Standardizing Methodology Metamodelling and Notation: An ISO Exemplar

Brian Henderson-Sellers[1] and Cesar Gonzalez-Perez[2]

[1] Faculty of Information Technology, University of Technology, Sydney, PO Box 123, Broadway, NSW 2007, Australia
[2] Instituto de Estudios Galegos Padre Sarmiento, Consejo Superior de Investigaciones Científicas, San Roque, 2, 15704 Santiago de Compostela, Spain
brian@it.uts.edu.au, cesargon@verdewek.com

Abstract. Standardization within a discipline often reflects its maturity. Within software engineering, standardization occurs in many areas – here we focus on a recent ISO standard that has been developed for a methodology metamodel: the Software Engineering Metamodel for Development Methodologies, ISO/IEC 24744. Since its publication as a pure metamodel (represented by several UML-style class diagrams) in February 2007, a follow-on project has been established to provide a complementary notation for all the methodological elements, both within the method domain and the endeavour domain. Here, we discuss not only the technical details but also the process by which standardization occurs.

Keywords: Metamodelling, methodology, notation, standards.

1 Introduction

Maturity in a discipline is often reflected when standardization occurs. The prime international standardization body is the International Organization for Standardization or ISO, headquartered in Geneva in Switzerland.

Standardization of methodological elements within software engineering assists development teams in following the same approach leading to interoperability at all levels and an increase in their efficiency and productivity. Often companies using an ISO standard have a head start in obtaining contracts, particularly with government departments worldwide, since adherence to an ISO standard can be indicative of a quality-focussed approach to software development. While there are many ISO standards relevant to software engineering, here we will focus on just one of these: the Software Engineering Metamodel for Development Methodologies, ISO/IEC 24744 [1].

All standards go through a rigorous development process. In ISO, software engineering standards mostly fall under the purview of Subcommittee number 7 (SC7) of the Joint Technical Committee 1 of ISO and IEC (International Electro-technical Commission). Within SC7, each standardization project is attributed to one of the several working groups for development. Development stages have a six-month duration during which work is undertaken to develop an initial, raw proposal into a standard acceptable to a very wide community. At each of these six-monthly stages, a vote is taken by National Bodies before the embryonic international standard can

R. Kaschek et al. (Eds.): UNISCON 2008, LNBIP 5, pp. 1–12, 2008.

proceed to the next stage, the voting community growing larger at each six month milestone. Final approval for this particular standard (24744) occurred after the November 2006 meeting in Kyoto and the standard was published February 13 2007.

In this paper, we focus on the technical details of the ISO/IEC 24744 International Standard. We first introduce metamodelling basics (Section 2) and then explain how these are reflected in the International Standard itself (Section 3). In Section 4, we go a little beyond the published standard to describe a notation that is currently the topic of a New Work Item (NWI) within SC7. This, in due course, is aimed at created an addendum to 24744 that will describe a recommended notation for describing methods and processes conformant to the metamodel in the existing standard.

2 Metamodelling Basics

Metamodelling is cognitively challenging and often ill understood (see examples in [2]). A metamodel is a model of models [3,4] where a model is *a statement about a given subject under study (SUS), expressed in a given language* [5]. The SUS that is the focus of our current study is that of methodologies so we are concerned with metamodels that are models of methodologies. These models provide the underpinning (quasi-formalism) for methodologies that exist in the real world e.g. the methodology used by a particular company on their projects (or endeavours in the wider sense). The relationship between a model and a metamodel is thus that the metamodel elements *represent* the model elements [6]. Together, the elements in the metamodel are the modelling language that can be used to describe such conformant models.

