Method Engineering for Multi-Perspective Software Development

Bashar Nuseibeh Anthony Finkelstein Jeff Kramer

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

(to appear in) Information and Software Technology Journal, February 1996, Butterworth-Heinemann.

Abstract

This paper examines the role of the method engineer in the context of multi-perspective software development. Such development is characterised by the existence of multiple development participants who hold multiple views on a system and its domain. These views may be described and developed using multiple representation schemes and development strategies respectively. The paper outlines the ViewPoints framework - an organisational framework developed to model such a scenario - and then examines the method engineering process required to support the kind of multi-perspective development described. The role of tool support in this context is also explored.

Keywords: viewpoints, method integration, meta-CASE, method engineering.

Introduction and Background

Multi-perspective software development engages multiple participants who invariably hold different views on a problem or solution domain. These participants include *clients* who have different, conflicting and complementary requirements of the software system they wish to acquire; and *developers* who must elicit, specify, analyse and validate these requirements, and then design and build a software system that satisfies the requirements. This is complicated by the fact that many of the above activities are distributed and concurrent, and pass through several iterations before (and frequently after) the software system is delivered. Moreover, these different participants deploy different notations and development strategies that are best suited for elaborating their respective areas of concern.

This "multiple perspectives problem" (Fig. 1) poses a number of challenging questions for any development environment or framework constructed to support such a heterogeneous development process. How does one structure, organise and manage these different perspectives? What are the relationships between different perspectives, and how are these relationships used to check consistency, and transfer and transform information held in different perspectives? And, how does one ensure distributed, concurrent and collaborative development between the different participants who hold these different perspectives?

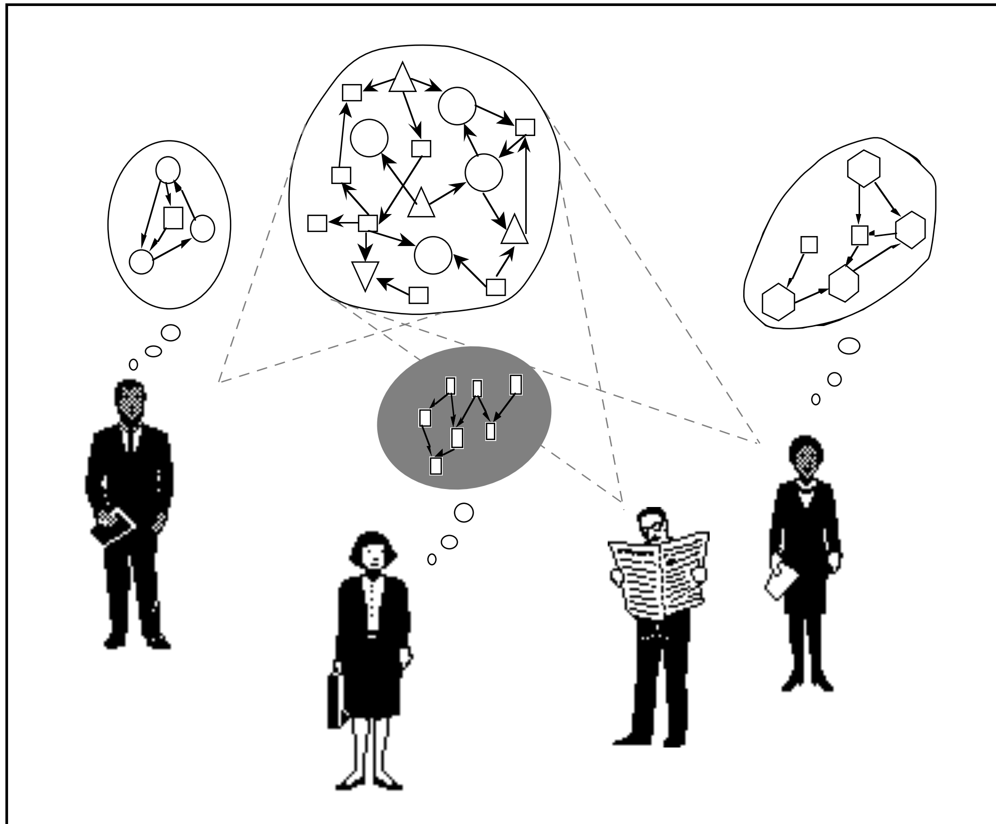

Figure 1: *The "multiple perspectives problem" - characterised by the existence of multiple views, notations, development strategies and development participants.*

In our previous work [1], we proposed a framework based on loosely coupled, locally managed, distributable objects called *ViewPoints*, which model the perspectives described above. Each ViewPoint encapsulates representation knowledge (the notation), development process knowledge (the development strategy) and specification knowledge (the product) for a particular domain. A *system specification* in this context is then a configuration or structured collection of ViewPoints. These ViewPoints represent the views or perspectives that clients and developers hold on different areas of concern. These areas of concern may be partially, totally or non-overlapping.

This paper explores the role of an additional participant in the above setting, that of the *method engineer*, and describes the kind of support such an engineer needs to provide to facilitate the multi-perspective development. In general, a method engineer is responsible for designing and constructing a development method that provides developers with systematic procedures and guidance on how to deploy one or more notations for describing a problem or solution domain. Increasingly, the role of the method engineer has become more application domain specific, with "customised" or "situation-specific" methods assembled for particular organisations or projects [2]. This has been made easier with the availability of improved tool support, including emerging *Computer Aided Method Engineering (CAME)* tools [3] such as *meta-CASE tools* [4, 5].

