Software Quality Journal, 13, 195–214, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

A Metamodel for Assessable Software Development Methodologies

CESAR GONZALEZ-PEREZ cesargon@it.uts.edu.au
TOM McBRIDE mcbride@it.uts.edu.au
BRIAN HENDERSON-SELLERS brian@it.uts.edu.au
Department of Software Engineering, Faculty of Information Technology, University of Technology, Sydney, Ultimo NSW 2007, Australia

Abstract. Software development methodologies usually contain guidance on what steps to follow in order to obtain the desired product. At the same time, capability assessment frameworks usually assess the process that is followed on a project in practice in the context of a process reference model, defined separately and independently of any particular methodology. This results in the need for extra effort when trying to match a given process reference model with an organisation's enacted processes. This paper introduces a metamodel for the definition of assessable methodologies, that is, methodologies that are constructed with assessment in mind and that contain a built-in process reference model. Organisations using methodologies built from this metamodel will benefit from automatically ensuring that their executed work conforms to the appropriate assessment model.

Keywords: capability assessment, metamodelling, software development methodologies, process assessment, SPICE/OOSPICE

1. Introduction

While process modelling has a long history, many authors (e.g., Curtis et al., 1992) express concern about the inadequacies of existing software lifecycle models which, among other things, lack the rigour required by automated and automatable processes and do not offer any facility by which to reason about the process itself in order to know whether or not a changed process is improved by the change. More recently, metamodels have been proposed (e.g., Conradi et al., 1994; Henderson-Sellers and Bulthuis, 1997) as a means of creating additional rigour for methodology/process modelling. In essence, using metamodels means modelling a methodology as if it were any other system, applying the same modelling ideas and procedures that are usually applied to business applications or other software-intensive systems.

A metamodel is, therefore, in this context, a model of a methodology or, indeed, of a family of related methodologies. Methodologies constructed from a metamodel usually offer a higher degree of formalisation[1] and better support for consistent extension and customisation, since the concepts that make their foundations are explicitly defined. Some metamodels are available as independent standards (such as SPEM, see (OMG, 2002)) while others are provided together with a broader methodological framework that may include a methodology (such as the OPEN Process Framework, (Firesmith and Henderson-Sellers, 2002) or OPEN/Metis, (Gonzalez-Perez, 2003), respectively).

Once a methodology is in place and an organisation uses it, steps are followed and some work is actually carried out. The organisation may want to *assess* the quality of the execution of such work; in this context, “assessment” means determining the quality of the work execution which, in turn, can be related to the quality of the resulting products.[2] The results of the assessment are often used to ascertain the *capability* of the organisation, either for internal purposes (such as process improvement or quality monitoring) or in a client/provider context, perhaps to meet a contractual requirement. These capability assessments are usually conducted using some well-established standard such as CMMI (SEI, 2002) or ISO/IEC 15504: 1998 (ISO, 1998), which define specific assessment methods and *process reference models*.

A process reference model usually defines the abstract properties that some formal or informal process must comply with in order to be assessable, for example in terms of *purpose* and *outcomes*. Assessors in charge of performing the capability assessment must check and ensure that the work carried out by the organisation conforms to the requirements of the process reference model as a preliminary step before the actual assessment takes place.

OOSPICE (Object-Oriented and Component-Based Software Process Improvement and Capability Determination) is an international project funded by the European Union that pursues, among other objectives, the delivery of a software development methodology (the OOSPICE methodology) oriented toward component-based development (CBD), plus an assessment methodology which, in turn, must include a specific process reference model (see Henderson-Sellers et al. (2002) and http://www.oospice.com for additional information). One of the deliverables of the project is a *unified* metamodel that aims to allow for the definition of all CBD methodologies that are assessable. An *assessable methodology* is a methodology that incorporates the necessary formal properties so that no external process reference model is needed, because it contains a built-in process reference model. The following sections show why a metamodel for assessable methodologies is necessary and how one has been built under the auspices of the OOSPICE project. We have chosen an iterative and incremental approach to discuss the metamodel, showing it at different stages of completeness and complexity. It must be noted that the metamodel is coherent and consistent at every single iteration, although later iterations support richer expressiveness.

2. Motivation

In order to reason about software process modelling, Curtis et al. (1992, p. 77) considered the different information people normally want to extract from a process model. Process modelling languages usually present one or more perspectives related to the sought information. Curtis et al. have hypothesized that a process model that addresses the four perspectives of Functional, Behavioural, Organisational and Informational will produce an integrated, consistent and complete model of the process. That is, a methodology will be better if it can represent several of these perspectives. During its development, the OOSPICE project requested the creation of a methodology that dealt primarily with the Functional perspective. Since this methodology was intended to be deployable and useable, and indeed has already been trialled in a small number of companies in Austria,[3] we recognize that its usefulness would have been enhanced if it also dealt with the Informational, Behavioural and Organisational perspectives—organizations that are users of the OOSPICE methodology are currently expected to determine local tailoring of requirements. This permits the OOSPICE methodology to remain context independent, yet customizable to specific circumstances.

Existing methodologies such as OPEN (Firesmith and Henderson-Sellers, 2002) are not intended to be assessable and lack the properties required by assessment methodologies such as SPICE. SPICE does have an exemplar process assessment model directed at assessment rather than software development. This assessment model is contained in Part 5 of ISO 15504:1998 ((ISO, 1998), also known as SPICE) and extends the SPICE process reference model by adding base practices that, if performed, would achieve the required outcomes. It also suggests work products an assessor might look for as evidence that the base practice is being performed, and the outcome is being achieved.

Since the SPICE process assessment model is intended only to aid assessment and was never intended to be a full software development methodology, the flow of work products from their creation to consumption is rarely consistent, there is little rationalisation of the work products into a realistic taxonomy to avoid duplication or confusion about whether, for example, a quality management strategy is the same type of work product as a project strategy, and process tasks that are almost identical exist in several of the processes. While such inconsistencies attract no penalty when a methodology is being developed, the penalties incurred when implemented are likely to be severe. These problems of inconsistency are readily identified when the methodology is grounded in a metamodel and implemented, as we have done, in a database type of software tool.

Databases are good at representing the relationships between things and at detecting irregularities in those relationships so we decided to develop a database that could represent the elements of the methodology by creating a structure isomorphic with the metamodel. The essential properties of the methodology were that it needed to be as assessable methodology. That is, not only did it have to represent the best available knowledge of processes for component based development that could be adopted and applied by any organisation, it also had to be assessable by an assessment methodology such as SPICE.

