View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UCL Discovery

Fine-Grain Process Modelling

Bashar Nuseibeh Anthony Finkelstein Jeff Kramer

Department of Computing, Imperial College
180 Queen's Gate, London, SW7 2BZ, UK
Email: {ban, acwf, jk}@doc.ic.ac.uk.

Abstract

In this paper, we propose the use of fine-grain process modelling as an aid to software development. We suggest the use of two levels of granularity, one at the level of the individual developer and another at the level of the representation scheme used by that developer. The advantages of modelling the software development process at these two levels, we argue, include respectively: (1) the production of models that better reflect actual development processes because they are oriented towards the actors who enact them, and (2) models that are vehicles for providing guidance because they may be expressed in terms of the actual representation schemes employed by those actors. We suggest that our previously published approach of using multiple "ViewPoints" to model software development participants, the perspectives that they hold, the representation schemes that they deploy and the process models that they maintain, is one way of supporting the fine-grain modelling we advocate. We point to some simple, tool-based experiments we have performed that support our proposition.

Software Process Modelling

Process modelling is the construction of abstract descriptions of the activities by which software is developed. In the area of software development environments, the focus is on models that are enactable; that is, executable, interpretable or amenable to automated reasoning. Modelling the software development process is a means of understanding the ways in which complex software systems are designed, constructed, maintained and improved [7]. A software process model may also be used for method guidance; that is, as a vehicle for answering questions such as "what should I do next?" or more importantly "how do I get out of this mess I'm now in?".

Software process modelling is a complex activity that can span the entire software development life cycle, from requirements analysis and specification to system implementation, evolution and maintenance. From an organisational point of view, it is desirable to model the overall development process including the coordination and interaction of a large number of development participants. From the individual developer's point of view, while coordination and interaction are still important, the focus is on modelling development activities that fall within the domain of concern or responsibility of that individual developer.

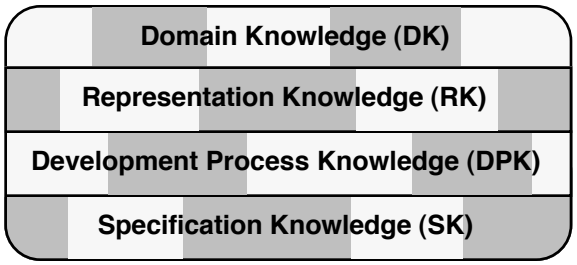
The development of large systems typically involves dealing with at least four kinds of knowledge:

- domain knowledge - "the world"
- representation knowledge - "the language"
- development process knowledge - "the strategy"
- specification knowledge - "the product"

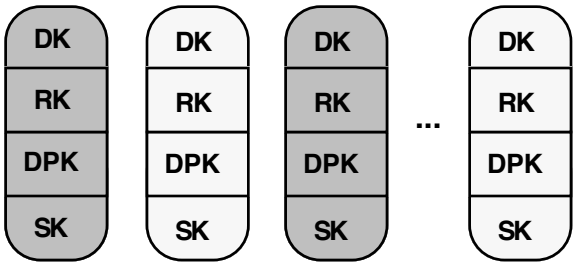
These have traditionally been identified and partitioned independently of each other, leading to possible mismatches between the partitions (fig. 1a). This has made the task of identifying and expressing the relationships and inter-dependencies between partitions of different kinds of knowledge much more difficult. What we suggest in this paper is that these different kinds of knowledge may be partitioned along the same lines, and then grouped together into multiple "objects" whose boundaries are defined by these partitions (fig. 1b).

Figure 1b reflects the fact that software development of complex systems involves many development participants who hold different views of the world and the software system they wish to construct. Moreover, these development participants may describe and elaborate their views using different representation schemes and by following different development strategies. Each participant has his or her own agenda of activities and goals and is only occasionally, if ever, concerned with the overall system development goals and objectives.

So what do we mean by fine-grain process modelling in this setting, and where does it fit in this view of the world?