In theory, the activities of method engineers precede those of software (application) developers. In practice however, the roles and activities of method engineers, software developers and clients are inextricably linked and often overlap (Fig. 2). Both clients and software developers will make demands on the method engineer: the client may require that the software developer use a particular method, and the system requirements may necessitate the customisation of that method for a particular project. The method engineer, while ultimately interacting mostly with a software developer by delivering a method for him to use, will do this based on feedback from both developer and client. His role may also extend throughout the development life-cycle (that is, even after delivering the method to the software developer), since the method deployed may be modified or enhanced (evolved) as the development proceeds.

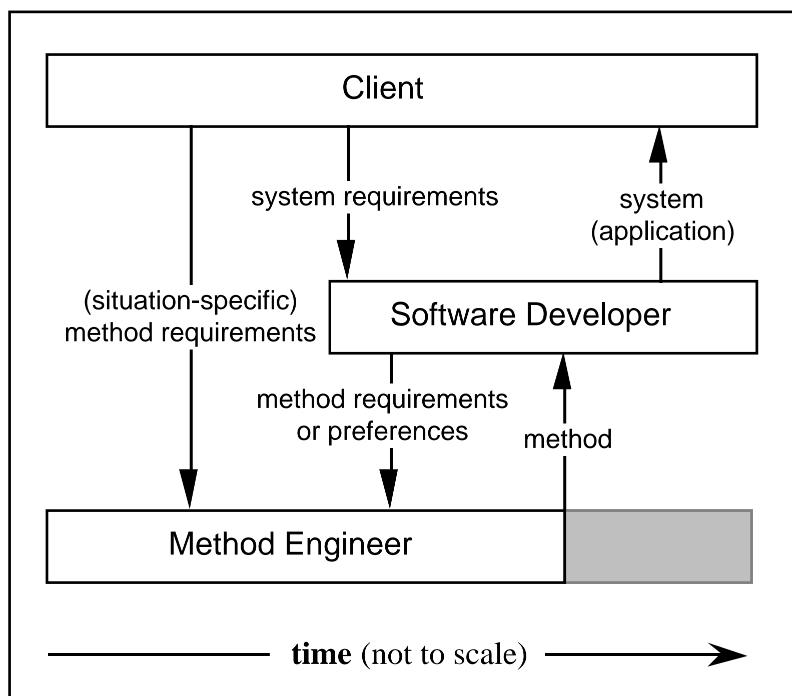

Figure 2: *An outline of the relationships between the three kinds of software development participants. The shaded portion of the method engineer's activities indicates that his involvement in the development process continues even after he delivers the software development method to the software application developer. Labelled vertical arrows represent information flows.*

The remainder of this paper is organised as follows. The next section, summarises multi-perspective software development within the ViewPoints framework, and defines the notion of a software engineering method in this context. The following section describes the method engineering process for ViewPoint-oriented development, and outlines some heuristics for method design and construction from reusable method fragments or components. The need for method integration is then highlighted, and is defined and discussed in this context. Finally, the scope for tool support - particularly for the method engineer - is outlined, and our experiences in using the proposed approach is summarised.

ViewPoint-Oriented Development

Composite systems, particularly those which deploy different technologies, often require a number of representation schemes and development strategies to describe their behaviour. Their development by a large number of participants further complicates matters by introducing different perspectives or views on overlapping areas of concern. To model this scenario we have defined *ViewPoints to be loosely coupled, locally managed, distributable objects that encapsulate representation knowledge, development process knowledge and specification knowledge about a system and its domain.* Thus a single ViewPoint contains a partial specification described in a suitable (formal) notation and developed by following a particular development strategy. A ViewPoint *owner* acts as the domain knowledge provider and as the enactor of the development process model contained within the ViewPoint.

A ViewPoint is internally divided into five "slots":

- **(1) Style** contains a definition of the representation scheme deployed by the ViewPoint,
- **(2) Work plan** contains a description or model of the development process,
- **(3) Specification** contains a partial specification described in the notation defined in the style slot, and developed by following the process described in the work plan slot,
- **(4) Domain** contains a label identifying the area of concern of the ViewPoint,
- **(5) Work record** contains the specification development status, history and rationale.

A ViewPoint owner thus enacts a ViewPoint *work plan* to produce a ViewPoint *specification* in a ViewPoint *style* for a particular ViewPoint *domain*. The ViewPoint *work record* maintains the development history (and hence development status and rationale) of a ViewPoint specification. This separation of concerns both between and within ViewPoints allows the representation within the framework of notions such as "roles" or "agents" in the process modelling sense, and "views" or "perspectives" in the specification sense. The framework also provides a very basic model for collaboration - however, detailed modelling of cooperative working groups is a complex activity covering many disciplines [6, 7], and is beyond the scope of this paper. What has concerned us thus far has been very specific interactions between particular ViewPoints related to each other by very specific syntactic or semantic relationships [8]. Moreover, and in this paper in particular, we are concerned with the method engineering process behind the creation and development of ViewPoints between which such relationships hold.