3. Core concepts

Metamodels are usually built as a set of (meta)classes (and the corresponding (meta) relationships) that can be instantiated into methodology elements. This is the approach followed by SPEM (OMG, 2002), OPEN (Firesmith and Henderson-Sellers, 2002) and UML[4] (OMG, 2001). These classes represent the concepts that the methodology must deal with, such as processes, tasks, activities, techniques, etc. However, different standards and different authors utilise different terms for the same concept and different meanings for the same terms, so a sound agreement regarding terminology is necessary before continuing. The following sections show the core concepts that the OOSPICE methodology (see (Henderson-Sellers et al., 2002) for further information) had to adopt from both the process side (i.e. the necessary concepts to define the work to be done) and the assessment side (i.e. the necessary concepts to incorporate a process reference model within the methodology).

3.1. The process side

Table 1 shows the core concepts (and associated terminology) adopted, from a process perspective, as a starting point for the OOSPICE methodology.

Table 1. Core concepts and terms used by the OOSPICE methodology from a process perspective. Similar concepts in other frameworks are included.

Term	Description	Related concepts (in other metamodels)
Process	A process is a unit of work execution within a given area of expertise.	Activity (OPEN) Discipline, Activity (SPEM)
Task	A task is a simple ongoing responsibility that must be addressed in order to achieve one or more specific outcomes.	Task (OPEN) Step (SPEM)
Technique	A technique is a particular way of achieving a stated objective, described in terms of how things must be done.	Technique (OPEN) Guidance (SPEM)
Work Product	A work product is an artefact relevant to the execution of some task. Work products can be created, modified or read by tasks.	Work Product (OPEN) Work Product (SPEM)

The idea of *process* comprises the central concept of the metamodel and the major type of component in the methodology. A process is atomic from an execution point of view; it is either completely executed or not executed at all. Every OOSPICE process is characterised by a name and a description. Examples of processes are *Application Architecture* and *Training*. Other metamodels have similar concepts but the granularity and atomicity is often not identical. For example, OPEN has an Activity concept, but this is at a more conceptual level and is not atomic, being more like an ongoing responsibility rather than focussing on atomicity of execution. In some contrast, a SPEM Discipline is slightly broader and more abstract than an OOSPICE process—it is a broad description of a field of expertise and is not atomic in terms of execution. A SPEM Activity, on the other hand, is slightly narrower and more focussed than an OOSPICE Process and again lacks of the notion of atomicity of execution.

In OOSPICE, a process is composed of tasks, which are usually described as "jobs to be performed" or "a set of partially ordered steps intended to reach some goal" (Curtis et al., 1992). Tasks express *what* must be done in order to complete a process, but include no intrinsic time ordering or causal relationships. Tasks in the OOSPICE metamodel are very similar to tasks in OPEN and steps in SPEM. Tasks are characterised by a name and a description. Examples of tasks are *Design user interface architecture* and *Make build/buy decisions*.

Techniques in OOSPICE express *how* things are to be done, as opposed to Tasks expressing *what*. They are very similar to techniques as defined in OPEN, and close to guidances in SPEM.[5] Examples of techniques are *Workshops* and *Gap Analysis*. Techniques are related to tasks in a many-to-many fashion, since each technique may be useful for many tasks, and each task may need different techniques. They are characterised by a name, a maturity rating, a definition and some usage guidelines. Each relationship between a technique and a task can be further characterised by the degree of usability of the technique for the task, as defined in Graham et al. (1997). This can take the values of mandatory, recommended, optional, discouraged and forbidden.

Finally, work products represent artefacts that are created, modified, queried or otherwise used by tasks. They are similar to work products as defined in OPEN and SPEM. Work products are related to tasks in a many-to-many fashion, since each task can interact with different work products and each work product can be used by different tasks. Work products are characterised by a name and a description. Furthermore, each relationship between a task and a work product must be characterised by the mode in which the work product is used by the task. The most common modes of use are read-only (the work product is queried or "read" but not modified), creation and modification. Examples of work products are *Project Plan* and *Customer Requirements*.

3.2. The assessment side

A process reference model is defined in ISO/IEC 15504: 1998 (ISO, 1998) as "a model comprising definitions of processes in a life cycle described in terms of process purpose and outcomes, together with an architecture describing the relationships between the processes." Table 2 shows the core concepts and terminology that the OOSPICE methodology had to take into account to comply with that definition.

Although the term "process" has already been used originally from a process perspective (see Section 3.1), the assessment side brings new meaning to the word. Although CMMI (SEI, 2002) defines "process area" with a broader scope than that of process in ISO/IEC 15504: 1998 (ISO, 1998), both standards agree in attaching a purpose and a set of outcomes (called goals in CMMI) to each process. The purpose is characterised by a textual definition, while outcomes are each characterised by a description of the expected result. Moreover, the set of outcomes of a process must be necessary and sufficient to achieve the process purpose. An example of a process purpose is *To determine whether the integration result achieves its requirements* (from the *Integration Testing* process). An example of an outcome is *The analysed test results are reported to stakeholders* (an outcome of the same process).

In addition, the assessment perspective provides a better differentiation between the concepts of process and task. From the definitions in Tables 1 and 2, it is clear that "process" is defined as being assessable and with an associated set of outcomes, while "task" is not assessable or does not have associated outcomes.

Table 2. Core concepts and terms used by the OOSPICE methodology from an assessment perspective. Similar concepts in other frameworks are included.

Term	Description	Related concepts
Process	A process is an assessable unit of work execution with well-defined purpose and outcomes.	Process (ISO/IEC 15504) Process Area (CMMI)
Process purpose	The process purpose describes the high-level objectives that the process should achieve.	Purpose (ISO/IEC 15504) Process Area Purpose (CMMI)
Process outcome	The process outcomes are the expected results of a successful enactment of the process.	Outcome (ISO/IEC 15504) Goal (CMMI)

Table 3. “Process” redefined.

Term	Description	Related concepts
Process	A process is an assessable unit of work execution within a given area of expertise and with well-defined purpose and outcomes.	(see Tables 1 and 2)

3.3. First iteration

In order to achieve a comprehensive set of concepts to incorporate into the OOSPICE metamodel, the process definition in Table 2 must be merged with that in Table 1. This can be nicely done to obtain a hybrid concept that accommodates both process and assessment needs, as shown in Table 3.

In Section 3.1 we introduced the notion that processes are characterised by a name and a description. In the light of the revised definition, a process description must be split into two different properties, namely its purpose (as defined in Table 2) and an explanation of the work performed by the process or *outline*.