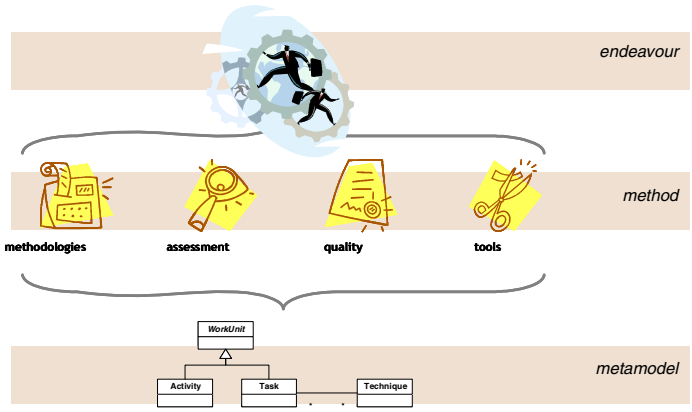

Fig. 1. Three layer architecture of recent standards (after [7])

This suggests some sort of layering or multi-domain representation – the metamodel domain, the methodology domain and, hinted at above, the endeavour domain. The metamodel domain is usually composed of standardized conceptual constructs, the method domain contains real-world methodology elements (methods, tools, coding standards) while the endeavour domain represents actual processes in use by the

people on a particular project/endeavour (Figure 1). Typically, standardization occurs in the metamodel domain.

3 An Exemplar – The ISO/IEC 24744 International Standard

ISO/IEC 24744 [1] is an International Standard that defines a metamodel for development methodologies. Although it is geared towards *software* development methodologies, there is nothing in it that prevents it from being applied to systems development methodologies or even other areas. The standard exists in the metamodel domain of Figure 1 and contains a number of elements that model entities in the method domain *plus* a number of element that model entities in the endeavour domain. This is unlike other method/process-focussed standards that only model entities in the method domain.

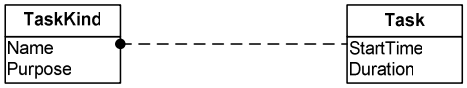

Fig. 2. Task/TaskKind powertype pattern in ISO/IEC 24744 (after [8])

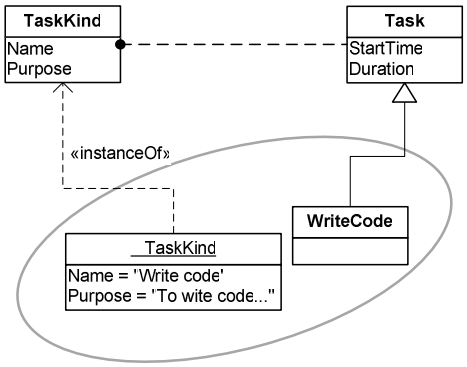

Fig. 3. The "instantiation" of a powertype pattern. A regular object is instantiated from the TaskKind class and a regular class is obtained by subtyping the Task class. Together these form a clabject (depicted by the ellipse) (after [8]).

To accomplish this duality, as discussed in [8] in more detail, ISO/IEC 24744 eschews the strict metamodelling paradigm and the well-known instantiation-related, multiple layering architecture of the Object Management Group and introduces the powertype concept of Odell [9] as a core model element in the metamodel. Pairs of concepts, one representing entities in the method domain and one representing entities in the endeavour domain, are abstracted to form powertype patterns [10] (Figure 2). The endeavour-focussed element (Task in Figure 2) represents elements (actual tasks) in the endeavour domain, while the method-domain element is modelled, in the example, by a conceptual class called TaskKind that represents all kinds of tasks that could possibly exist in the method domain. In the method domain, an instance of

TaskKind is also a subtype of Task (where TaskKind and Task are both in the metamodel domain) (Figure 3). This means it has both class-like and object-like characteristic. Such an entity has been called a "clabject" by Atkinson [11].

ISO/IEC 24744 makes wide usage of clabjects and powertype patterns. Its scope includes work units, work products, producers, stages and model units (for details see relevant subsection in Section 4). The overall architecture of ISO/IEC 24744 is shown in Figure 4. Powertype patterns are used for subtypes of Template and Endeavour elements, whilst regular instantiation semantics are sufficient for the subtypes of Resource. The differentiation between templates and resources is practical, with clear semantics (see also [12]). Nevertheless, it is appropriate for both template and resource concepts to be implemented in the metamodel equally, as regular classes. It is their usage that is different. Furthermore, we must emphasize that the instances of resources and templates (collectively called methodology elements) reside in the Method domain, in contrast to the instances of endeavour elements, which reside in the Endeavour domain.