Method Engineering in the ViewPoints Framework

In general, the notion of a software engineering method is used to mean a collection of procedures and heuristics for systematically deploying one or more notations to describe a system [9]. Taking on board our ViewPoints model of software development, we can think of a method as a collection of method fragments, in the same way as we treat a system specification as a collection of ViewPoints. Each method fragment describes how to develop a single ViewPoint specification (i.e., a partial system specification). Clearly however, several ViewPoints may deploy the same notation and development strategy to produce different ViewPoint specifications for different domains. There is therefore a need to define a reusable ViewPoint "type" or "template" from which several ViewPoints (instances) may be instantiated.

Looking at a ViewPoint, we see that the style and work plan slots may be treated as reusable units: one can deploy the same representation scheme and development strategy many times to produce different partial specifications. We call a ViewPoint in which only these two slots have been defined a ViewPoint *template*, and we can think of it as a definition of a single development technique or "primitive" method. We can now treat a software engineering method as a configuration or structured collection of ViewPoint templates (Fig. 3).

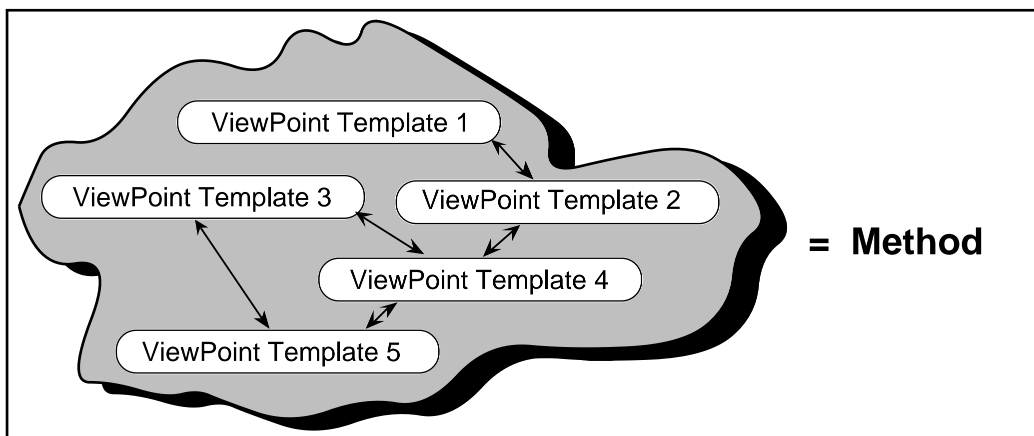

Figure 3: *A software engineering method is a configuration (structured collection) of ViewPoint templates. These templates describe the various development techniques that the method comprises. The arrows indicate the inter-ViewPoint relationships that must hold for partial specifications in the different ViewPoints to be consistent.*

In [10], the principles of configuration programming used to describe the *structure* of distributed systems were used to describe the structure of the JSD method [11] in terms of the ViewPoint templates it contains (or more accurately, in terms of the ViewPoint templates the method engineer *decided* it should contain). Configuration programming [12] advocates the use of an interconnected-component model for software design and construction through evolution. This has proved to be particularly useful because software engineering methods in general do not deploy a sequential series of procedures, but are

rather composed of several loosely coupled stages each containing one or more development techniques or sub-techniques (by development technique we mean a representation scheme and a procedure of deploying it). Thus by treating ViewPoint templates as components in a configuration, a system specification is then developed by creating instances of those templates, which are created or "spawned" dynamically as the development proceeds.