The concepts introduced so far can be used to build the first iteration of the OOSPICE metamodel, which is shown in Figure 1 as a UML class diagram.[6]

In our metamodel, `Process`[7] is defined by its name, purpose, outline, set of tasks and set of outcomes. This definition allows it to be used in the two different aspects identified earlier:

- From the process side, it describes the work to be done through the process name, outline and set of tasks.
- From the assessment side, it contributes to a built-in process reference model through its name, purpose and set of outcomes.

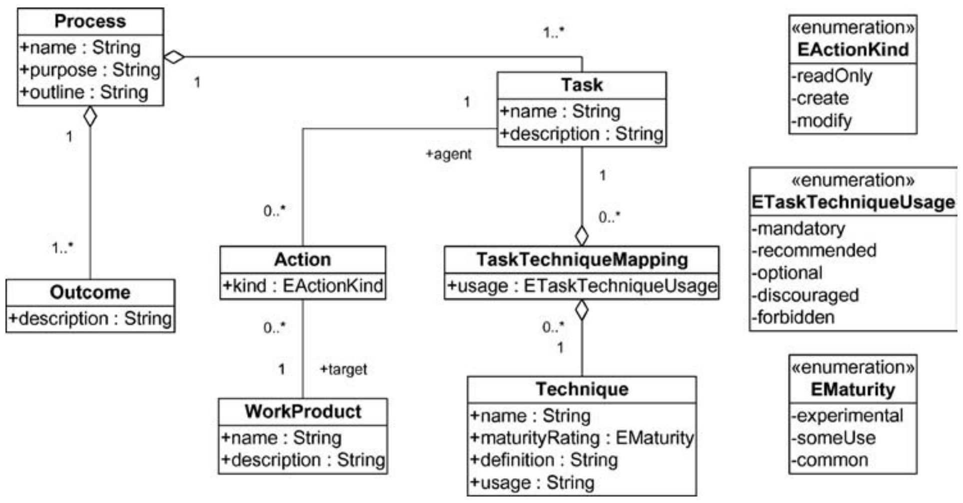

Figure 1. First iteration of the OOSPICE metamodel.

Moreover, tasks are related to techniques through a specific “mapping” class (`TaskTechniqueMapping` in Figure 1) that holds usage information, as described in the previous sections, and to work products through another intermediate class, called `Action`, that describes *how* a specific task acts upon a specific work product.

Multiplicities involving `Action` and `TaskTechniqueMapping` in Figure 1 might seem odd, especially those marked `0..*`. However, they are necessary to provide for the following cases: (i) a work product is introduced in the process without any action being performed on it (externally defined work products, typically), (ii) a task does not result in the modification of any work product (such as receiving a supplied component), (iii) a task does not use any techniques (simple tasks that do not need *how to* guidance such as *Accept the supplied product*; this involves just receiving a product) and (iv) a technique is not used by any task (since techniques form an independent pool, this is not uncommon).

4. Refinement and additional issues

Although the first iteration of the OOSPICE metamodel (see Figure 1) is useful to describe simple assessable methodologies, it can be further refined in order to provide support for more advanced needs. These needs include, but are not limited to, the following:

- Processes are often dealt with in thematic groups, to some extent loosely similar to OPEN activities or SPEM disciplines, but less granular. This allows, in theory, for a set of related processes to be replaced. While we are primarily interested in software development, the industry standard process reference model is ISO 12207 (ISO, 2002), which contains processes related to acquisition and supply, and human relations. If an organisation already had well embedded processes for supply chain management, SCOR perhaps (Supply-Chain Council, 2003), they may wish to retain them rather than try to implement those of ISO 12207.
- Different processes happen to include tasks with very similar (or even exactly equal) names and descriptions. Some optimisation (in the sense of shared information) can probably be achieved.
- Although outcomes must express the expected results of performing the process regardless of the tasks or techniques used to actually do the work, some relationship between outcomes and tasks is usually established by assessors in order to verify that outcomes have been achieved.
- Not all actions are possible at any time. The usage of work products by tasks is regulated by the *state* of the different work products at any point in time. Some way of modelling these constraints would be useful.

The following sections describe the solutions adopted to address these issues.

4.1. Domains

A *domain* in OOSPICE is a group of processes that are applicable in a common situation or realm. Domains usually group processes belonging to closely related areas of expertise, serving as a convenient way to deal with several processes as a block. A domain is characterised by a name and a description. It is similar to both an OPEN activity (which is narrower and more focussed than a domain) and a SPEM discipline (slightly narrower and more focussed than an OOSPICE domain). The domains defined in OOSPICE are *Engineering* (processes for designing and building a system), *Organisation* (processes working at the organisation level), *Management* (processes for project management), *Customer-Supplier* (controlling the relationships between customers and suppliers), *Human Resources* (at both organisation and project levels) and *Support* (supporting processes to be used by other processes).

A `Domain` class should, therefore, be introduced in the metamodel reflecting this concept.

4.2. Task pool

After using the first iteration of the metamodel for a while, it is easy to see that different processes often have some tasks with similar or identical names and descriptions. For example, process *Verification* includes a task called *Communicate verification result* that is described as "Distribute verification results to all stakeholders"; at the same time, the process *Measurement* includes a task called *Communicate measurement information* that is described as "Distribute measurement information to all stakeholders". There is, however, a major difference between both tasks, namely that they interact with very different sets of work products (through the appropriate actions). While the task in *Verification* uses (in a read-only manner) the work product *Verification Result*, the task in Measurement utilises work products *Measurement Information Product* (the information to communicate) and *Communication Plan* (a work product establishing how communication is to be done).

It seems that, although the definitions of some tasks can be constant across different processes, the actions that they perform are strongly determined by the particular process in which the tasks occur. It appears reasonable to keep the constant properties of tasks in a single place (the Task class) and adding a new class to the metamodel, named `ProcessTaskMapping`, to handle information that can vary from process to process, namely the set of actions that the task performs and a descriptive name for the task *within* a given process. This new class is, in fact, a refinement of the one-to-many relationship between `Process` and `Task` depicted in Figure 1.

Using this approach, a *pool of tasks* is kept for any methodology based on the OOSPICE metamodel, regardless of the existing processes. Tasks are not part of processes anymore, but autonomous entities that can be mapped to processes in a many-to-many fashion through intermediate process-task mappings which, in turn, define the particularities of each task within each process.

4.3. Tasks and outcomes

During the performance of an assessment, assessors usually look for evidence that outcomes have been achieved. Although the particular ways (i.e. tasks and techniques) employed by an organisation to achieve an outcome are not of direct interest to the assessment process, it is helpful to maintain a formal link between each outcome and the tasks utilised to achieve it. Having such a link enables methodologists to ensure that the appropriate tasks are in place to fulfil every outcome.

