Towards the Use of Granularity Theory for Determining the Size of Atomic Method Fragments for Use in Situational Method Engineering

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

[1] School of Software, Faculty of Engineering and Information Technology,
University of Technology, Sydney, P.O. Box 123, Broadway, NSW 2007, Australia
Brian.Henderson-Sellers@uts.edu.au
[2] The Heritage Laboratory (LaPa), Spanish National Research Council (CSIC)
Santiago de Compostela, Spain
cesar.gonzalez-perez@iegps.csic.es

Abstract. Situational method engineering defends the idea that methodologies should be constructed by assembling pre-existing method fragments from a repository. The structure of the repository, the kinds of fragments that it can store, as well as the possible relationships among them, are dictated by an underlying metamodel. One of the aspects that must be studied is that of the granularity of the individual method fragments in the context of the metamodel to which the fragments are conformant. This becomes especially relevant in a service-oriented method engineering context, where interoperability and composability of fragments from multiple repositories is a key issue. This paper applies some theoretical works on granularity to the study of both the granularity and the size of method fragments, recommending some best practices that should be adopted in order that the resultant method fragments are atomic and therefore likely to be consistent in quality thus leading to higher quality constructed methodologies and paving the way for easier composition and interoperation of fragments.

Keywords: granularity, method fragments, situational method engineering.

1 Introduction

Situational method engineering (SME) [1, 2] describes the creation and use of a software development method(ology)[1] from small, atomic methodological pieces, known as "method fragments" or, at a larger scale (i.e. non-atomic), "method chunks" [3], typically conformant to the conceptual definitions in an underpinning metamodel [4].

While much of the literature focusses on method construction e.g. [5-7], little has focussed on the *details* of the atomic elements of a method (method fragments) themselves. In particular, an issue, as yet little discussed, is the *granularity* both of elements in the metamodel and of the atomic modelling elements (method fragments) conformant to it. Firstly, the elements in the metamodel may each have fine

[1] Method and methodology are taken to be synonyms for the purposes of this paper.

J. Ralyté, I. Mirbel, and R. Deneckère (Eds.): ME 2011, IFIP AICT 351, pp. 49–63, 2011.

granularity or coarse granularity [8], the latter resulting from an abstraction mapping from a highly detailed system to a less detailed one. This, naturally, affects the granularity of method fragments generated from the metamodel. However, there is a second and orthogonal granularity issue – the granularity and the "size" of the fragment that conforms to such a definition. If fragments are too coarse-grained, thus containing restricted information and/or detail, it is likely that they will highly specific to a single situation (organizational context) i.e. their reusability may be limited and there may be partial overlaps between the specifications of fragment pairs [9]. In addition, it is arguable that fragments coming from repositories constructed on top of metamodels with very different granularities would suffer from interoperability and composability issues, since the abstraction levels at which they have been defined are naturally different. These three granularity issues are highly relevant to issues of SME, e.g. in terms of method construction, fragment storage, fragment interoperability and composability. In particular, a strategic, long-term research goal is an evaluation of the quality of the method fragments and the consequent quality of any methodology constructed from the fragments within the SME approach. In this paper, we concentrate on a precursor for a future quality evaluation by focussing on the granularity of method fragments in the context of their conformance to a metamodel.

Using conclusions from our earlier study of the granularity of metamodels [8], in the context of SME there are two major areas needing to be addressed:

1. the impact on method fragments of the scale and granularity of the metamodel e.g. definitions of method fragment types such as Activity, Task, Step etc. as compared to simply ProcessElement (Figure 1) [similarly Phase, Lifecycle etc.] i.e. the granularity of the metamodel [8].
2. the size of method fragments generated (usually by instantiation) from such method element definitions (made at the meta-level). Here, we focus on fragment generation from the Work Unit metaclass (Figure 2) but note that Work Product and Producer fragments are equally relevant, but are not discussed in any great detail here since analogous arguments apply.

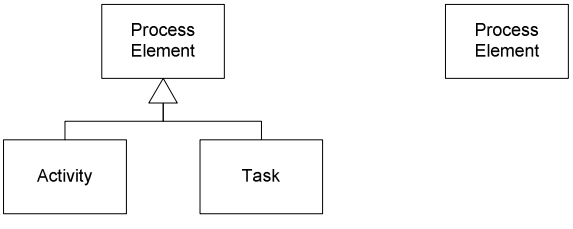

Fig. 1. Granularity of a metamodel might be fine-grained (left) or more coarse-grained (right) (after [8]) With kind permission of Springer Science+Business Media

In this paper, we summarize how to apply a theory of granularity abstraction e.g. [10-13] to the size of method fragments (we focus on fragments as being the only atomic element in SME, eschewing for the present the larger scale method chunk [14] and method component [15]). Following an overview of the theory of granularity (Section 2) and the typical structure of method fragments in Section 3, we then analyze in detail how fragment size and granularity can be optimized using these

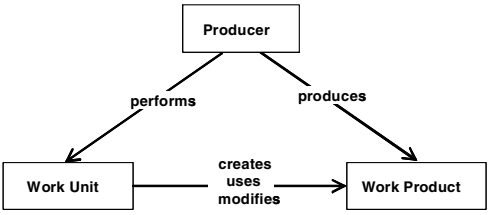

Fig. 2. The triangle of Producer, Work Unit and Work Product that underpins SPEM, OPF and ISO/IEC 24744 standards for software engineering process modelling (after [8]) With kind permission of Springer Science+Business Media

theories. In Section 4, we apply these ideas to a case study of one particular SME methodology: OPEN [16]. We conclude with some recommended guidelines for fragment specification (Section 5).