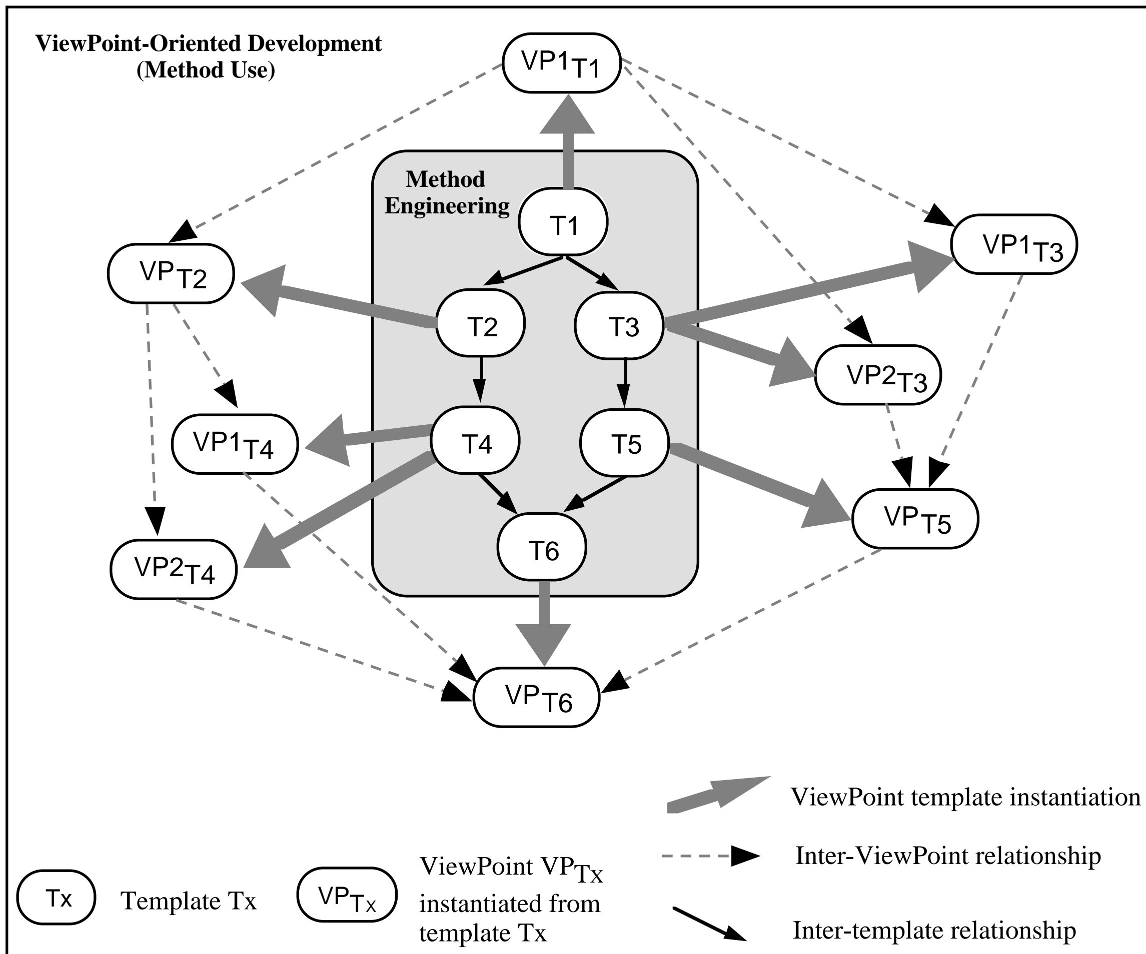

Figure 4: *Method engineering includes constructing the ViewPoint templates that make up a method, and defining the inter-template relationships between them (shaded box). ViewPoint-oriented development includes instantiating the ViewPoint templates to create new ViewPoints and instantiating the inter-template relationships to establish inter-ViewPoint relations.*

Fig. 4 schematically outlines top level activities of method engineering and ViewPoint-oriented development. What the diagram does not show is the iterative process of refinement in which individual ViewPoints are (partially) developed, and then modified and evolved as a result of traversing the inter-ViewPoint relationship arrows which represent inter-ViewPoint consistency checks, transformations and transfers.

Fig. 5 illustrates in a little more detail the method engineering tasks of method design and construction. From somewhat uncertain requirements of a software engineering method, a method engineer identifies the development techniques that the target method will deploy. The next step is then to identify the ViewPoint templates that need to be

constructed to describe these techniques. A good heuristic in such situations is to identify as many "simple" development techniques as possible (e.g., those that deploy simple notations) and then choose a single template to describe each technique. Of course this may not always be possible, since a method engineer may be restricted in terms of the kinds of techniques or notations required by his "customers"[1].

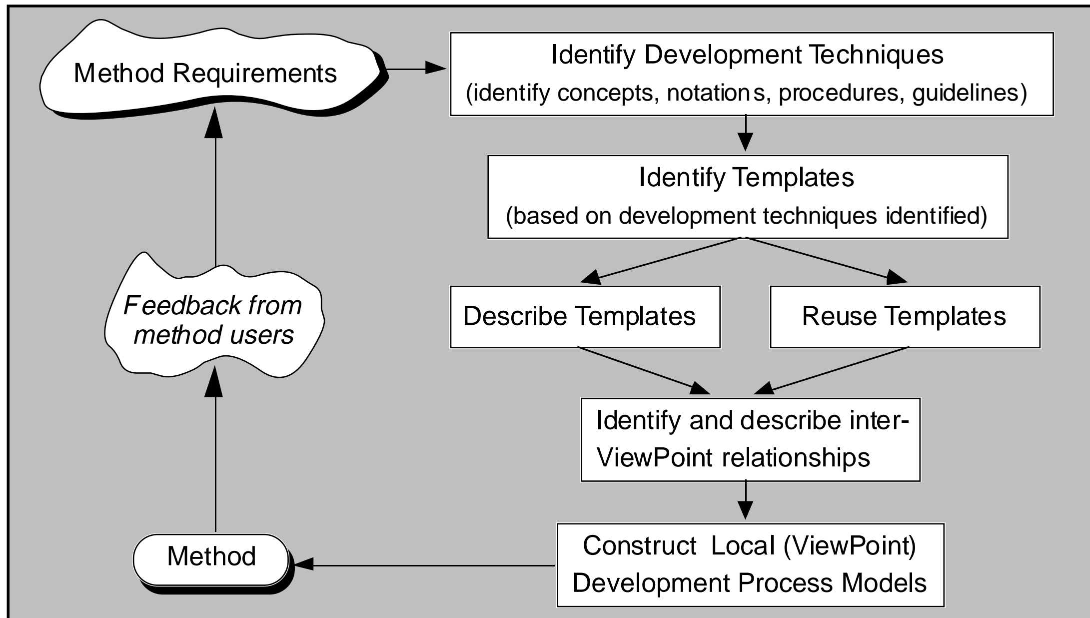

Figure 5: *Method design and construction in the ViewPoints framework.*

Defining ViewPoint templates involves elaborating the style and work plan for each template. Representation styles may usually be defined using a meta-language or model, such entity-relationship modelling for graphic-based notations, or BNF grammars for text-based notations. Work plans may be defined in terms of the actions that *may* be performed to build a ViewPoint specification in the defined ViewPoint style. This usually includes basic assembly (editing) actions, in- and inter-ViewPoint checking actions and inconsistency handling actions [13]. Work plans also contain some form of process model to guide a ViewPoint developer about *when* it is advisable to perform any of the available actions.