However, outcomes are defined in relation to a specific process (see Figure 1), while tasks are independent of these (see Section 4.2). Therefore, the link should be established between each outcome and the particular involvement of a given task in the outcome's process. As seen in Section 4.2, such involvements are modelled in our metamodel through process-task mappings. Thus, a formal relationship must be established between classes `Outcome` and `ProcessTaskMapping`.

4.4. Time ordering and constraints

The set of tasks mapped to a process (see earlier sections) is not ordered. This means that the tasks are not steps to follow but ongoing responsibilities that must be addressed at some time within the execution of the process. The appropriate moment to execute each task is only given by the state of the necessary input work products, i.e. those work products that are either read-only or modified by the task. This approach allows time-ordering within a process by specifying a set of constraints that must be satisfied before any task can be executed. Such constraints are called *preconditions*. They assert that a specific property of a specific work product must have a specific value. An example precondition could assert that the *RevisionStatus* property of the *Domain Assessment* work product must have the value *Reviewed* before the task *Develop vision statement* in process *Application Requirements Engineering* can proceed.

Although preconditions regulate when tasks can be executed, a better degree of control (as well as a more elegant metamodel) is attained if they are attached to actions. From Sections 3.3 and 4.2, an action describes how a particular task in the context of a particular process (i.e. a process-task mapping) acts upon a particular work product; actions are therefore the actual "atoms" of execution in OOSPICE, since each of them deals with a single work product and a single task in the context of a single process. A precondition attached to an action is capable of making an assertion about the state of such a work product in order to permit or prevent the execution of such a task in the process.

Once the need for preconditions is established, we must face the need for some way to alter the state of work products as a consequence of executing actions.[8] In a nice (but not very original) demonstration of symmetry, *postconditions* can be introduced to specify the state of work products *after* an action is performed. An example postcondition could assert that the *RevisionStatus* property of the *Application Vision Statement* work product takes the value *Complete* after being created by task *Develop vision statement* in process *Application Requirements Engineering*.

Using this approach, any action in a given methodology can have as many preconditions and postconditions as necessary. All preconditions must be met for the action to be executable. An action for which all preconditions hold at a given point in time is called an *executable action* at that time; an executable action can be actually executed by the organisation at any moment. Correspondingly, all postconditions are guaranteed after execution of the action.

Any task in a given methodology can, in turn, have any number of actions attached to it through a given process-task mapping. Since none, some or all of such actions may be executable at a given point in time, the "executability" of a process-task mapping must be defined in term of the "executability" of its constituent actions: a process-task mapping is executable if and only if all of its actions are executable. However, from the perspective of task performance, a process-task mapping cannot be considered to have been executed until all of its actions have been executed. These definitions result in a somewhat paradoxical consequence since a process-task mapping can be completely executed without having been executable at any point in time, if its constituent actions become executable and are executed at non-overlapping moments. Although this could seem counterintuitive, it makes sense when we realise that a process-task mapping is not atomic from a temporal point of view, since at any point in time some of its actions can have already been executed and some others may be still pending. The "executability" of a process-task mapping refers to the process-task mapping as a whole.

4.5. Other issues

Some additional issues can be found with the first iteration of the metamodel (see Figure 1). For example, some work products are simple variations of a generic concept. For example, both *Acquisition Strategy* and *Reuse Strategy* are strategies, and therefore can be defined as "... recording how an organisation intends to achieve some outcomes according to applicable policies and within identified constraints". At the same time, they focus on different applications of the general concept (acquisition vs. reuse), and detailed descriptions for each of them are also valuable. An `isSubtypeOf` relationship can be introduced in the metamodel between the `WorkProduct` class and itself, so any work product can be described through its attributes and declared to be a specialised subtype of a more generic one. A conventional generalisation/specialisation relationship is out of scope in this situation, since specific work products are not classes in our metamodel, but objects at the methodology level.

Simultaneously, many methodology elements are defined following well-known best practices or standards. For example, most methodologies would want a process named *Requirements Engineering* or something similar. In such cases, it is useful to have a formal way to document the industry, academic or bibliographic sources from which each methodology element has been taken. The metamodel can be augmented with a `Source` class, which models a generic kind of source (be it a standard, a journal paper or whatever other document), plus a `Reference` class that models a specific reference to a given source. Sources can be characterised by a short name and a formal citation, while references are characterised through a pointer to a specific source plus a comment often used to detail the relevant pages or section within the source. At the same time, methodology elements must be given the possibility of having references attached. Thus, an interface IUsingSources is introduced and realised by classes `Domain, Process, Task, Technique` and `WorkProduct`. This interface is defined as having a one-to-many relationship to `Reference`.

4.6. Second iteration

The second iteration of the OOSPICE metamodel incorporates all the refinements discussed in the previous sections, as shown in Figure 2.

The major differences between the second and the first iterations of the metamodel are those concerning the introduction of domains; sources and references; and constraints, preconditions and postconditions. Relationships have been also altered, especially the one

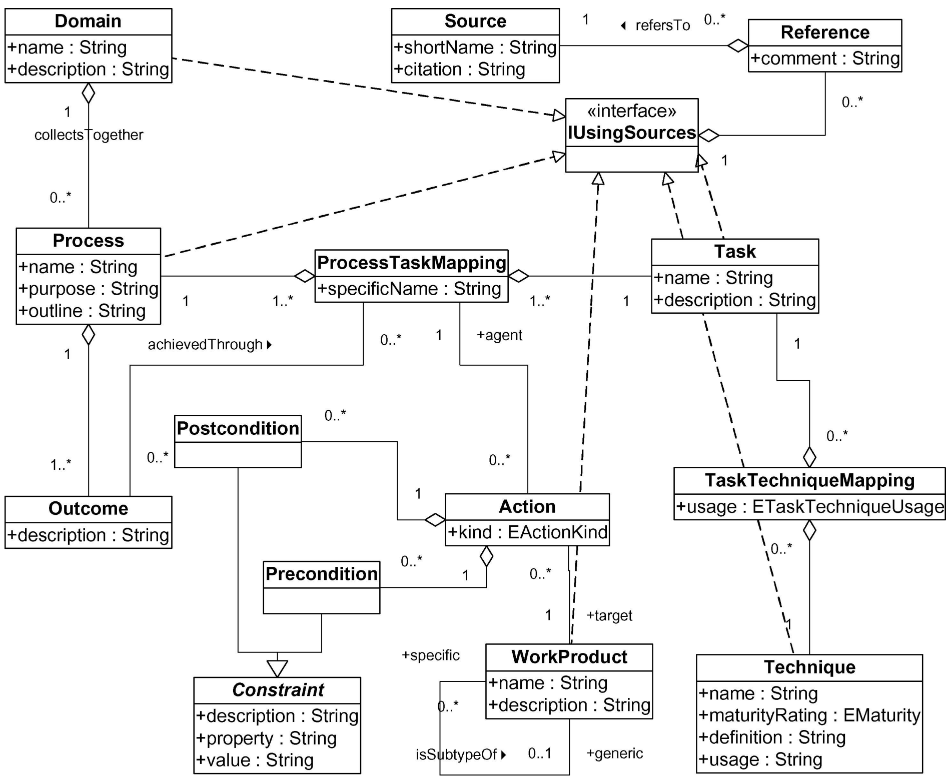

Figure 2. Second iteration of the OOSPICE metamodel.

between Process and Task by the introduction of the ProcessTaskMapping.

