IEEE Proc. of Int. Workshop on CASE (CASE 92), 6-10th July 1992, Montreal, Canada, pp.50-60.

ViewPoints: A Vehicle for Method and Tool Integration

Bashar Nuseibeh Anthony Finkelstein

Department of Computing
Imperial College, London, SW7 2BZ
Email: ban@doc.ic.ac.uk

Abstract

This paper proposes an object-based framework for the development of heterogeneous, composite systems. Such systems require the use of multiple notations and development strategies to describe multiple developer perspectives. The framework employs coarse-grain objects, called ViewPoints, that represent "agents" having "roles-in" and "views-of" a problem domain. These ViewPoints are loosely coupled, locally managed encapsulations, integrated via inter-ViewPoint consistency relations and transformations.

Tool integration is treated as a special case of method integration, and is demonstrated by The√iewer - a prototype support environment presented in this paper. The√iewer supports the proposed framework, and illustrates how ViewPoints may be used for method design, description, integration and use. Developed in Objectworks/Smalltalk, it maps the object-based framework onto an object-oriented implementation. The top level architecture and implementation of The√iewer is also briefly presented.

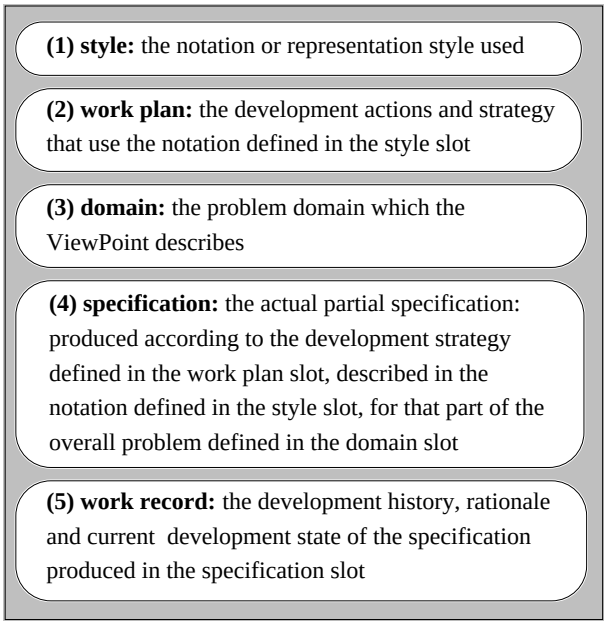
1: Introduction

The development of heterogeneous, composite systems requires the use of a number of different methods at different stages of the development process. These methods employ different notations and development strategies that must be integrated to produce system specifications.

This paper proposes an object-based, organisational framework for systems engineering methodology. The framework uses *ViewPoints* [4] to describe system development participants, their roles in the development process, and their views of the problem domain. The framework is used as a basis for the provision of integrated environment support for the specification, design and implementation of *large, heterogeneous, composite systems* [16]. Such systems deploy a variety of different technologies and require a variety of expertise for their development. The ViewPoints framework acknowledges the need to support different development notations and strategies, and attempts to provide a mechanism for their customisation and integration.

2: ViewPoints

Large systems development projects typically consist of a number of participants, engaging in the partial specification of system components. These participants frequently employ different notations and development strategies to produce descriptions of different (or the same) problem domains. *We define a ViewPoint as a loosely coupled, locally managed object, encapsulating representation knowledge, development knowledge and specification knowledge of a particular problem domain.* A ViewPoint encapsulates this systems development knowledge in five separate *slots:*



Each ViewPoint is associated with a particular development participant called the ViewPoint *owner*. The ViewPoint owner is responsible for enacting the

ViewPoint work plan to produce a ViewPoint specification, in the ViewPoint style, for the owner's domain of responsibility.

Clearly, a number of ViewPoints may employ the same style (eg, functional decomposition) and the same work plan (eg, a top-down process), to produce different specifications for different domains. We therefore define a reusable *ViewPoint Template* (Figure-1) in which only the style and work plan slots are elaborated.