Strictly speaking, inter-ViewPoint check actions may be defined separately from the individual ViewPoint templates because they are not part of any single development technique *per se*, but rather describe relationships that the method engineer wishes to express between these techniques. They are also more method-dependent and therefore less reusable than the remaining elements of a ViewPoint template. Thus, although they are included as part of the ViewPoint template definition to maximise the distributability of ViewPoints, they are nevertheless maintained separately so that they may be more

---

1 "Customers" of method engineers are software developers who deploy (use) the method(s) produced by method engineers.

easily located and modified when reusing templates. Inter-ViewPoint check actions are key actions to achieving method integration, which is discussed in the next section.

Modelling the process by which multiple development participants develop multiple partial specifications using multiple representations is a complex task. *Process programming* has been proposed as a way of describing and then executing such process activities [14]. This however, has proved somewhat elusive for reasons initially voiced by [15] and since then by others including [16]. What has been more effective has been the notion of so-called "fine-grain process modelling" [17] to provide development *guidance* at the level of the individual developer deploying a single representation. This is the (pragmatic) approach used in the ViewPoints framework [18]. "Simple" process models are defined locally (that is, one model in the work plan slot of each individual ViewPoint), each of which can then be used to provide guidance for the individual ViewPoint specification developer in terms of the notation he or she is using (that is, the particular notation defined in the ViewPoint style slot). The notion of method guidance in the context of requirements engineering methods was also explored in the TARA project described in [19].

The outcome of the various method engineering activities described above is a *method* that is aimed at satisfying the initial requirements. This method may now be deployed by the ViewPoint-oriented developer (the method user).

Method Integration

"... having divided to conquer, we must now reunite to rule." [20]. In other words, having advocated distributed, multi-perspective software development by deploying multiple ViewPoints instantiated from multiple templates, it is then necessary to ensure that these ViewPoints are somehow "glued" together to form some kind of integrated system specification. To achieve such integration, the multiple templates that together constitute "the method", must themselves be integrated in order for the ViewPoints that are instantiated from them to be integrated. This *method integration* is useful for combining the strengths and reducing the weaknesses of selected methods. It is generally achieved by (1) integrating common features of several methods, (2) extending a "main" method with features taken from "supplementary" methods, or (3) restricting a main method by replacing or overriding some of its features with features taken from supplementary methods [9].

In [8], we advocated the use of pairwise *inter-ViewPoint relationships or rules* that act as the integration vehicle. These rules are defined by the method engineer, and express the syntactic (and therefore to some extent the semantic) relationships between methods' constituent templates. Typically these rules might describe an equivalence relation

between two elements of notations described in two different template style slots. Thus, they may be invoked and applied by the method user (the software developer) to ascertain whether or not two partial specifications are consistent; or alternatively they may be used more dynamically to transfer, translate and/or transform partial specification information from one ViewPoint to another in order to make the two ViewPoints consistent. Consistency in this context then, is the satisfaction of the defined inter-ViewPoint rules, and integration is achieved by achieving consistency.

The existence of many inter-ViewPoint rules means that we may speak of *partial consistency* (and partial integration); i.e., only *some* inter-ViewPoint rules may hold. In fact, we can go further by proposing that "absolute" or "total" consistency is not realistically possible, or even desirable, in many circumstances - and therefore our inter-ViewPoint rules only represent our "best effort" in defining what we mean by total consistency. This satisfies one of our objectives of tolerating inconsistency during development and supporting multiple, possibly conflicting, views. In [13], we take this further by exploring the notion of inconsistency handling, in which we define rules (of the form Inconsistency implies Action), that specify how to act in the presence of inconsistency. We have also adopted this approach in managing inconsistencies in evloving requirements specifications [21].

Consistency checking as a vehicle for method integration is beyond the scope of this paper. To summarise our approach to consistency management however, we treat the application of inter-ViewPoint rules as the enactment of a tuple <R, I>,

where, R is the inter-ViewPoint rule of the general form: $VP_S \Re VP_D$,

where, VP_S and VP_D are the source and destination ViewPoints, respectively, between which the inter-ViewPoint rule is defined and the relationship $\Re$ holds,

and, I is the inconsistency handling rule of the form: $\perp \Rightarrow A$,

where, $\perp$ is an inconsistency,

and, A is the set of inconsistency handling actions.

Thus, for every inter-ViewPoint rule, R, there is an associated rule, I, in the tuple <R, I>, which specifies how to handle inconsistencies resulting from the failure of the rule.

Tool Support: CAME and Meta-CASE

Two kinds of computer-based toolkits appear to be necessary to support method engineering in the ViewPoints framework. The first includes tools for defining ViewPoint templates (such as structured text and graphics editors), and tools for cataloguing and accessing templates from reuse libraries. The second kind, collectively known as *meta-*

CASE tools, lie on the boundary between CAME and traditional CASE tools. Meta-CASE tools take formal descriptions of methods as input, and generate CASE tools to support these methods. Two such tools include the IPSYS ToolBuilders Kit™ [4] and VSF [5].