The second iteration of the metamodel allows the detailed description of assessable methodologies, giving support for task pooling, constraint-based time ordering, work product generalisation hierarchies, process grouping and source referencing.

Using the UML realization relationship to connect a class and its interface, we show how Domain, Process, WorkProduct, Technique and Task have a common interface IUsingSources which permits each of them to access referential source material.

5. Capability levels

Most methodological frameworks for software development acknowledge that the customisation and adaptation of the methodologies to the specific needs of their users is a key necessity (see for example (OMG, 2002; Firesmith and Henderson-Sellers, 2002). Even “fixed” process definitions without an explicit metamodel, such as ISO/IEC 12207: 1995/Amd 1: 2002 (ISO, 2002), allow for tailoring. This customisation, tailoring or process construction, whatever it is named, is always described in terms of adding or deleting methodology elements such as processes, activities, task or techniques. However, when adapting a methodology, a whole family of customisations can be directly related to the capability level of the user organisation. The following sections explore this situation in full detail.

5.1. Capability context

Capability levels are sets of common features that provide a major enhancement in the capability to perform processes. ISO/IEC 15504: 1998 (ISO, 1998) defines the following six capability levels, used also in OOSPICE (Henderson-Sellers et al., 2002):

- Not Performed (level 0): the organisation fails to successfully execute the process.
- Informal (level 1): the process is successfully executed but may not be rigorously planned and tracked.
- Planned (level 2): the process is planned and tracked while it is performed; work products conform to specified standards and requirements.
- Defined (level 3): the process is performed according to a well-defined specification that may use tailored versions of standards.
- Controlled (level 4): measures of process performance are collected and analysed, leading to a quantitative understanding of process capability and an improved ability to predict performance.
- Optimised (level 5): continuous process improvement against business goals is achieved through quantitative feedback.

For example, many processes would include a planning task (in order to ensure that the process is carried out in a planned fashion), but this means that the organisation performs at a level 2 or higher. In other words, a process containing a task that needs the performance of any kind of planning cannot be applied to an organisation performing at level 1. Similarly, a process involving a task needing some process performance measures could not be approached by any organisation performing at level 3 or lower.

Although an organisation's capability level for a specific process is usually known *after* performing the process and making a formal assessment based on it, the ability to select a particular desired capability level *a priori* can help organisations select the most appropriate version of a methodology. A metamodel that directly supports the concept of capability level (together with an appropriate tool set, probably[9]) would permit the definition of methodologies that are dynamically tailorable along their capability level dimension, thus reflecting the capability context of the user organisation. With this approach, a process is not mapped to a fixed set of tasks and outcomes anymore; instead, these sets are dynamically constructed depending on the chosen capability level. For example, in defining a process at level n, only those tasks and outcomes applicable to capability level n or below are considered (see Figure 3 for an example).

This model implies that every single task and outcome in the methodology has a *minimum capability level* above which it makes sense, and below which it does not. Continuing with the example shown in Figure 3, the task *Verify work products* on the right-hand side of the figure is defined as having a minimum capability level of 2, which means that this task will never appear (and therefore no attempt will be made to perform it) as part of its process when the process' capability level is lower than 2.

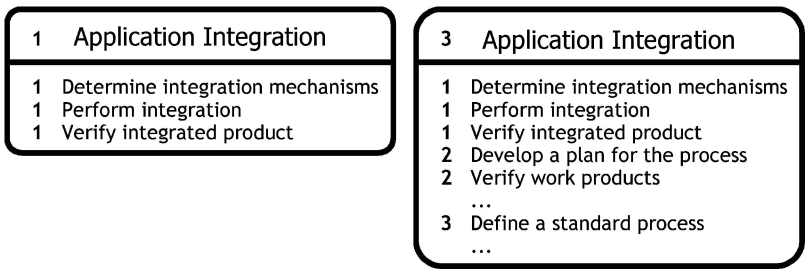
1 Application Integration
1 Determine integration mechanisms
1 Perform integration
1 Verify integrated product
3 Application Integration
1 Determine integration mechanisms
1 Perform integration
1 Verify integrated product
2 Develop a plan for the process
2 Verify work products
...
3 Define a standard process
...

Figure 3. Example of dynamic process definition depending on capability level. Tasks for the same process, Application Integration, are shown for capability levels 1 and 3.

With regard to outcomes, and since the “fixed” outcomes of a process (i.e. the outcomes necessary and sufficient to achieve the process purpose, as defined in Section 3.2) are *sufficient* to achieve the process purpose, the dynamically added set of outcomes for a process are not related to the purpose of the process itself, but to the definition of the associated capability level.

As a side effect of this approach, some processes present an empty set of tasks and outcomes below certain capability levels. For example, the *Process Establishment* process (which aims to establish a set of formally defined processes across an organisation) appears “empty” at capability levels 1 and 2. We can say, therefore, that this process has a *derived* minimum capability level of 3, since all of its tasks and outcomes have minimum capability levels equal or greater than 3.

5.2. *Generic and specific methodology elements*

At the same time, many (if not all) of the tasks and outcomes that may or may not appear as part of processes depending on capability levels *are common to all processes*. Continuing with our example, the task *Verify work products* that appears in process *Application Integration* in Figure 3 also appears in every other process at capability level 2 or higher, since it makes abstract reference to the process’ work products and demands that they are verified, regardless of their contents or character. Such tasks and outcomes may be called *generic*, as opposed to *specific* ones that make explicit references to the aim and character of a given process. For example, the task *Determine integration mechanisms* (see Figure 3 again) is implicitly linked to the *Application Integration* process, since it deals with the objectives of the process, i.e. determining the mechanisms to integrate an application. However, tasks such as the aforementioned *Verify work products* usually deal with the processes themselves, and are therefore generic.