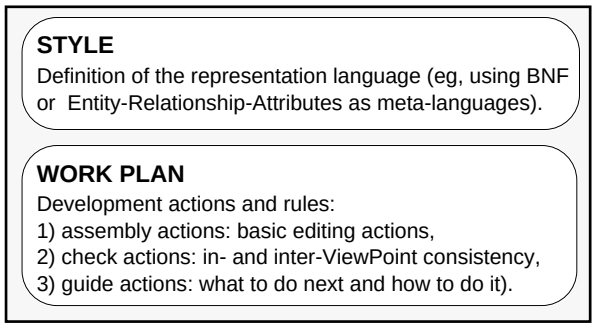


Figure-1: *The two slots of a ViewPoint Template.*

A ViewPoint template is in effect a ViewPoint type, such that a single ViewPoint template may be instantiated several times to yield several different ViewPoints (Figure-2), and by extension several ViewPoint specifications (e.g., the sample ViewPoint in Figure-3).

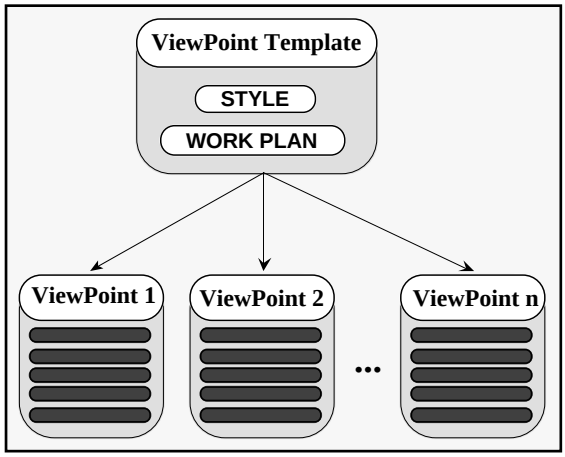


Figure-2: *A single ViewPoint Template (describing a single development technique: notation and process), may be instantiated to produce several ViewPoints, each ViewPoint containing a specification for a different domain in the overall problem. Thus one might envisage producing several dataflow diagrams in the course of a system specification project - but the notation and development strategy for producing dataflow diagrams would be defined only once in a ViewPoint template.*