(a) Global view of software development knowledge



(b) Proposed view of software development knowledge

Figure 1: *(a) Software process models have traditionally represented the software development process globally by identifying, partitioning and relating different kinds of knowledge independently (b) We believe that the different kinds of knowledge may be partitioned along the same lines and treated as single objects.*

Fine-Grain Software Process Modelling

The division of software development knowledge into many smaller units is the first step towards achieving a finer level of granularity of software process modelling. This decomposition into small units of knowledge results in many "smaller" process models each of which is typically associated with a single developer, representation or both. The models may represent short-lived processes that deal with individual activities or participants in the development life cycle. Fine-grain process modelling at this level of granularity is modelling at the *developer level*. This is in contrast with the more coarse-grain modelling that is concerned with more managerial and organisational activities such the synchronisation of tool invocations.

"Process integration" in this setting is then more than the usual (but non-trivial) task of producing a single, coherent process model for the overall development life cycle (e.g., [1], [9] and [15]). It is also concerned with "gluing" together many individual process models that must interact and cooperate in a coordinated manner in order to achieve the overall objectives of the development. Such coordination requires more than just synchronisation and concurrency control (e.g., [3]) or communication between agents in a cooperative setting (e.g., [16]) which are challenging enough, but also the reconciliation of fundamentally different development strategies encapsulated in different process models. This latter reconciliation may not in fact be entirely necessary, as different individual process models should be able to coexist, and only those areas of overlap need to be reconciled.

An even finer level of granularity than the developer level is *representation level* process modelling; that is, modelling a development process at the level of actions or activities that relate to or manipulate elements of a representation scheme. Fine-grain process modelling at this level is concerned with providing links between development process knowledge and representation knowledge which may then be used to produce a specification for a particular problem domain (fig. 2). A process model at the representation level is useful for the individual developer because it may be used to provide *guidance* expressed in a language the developer understands best - the language he or she is using! Thus, guidance can take the form of a recommendation on what to do next in order to advance the specification process, or advice on handling existing specification inconsistencies. This is in contrast with processes that are "unaware" of the representation schemes they manipulate and therefore treat them as coarse-grain "vanilla" objects with no internal structure or semantics.

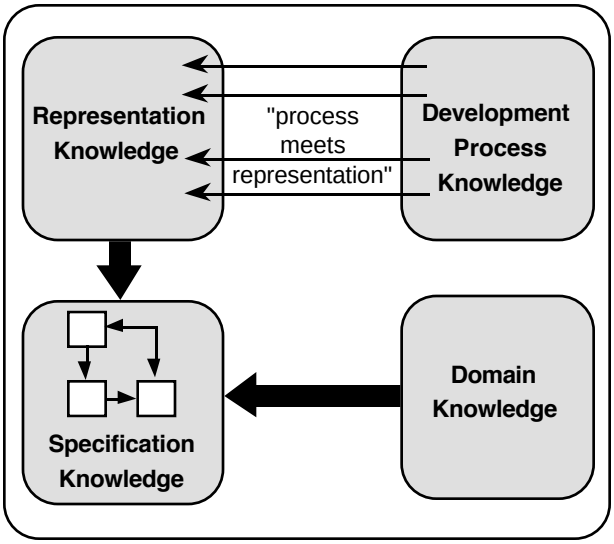


Figure 2: *Representation level process modelling provides a means of describing the relationship between development process knowledge and representation knowledge. This, together with any domain knowledge may be used to construct a specification.*

Developer and representation level process modelling are complementary activities. Creating developer level process models results in simpler representation schemes and process models, which are easier to relate, making representation level modelling easier. Conversely, representation level process models may become easier to integrate since the elements of overlap are mostly syntactic and are thus easier to transform, translate and check.

ViewPoints

In our previous work [5, 6] we have advanced the use of multiple, overlapping "ViewPoints" to model multiple development participants, who hold multiple views of a problem domain - described and developed using multiple representation schemes and development strategies, respectively. We defined ViewPoints as loosely-coupled, locally-managed, distributable objects, encapsulating representation knowledge, development process knowledge and (domain-specific) specification knowledge. Each ViewPoint thus captures a partial specification, the notation in which it is described, and the process deployed to develop it. Communication, consistency checking and information transfers between ViewPoints is done via one-to-one inter-ViewPoint rules [11].