Although all of the generic tasks and outcomes in the current version of the OOSPICE methodology have a minimum capability level greater than 1, i.e. are dynamically added or removed to/from processes depending on the capability context of the organisation, we must emphasise that this circumstance is not imposed by the metamodel.

5.3. Third iteration

The OOSPICE metamodel must be modified to accommodate minimum capability levels in processes, tasks and outcomes, as well as to differentiate between specific tasks and outcomes (which are explicitly linked to a given process) and generic ones (which are not explicitly linked to any process but are automatically included in processes above a certain capability level). Figure 4 shows the metamodel with the necessary modifications to reflect the addition of support for capability levels and generic *vs.* specific methodology elements.

6. The OOSPICE methodology

When we implemented this metamodel as a database in which to describe the OOSPICE methodology, the total number of tasks in the generated methodological repository was reduced from 229 to 178 as a result of the identification of overlapping and redundant tasks not observed manually. Some 76 errors were located in the work product flows and the number of work products was reduced from somewhere around 300 to 217 for similar reasons. These changes reduced the methodology's size and superficial complexity.

The ease with which processes could be displayed and reported with differing levels of capability exposed a common misconception of people unfamiliar with capability level based methodologies. The misconception is to see the processes displayed at the lowest capability, which they always are, and think that the process will never require anything more than that. The objection is that software development at the lowest maturity level would not be acceptable practice in many organisations and certainly not something OOSPICE should be recommending, even tacitly. Capability level based methodologies are able to implement different capability levels for the same process to meet specific circumstances. Most of the core software development processes would normally be required to be performed at least at a capability level of 2 if not 3, but a process that is seldom required because it is not critical to the project might not need more than the lowest capability level. Since the OOSPICE methodology could display processes at varying capability levels the tendency to try to add more to the basic process was reduced because it could be seen that the intended additions were already represented in the higher capability levels.

Tailoring a methodology is always difficult because it requires a lot of work and is usually restricted to selecting which processes will be performed and to incorporating the Behavioural and Organisational aspects of them. Seldom are the original processes altered to deal with, for example, a higher level of software criticality. This OOSPICE model quite easily copes with selecting specific processes, and omitting others, by setting their required maturity level to 0 so that they won't be displayed or reported. Differing levels of project criticality see the capability level adjusted accordingly.

The database (and metamodel) have only been developed to support an assessable software development methodology but most of the information is already present to meet the requirements of a SPICE compliant assessment. Tools could readily be developed for that. The current state is that the metamodel is implemented in a Microsoft Access database for reasons of cost and convenience. It must be regarded as a prototype waiting for an excuse, or funds, to re-implement as a commercially acceptable tool both for software development and for process assessment.

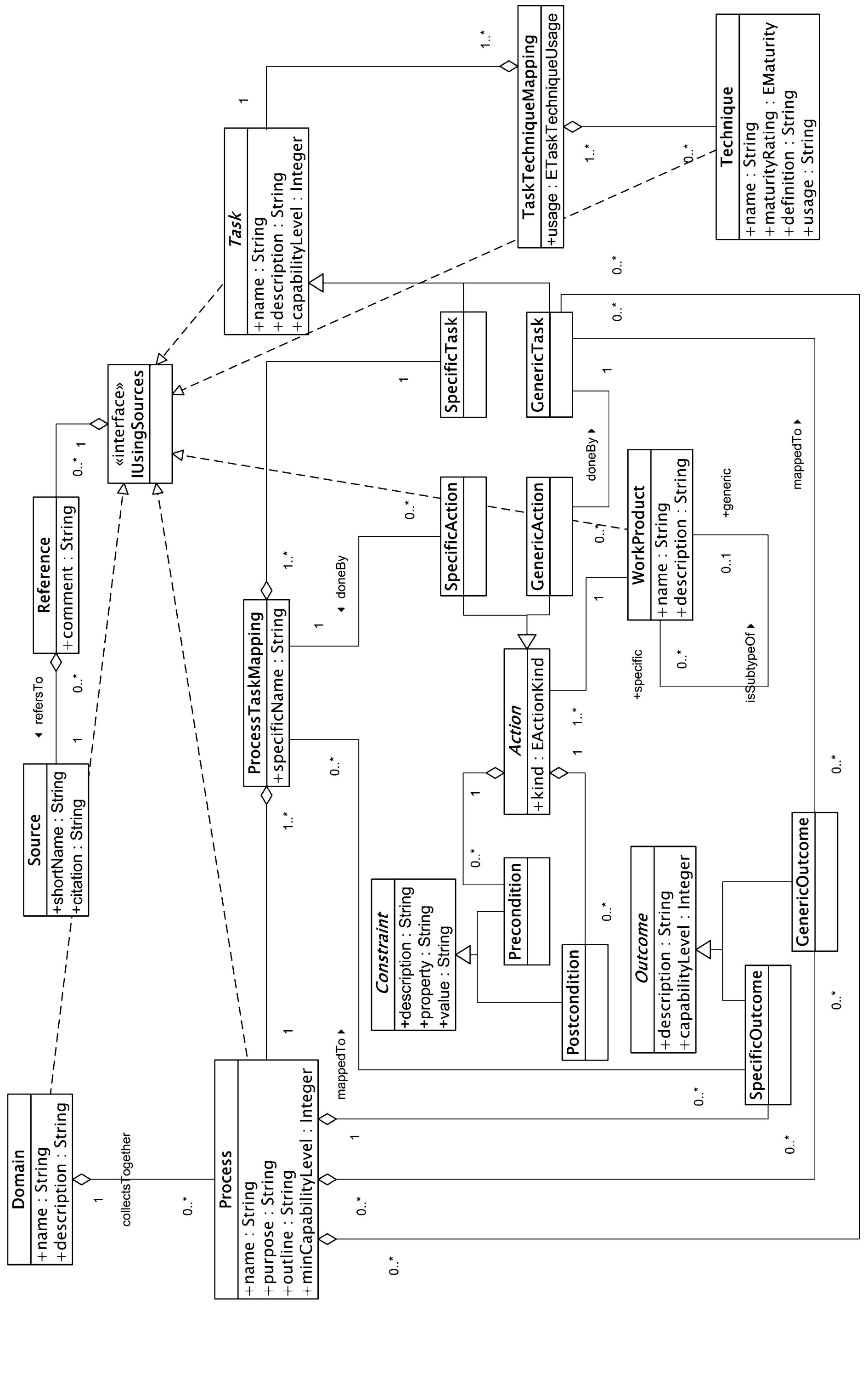

Figure 4. Third iteration of the OOSPICE metamodel.