The Viewer [22] is a prototype environment supporting the ViewPoints framework developed by the Distributed Software Engineering Group at Imperial College. It provides rudimentary tools of the two kinds outlined above. These may be accessed by the so-called Template Browser shown in Fig. 6. The Template Browser allows the definition of ViewPoint templates (both style and work plan slots), and partially generates tools for supporting the development of ViewPoints (partial specifications) instantiated from these templates.

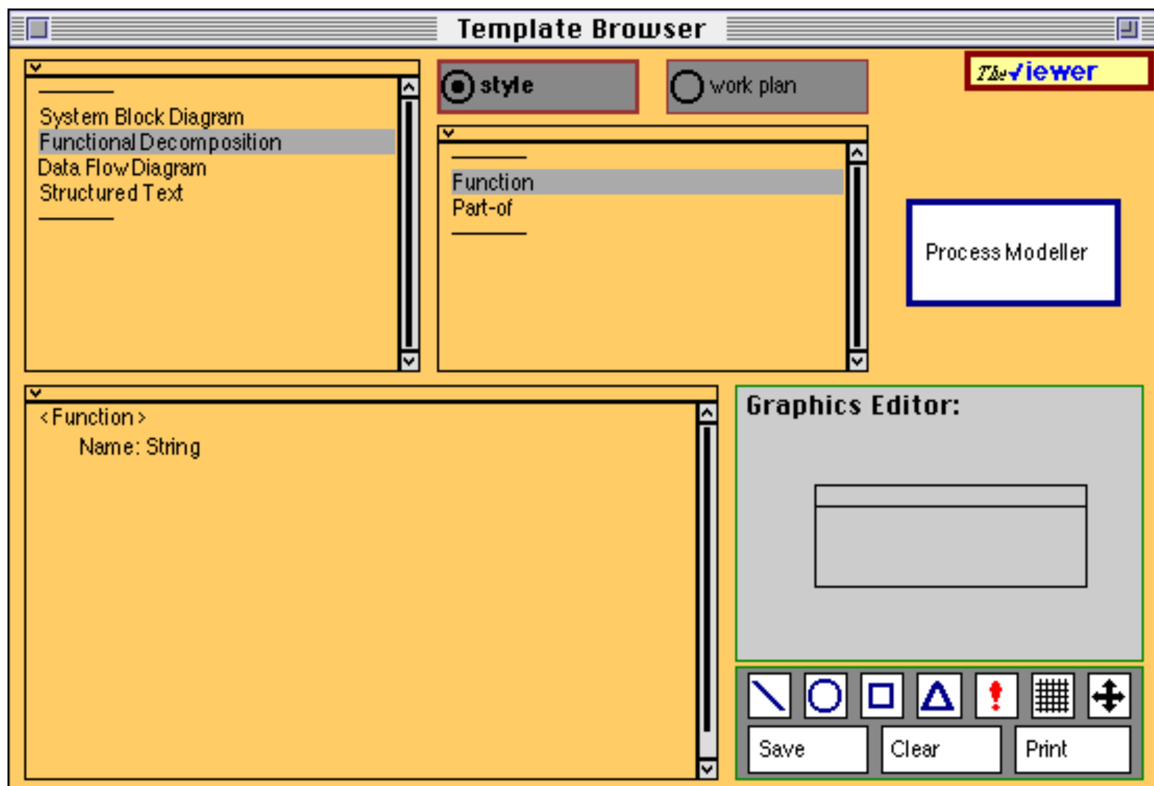

Figure 6: *The Template Browser of The Viewer. Template names are shown in the top-left hand of the window. A graphical icon editor is provided in the bottom-right part of the window, while other textual attributes of style and work plan are defined in the bottom-left window pane.*

For example, the method engineer may define the graphical attributes of a notation in the graphical icon editor. This is then "picked-up" by *The Viewer* and added to the assembly (editing) menu of the ViewPoint specification editor. In this way, the method engineer may also play the role of the CASE tool developer, where the method description (if completely formalised) may also serve as the tool description. Moreover, the inter-ViewPoint rules used to describe the relationships between ViewPoint templates (and later between ViewPoints), may also be used as the tool integration rules (Fig. 7). This is consistent with the view that tool integration is the "implementation" of the "intentions" of method integration [23].

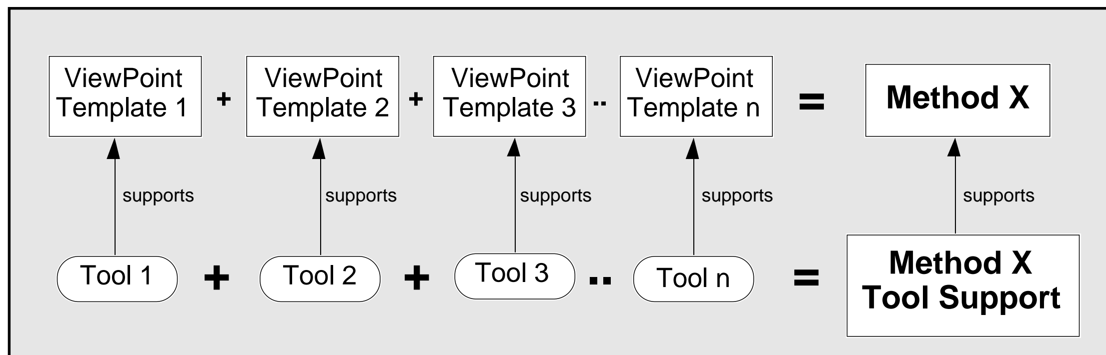

Figure 7: *CASE tool developers may construct tools to support each of a method's constituent development techniques (templates). In the case where the template descriptions are concise and formal enough, the description of a template may serve as the description of the tool required to support it - provided the meta-CASE tools technology is in place to generate it* [22].