ViewPoints are related to each other by many (simple?) mappings between representations. Consistency checks may be described and distributed among the various ViewPoints, so that invoking and applying them during ViewPoint development may then be used to *drive* the development process further (e.g., to perform inconsistency handling, information transformation and transfer, or simply some basic development steps such as further editing of ViewPoint specifications).

Experimental Tool Support

Supporting the ViewPoints framework, *The Viewer* environment and sample tools [10] illustrate the role of fine-grain process modelling in multi-perspective development.

The Viewer distinguishes between "method design" and "method use". During method design, the development techniques that make up a method are defined. Thus for each ViewPoint "type", a method designer may describe the notation and process which ViewPoints instantiated from that type will deploy. Since the representation and process are defined in the same ViewPoint, one can explicitly refer to representation level information when defining and constructing the ViewPoint process model. In *The Viewer* for example, we have experimented with a precondition → [Action] postcondition notation, so that if some preconditions hold, and an Action is performed, further postconditions then apply. Figure 3 is a screen-dump of the rudimentary "process modeller" provided by *The Viewer* during method design. Individual actions, pre- and postconditions may also be annotated with text to provide further context-sensitive guidance.

Note that the actual notation used to represent process models is not central to our argument. The above precondition/action/postcondition notation was chosen because of its simplicity and our past experience in using Modal Action Logic [4].

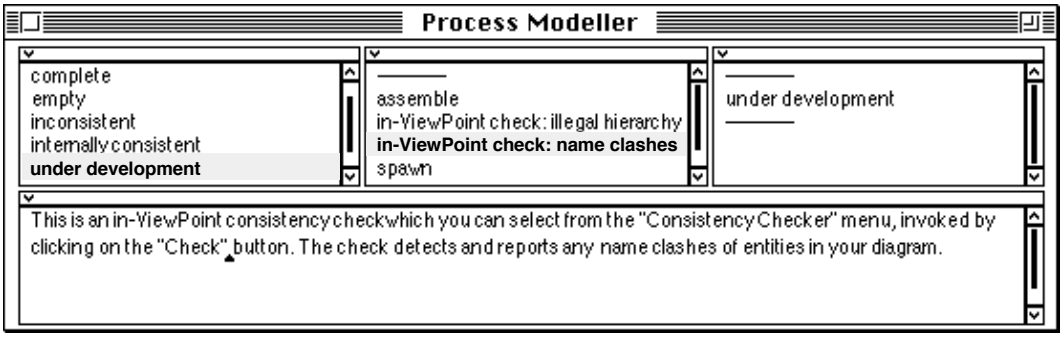


Figure- 3: *The Viewer's process modeller. The top three window panes (from left to right) contain the preconditions, actions and postconditions, respectively, for a particular ViewPoint type (development technique). The bottom pane contains textual annotations that may be added by the method designer to any particular condition or action.*

The benefit of using such a process modeller for each ViewPoint is that it allows us to describe actions of the development process in whatever granularity we choose. So, in fig. 3, the action "assemble" refers to any type of basic editing action (we don't want to be more specific in this case), whereas the next two actions are very specific consistency checks whose outcome may affect the development process in different ways.

During method use (ViewPoint development), a ViewPoint specification developer requesting guidance is presented with a list of possible actions

that he may perform in his current ViewPoint state (defined by the preconditions that hold at that time). He is also presented with the text annotations provided by the method designer to help him decide on the actions to take next. If he is still unable to understand the recommended actions, then he has the option of selecting a particular action and asking the tool to "enact" or "perform" that action on his behalf. He may of course, wish to ignore the guidance provided altogether (but *The Viewer* records the fact that guidance was given and not taken - a useful management and monitoring option). A guidance window provided for the method user by *The Viewer* is shown in fig. 4.