Interestingly, a second database was created by the same authors using the same principles to check the consistency of a revision of ISO 15504-5, which is the SPICE process assessment model. The assessment model is primarily focussed only on those aspects of the process tasks and work products directly relevant to assessment. Hence, a consistent flow of work products through the various processes and tasks is not essential for assessment purposes but is required to gain acceptance in the software development community. The SPICE process assessment model is, essentially, a software development methodology and logical inconsistencies will discourage its adoption. This second application provides a cross-check on the OOSPICE metamodel/database approach described here.

7. Further iteration

Although the *current version* of the OOSPICE metamodel (as depicted in Figure 4) is capable of defining assessable methodologies with a fine degree of detail, some further refinements are possible. In particular, the introduction of powertype patterns in the metamodel, following the approach proposed by Gonzalez-Perez and Henderson-Sellers (2004), can enhance it, at least, in the following areas:

- Project-level elements can be taken into account as well as methodology-level elements.
- Properties of enacted work products can be directly supported, which is very useful for the formalisation of actions.

The following sections describe these issues in a little more detail, laying the baseline work for future research endeavour.

7.1. Project-level elements and powertype patterns

When a development team *enacts* a methodology with the objective of obtaining some specific product, they deal with actual entities pertaining to both the modelling and process domains, such as documents, models and tasks. A document or a task, as seen by a developer, is an entity with certain properties, which can be formalised in the same way as any other information. For example, any project manager would certainly be aware that a task has always a description, a start date, a due date and a responsible developer (or developers) assigned to it. Similarly, a document probably has a title, a set of authors, a version number and a date of last modification. An example of a task could be the task with description “Code method bodies for class Customer”, started on the 3rd of March 2003, due on 20th March 2003 and assigned to developers Carrie and John. A document example could be the document titled “System Architecture Specification” version 1.3, prepared by Carrie and Liz and last modified on 5th February 2003.

Tasks and documents, as described in our example, are *project-level elements* (as opposed to *methodology elements*), since they occur in the realm of a specific project. Tasks could be formalised as a Task class with attributes `Description`, `StartDate`, `DueDate` and `ResponsibleDevelopers`; documents could be formalised as a `Document` class with attributes `Title, Version, Authors` and `LastModificationDate`.

However, metamodels such as the OOSPICE metamodel in its current version (see Figure 4), SPEM (OMG, 2002) or OPEN (Firesmith and Henderson-Sellers, 2002) do not include classes such as these. The `Task` class in the OOSPICE metamodel (as well as similar classes in SPEM and OPEN) actually represent *kinds of tasks*, such as *Code method bodies* or *Verify work products*, while the `Task` class in our example represents actual tasks being performed during the project. For the sake of readability, we can *rename* classes describing kinds of elements by adding the "Kind" suffix to them, as proposed in Gonzalez-Perez and Henderson-Sellers (2004), so the `Task` class in the OOSPICE metamodel would become `TaskKind, Process` would become `ProcessKind`, etc.[10] At the same time, tasks being performed during a project (proper *tasks*) and task kinds as defined by the methodology are strongly related, since *every task (in our new parlance) is an instance of a task kind, which in turn is, at the same time, both a subtype of the class* `Task` *and an instance of the class* `TaskKind`. `Task` and `TaskKind`, therefore, form a powertype pattern (as defined in Odell (1994) and Martin and Odell (1992)), because `TaskKind` is a powertype of `Task`.

Powertypes, originally described in Martin and Odell (1992) and initially applied to methodologies in Gonzalez-Perez and Henderson-Sellers (2004), allows a metamodel to exert control on both methodology elements and project elements. For example, the class `Task`, representing actual tasks being carried out during a project, may have attributes `Description`, `StartDate` and `DueDate`, while the class `TaskKind`, representing kinds of tasks in the methodology, may have attributes Name and `MinCapabilityLevel`.

7.2. *Support of work product properties*

The powertype approach appears to be useful, for example, to formalise the description of work product properties, which are essential in the definition of constraints (see Section 4.4). Every constraint, either a precondition or a postcondition, make reference to a specific "property" of a specific work product. The concept of "work product property" has not been formally defined yet, since we need powertype-based metamodelling to give a proper definition. Properties of work products as used in constraints refer to *actual work products*, not to work product kinds, so giving a formal definition before introducing powertypes would have been impossible. For example, the property `RevisionStatus` of a work product refers to an attribute of the `WorkProduct` class, which represents actual work products (physical and electronic documents, for example) as generated and used during the performance of a project. It is not an attribute of class WorkProductKind, which represents different kinds of work products that may be used.

By introducing powertype-based metamodelling and subsequent support for project-level elements, work product properties can be represented in the metamodel as attributes of the `WorkProduct` class.

7.3. *Fourth iteration*

A potential fourth iteration of the OOSPICE metamodel would include renamed classes for methodology-level elements, such as `ProcessKind`, `TaskKind` and `WorkProductKind`. In addition, it would pair such classes with newly introduced ones (`Process`, `Task` and

`WorkProduct`, for example) in powertype patterns. The overall structure of the metamodel, however, would stay unmodified.

8. Conclusions

This paper has described a metamodel to build and support methodologies that are assessable, i.e. that can be directly used to determine the quality of the executed work. In particular, the OOSPICE metamodel is rich enough to express this kind of methodologies, by introducing the concept of capability level in the metamodel and allowing for methodologies to be dynamically adjusted depending on the user organisation's capability context. As a potential future enhancement, we have proposed the use of powertype patterns in the metamodel to aid in the formalisation of project-level elements. Such an enhancement would replace one metamodelling philosophy (layered, so-called "strict" metamodelling) by another in which layering based on the "is-instance-of" relationship between layers is no longer valid. In any case, the built-in process reference model that the OOSPICE metamodel supports makes process assessment much more straightforward than with conventional approaches.

We can assume that current assessment frameworks such as SPICE and CMMI are capable of process assessment. The metamodel presented in this paper incorporates all the concepts from these approaches, abstracting the concepts to a higher abstraction level, that of the metamodel. Since approaches such as SPICE can now be created from this underpinning metamodel, it is reasonable to assume that such generated (and widely validated) assessment approaches contain the structures necessary to record and manage assessment information. Therefore, we can conclude that our metamodel indeed is able to support process assessment.

Acknowledgments

We wish to thank the Australian Research Council for providing funding for the Australian contribution to the OOSPICE project. This is contribution number 03/07 of the Centre for Object Technology Applications and Research (COTAR).