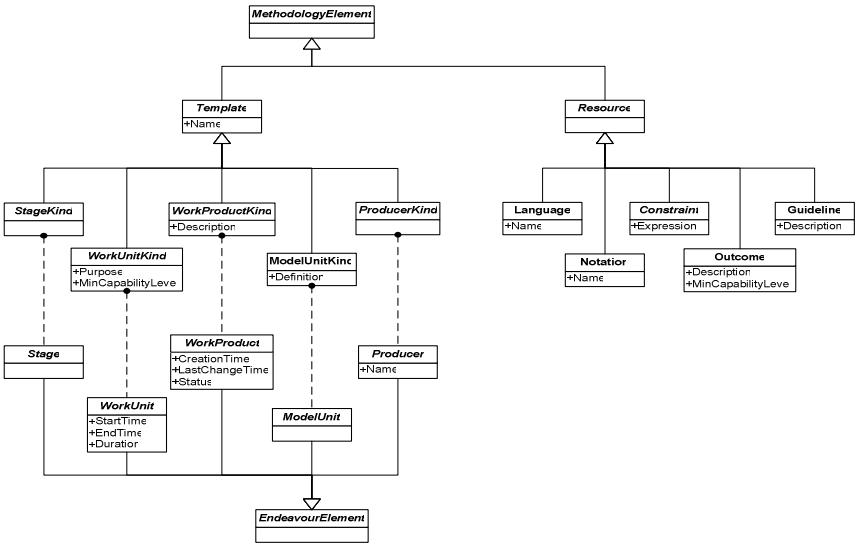

Fig. 4. Overall architecture of ISO/IEC 24744 (after [1])

4 Adding a Notation

While the published International Standard [1] does not contain any recommended notation (but just an abstract syntax and semi-formal semantics), at the time of writing, a new project under Working Group 19 of the SC7 subcommittee of ISO, has been commenced to standardize a notation especially designed to depict ISO/IEC 24744 concepts. Here is a summary of the progress to date on this soon-to-be-standardized set of symbols.

The proposed notation is mainly graphical and supports most of the template concepts found in 24744 (left-hand side of Figure 4).

Although the metamodel of ISO/IEC 24744 contains classes that represent concepts from the Method domain and classes that represent concepts from the Endeavour domain, the notation so far gives more support for the former. It has been designed to depict concepts from the Method domain in order to help the methodologist or method engineer represent method fragments and complete methodologies. The notation does not yet comprehensively cover the Endeavour domain, i.e. it is not capable of depicting concepts pertaining to specific enactments of any methodology. There is an exception to this, namely enactment diagrams. Enhancement of the notation presented here to cover Endeavour domain concepts is for future consideration in the ISO standardization process.

The notation has been designed to be easy to draw by hand as well as using a drawing package on a computer. Special care has been taken to follow semiotic principles, as suggested in [13, 14] in choosing symbols that convey the underlying concept, at least in most situations and to readers of most cultures and backgrounds. In addition, the symbols adopted by the notation exhibit visual resemblance (based on shapes and colours) to each other that mimic the structural relationships of the underlying concepts in the metamodel, establishing common "visual themes" for closely related concepts. Colour is extensively used by this notation, since it helps identify symbols and symbol patterns with ease when displayed on a computer display or a colour printout. However, care has been taken to guarantee that greyscale and black and white versions of the same symbols are perfectly readable and identifiable. In this regard, colour does enhance diagram readability but can be avoided without great loss.

This notation introduces the novel concept of *abstract symbols*, i.e. symbols that depict instances of abstract classes. In principle, most notations only include symbols to depict instances of concrete classes, since abstract classes do not have direct instances. However, it is suggested that in some scenarios it is convenient to represent an entity in a diagram for which only the abstract type is known. For example, consider the case where a work product kind representing a certain system must be depicted in a diagram. A notation with only concrete symbols would force the designer to choose a specific concrete type of work product (document, model, software item etc.) in order to depict it. This notation thus includes an "abstract work product kind" symbol that allows the designer to depict the above-mentioned system without specifying whether it is software, hardware, composite etc. Abstract symbols usually consist of the simple shape from which all the concrete symbols in a visual theme are generated.