2 Theory of Granularity

Granularity theory e.g. [10,12] is based on *abstraction* e.g. [11,17,18]. Abstraction, roughly speaking [11], describes an approach in which one focusses on relevant characteristics of a problem (which is often called the "subject under study" (SUS)), whilst discarding its "irrelevant details" [13,18]. These authors [11] propose three informal properties:

1. the abstraction process maps a representation of the problem to a new, more "abstract" representation. (This is the essence of modelling).
2. by throwing away details, the result of the abstraction process provides a simpler problem (a.k.a. "an abstraction") to be solved than the original.
3. by preserving relevant, desirable properties, the solution of the abstracted problem can be transferred to the original, more complex problem, thus solving it.

Since an abstraction, α, necessarily contains less detail than the SUS on which it is based (property 2 above), this may result in several entities in the SUS, e_i, being mapped to a single one in the abstraction i.e.

$$\alpha(e_1) = \alpha(e_2) \neg \Rightarrow e_1 = e_2 \tag{1}$$

[19]. This loss of detail creates a simpler system from the SUS for the purposes of understanding the original SUS e.g. [11,13], although this property is excluded from the formal developments of [11] on account of its complexity. Rather, an abstraction can be defined formally as a mapping, α, between two formal systems, which may be similar or dissimilar e.g. [12]. Here, a formal system, Σ, is a set of formulae, Θ, written in a language Λ i.e.

$$\Sigma = (\Lambda, \Theta) \tag{2}$$

The definition (which addresses property 1 above) given by [11] is:

$$\alpha : \Sigma_1 \Rightarrow \Sigma_2 \tag{3}$$

where the mapping, α, is between a pair of formal systems (Σ_1, Σ_2) with languages Λ_1 and Λ_2 respectively and there is an effective total function, α_Λ, that maps "ground language" Λ_1 to "abstract language" Λ_2 i.e.

$$\alpha_\Lambda : \Lambda_1 \rightarrow \Lambda_2 \tag{4}$$

Of particular relevance to modelling are atomic abstractions, which are fragments that comprise no other method fragments [20] and are therefore instances of the finest granular classes in the corresponding metamodel.

Atomic abstractions can be classified [13] as symbol abstractions, arity abstractions and truth abstractions. Symbol abstractions can operate on constants, functions and predicates and are thus the most relevant to our present discussion. For example, for a symbol abstraction we have

$$x_1, \ldots\ldots x_n \in \Lambda_1, x \in \Lambda_2 \text{ and } \alpha(x_i) = x \text{ for all } i \in [1,n] \tag{5}$$

where x is either a constant, a function or a predicate.

Based on [10], Ghidini and Giunchiglia [13] suggest that symbol abstractions are in fact *granularity abstractions* (e.g. classification, generalization, aggregation). An abstraction α is defined by [12, citing 10 and 11] as a granularity abstraction if and only if

(i) f maps individual constants in Λ to their equivalence class under the indistinguishability relation ~ in Λ.
(ii) f maps everything else, including the predicates in Λ, to itself. (6)

Equation 5 maps *many* individual elements in a set to a single entity in a second set. A natural consequence of this, it is noted, is that granularity abstractions tend to lose information e.g. [21]. As a measure of the degree of granularity, we previously proposed [8] a simple measure: of the system granularity, G_S, as being related to the number of entities, n, in each system. Since it is reasonable to propose that the fine-grained system should have a smaller value for G_S than for a coarse-grained system, we hypothesized that the grain size (system granularity value) is thus a reciprocal measure of the number of granularity abstraction mappings (Equation 5 or 6) between two entities [10]. Thus

$$G_S = 1/n \tag{7}$$

This measure refers to entities represented in a single system or model. As noted above, granularity refers to the degree of decomposition/aggregation, generalization or classification levels often observed in terms of the number and size of extant entities and generally regarded as orthogonal. On the one hand, with a composition granularity abstraction, they take the role of "parts" in a whole-part (aggregation or meronymic) relationship. Thus moving from the "parts" (fine detail) to the "whole" (coarse detail) loses detail, reinforcing the notion that in many senses granularity is a kind of abstraction. In the OO literature, this removal of detail in the process of moving between granularity levels can be modelled not only by a whole-part relationship but also by a generalization relationship between two sets – the generalization granularity abstraction; or by an instance-of relationship between objects and their class – the classification granularity abstraction [21]. Consequently, making such parts or subclasses visible/invisible changes the granularity value of the overall system/model.

Granularity is thus a kind of abstraction that uses aggregation, generalization or classification relationships between entities to achieve simplification. This mechanism produces entities that are more coarse granular than the original, fine grained entities.