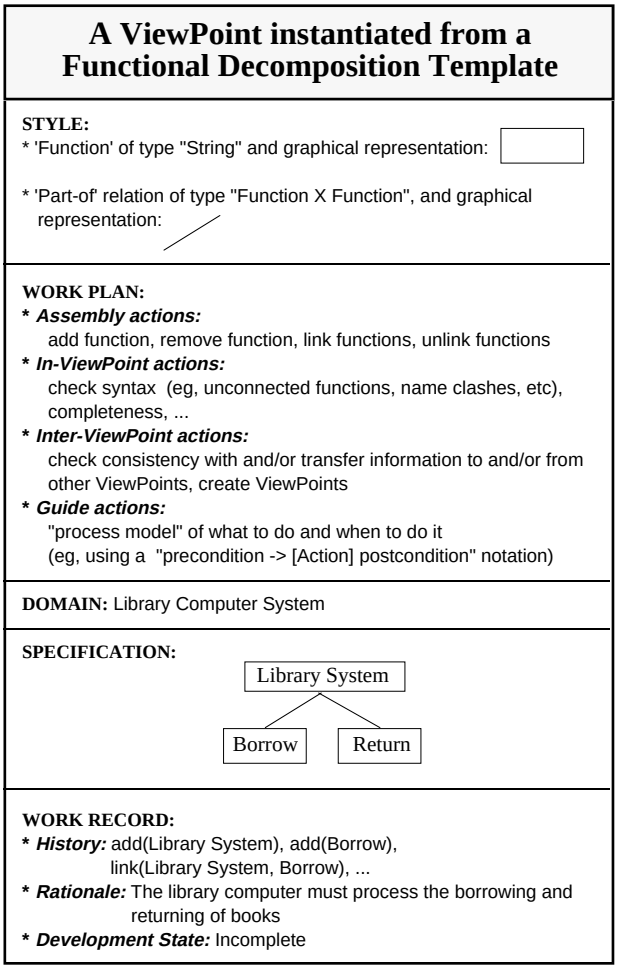


Figure-3: *A simplified example of a "typical" ViewPoint instantiated from a Functional Decomposition Template. Many such ViewPoints may be instantiated from that single template, each producing a different functional hierarchy (specification) for a different domain. The Functional Decomposition Template is simply a ViewPoint with only the Style and Work Plan slots filled.*

A software engineering *method* in this context is a collection of ViewPoint templates, representing the constituent development techniques of the method. Customised methods, or combinations of methods, may thus be constructed by grouping together the relevant templates. The binding of these templates is via *Inter-ViewPoint Rules* that relate the style components of the various templates. These rules are effectively the method integration mechanism, and are used to check consistency between ViewPoints.

This ViewPoint-Oriented Systems Engineering (VOSE) framework has a number of analogies in the object-oriented world [2]. The framework is "object-based" as opposed to

"object-oriented" to highlight the (deliberate) omission of inheritance. Strictly speaking, the provision of ViewPoint templates (types) means VOSE is "class-based" [17]. It is however, a systems development paradigm rather than a language, with more emphasis on objects (ViewPoints) rather than classes (Templates). Table-1 lists some of the mechanisms and terminology commonly used in object-oriented paradigms, with their VOSE equivalents where applicable.

Object-Orientation	VOSE
Object	ViewPoint
Class	Template
Encapsulation of state and behaviour	Encapsulation of state in the specification, domain and work record slots; and behaviour in the style and work plan slots
Information Hiding: Objects can only be changed by object operations	Information Hiding: ViewPoint specifications can only be changed by work plan actions
Inheritance	Not Available (deliberately omitted to maximise encapsulation of templates and distribution of ViewPoints ... inheritance works against efficient distribution)
Polymorphism: the binding of a message to different methods, depending on the type of the receiving object	Polymorphism simulated (as we neither have inheritance nor dynamic binding): identical work plan actions may be used to build different specifications, depending on the ViewPoint Template instantiated
Message Passing	Message Passing - available, but not in the strict object-oriented sense; rather, in the distributed software engineering context: via inter-ViewPoint consistency checking mechanisms and bindings

Table-1: *Analogous mechanisms and terminology between object-oriented [2] and ViewPoint-oriented systems engineering (VOSE).*

The omission of inheritance reflects the firm commitment of VOSE to distributed development by a number of participants in a cooperative setting. Inheritance, by its nature, imposes dependency between classes (templates) and the superclasses from which they inherit. In a distributed environment, classes and their superclasses may be located on different nodes that are separated geographically. The implementation of inheritance in such settings is likely to be, at least, inefficient. Furthermore, each ViewPoint template is essentially the description of a specification development technique. Thus, it is a reusable component that may be customised and used in different methods. A coupling relationship, such as inheritance, between it and another template reduces its reusability.

In addition to ViewPoint template reuse, VOSE also provides the opportunity for the reuse of actual specifications and designs. A ViewPoint (ie, an instantiated Viewpoint template) encapsulates a specification and its development rationale for a particular problem domain. This encapsulation and modularity makes ViewPoints highly reusable, although mechanisms for managing and integrating reuse into the development process are still required - a remain elusive.

ViewPoint development, management and reuse all benefit from automated support, and the next section describes *The√iewer* - a prototype environment providing such support.

3: Tool Support: *The√iewer*

The√iewer is a VOSE support environment implemented in Objectworks/Smalltalk Release 4. The smalltalk system was chosen as the development environment for a number of diverse reasons:

(1) The inherent object-based nature of VOSE allows the convenient representation of ViewPoints as smalltalk objects.

(2) The suitability of the smalltalk environment for rapid prototyping and exploratory programming facilitates the evolutionary development of the VOSE framework.