4.1 Stages

A stage is a managed time frame within an endeavour. Stages are partitioned into stage kinds by the StageKind class (Figure 4). In addition to the abstract StageWithDurationKind, three of its subtypes are covered by this notation: TimeCycleKind, PhaseKind and BuildKind. These are represented by broad symbols that can contain other elements. A rectilinear theme has been chosen. Colours, when used, for all these symbols belong to the blue-purple range (Figure 5).

The symbol used to depict a time cycle kind is composed of two navy blue horizontal brackets with their right-hand side end bent outwards, simulating a truncated arrow head. These brackets delimit a quasi-rectangular area within which symbols for other stage kinds can be shown. This symbol tries to convey the meaning that a time

cycle kind comprises a collection of other stage kinds - hence the bracket analogy. A similar argument underpins the shape for PhaseKind and the more iteratively focussed BuildKind with its double point, resembling dual arrow heads. It too can serve as a container for other elements, which can be shown inside it.

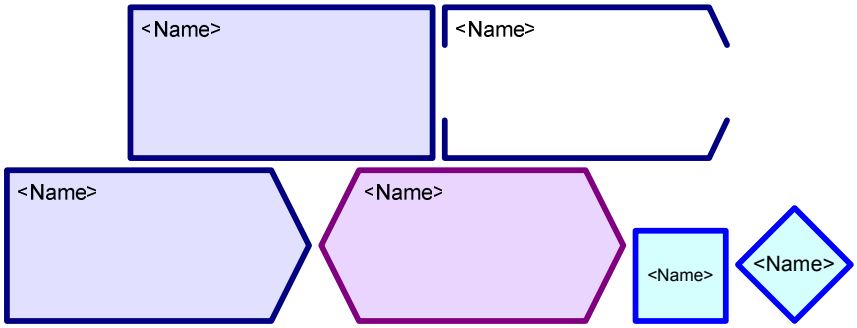

Fig. 5. The "stage family" has six symbols representing StageWithDurationKind, TimeCycle-Kind, PhaseKind, BuildKind, InstantaneousStageKind and MilestoneKind

In contrast, an instantaneous stage is a managed point in time within an endeavour. InstantaneousStageKind is thus another abstract class, depicted by an abstract symbol (a square). This symbol tries to convey the idea of a point in time, hence the similarity with other stage-related symbols (overall rectangular shape) but a smaller area. Since instantaneous stages are points in time rather than time spans, no other symbols need to be shown inside this one.

The symbol used to depict a milestone kind is a small square rotated 45 degrees. This symbol tries to convey the idea of an event-marking point in time, hence the similarity with other stage-related symbols (points facing left and right) but a smaller area. It also resembles the symbol used by Microsoft PowerPoint™ and other project management software tools to depict milestones. Since milestones are points in time rather than time spans, no other symbols will need to be shown inside this one.

Temporality is represented by the "pointed" nature of the left and/or right sides of these symbols – other than the two symbols (rectangle and square) representing the abstract supertypes (StageWithDurationKind and InstantaneousStageKind).

4.2 Work Units

A work unit is a job performed, or intended to be performed, within an endeavour. Work units are partitioned into work unit kinds by the WorkUnitKind class according to their purpose within the endeavour (Figure 4).Three subtypes of WorkUnitKind are covered by this notation: ProcessKind, TaskKind and TechniqueKind. None of these concepts are involved in whole/part relationships that may need nesting of symbols, so the symbols chosen to depict them are basic curvilinear shapes and easily resizeable to accommodate long names or abbreviations. However, since work unit symbols often occur on the same diagram, as well as their contrasting shape outlines, an additional colour coding can be added so that the user can readily discriminate between them in process diagrams that support colour (see Figure 6).