Granularity abstractions as applied to metamodels are discussed in [8] where the values of granularity for several current metamodels are given. In the next section, we see how these ideas can be used in the assessment of the granularity and size of method fragments generated to be conformant to a given metamodel.

3 The Granularity of Method Fragments

Situational method engineering relies on the use of stored fragments that each conform to some element in an agreed metamodel, where the metamodel is a model of models e.g. [22] and can thus be thought of as a language [23,24]. While, in principle, a metamodel may focus on the definition of work products, metrics, methodologies etc. (in the software engineering context), here, we assume that the metamodel focuses on software development methodologies, and examine the granularity of those fragments that conform to one particular metamodel element: WorkUnit (Figures 2 and 3).

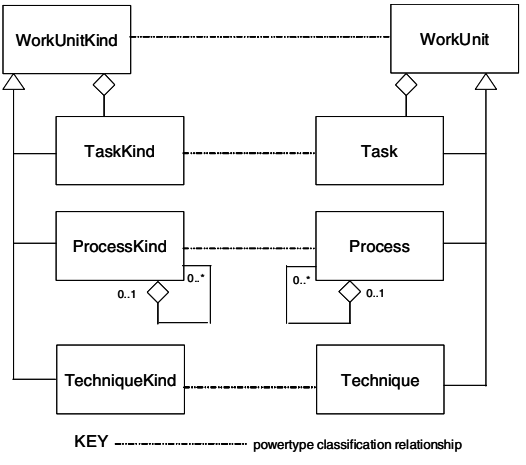

Fig. 3. The WorkUnit/WorkUnitKind metalevel classes together with the subtypes as defined in ISO/IEC 24744

In [8], we examined a number of methodology metamodels in which the values of G_S ranged from 0.25 to 1. Clearly this has a direct impact on the granularity and size of the conformant fragments. Assuming the overall size of the system is S, then if only one fragment were to be generated conformant with each meta-element, then for n meta-elements, the overall size would be related to the fragment sizes by

$$S=\Sigma^{n}_{i=1} f_i \tag{8}$$

where f_i is the size of the i-th fragment. For a constant size, S, this means that the fragment size is bigger for smaller n. Since it is likely that larger fragments are not

atomic and therefore of less than optimal quality, this size evaluation could be contributory to an overall evaluation of the quality of the method fragments in a repository and, by inference, the quality of the constructed methodology.

In this section, we highlight those fragments that depict work unit kinds i.e. what work needs to be done and how – but neglecting the "who", the "when" and the "what" for the sake of simplicity. Each work unit kind fragment is conformant to the WorkUnit/WorkUnitKind metaclass of Figure 2 or one of its subclasses: Process/ProcessKind, Task/TaskKind and Technique/TechniqueKind (Figure 3). We need to note for the discussion in Section 4 that this metamodel permits a Process/ProcessKind to consist of several Task/TaskKind fragments. In the OPEN Process Framework e.g. [6,16], this large agglomeration was known as an "Activity".

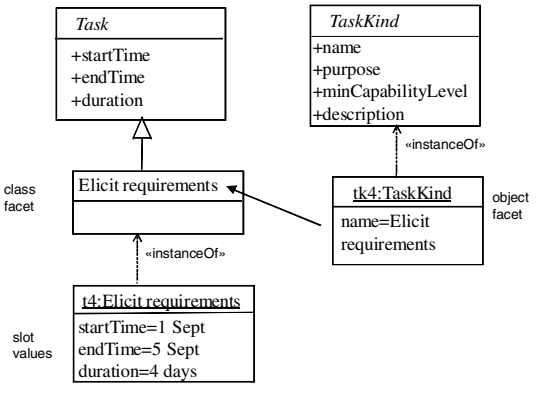

Fig. 4. Powertype pattern for Task/TaskKind and the class and object facets created together with an example of the process "object" in the Endeavour Domain

In the following example, we illustrate the relationship between a fragment and its defining metalevel class with the subclass of WorkUnit/WorkUnitKind named Task/TaskKind. Using ISO/IEC 24744, each concept is depicted in the metamodel using a powertype pattern [25] as shown in Figure 4. Powertype instantiation then depicts, at the model level (i.e. in the Method Domain), an entity with both a class facet (here ElicitRequirements) and an object facet (here t4:TaskKind). The powertype pattern is powerful because it not only permits representations at the model level but also, by instantiating the class facet of the method fragment (in the Model Domain), permits the allocation of project-specific values in the Endeavour Domain [26]. The method fragment is thus a combination of allocated values (from the object facet) and specified but unallocated attributes (from the class facet) (Figure 5).

It is clear from Figures 4 and 5 that any method fragment conformant to the Task/TaskKind powertype pattern that is defined in the ISO/IEC 24744 metamodel will, by definition, contain the exact same number of fields. Whilst most fields will necessarily be brief, the Description field is unconstrained. The long-term research question therefore devolves to an evaluation of "How long (number of words/number of concepts etc.) should a method fragment (such as Task/TaskKind in Figure 5) be in order for the fragment to be regarded as of "good quality"?"