(3) The accessibility of smalltalk on a number of different platforms: *The√iewer* was developed on an Apple Macintosh IIci, and runs under X-Windows on Sun workstations and Windows on IBM PCs.

(4) The desire to offer, at the prototyping stage, a programmable platform for the VOSE tool developer.

(5) Previous experience in implementing graphical CASE tools and consistency checking mechanisms in smalltalk [13].

3.1: Scope

The startup window of *The√iewer* (Figure-4) conveniently illustrates the scope of the VOSE environment. On the one hand, a "method designer" is provided with the opportunity to design, describe and integrate ViewPoint templates that constitute a method. On the other hand, a "method user" may instantiate pre-defined templates to yield concrete ViewPoints, whose specifications may be developed, checked and managed within the boundaries of a system development project.

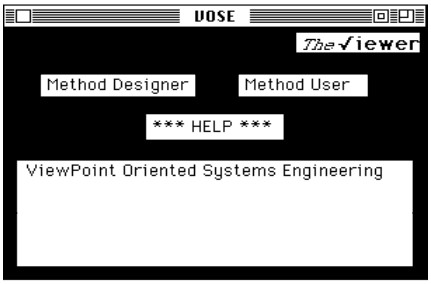


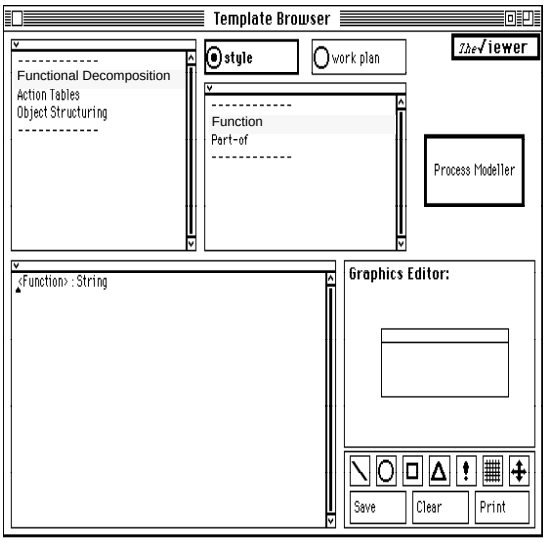
Figure-4: *The startup window of The√iewer defining the scope of the VOSE environment. The "Method Designer" button creates a Template Browser (Figure-5), while the "Method User" button creates a ViewPoint Configuration Browser (Figure-6).*

3.2: Method Design

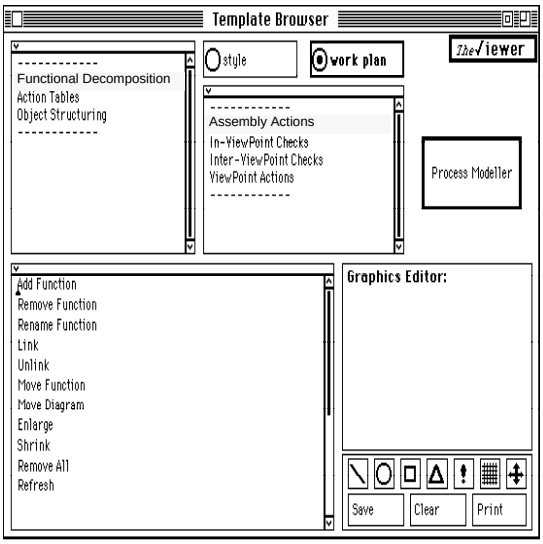
In the VOSE context, method design takes the form of ViewPoint template description and integration. The constituent techniques of "the method" are chosen, and then a template for each is elaborated. *The√iewer* provides a Template Browser (Figure-5) to facilitate such activities. In Figure-5 a customised example method is being defined. It consists of three simple graphical development techniques: functional decomposition, object structuring and action tables. The functional decomposition template has been selected in this example.

