements and classes.

5 Conclusion and Future Work

This paper has discussed how useful is to introduce two complementary modeling principles, three layer hierarchy and powertype patterns, as a baseline to define traceability methodology metamodels. The main contribution of TmM is that it totally provides a solution approach to the challenge numbered as *D-GC1* of *Center of Excellence for Traceability Technical Report (COET-GCT-06-01)* [9]. TmM provides a complete solution approach to the required meta-model to represent semantic information of traceability links, as is demanded in D-GC1. As well, TmM overcomes some of the current model limitations including, challenges (D-GC1), (C-GC4), (C-GC4.2) from [9], as it was described in section 1.

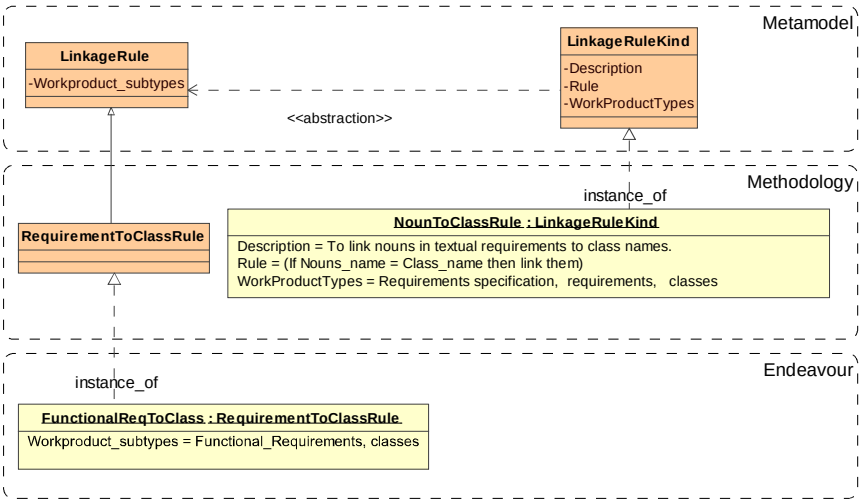

Fig. 6. Instantiating the LinkageRule Class.

Based on the support provided by SEMDM [8], the powertype metamodeling offered naturally the platform to model a traceability concept as a pattern, which allows, through its instantiation, to generate two classes: a class to be used at the methodology layer, and another class to be used at the project layer. Both classes represent the same traceability concept but with a different purpose. It is also possible to define a set of traceability core concepts, not by extension but by intension. Thanks to providing mechanisms to extend the metamodel, this set of core concepts is the base to define, at methodology level, those features specific to a domain or project, and at the desired level of granularity. That is, the metamodel does not preclude any more which concepts will be present in the methodology. At present TmM specification is being finalized and instantiated to challenging applications and domains such as agile development of large software intensive systems. After TmM validation it will be implemented in tools to support the embedded systems development.

Acknowledgement. This research work has been partially sponsored by the Mexican National Council of Science and Technology (CONACyT), the projects OVAL/PM TIC2006-14840 from Spanish MEC, and FLEXI ITEA2 6022 FIT-340005-2007-37 from Spanish MINER.

References

1. Dahlstedt, Å., Persson, A.: 5. In: Requirements Interdependencies: State of the Art and Future Challenges. Springer-Verlag, Berlin Heidelberg (2005) 95–116