To begin to answer this question, rather than simply a length evaluation, one should consider whether the Description really describes an atomic concept or not. In a particular context, we can determine whether or not a concept is atomic, such that it could be regarded as being of good quality, by "inverting" Equations 5, 7 and 8 to seek (a) a maximum value of n and, in parallel, (b) a minimum value for each f_i. In other words, we seek the set with the largest number of elements that satisfies Equation 5 and a parallel set $\{f_1 \ldots\ldots f_n\}$ such that each f_i is a minimum, whilst retaining a conformance of each of these elements to the relevant class in the metamodel.

Task Kind

Name:	Elicit requirements
Purpose:	To develop and refine a formal and stable requirements specification.
Minimum capability level:	1
Description:	Requirements are to be elicited from clients, domain experts, marketing personnel and users. Usual sub-tasks include defining the problem, evaluating existing systems, establishing user requirements, establishing distribution requirements and establishing database requirements.

startTime
endTime
duration

Fig. 5. The details of a Task/TaskKind fragment called Elicit Requirements

For example, a Task/TaskKind fragment for "Draw a use case diagram" could be considered atomic[2]. In contrast, one could argue that "Create a design for an atomic reactor control system" will necessarily involve a large number of (sub)tasks. Thus, if we are able to break down the fragment into a larger number of other fragments, we can readily deduce that the original fragment was not atomic and hence of poor quality. Adding a quantitative value to that "quality" does not, however, seem possible at this time. This is similar to the discussions well over a decade ago regarding "How big is a good quality object (or rather coded class) when using an object-oriented programming paradigm?" Indeed, our earlier research in this area [27] suggested strongly that there can never be an absolute cutoff threshold number but rather that there is a distribution that can be analyzed statistically such that the larger the size, the lower the probability (but not zero) that the class (and, by extension here, the method fragment) is of good quality.

In the next section, as an exemplar we investigate the current sizes of those fragments stored in the OPF repository [16], specifically those conformant to the WorkUnit/WorkUnitKind meta-element and its subtypes of Process/ProcessKind (OPEN's Activity) and Task/TaskKind (also called Task in OPEN). We analyze these in terms

[2] Although one could argue that it could be broken down as "Draw a symbol for each use case", "Add actors to use case diagram" etc., nevertheless in terms of the atomicity of "Task" it is reasonable to take as atomic since these more detailed elements such as "Add actors to use case diagram" are either Steps within the Task or associated Techniques linked with the Task.

of the granularity theory outlined in this and previous sections before making recommendations to improve the overall consistency in terms of their granularity.

4 Case Study Based on the OPF Metamodel and Fragment Definitions

4.1 The Current Situation

In the original published version of OPEN [16], modelling was seen as beginning to subsume and replace the subactivities of object-oriented analysis and object-oriented design. At the time, the general ideas of using models were gaining strength as a result of the publication of the Object Management Group's Unified Modeling Language or UML [28]. Despite the use of the word "modelling" within the subactivity called Evolutionary Development, itself embedded within the Build Activity of the OPEN Process Framework (as it was later renamed in [6]), in the formal description, modelling was really captured totally in the Task: *Construct the object model* (described in more detail in [16, p 160-162]) – see abbreviated version in the Appendix). However, the description of this "task" was extensive in both detail and scope and, indeed, in a later publication [6], the increasing size of this task was noted (page 274) where it is stated: "In this fairly Large-scale Task, …". Indeed, the description of this particular task includes a suggested list of supportive techniques that can be used to accomplish this task. The number of suggested techniques is 37. These are clearly not 37 alternatives from which one is to be chosen but rather a suite from which strictly more than one is necessary. Thus, this task cannot be considered to be atomic – as argued above, atomicity should be the goal. Thus, for instance when the model is of a Task implemented by a Technique (as in the OPF metamodel – now replaced by ISO/IEC 24744 [29]) – then there should probably (or at least on most occasions) be only one technique for each task – or at worst a *choice* of one technique from a possible suite of alternatives. What is not correct is that there should be a concatenation or suite of techniques that are mandatory to accomplish the task.

The Task *Construct the object model* is described [16] as "the prime technical focus of any OO methodology" where a number of model-descriptive diagrams are constructed. Six diagrams were listed (pre-UML) and of course now the whole gamut of 13 UML 2 diagram types would require support. The fact that so many diagram types are involved suggests strongly that the granularity of this current Task is much too coarse and that its scope is more akin to that of an OPF Activity than a single Task.

These granularity problems were compounded when the OPF repository of method fragments was extended to support agent-oriented method construction. During an extensive analysis of a large number of existing agent-oriented methodologies (summarized in [30]), a new OPF Task: *Construct the agent model* was introduced in parallel to the existing Task: Construct the object model. As more agent-oriented (AO) methods were analyzed, this AO Task grew in size. Thus, for instance, in the analysis of the PASSI methodology e.g. [31-33], which followed eight other analyses, a mapping from several PASSI tasks to the OPF Task: *Construct the agent model* was made. Specifically, Henderson-Sellers *et al.* [33] identified (from PASSI):

- Agent identification
- Task specification (actually suggested as a subtask)

- Ontology description (also OPF Task: Define ontologies)
- Role description (needing some extension)
- Agent structure definition (plus additional subtask)
- Agents behaviour description (new subtask)

That six PASSI tasks were covered by a single OPF task immediately suggests a granularity problem with the OPF Task: Construct the agent model – since, by definition, a task is atomic, being the smallest fragment that can be project managed.