A process is a large-grained work unit that operates within a given area of expertise within the endeavour. The symbol used to depict a process kind is a rounded rectangle or "roundangle". When colour is available, line colour is medium green and fill colour is light green. In some contrast, a task is a small-grained work unit that focuses on what must be done in order to achieve a given purpose within the endeavour. The symbol used to depict a task kind is an ellipse with more intense shades of green than ProcessKind. TechniqueKind, shown with similar colours to ProcessKind, details *how* tasks are to be accomplished. The minimum capability level of work unit kinds is optionally shown inside a circle toward the top or left as indicated ("n" in the figure).

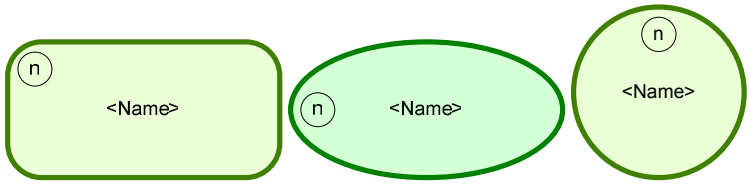

Fig. 6. The WorkUnitKind family of icon shapes and colours: ProcessKind, TaskKind and TechniqueKind

4.3 Work Products

A work product is an artefact of interest for the endeavour. Work products are partitioned into work product kinds by the WorkProductKind class according to the nature of their contents and the intention behind their usage. Five subtypes of WorkProductKind are covered by this notation: DocumentKind, ModelKind, SoftwareItemKind, HardwareItemKind and CompositeWorkProductKind. All of them are represented by tall symbols with colours in the red-pink range. The former four correspond to "simple" work products, and therefore are represented by reasonably simple rectangular shapes, but with different adornments. CompositeWorkProductKind, on the other hand, uses a symbol that tries to convey a sensation of depth to represent multiplicity.

The most general is the abstract symbol for WorkProductKind itself. This is depicted by a vertically-oriented rectangle. Line colour is red and fill colour is pale red/pink (Figure 7). This symbol captures the overall shape used for the other (concrete) types of work product kinds. A document is a durable depiction of a fragment of reality. Documents are partitioned into document kinds by the DocumentKind class according to their structure, type of content and purpose. The symbol used to depict a document kind is a vertical rectangle with a dog-eared top right corner. This symbol depicts a sheet of paper.

A model is an abstract representation of some subject that acts as the subject's surrogate for some well-defined purpose. Models are partitioned into model kinds by the ModelKind class according to their focus, purpose and level of abstraction. The symbol currently used to depict a model kind consists of a vertically-stacked pair of horizontal rectangles linked by a short vertical line. This symbol is reminiscent of two nodes and an arc in a model.

A software item is a piece of software of interest to the endeavour. Software items are partitioned into software item kinds by the SoftwareItemKind class. The symbol

currently used to depict a software item kind is a square with a bottom left chamfered corner and a smaller rectangle adjacent to the bottom side. This symbol depicts a floppy disk, commonly used to represent the concept of software

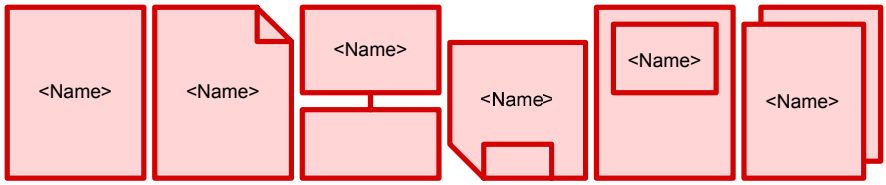

Fig. 7. Symbols for WorkProductKind, DocumentKind, ModelKind, SoftwareItemKind, HardwareItemKind and CompositeWorkProductKind, respectively

A hardware item is a piece of hardware of interest to the endeavour. Hardware items are partitioned into hardware item kinds by the HardwareItemKind class according to their mechanical and electronic characteristics, requirements and features. The symbol currently used to depict a hardware item kind is a vertical rectangle with a small rectangle nested in its upper half.

A composite work product is a work product composed of other work products. Composite work products are partitioned into composite work product kinds by the CompositeWorkProductKind class. The symbol used to depict a composite work product kind is a pair of vertical rectangles "stacked" along the z-axis, simulating perspective. This symbol tries to convey the idea of composition, i.e. a work product made of multiple components.