2. Alarcon, P., Garbajosa, J., Crespo, A., Magro, B.: Automated integrated support for requirements-area and validation processes related to system development. In: INDIN 04, ISBN 0-7803-8513-6 (2004) 287–292
3. Cockburn, A.: Selecting a project's methodology. IEEE Softw. **17**(4) (2000) 64–71
4. Espinoza, A., Alarcón, P.P., Garbajosa, J.: Analyzing and systematizing current traceability schemas. In O'Conner, L., ed.: 30th Annual IEEE/NASA SEW 2006, Proceedings, Columbia, Maryland, IEEE Computer Society (April 2006) 21–32
5. von Knethen, A., Grund, M.: Quatrace: A tool environment for (semi-) automatic impact analysis based on traces. In: Proceedings of the ICSM '03, Washington, DC, USA, IEEE Computer Society (2003) 246
6. Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. Software Engineering, IEEE Transactions on **27**(1) (Jan 2001) 58–93
7. Kelleher, J.: A reusable traceability framework using patterns. In: Proceedings of the TEFSE '05, New York, NY, USA, ACM Press (2005) 50–55
8. ISO/IEC-JTC1/SC7: ISO/IEC 24744:2006 Software Engineering - Metamodel for Development Methodologies. ISO/IEC. (2006)
9. Antoniol, G., Berenbach, B., Eyged, A., Ferguson, S., Maletic, J., Zisman, A.: Problem statements and grand challenges in traceability. Technical Report COET-GCT-06-01-0.9, Center of Excellence for Traceability (Sep 10, 2006)
10. Henderson-Sellers, B., Gonzalez-Perez, C.: The rationale of powertype-based metamodelling to underpin software development methodologies. In: Proceedings of the APCCM '05, Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2005) 7–16
11. Gonzalez-Perez, C., Henderson-Sellers, B.: A powertype-based metamodelling framework. Software and System Modeling **5**(1) (2006) 72–90
12. Asuncion, H.U., François, F., Taylor, R.N.: An end-to-end industrial software traceability tool. In: Proceedings of the ESEC-FSE '07, New York, NY, USA, ACM (2007) 115–124
13. Maeder, P., Philippow, I., Riebisch, M.: A traceability link model for the unified process. In: Proceedings of the SNPD '07, Washington, DC, USA, IEEE Computer Society (2007) 700–705
14. Tabares, M.S., Moreira, A., Anaya, R., Arango, F., Araujo, J.: A traceability method for crosscutting concerns with transformation rules. In: Proceedings of the ICSEW '07, Washington, DC, USA, IEEE Computer Society (2007) 95
15. Morris, S.J., Gotel, O.C.Z.: Model or mould? a challenge for better traceability. In: Proceedings of the MISE '07, Washington, DC, USA, IEEE Computer Society (2007) 1
16. Tekinerdogan, B., Hofmann, C., Aksit, M.: Modeling traceability of concerns in architectural views. In: Proceedings of the AOM '07, New York, NY, USA, ACM (2007) 49–56
17. Fletcher, J., Cleland-Huang, J.: Softgoal traceability patterns. In: Proceedings of the ISSRE '06, Washington, DC, USA, IEEE Computer Society (2006) 363–374
18. Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y.: Model traceability. IBM Syst. J. **45**(3) (2006) 515–526
19. Pohl, K., Dömges, R., Jarke, M.: Towards method-driven trace capture. In: Proceedings of the CAiSE '97, London, UK, Springer-Verlag (1997) 103–116
20. von Knethen, A.: A trace model for system requirements changes on embedded systems. In: Proceedings of the IWPSE '01, ACM Press (2001) 17–26
21. Feiler, P., Humphrey, W.: Software process development and enactment: concepts and definitions. Software Process, 1993. Continuous Software Process Improvement, Second International Conference on the (Feb 1993) 28–40

22. Fuggetta, A.: Software process: a roadmap. In: Proceedings of the ICSE '00, New York, NY, USA, ACM (2000) 25–34
23. Kelleher, J., Simonsson, M.: Utilizing use case classes for requirement and traceability modeling. In: MS'06: Proceedings of the 17th IASTED, Anaheim, CA, USA, ACTA Press (2006) 617–625
24. Object Management Group: Meta Object Facility (MOF) Core Specification - Version 2.0. Object Management Group, Inc. (OMG), OMG Headquarters, Needham, MA, USA. (January, 2006)
25. Object Management Group: Software Process Engineering Metamodel Specification - Version 1.1. Object Management Group, Inc. (January 2005)
26. Firesmith, D.G., Henderson-Sellers, B.: The OPEN Process Framework. Addison-Wesley (2002)
27. Atkinson, C., Kühne, T.: Profiles in a strict metamodeling framework. Sci. Comput. Program. **44**(1) (2002) 5–22
28. Espinoza, A.: TmM metamodel for traceability methodologies definition. internal technical report. Technical report, Universidad Politecnica de Madrid (March, 2008)
29. Spanoudakis, G., Zisman, A., Pérez-Miñana, E., Krause, P.: Rule-based generation of requirements traceability relations. Journal of Systems and Software **72**(2) (July 2004) 105–127
30. Costello, R.J., Liu, D.B.: Metrics for requirements engineering. J. Syst. Softw. **29**(1) (1995) 39–63
31. Becker, D.: Measuring requirements traceability from multiple angles at multiple lifecycle entry points. In: Proceedings of the RE 2003., IEEE (8-12 Sept. 2003) 291