The current situation is thus that there are two OPF Tasks: *Construct the object model* and *Construct the agent model* (Figure 6(a)). These two tasks would appear at first glance to have the same focus and scope but to be applied to two different technologies (the object model and the agent model). It is therefore appropriate to challenge the need for two such tasks as an exemplar of some consequences of imprecise definition of appropriate "granularity". Consequently, we seek to evaluate the efficacy of uniting these two originally disparate tasks. We show that, by focussing on *modelling*, we can ensure that a method fragment is created that is technology independent. Of course adding to the existing *Construct the object model* will make it even larger (Figure 6(b)), thus adding to the problems identified in the previous paragraph. In other words, using the theoretical discussions in earlier sections of this paper, we investigate (in Section 4.2 below) whether the converged modelling task outlined above is at the appropriate granularity.

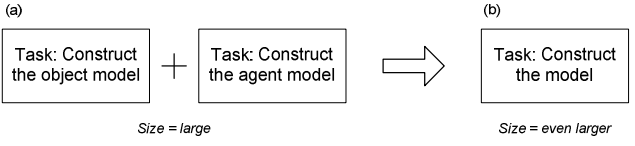

Fig. 6. Amalgamating two existing tasks, both already too large, creates an unacceptably large "tasks". Part (a) reflects the left hand side of Equation 5 with n=2 whilst part (b) represents the single element (x) with n=1.

4.2 The Proposed New Situation

In this section, we investigate two issues: (i) replacing the "bloated" *Construct the object/agent model* tasks of the original OPF with a larger set of much smaller granularity (n much larger or G much smaller: (Equation 7)) by ensuring that each task is an atomic abstraction i.e. a granularity abstraction (Equations 5 and 6) and (ii) seeking a balance between tasks and activities in this modelling area. These two issues are considered in the two following subsections.

4.2.1 New Tasks Derived from the Two Existing Modelling Tasks

We now examine the text in Appendix, which has been abstracted from the pertinent parts of the original description in [16]) for the Task: *Construct the object model*. We undertake essentially a textual analysis by identifying nouns that represent work products that need a task to produce them and verbs that represent the actions that are the core of a task. The list we have constructed includes:

- Identify classes and objects
- Identify roles to be played by objects
- Identify responsibilities of each class
- Add stereotypes[3]
- Implement responsibilities as class operations/methods and attributes
- Identify class-class relationships including possible meronymic (whole-part) representations
- Identify inheritance hierarchies
- Add constraints such as cardinalities
- Specify object behaviour, including its lifecycle
- Define state transition diagrams for each class, as necessary.

Then, from our studies of agent modelling, we can identify tasks such as

- Identify each agent in the system-to-be
- Specify the tasks associated with each agent
- Describe the roles that an agent may play in the system-to-be
- Describe the ontology associated with each agent
- Define the internals of each agent (agent structure)
- Design the behavioural aspects of each agent
- Design the interactions between agents

These two lists are of course peers but provide selection lists affiliated with the decision on which technology, objects or agents (or possibly a hybrid architecture) is to be used in any particular situation i.e. in constructing a particular situational method. The tasks are all atomic although space precludes formal proof of this.

4.2.2 Elevation to Activity Status and Merger with a Pre-existing Activity

Although tasks are the main focus of the work unit idea in the OPF, there is also an element called Activity. The need for an activity is twofold: firstly, to gather together a number of tasks with a "placeholder" that is at a higher abstraction level and, secondly, to be used in full lifecycle method construction on the grounds that, whilst a full-scale method may need many tens if not hundreds of tasks (and associated techniques), it will only have a handful (around a dozen or so) elements of the granularity of an Activity.

We therefore argue for the introduction of a new concept at a higher abstraction level than the tasks described above (i.e. an activity) called *Construct the model using the selected technology/paradigm*. Such a "promotion" would be in line with the philosophy underpinning the MDA and model transformations as well as the use of meronymy for creating a granularity abstraction hierarchy. This new activity then consists of a large number of tasks, these tasks being those listed above in Section 4.2.1 where each one meets the notion of abstraction atomicity defined in Section 3.

There is, however, one further technical discussion point in that there is a pre-existing Activity in the OPF called "Design" [6] that has these modelling concerns as one of its tasks. By elevating the model construction to an activity granularity, we in-

[3] Only when necessary and appropriate (ensuring correct definitions of each stereotype are available).

troduce a conflict of terminology. However, since publication, the OPF metamodel has been replaced by that of ISO/IEC 24744 and, although we have retained the terminology of the original OPF by using the name Activity rather than the name Process as preferred in ISO/IEC 24744, we can now take advantage of the recursive relationship in ISO/IEC 24744 that supports both processes and sub-processes (Figure 3) and thus consider the newly introduced *Construct the model using the selected technology/paradigm* as a subactivity or subprocess of the OPF *Design Activity/Process*.