Finally, because no single process model is hard-coded into *The Viewer*, a ViewPoint developer may artificially "force" a ViewPoint into a particular state by selecting preconditions that apply. This may be used to the developer's advantage by (1) imposing stricter control over unwanted "automatic" forward chaining when preconditions hold, and (2) giving the developer the freedom to explore future development states and asking "what-if..." type of questions.

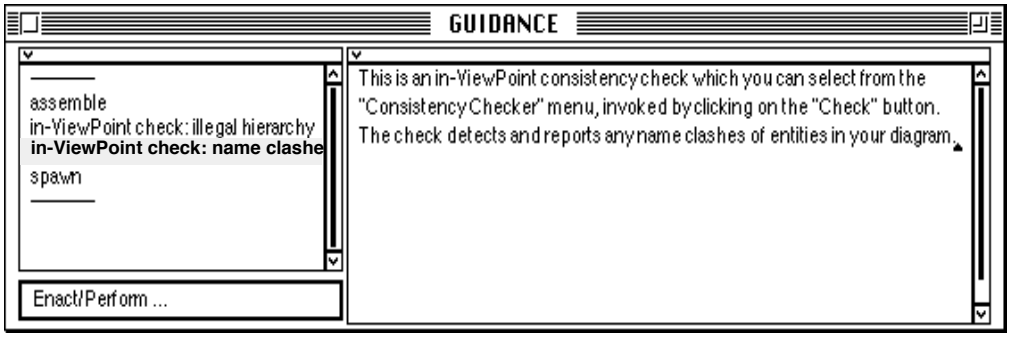


Figure 4: *The Viewer's process guidance window. The left window pane lists the possible recommended actions at this stage of development. The right window pane displays the context-sensitive guidance provided by the method designer when the process model was defined (figure 3). The button at the bottom-left corner "enacts" or "performs" the action selected from the list above.*

The Way Ahead

The sample tools described in the previous section demonstrate the feasibility of supporting fine-grain process models at the representation level. Developer level process modelling is more problematic. We believe that within the ViewPoints framework, inter-ViewPoint rules may be used to express the relationships between different ViewPoints and therefore different process models in a software development project. In such a setting, the invocation of such rules and their application is a means of integrating different process models and driving the overall development process. There are different approaches to rule invocation and application.

The "stupid" approach is to attempt to apply these inter-ViewPoint rules at all times during development, in which case the developer is constantly reminded of inconsistencies that exist between different specifications maintained by different ViewPoints. This is reminiscent of the support provided by first-generation CASE tools.

A step up from this is the more "pragmatic" approach in which the invocation and application of (inter-ViewPoint) consistency rules may be controlled (switched on and off) by the developer. With this approach, developers may elaborate their own areas of concern freely, only worrying about resolving conflicts at particular points during the development.

The approach we favour, which we realistically term the "problematic" approach, is to allow the process model to guide the developer not only during individual specification development, but also during inter-developer (*c.f.* inter-ViewPoint) communication. This should allow individual developers the freedom to develop their own domains of responsibility, *and* also help them resolve conflicts if and when they occur. In many instances however, conflicts and inconsistencies may occur which do not require immediate resolution, if at all. In such cases the process models should provide a means for *inconsistency handling* [8]. Thus one may have, for example, "process rules" of the form INCONSISTENCY implies ACTION, which specify how to act in the presence of inconsistency. These, and other such rules, in effect *drive* the software development process.

An analogous attitude towards inconsistencies in software development was proposed by Balzer [2], who in advocating "tolerating inconsistency" suggested that inconsistencies may be marked temporarily, avoided and then returned to later for possible resolution.

Inconsistency handling, conflict resolution and process integration are all areas that still require

further work, but we believe that studying fine-grain processes that interact with individual representations and developers brings us a step closer to understanding and supporting more effectively the overall software development process. This view is confirmed by our observation of other work in the area which has addressed issues of process model granularity; e.g., Perry's work outlined in [12], [13] and [14].