4.4 Producers

A producer is an agent that has the responsibility for executing work units. Producers are partitioned into producer kinds by the ProducerKind class according to their area of expertise. Three subtypes of ProducerKind are covered by this notation: TeamKind, RoleKind and ToolKind. All of them are based on a schematic depiction of a torso using half an ellipse with colours in the orange-yellow range (Figure 8).

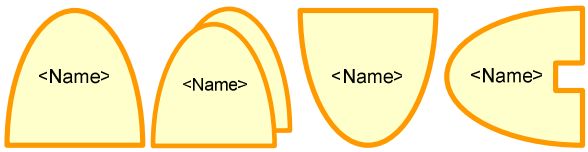

Fig. 8. Symbols for ProducerKind, TeamKind, RoleKind and ToolKind, respectively

The most generic is the abstract symbol for the superclass, ProducerKind. The symbol used to depict a work product kind is half an ellipse standing on its flat side.

A team is an organized set of producers that collectively focus on common work units. Teams are partitioned into team kinds by the TeamKind class according to their responsibilities. The symbol used to depict a team kind is a pair of half ellipses standing

on their flat side and "stacked" along the z-axis, simulating perspective. This symbol depicts multiple human torsos, and hence the team.

A role is a collection of responsibilities that a producer can take. Roles are partitioned into role kinds by the RoleKind class according to the involved responsibilities. The symbol used to depict a role kind is half an ellipse standing on its round tip – a "mask" symbol, used to show a person who is playing a (theatrical) role.

A tool is an instrument that helps another producer to execute its responsibilities in an automated way. Tools are partitioned into tool kinds by the ToolKind class. The symbol used to depict a tool kind is a vertical half ellipse with its round tip pointing leftwards and a square indentation in the centre of its flat side - depicting the head of an open-end wrench or spanner, a prototypical tool.

4.5 Other Groups of Symbols

There are other supporting symbols proposed as part of the ISO/IEC 24744 notation, including Actions and other similar relationships linking pairs of methodology elements. For example, an action is a usage event performed by a task upon a work product. Actions are partitioned into action kinds by the ActionKind class according to their cause (the specific task kind), their subject (the specific work product kind) and their type of usage (such as creation, modification etc.). An action is depicted (Figure 9) by an arc that goes from the symbol for the associated task kind to the symbol for the associated work product kind. The arc is a plain line with a small circle on the end of the work product kind. The type of usage is specified inside the small circle using an abbreviation ("t" in the figure – there are various options not listed here – see [15]). The role of the work product kind for this particular action kind, if any, can be shown close to the work product end. The optionality of the action kind can be shown using a deontic marker ("d" in the figure).

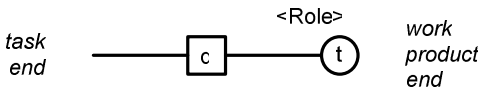

Fig. 9. Notation for actions

Similarly, a work performance is an assignment and responsibility association between a particular producer and a particular work unit. Work performances are partitioned into work performance kinds by the WorkPerformanceKind class according to the purpose of their inherent assignment and responsibility association. The symbol used to depict a work performance kind is an arc that goes from the symbol for the associated producer kind to the symbol for the associated work unit kind. A plain arc is used. The recommended assignment of the work performance kind can also be shown using a deontic marker.

4.6 Diagram Types

As with any kind of modelling using a graphical notation, various views of the total model can be made. In each a different aspect of the system is stressed. With ISO/IEC 24744, we can identify the need for the following diagram types:

- Lifecycle diagrams, which represent the overall structure of a method. They can depict both the temporal aspects (reading them left to right as time passes) and the content aspects (Figure 10).