5 Discussion and Related Work

Abstraction has long been recognized as a keystone of software engineering modelling [34, 35 ch. 3]. Two fundamental abstraction mechanisms are stated in [36] as being composition and classification. Whilst both are useful in discussing granularity [21], we have here concentrated on the former mechanism in our example of a Process/ProcessKind fragment in terms of an aggregation of Task/TaskKind fragments.

The granularity of method fragments in SME is discussed in [37, 38] and used by [39] – five categories are proposed qualitatively: method, stage, model, diagram and concept. Their finest granularity level (concept) is akin to the notion of atomicity discussed here. In the (object-oriented) methodological research literature, to the best of our knowledge, no-one has attempted to underpin discussions of granularity with theory – as discussed in Section 2 here. Indeed, Bettini and Ruffini [40] note, as do we, a history dating back to around 1985 with the publication of Hobbs [10] – although the focus in [40] is on the introduction of temporal constraints into discussions of granularity (and therefore out of scope for our discussion in this paper). In Unhelkar and Henderson-Sellers' [41] discussion of granularity, they focus on a qualitative evaluation of object-oriented designs in the context of reuse.

The introduction of two levels of granularity as in Section 4.2.2 (which distinguishes between the coarse granular entity labelled Activity and the finer granularity of the Tasks and their meronymic relationship) is a direct reflection of the basic granularity abstraction as defined in Equation 5. In practical terms it allows the use of both Tasks (small granularity, G) and elements, called here Activities (or Processes if using ISO/IEC 24744), of a larger granularity. However, as seen from Equations 5, 7 and 8, there is no unique mapping between these granularities. In the context of the application of granularity theory as expressed by these equations to SME situations such as described in Section 4.2.2, there remains an ambiguity regarding whether a given task or collection of Tasks is or is not at a high enough granularity (large value of G) to be called an Activity. This is seen for instance, qualitatively, in the discussions in [6] in their discussion of Environment Engineering and Project Management.

6 Conclusions and Recommendations

In this paper we have analyzed the theory of granularity in the context of method fragments used in situational method engineering and argued that it is important that each method fragment has a granularity that is atomic i.e. it cannot be broken down into a meronymically-based hierarchy i.e. the value of G (Equation 7) has been minimized. We have also argued that consistently-sized fragments should enhance composability

and interoperability of fragments across repositories in a service-oriented method engineering context, although this is a topic for further, future elucidation.

We have applied these ideas to one specific set of method fragments – those found in the repository of the OPEN Process Framework and documented in several books and research papers. We have taken as an illustration a single example – the previously documented concern that the task called *Construct the Object Model* supplemented by the task *Construct the Agent Model* had accreted to the extent that it can no longer be regarded as atomic. We have therefore analyzed the documented descriptions of these two so-called tasks and identified a larger number of smaller granularity tasks (atomic granularity) and propose that these should replace the two earlier tasks in the OPF repository.

Future work entails applying the same idea to all the other tasks within the OPF repository to locate any other tasks that are no longer atomic. This size increase is likely to have occurred when new software ideas, such as the introduction of web technologies and aspects, were included by the expediency of adding information to pre-existing method fragments, rather than the introduction of brand new (atomic size) method fragments. We therefore recommend for future work the systematic review of all method fragments in the OPF repository and, similarly, for other pre-existing method fragment repositories from other authors. Determining size and atomicity properties across repositories, even as aggregated values, is also suggested as a tentative way to document the abstraction level at which the concepts that underpin each particular repository have been captured. This should be explored from the perspective of tool-assisted composition of method fragments in a service-oriented SME context.

Of more widespread applicability are questions that could form the topic for future research that relates to the effect of changing granularity on the usability of methodologies. In addition, a more detailed study of the effects of granularity on reusability of fragments in comparison to chunks, for example building on the studies in [9], would make a valuable contribution to the SME literature.

Acknowledgements

This is paper number 10/06 of the Centre for Object Technology Applications and Research within the Centre for Human Centred Technology Design of the University of Technology, Sydney.

References

1. Welke, R., Kumar, K.: Method Engineering: A Proposal for Situation-Specific Methodology Construction. In: Cotterman, W.W., Senn, J.A. (eds.) Systems Analysis and Design: A Research Agenda. J. Wiley & Sons, Chichester (1991)
2. Brinkkemper, S.: Method Engineering: Engineering of Information Systems Development Methods and Tools. Inf. Software Technol. 38(4), 275–280 (1996)
3. Ågerfalk, P.J., Brinkkemper, S., Gonzalez-Perez, C., Henderson-Sellers, B., Karlsson, F., Kelly, S., Ralyté, J.: Modularization Constructs in Method Engineering: Towards Common Ground? In: Ralyté, J., Brinkkemper, S., Henderson-Sellers, B. (eds.) Situational Method Engineering: Fundamentals and Experiences. Proceedings of the IFIP WG 8.1 Working Conference. IFIP, vol. 244, pp. 359–368. Springer, Boston (2007)