Our experiences in attempting to follow this line of investigation, using the ViewPoints framework as a vehicle, have, if nothing else, clarified the exact technical problems we need to tackle in order to progress further, and have therefore set our research agenda. In particular, we need to experiment within our ViewPoints framework with inter-ViewPoint communication protocols and investigate mechanisms for inter-ViewPoint rule invocation and application.

Acknowledgements

This work was partly funded by the UK Department of Trade and Industry (DTI) as part of the Advanced Technology Programme (ATP) of the Eureka Software Factory (ESF).

References

[1] A.W. Brown and J.A. McDermid (1992), "Learning from IPSE's Mistakes", IEEE Software, 23-28, March 1992.

[2] B. Balzer (1991), "Tolerating Inconsistency", Proceedings of 13th International Conference on Software Engineering (ICSE-13), 13-17th May 1991, Austin, Texas, IEEE CS Press, 158-165.

[3] N. Barghouti (1992), "Supporting Cooperation in the MARVEL Process-Centered Environment", ACM Software Engineering Notes, 17(5):21-31, December 1992.

[4] R. Cunningham, A. Finkelstein, S. Goldsack, T. Maibaum and C Potts (1985), "Formal Requirements Specification - The FOREST Project", Proceedings of 3rd International Workshop on Software Specification and Design, IEEE CS Press.

[5] A. Finkelstein, J. Kramer, and M. Goedicke (1990), "ViewPoint Oriented Software Development", Proceedings of International Workshop on Software Engineering and its Applications, Toulouse, France, December 1990.

[6] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein and M. Goedicke (1992), "Viewpoints: A framework for integrating multiple perspectives in system development", International Journal on Software Engineering and Knowledge Engineering, 2(1):31-57, March 1992, World Scientific Publishing Company.

[7] A. Finkelstein, J. Kramer and M. Hales (1992), "Process Modelling: a critical analysis", Integrated Software Reuse: management and techniques, P. Walton & N. Maiden (eds.), Chapman & Hall and UNICOM, 1992, 137-148.

[8] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer and B. Nuseibeh (1993), "Inconsistency Handling in Multi-Perspective Specifications", (to appear in) Proceedings of 4th European Software Engineering Conference (ESEC 93), Garmisch, Germany, 13-17th September 1993, Springer-Verlag.

[9] P. Mi and W. Scacchi (1992), "Process Integration in CASE Environments", IEEE Software, 45-53, March 1992.

[10] B. Nuseibeh and A. Finkelstein (1992), "ViewPoints: A Vehicle for Method and Tool Integration", Proceedings of 5th International Workshop on CASE (CASE 92), Montreal, Canada, 6-10th July 1992, IEEE CS Press, 50-60.

[11] B. Nuseibeh, J. Kramer and A. Finkelstein (1993), "Expressing the Relationships Between Multiple Views in Requirements Specification", Proceedings of 15th International Conference on Software Engineering (ICSE-15), Baltimore, Maryland, USA, 17-21st May 1993, IEEE CS Press, 187-196.

[12] D.E. Perry (1990), "Policy and Product-Directed Process Instantiation", Proceedings of 6th International Software Process Workshop (ISPW6), Hakodate, Japan, October 1990, IEEE CS Press.

[13] D.E. Perry (1991), "Policy-Directed Coordination and Cooperation", Proceedings of 7th International Software Process Workshop (ISPW7), Yountville, USA, October 1991, IEEE CS Press.

[14] D.E. Perry (1992), "Humans in the Process: Architectural Implications", Proceedings of 8th International Software Process Workshop (ISPW8).

[15] I. Thomas and B.A. Nejmeh (1992), "Definitions of Tool Integration for Environments", IEEE Software, 29-35, March 1992.

[16] L.G. Williams (1988), "Software Process Modelling: A Behavioural Approach", Proceedings of 10th International Conference on Software Engineering (ICSE-10), 11-15th April 1988, Singapore, 175-186.