Notes

1. By "formalisation" we mean rigour and consistency in its definition, not rigidity in its usage.
2. That process quality largely determines product quality is a fairly well accepted principle in software engineering, as stated by Humphrey (1989).
3. Unfortunately, reports on these trials are not publicly available due to confidentiality and intellectual property issues.
4. Although the UML standard does not deal with methodologies, it makes use of a metamodel in the sense of our discussion.
5. Although SPEM includes a Technique element, OOSPICE techniques are closer to SPEM guidances because of their abstract character.
6. Given the ambiguity in UML regarding whole/part relationships, we will use "white diamonds" in class diagrams to depict this kind of relationship, without any implication about secondary attributes (see Barbier

and Henderson-Sellers (2000)), i.e. to simply mean that a whole (white diamond end of relationship) consists of parts (the unadorned end of the relationship).

7. The use of this font indicates a name of an element in the metamodel.
8. If we did not include a way to alter the state of work products, the enactment of a methodology would not be able to progress at all.
9. Although the use of automated tools for process construction is not absolutely necessary, it can make customisation much easier and less prone to errors.
10. Notice that not every single class in the metamodel must be renamed in this way, but only classes representing *template* elements as described in Gonzalez-Perez and Henderson-Sellers (2004).

References

Barbier, F. and Henderson-Sellers, B. 2000. The whole-part relationship in object modelling: A definition in cOlOr, *Information and Software Technology* 43:19–39.

Conradi, R., Fernström, C., and Fuggetta, A. 1994. Concepts for evolving software processes, In *Software Process Modelling and Technology*, eds. A. Finkelstein, J. Kramer, and B. Nuseibeh, pp. 9–31, John Wiley & Sons, New York.

Curtis, B., Kellner, M. I., and Over, J. 1992. Process modelling, *Communications of the ACM*. 35(9): 75–90.

Firesmith, D. G. and Henderson-Sellers, B. 2002. *The OPEN Process Framework*, The OPEN Series. Addison-Wesley, London.

Gonzalez-Perez, C. 2003. *OPEN/Metis White Paper* (PDF). Accessed on 18th February 2003. http://www.openmetis.com

Gonzalez-Perez, C. and Henderson-Sellers, B. 2005. *A Powertype-Based Metamodelling Framework*, Software and Systems Modelling. [submitted].

Gonzalez-Perez, C. and Henderson-Sellers, B. 2005. Templates and resources in software development methodologies, *Journal of Object Technology* 4(3).

Graham, I., Henderson-Sellers, B., and Younessi, H. 1997. The OPEN process specification, *The OPEN Series*, Harlow (Essex), Addison-Wesley Longman, UK.

Henderson-Sellers, B., Bohling, J., and Rout, T. 2002. Creating the OOSPICE model architecture—a case of reuse, In *SPICE 2002*. Venice, Italy, pp. 13–15 March 2003. Qualital.

Henderson-Sellers, B. and Bulthuis, A. 1997. *Object-Oriented Metamethods*, Springer-Verlag, New York.

Henderson-Sellers, B., Stallinger, F. and Lefever, B. 2002. *Bridging the Gap from Process Modelling to Process Assessment: The OOSPICE Process Specification for Component-Based Software Engineering*. In 28th *EUROMICRO Conference*. Dortmund, Germany, 4–6 September 2002. IEEE Computer Society: Los Alamos, CA.

Humphrey, W. S. 1989. *Managing the Software Process*, Addison-Wesley, Reading, MA.

ISO, 1998. *Software Process Assessment*. ISO/IEC 15504: 1998. International Standards Organization/International Electrotechnical Commission.

ISO, 2002. *Software Life Cycle Processes, Amendment 1*. ISO/IEC 12207: 1995 / Amd 1: 2002. International Standards Organization/International Electrotechnical Commission.

Martin, J. and Odell, J. J. 1992. *Object-Oriented Analysis & Design*, Prentice-Hall, Englewood Cliffs, NJ.

Odell, J. J. 1994. Power types, *Journal of Object-Oriented Programming* 7(2): 8–12.

OMG, 2001. *Unified Modelling Language Specification*. formal/01-09-68 through 80 (13 documents). Object Management Group.

OMG, 2002. *Software Process Engineering Metamodel Specification*. formal/2002-11-14. Object Management Group.

SEI, 2002. *CMMI for Systems Engineering/Software Engineering/Integrated Product and Process Development/Supplier Sourcing, Continuous Representation*. CMMI-SE/SW/IPPD/SS, V1.1, Continuous. Carnegie Mellon Software Engineering Institute.

Supply-Chain Council, 2003. *Supply Chain Operations Reference Model* (website). Accessed on 9th October 2003. http://www.supply-chain.org

Cesar Gonzalez-Perez is a post-doctoral research fellow in the Faculty of Information Technology at UTS, where he is currently researching with Professor Henderson-Sellers in object-oriented methodologies, with particular emphasis on metamodelling and component-based, assessable methodologies. He is the founder and former technical director of Neco, a company based in Spain specializing in software development support services, which include the deployment and use of the OPEN/Metis methodology at small and mid-sized organizations. He has also worked for the University of Santiago de Compostela in Spain as a researcher in computing & archaeology, and received his Ph.D. in this topic in 2000.

Tom McBride has more than twenty years in the computer industry in positions ranging from computer operator, developer, project manager to QA manager. He is significantly involved in standards development, both locally in Australia and internationally for the International Standards Organisation. Tom is Chairman of the Australian Computer Society National Standards Committee and is assisting the development of the OOSPICE Component Based Development methodology. He is also a lecturer in software development-related subjects at the University of Technology, Sydney and is currently enrolled as a Ph.D. student investigating coordination in software development.

Brian Henderson-Sellers is Director of the Centre for Object Technology Applications and Research and Professor of Information Systems at UTS. He is author of eleven books on object technology and is well-known for his work in OO methodologies (MOSES, COMMA, OPEN, OOSPICE) and in OO metrics.

Brian has been Regional Editor of Object-Oriented Systems, a member of the editorial board of Object Magazine/Component Strategies and Object Expert for many years and is currently on the editorial board of Journal of Object Technology and Software and Systems Modelling. He was the Founder of the Object-Oriented Special Interest Group of the Australian Computer Society (NSW Branch) and Chairman of the Computerworld Object Developers' Awards committee for ObjectWorld 94 and 95 (Sydney). He is a frequent, invited speaker at international OT conferences. In 1999, he was voted number 3 in the Who's Who of Object Technology (Handbook of Object Technology, CRC Press, Appendix N). He is currently a member of the Review Panel for the OMG's Software Process Engineering Model (SPEM) standards initiative and is a member of the UML 2.0 review team. In July 2001, Professor Henderson-Sellers was awarded a Doctor of Science (D.Sc.) from the University of London for his research contributions in object-oriented methodologies.