4. Henderson-Sellers, B.: Method Engineering for OO System Development. Comm. ACM 46(10), 73–78 (2003)
5. Ralyté, J., Rolland, C.: An Assembly Process Model for Method Engineering. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 267–283. Springer, Heidelberg (2001)
6. Firesmith, D.G., Henderson-Sellers, B.: The OPEN Process Framework. An Introduction. Addison-Wesley, Harlow (2002)
7. Henderson-Sellers, B.: Process Metamodelling and Process Construction: Examples using the OPEN Process Framework (OPF). Annals of Software Engineering 14, 341–362 (2002)
8. Henderson-Sellers, B., Gonzalez-Perez, C.: Granularity in Conceptual Modelling: Application to Metamodels. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 275–288. Springer, Heidelberg (2010)
9. Henderson-Sellers, B., Gonzalez-Perez, C., Ralyté, J.: Comparison of Method Chunks and Method Fragments for Situational Method Engineering. In: Procs. 19th Australian Software Engineering Conference. ASWEC 2008, pp. 479–488. IEEE Computer Society, Los Alamitos (2008)
10. Hobbs, J.: Granularity. In: Procs. Int. Joint Conf. on Artificial Intelligence, IJCAI 1985 (1985)
11. Giunchiglia, F., Walsh, T.: A Theory of Abstraction. Artificial Intelligence 57(2-3), 323–390 (1992)
12. Mani, I.: A Theory of Granularity and its Application to Problems of Polysemy and Underspecification of Meaning. In: Cohn, A.G., Schubert, L.K., Shapiro, S.C. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR 1998), pp. 245–257. Morgan Kaufmann, San Mateo (1998)
13. Ghidini, C., Giunchiglia, F.: A Semantics for Abstraction. In: López de Mántaras, R., Saitta, L. (eds.) Procs. 16th European Conf. on Artificial Intelligence, ECAI 2004, pp. 343–347. IOS Press, Amsterdam (2004)
14. Ralyté, J., Rolland, C.: An Approach for Method Engineering. In: Kunii, H.S., Jajodia, S., Sølvberg, A. (eds.) ER 2001. LNCS, vol. 2224, pp. 471–484. Springer, Heidelberg (2001)
15. Wistrand, K., Karlsson, F.: Method Components – Rationale Revealed. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 189–201. Springer, Heidelberg (2004)
16. Graham, I., Henderson-Sellers, B., Younessi, H.: The OPEN Process Specification. Addison-Wesley, Harlow (1997)
17. Kaschek, R.: A Lttle Teory of Astraction. In: Rumpe, B., Hesse, W. (eds.) Modellierung 2004. Proceedings zur Tagung in Marburg/L. LNI, vol. 45, pp. 75–92. Springer, Berlin (2004)
18. Keet, M.: Enhancing Comprehension of Ontologies and Conceptual Models through Abstractions. In: Basili, R., Pazienza, M.T. (eds.) AI*IA 2007. LNCS (LNAI), vol. 4733, pp. 813–821. Springer, Heidelberg (2007)
19. Kühne, T.: Matters of (Meta-)modeling. Softw. Syst. Model. 5, 369–385 (2006)
20. Cervenka, R., Trencansky, I.: AML. The Agent Modeling Language. Birkhäuser, Basel (2007)
21. Keet, C.M.: A Taxonomy of Types of Granularity. In: Procs. IEEE International Conference on Granular Computing (GrC 2006), Atlanta, USA, May 10-12, pp. 106–111. IEEE Computer Society, Los Alamitos (2006)
22. Favre, J.-M.: Foundations of Model (Driven) (Reverse) Engineering: Models. Episode I: Stories of The Fidus Papyrus and of The Solarus. In: Bézivin, J., Hockel, R. (eds.) Procs. Dagstuhl Seminar 04101 “Language Engineering for Model-Driven Software Development” (2005)