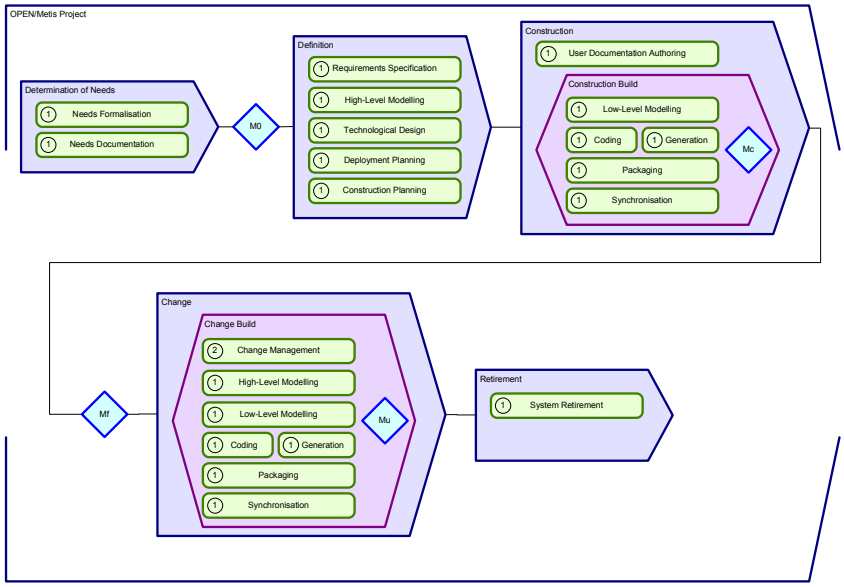

Fig. 10. A lifecycle diagram showing the content structure as well as the temporal structure of a method. Notice how stage kinds can contain both nested stage kinds (such as "Construction Build" inside "Construction") and process kinds ("User Documentation Authoring").

- Process diagrams, which describe the details of the processes used in a method. These may include the relationships between process kinds, the links between process kinds and task kinds, and the producer kinds assigned to the work unit kinds by the appropriate work performance kinds. Process diagrams, therefore, focus strongly on the "*what*" aspect of a method, being also able to show the "*who*" (Figure 11).
- Action diagrams (Figure 12), which show the usage interactions between task kinds and work product kinds. Action diagrams represent the usage that task kinds make of work product kinds. These usages are basically modelled as action kinds in ISO/IEC 24744. Action diagrams, therefore, serve to visualize the bridge between the process and product sides of a method.
- Task-Technique diagrams, which some developers may find useful to formalize which techniques are useful for which tasks and vice versa. Such diagrams would provide an alternative to the more well-established textual descriptions such as a deontic matrix [17].

The notation proposed so far for ISO/IEC 24744 focusses on representing the method domain. In the endeavour domain, diagram types are also needed. The only one so far tentatively proposed is the "enactment diagram", which represent a specific enactment of a method (or part of a method) and its relationship to the method specification.

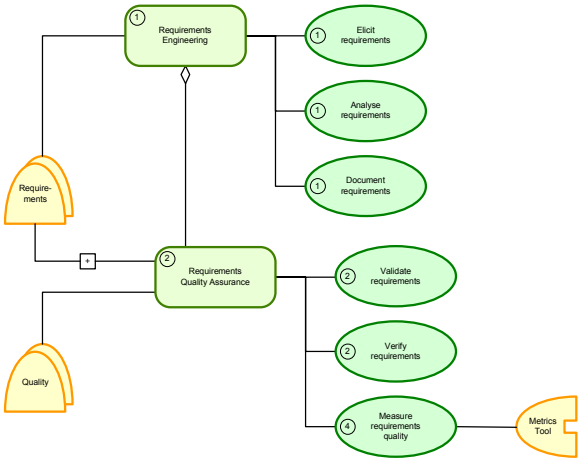

Fig. 11. A process diagram showing the details of the "Requirements Engineering" and "Requirements Quality Assurance" processes

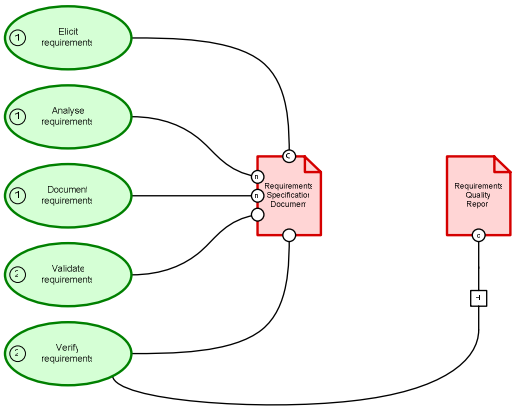

Fig. 12. An action diagram showing how requirements-related task kinds interact with requirements-related work products