Experiences

The ViewPoints framework and *The Viewer* have been deployed and used in a number of case studies and research environments. A number of "standard" methods have been described using ViewPoint templates including CORE [24], the Constructive Design Approach [25] and SCS [26]. Moreover, an object-oriented method developed by Hewlett-Packard (UK) called FUSION [27] has been described in terms of ViewPoint templates and partly supported by *The Viewer* [28]. Siemens (Germany) have also used the framework to describe an extended form of Petri Nets and developed ViewPoint-based tools to build and simulate such nets [29]. Finally, the feasibility of integrating human factors development techniques into software engineering methods has also been explored in this context.

The above experiences have provided us with feedback on improving the structure and organisation of the ViewPoints framework, and in the case of the industrial partners it has provided them with a fresh outlook on their methods and tools, and pointed to a number of improvements that may be incorporated into their methods.

These experiences have also provided us with important insight into what constitutes a "good" software engineering method in this context. We hold the view that useful software engineering methods should comprise many simple, highly redundant, preferably graphical notations, with many simple consistency checks described and enacted between them to achieve method integration. This does not appear to produce an $n * (n - 1)$ explosion of consistency checks for n number of ViewPoints, since in practice not all notations in a method are necessarily related. In fact, "good" software engineering methods have a small number of well-defined relationships between *some* of the constituent development techniques they deploy.

Conclusions

In this paper we have examined method engineering in the context of multi-perspective software development, as exemplified by the ViewPoints framework. In particular, we have explored the design and construction of methods in terms of ViewPoint templates - the reusable method fragments which serve as types from which ViewPoints (containing partial specifications) may be instantiated.

The importance of method integration was highlighted, and it was proposed that this be achieved via inter-ViewPoint rules that express the relationships between methods' constituent templates (and therefore the ViewPoints instantiated from them).

The role of computer-based tool support for the method engineer in the context of the ViewPoints framework was explored, and included traditional structured editors for defining ViewPoint templates, cataloguing tools for reusing templates from libraries, and meta-CASE tools technology for generating CASE tools to support these templates.

Further work is necessary to examine the nature of interactions between ViewPoints after they have been instantiated from their templates. In particular, we need to explore ways in which the method engineer can express these interactions during method design and construction. Such a study would provide us with a better understanding of the multi-perspective development process, and is therefore currently underway.

Acknowledgements

We would like to thank the anonymous reviewers for their constructive comments on an earlier draft of this paper. The work described 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). Partial funding was also provided by the ESPRIT Basic Research Action PROMOTER and the UK EPSRC project VOILA.

References

1. Finkelstein, A., J. Kramer, B. Nuseibeh, L. Finkelstein and M. Goedicke (1992); "Viewpoints: A Framework for Integrating Multiple Perspectives in System Development"; *International Journal of Software Engineering and Knowledge Engineering*, 2(1): 31-58, March 1992; World Scientific Publishing Co.

2. Kumar, K. and R. Welke (1992); "Methodology Engineering: A Proposal For Situation-Specific Methodology Construction"; *(In) Challenges and Strategies for Research in Systems Development;* W. W. Cotterman and J. A. Senn (Ed.); 257-269; John Wiley Series in Information Systems, John Wiley & Sons Ltd., Chichester, UK.

3. Harmsen, F. and S. Brinkkemper (1993); "Computer Aided Method Engineering based on existing Meta-CASE Technology"; *Proceedings of 4th European Workshop on the Next Generation of CASE Tools (NGCT '93)*, Sorbonne, Paris, France, 7-8th June 1993, Memorandum Informatica 93-32, University of Twente, Holland.

4. Alderson, A. (1991); "Meta-CASE Technology"; *Proceedings of European Symposium on Software development Environments and CASE Technology*, Königswinter, Germany, June 1991, 81-91; LNCS, 509, Springer-Verlag.

5. Pocock, J. N. (1991); "VSF and its Relationship to Open Systems and Standard Repositories"; *Proceedings of European Symposium on Software development Environments and CASE Technology*, Königswinter, Germany, June 1991, 53-68; LNCS, 509, Springer-Verlag.

6. Joosten, S. and S. Brinkkemper (1993); "Modelling of Working Groups in Computer Supported Cooperative Work"; *Proceedings of the 18th Annual International Conference on Information Technologies and Programming*, Sophia, Bulgaria.

7. Easterbrook, S., A. Finkelstein, J. Kramer and B. Nuseibeh (1994); "Coordinating Distributed ViewPoints: The Anatomy of a Consistency Check"; *Concurrent Engineering: Research and Applications*, 2(3): 209-222, CERA Institute, West Bloomfield, USA.

8. Nuseibeh, B., J. Kramer and A. Finkelstein (1994); "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification";*Transactions on Software Engineering*, 20(10): 760-773, October 1994; IEEE Computer Society Press.