23. Gonzalez-Perez, C., Henderson-Sellers, B.: Metamodelling for Software Engineering. J. Wiley & Sons, Chichester (2008)
24. Bertoa, M.F., Vallecillo, A.L.: Quality Attributes for Ssoftware Metamodels. In: Procs 13th TOOLS Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2010). IEEE Computer Society Press, Los Alamitos (2010) (in Press)
25. Henderson-Sellers, B., Gonzalez-Perez, C.: Connecting Powertypes and Stereotypes. J. Object Technol. 4(7), 83–96 (2005)
26. Gonzalez-Perez, C., Henderson-Sellers, B.: A Powertype-based Metamodelling Framework. Software and Systems Modeling 5(1), 72–90 (2006)
27. Haynes, P., Henderson-Sellers, B.: Cost Estimation of OO Projects: Empirical Observations, Practical Applications. American Programmer 9(7), 35–41 (1996)
28. OMG: UML Semantics, Version 1.0, OMG document ad/97-01-03 (January 13, 1997)
29. ISO/IEC: Software Engineering – Metamodel for Software Development. ISO/IEC 24744, Geneva, Switzerland (2007)
30. Henderson-Sellers, B.: Creating a Comprehensive Agent-oriented Methodology - Using Method Engineering and the OPEN Metamodel. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-Oriented Methodologies, ch. 13, pp. 368–397. Idea Group, Hershey (2005)
31. Burrafato, P., Cossentino, M.: Designing a Multi-agentSsolution for a Bookstore with the PASSI Methodology. In: Procs. 4th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS 2002), Toronto (May 2002)
32. Cossentino, M.: From Requirements to Code with the PASSI Methodology. In: Henderson-Sellers, B., Giorgini, P. (eds.) Agent-Oriented Methodologies, pp. 79–106. Idea Group, Hershey (2005)
33. Henderson-Sellers, B., Debenham, J., Tran, N., Cossentino, M., Low, G.: Identification of Reusable Method Fragments from the PASSI Agent-oriented Methodology. In: Kolp, M., Bresciani, P., Henderson-Sellers, B., Winikoff, M. (eds.) AOIS 2005. LNCS (LNAI), vol. 3529, pp. 95–110. Springer, Heidelberg (2006)
34. Hazzan, O., Kramer, J.: Abstraction in Computer Science and Software Engineering: A Pedagogical Perspective (2006), http://edu.technion.ac.il/Faculty/OritH/HomePage/FrontierColumns/OritHazzan_SystemDesigFrontier_Column5.pdf (accessed 28.4.2010)
35. Meyer, B.: Object-Oriented Software Construction, 2nd edn. Prentice-Hall, Englewood Cliffs (1997)
36. Jørgensen, K.A.: Modelling on Multiple Abstraction Levels. In: Procs. 7th Workshop on Product Structuring – Product Platform Development, Chalmers University of Technology, Göteborg (2004)
37. Brinkkemper, S., Saeki, M., Harmsen, F.: Assembly Techniques for Method Engineering. In: Pernici, B., Thanos, C. (eds.) CAiSE 1998. LNCS, vol. 1413, pp. 381–400. Springer, Heidelberg (1998)
38. Brinkkemper, S., Saeki, M., Harmsen, F.: Meta-Modelling Based Assembly Techniques for Situational Method Engineering. Information Systems 24(3), 209–228 (1999)
39. Sunyaev, A., Hansen, M., Kremar, H.: Method Engineering: A Formal Description. In: Papadopoulos, G.A., Wojtkowski, W., Wojtkowski, G., Wrycza, S., Zupančić, J. (eds.) Information Systems Development, pp. 645–654. Springer, New York (1999)
40. Bettini, C., Ruffini, S.: Deriving Abstract Views of Multi-granularity Temporal Constraint Networks. In: Hameurlain, A., Cicchetti, R., Traunmüller, R. (eds.) DEXA 2002. LNCS, vol. 2453, pp. 295–317. Springer, Heidelberg (2002)
41. Unhelkar, B., Henderson-Sellers, B.: ODBMS Considerations in the Granularity of a Reusable OO Design. In: Mingins, C., Meyer, B. (eds.) TOOLS 15, pp. 229–234. Prentice Hall, Upper Saddle River (1995)

Appendix: Task: Construct the Object Model

Textual description abstracted from the original source [16].

Associated Techniques
Abstract classes, association, blackboarding, BNF, collaborations, composition structures, connascance, contract specification, data-flow modelling, delegation, ER modelling, event charts, event modelling, formal methods, fuzzification, generalization, genericity specification, hypergenericity, implementation inheritance, information engineering, object lifecycle histories, ownership modelling, partitions, pattern recognition, Petri nets, power types, polymorphism, responsibilities, role modelling, service identification, state machines, stereotypes, task cards, transformations of the object model, usage, use cases, visibility.

Description
…….

Building this single object model can use a wide range of techniques (as listed above) dependent on whether the focus is on task modelling, business object modelling or system object modelling. At each stage, candidate CIRTs[4] are identified and the relationships between them expressed in terms of CIRT responsibilities leading to the use of associations, aggregations, containments, dependencies, collaborations and, later, inheritance structures. It is unnecessary that these relationships be fully defined or be accurate in the cardinality. It is better to draw informal connections (unlabelled and with deferred cardinality) than none at all between CIRTs[5]—mandatory constraints and cardinalities should not be enforced too soon as these are likely to change as the object model is continually refined. We find that often these rough sketches will aid in a rapid elimination of redundant or duplicate CIRTs. Once the initial relationships are identified they should be depicted in the appropriate object-oriented diagrams and fully documented.

…….

Whilst this is more realistically a representational issue, most methodologies, including OPEN, offer a suite of complexity management tools within the guidelines of the method itself. These are aimed at creating a self-consistent suite of diagrams which, together, document the totality of the one model. In earlier methods, the way these various model 'views' linked together was somewhat suspect. It is important that these orthogonal views, often at different abstraction levels (more or less detailed), *all* represent the same 'truth' in the model being created. For example, changing a message send in the dynamic model should change it similarly in its service representation within the CIRT interface; changing the name of a CIRT should be reflected in the use cases and vice versa. This is important particularly when CASE tool support is sought. The tool needs to have a global view despite the fact that any one diagrammatic representation…only shows a subset of the total information available for the model.

[4] In the original OPEN, CIRT was used as supertype of class, instance (object), role and type.
[5] This is possible using the TBD relationship icon in COMN but not possible in UML.