5 To the Future

The ISO process for the notation to accompany the metamodel [1] will continue over the next several years with a likely final publication in 2009/10. During that process, elements of shape, topology, colour etc. may well change from those suggested above. In addition, it is anticipated that diagram types will be suggested to support the endeavour domain. At each six-monthly stage, international feedback is sought via the National Bodies that each have voting rights on ISO/IEC standards under SC7. Best practice is achieved by exposure to a wide-ranging audience, initially just software engineering experts in each country worldwide and then over a wider ISO electorate.

References

1. ISO/IEC: 24744: Software Engineering - Metamodel for Development Methodologies. International Organization for Standardization/International Electrotechnical Commission, Geneva (2007)
2. Henderson-Sellers, B.: On the Challenges of Correctly Using Metamodels in Method Engineering. In: Fujita, H., Pisanelli, D. (eds.) New Trends in Software Methodologies, Tools and Techniques. Proceedings of the sixth SoMeT 2007, pp. 3–35. IOS Press, Amsterdam (2007)
3. Flatscher, R.G.: Metamodeling in EIA/CDIF – Meta-metamodel and Metamodels. ACM Trans. Modeling and Computer Simulation 12(4), 322–342 (2002)
4. OMG: MDA Guide Version 1.0.1. OMG document omg/03-06-01 (2003)
5. Gonzalez-Perez, C., Henderson-Sellers, B.: Modelling Software Development Methodologies: A Conceptual Foundation. J. Systems and Software 80(11), 1778–1796 (2007)
6. Seidewitz, E.: What do Models Mean? IEEE Software 20, 26–31 (2003)
7. Henderson-Sellers, B.: Method Engineering: Theory and Practice. In: Karagiannis, D., Mayr, H.C. (eds.) Information Systems Technology and its Applications. 5th International Conference ISTA 2006, Gesellschaft für Informatik, Bonn. Lecture Notes in Informatics (LNI), vol. P-84, pp. 13–23 (2006)
8. Gonzalez-Perez, C.: Supporting Situational Method Engineering with ISO/IEC 24744 and the Work Product Pool Approach. In: Ralyté, J., Brinkkemper, S., Henderson-Sellers, B. (eds.) Situational Method Engineering: Fundamentals and Experiences. Proceedings of the IFIP WG 8.1 Working Conference, Geneva, Switzerland, September 12-14, 2007. IFIP Series, vol. 244, pp. 7–18. Springer, Berlin (2007)
9. Odell, J.: Power Types. Journal of Object-Oriented Programming 7(2), 8–12 (1994)
10. Henderson-Sellers, B., Gonzalez-Perez, C.: Connecting Powertypes and Stereotypes. J. Object Technol. 4(7), 83–96 (2005)
11. Atkinson, C.: Metamodelling for Distributed Object Environments. In: First Int. Enterprise Distributed Object Computing Workshop (EDOC 1997), Brisbane, Australia (1997)
12. Gonzalez-Perez, C., Henderson-Sellers, B.: Templates and Resources in Software Development Methodologies. J. Obj. Technol. 4(4), 173–190 (2005)
13. Constantine, L.L., Henderson-Sellers, B.: Notation Matters: Part 1 - Framing the Issues. Report on Object Analysis and Design 2(3), 25–29 (1995)
14. Constantine, L.L., Henderson-Sellers, B.: Notation Matters: Part 2 - Applying the Principles. Report on Object Analysis and Design 2(4), 20–23 (1995)
15. Gonzalez-Perez, C., Henderson-Sellers, B.: Metamodelling for Software Engineering. J. Wiley & Sons, Chichester (2008)
16. Henderson-Sellers, B., Edwards, J.M.: BOOKTWO of Object-Oriented Knowledge: The Working Object, p. 594. Prentice-Hall, Sydney, Australia (1994)