9. Kronlöf, K. (Ed.) (1993); *Method Integration: Concepts and Case Studies*, Wiley Series in Software-Based Systems, John Wiley & Sons Ltd., Chichester, UK.

10. Kramer, J. and A. Finkelstein (1991); "A Configurable Framework for Method and Tool Integration"; *Proceedings of European Symposium on Software development Environments and CASE Technology*, Königswinter, Germany, June 1991, 233-257; LNCS, 509, Springer-Verlag.

11. Cameron, J. (1989); *JSP & JSD: The Jackson Approach to Software Development*; 2nd Edition, IEEE Computer Society Press, Washington, USA.

12. Kramer, J. (1991); "Configuration Programming - A Framework for the Development of Distributed Systems"; *Proceedings of International Conference on Computer Systems and Software Engineering (CompEuro 90)*, Tel-Aviv, Israel, May 1990, 374-384; IEEE Computer Society Press.

13. Finkelstein, A., D. Gabbay, A. Hunter, J. Kramer and B. Nuseibeh (1994); "Inconsistency Handling in Multi-Perspective Specifications"; *Transactions on Software Engineering*, 20(8): 569-578, August 1994; IEEE Computer Society Press.

14. Osterweil, L. (1987); "Software Processes Are Software Too"; *Proceedings of 9th International Conference on Software Engineering (ICSE-9)*, Monterey, California, USA, 3rd March - 2nd April 1987, 2-13; IEEE Computer Society Press.

15. Lehman, M. M. (1987); "Process Models, Process Programs, Programming Support"; *Proceedings of 9th International Conference on Software Engineering (ICSE-9)*, Monterey, California, USA, 30th March - 2nd April 1987, 14-16; IEEE Computer Society Press.

16. Finkelstein, A., J. Kramer and M. Hales (1992); "Process Modelling: A Critical Analysis"; *(In) Integrated Software Engineering with Reuse: Management and Techniques;* P. Walton and N. Maiden (Ed.); 137-148; Chapman & Hall and UNICOM, UK.

17. Nuseibeh, B., A. Finkelstein and J. Kramer (1993); "Fine-Grain Process Modelling"; *Proceedings of 7th International Workshop on Software Specification and Design (IWSSD-7)*, Redondo Beach, California, USA, 6-7 December 1993, 42-46; IEEE Computer Society Press.

18. Leonhardt, U., A. Finkelstein, J. Kramer and B. Nuseibeh (1995); "Decentralised Process Enactment in a Multi-Perspective Development Environment"; *Proceedings of 17th International Conference of Software Engineering*, Seattle, Washington, USA, 255-264; IEEE Computer Society Press.

19. Finkelstein, A. and J. Kramer (1991); “TARA: Tool-Assisted Requirements Analysis”; *(In) Concept Modelling, Databases and CASE: An Integrated View of Information Systems;* P. Loucopoulous and R. Zicari (Ed.); 413-432; Wiley, UK.

20. Jackson, M. (1990); “Some Complexities in Computer-Based Systems and Their Implications for System Development”; *Proceedings of International Conference on Computer Systems and Software Engineering (CompEuro ‘90)*, Tel-Aviv, Israel, 8-10th May 1990, 344-351; IEEE computer Society Press.

21. Easterbrook, S. and B. Nuseibeh (1995); “Managing Inconsistencies in an Evolving Specification”; *Proceedings of 2nd International Symposium on Requirements Engineering (RE 95)*, 27-29th March 1995, York, UK, IEEE Computer Society Press.

22. Nuseibeh, B. and A. Finkelstein (1992); “ViewPoints: A Vehicle for Method and Tool Integration”; *Proceedings of 5th International Workshop on Computer-Aided Software Engineering (CASE ‘92)*, Montreal, Canada, 6-10th July 1992, 50-60; IEEE Computer Society Press.

23. Wybolt, N. (1991); “Perspectives on CASE Tool Integration”; *Software Engineering Notes*, 16(3): 56-60, July 1991; SIGSOFT & ACM Press.

24. Mullery, G. (1985); “Acquisition - Environment”; *(In) Distributed Systems: Methods and Tools for Specification;* M. Paul and H. Siegert (Ed.); LNCS, 190, Springer-Verlag.

25. Kramer, J., J. Magee and A. Finkelstein (1990); “A Constructive Approach to the Design of Distributed Systems”; *Proceeding of 10th International Conference on Distributed Computing Systems*, Paris, France, 28th May-1st June, 580-587; IEEE Computer Society Press.

26. Castro, J. and A. Finkelstein (1991); “VSCS: An Object Oriented Method for Requirements Elicitation and Formalisation”; *FOREST project report,* NFR/WP2.2/IC/R/002/A; Department of Computing, Imperial College, London, UK.

27. Coleman, D., P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes and P. Jeremaes (1993); *Object-Oriented Development: The Fusion Method*; Prentice-Hall, Engelwood Cliffs, NJ, USA.

28. Ballesteros, L. A. R. (1992); “Using ViewPoints to Support the FUSION Object-Oriented Method”; *M.Sc. Thesis,* Department of Computing, Imperial College, London, UK.

29. Graubmann, P. (1992); “The HyperView Tool Standard Methods”; *REX project report,* REX-WP3-SIE-021-V1.0; Siemens, Munich, Germany.