In Figure-5a, the style slot of the functional decomposition template is being described, while Figure-5b shows part of the work plan description. The consistency relations and transformations between templates of the method are defined in the "Inter-ViewPoint Checks" part of the work plan (shown but not selected in Figure-5b).

The Graphics Editor at the bottom right of the Template Browser provides the method designer with tools to draw and customise icons that may be part of a representation style and that may then become part of method user tools.



(a)



(b)

Figure-5: *Two snapshots of The√iewer's Template Browser. ViewPoint template names are listed in the top left window pane. For a selected template the user may describe (a) the style, by clicking on the 'style' button, or (b) the work plan, by clicking on the 'work plan' button'. The descriptions may be textual (bottom left) and/or graphical (bottom right). The 'Process Modeller' button provides tools defining ViewPoint state transitions and context-sensitive guidance.*

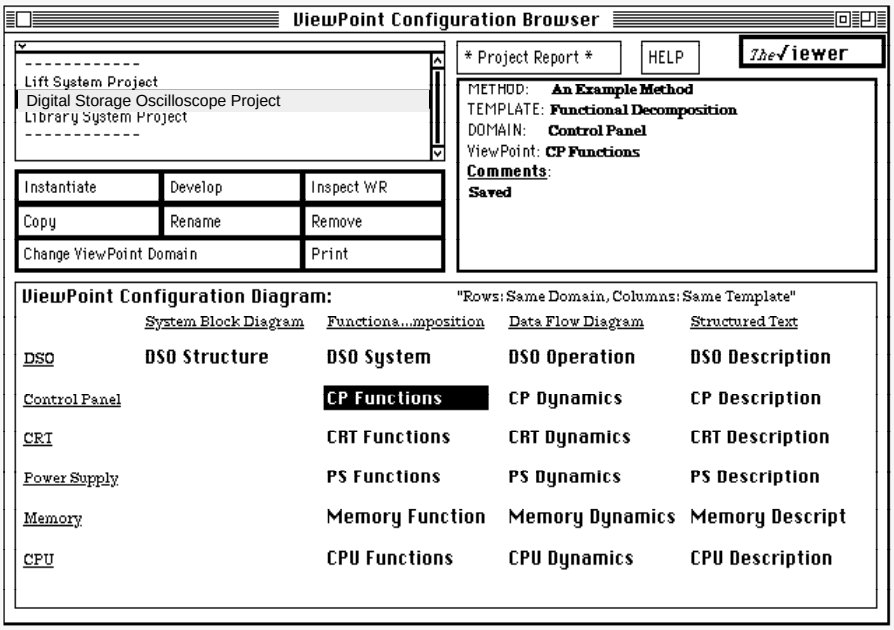


Figure-6: *The√iewer's ViewPoint Configuration Browser. The list of projects created is shown in the top left window pane, and the panel of command buttons below it apply to the ViewPoint selected from the ViewPoint Configuration Diagram in the bottom window pane. The "Project Report" button produces the Project Analyser shown in Figure-7.*

3.3: Method Use

Development (Project) Management: In VOSE, a systems engineering project revolves around instantiating ViewPoint templates, developing ViewPoint specifications, and checking for consistency between them. A project under development therefore consists of a number of ViewPoints, instantiated from the constituent templates of the method on which the project is based. These ViewPoints are displayed in a ViewPoint Configuration Diagram, which tabulates ViewPoints instantiated from the same template in columns, and those relating to the same domain in rows. *The√iewer* employs a ViewPoint Configuration Browser (Figure-6) as the overall project management tool. This browser allows the creation of projects based on pre-defined methods, the instantiation of the methods' constituent templates, and the presentation of selected projects' ViewPoint configuration diagrams. A ViewPoint configuration diagram may be navigated by selecting the required ViewPoint and executing the required command.

The ViewPoint Configuration Browser also provides a number of monitoring tools such as the Project Analyser (Figure-7) and the Work Record Inspector (Figure-8).

The Project Analyser simply lists the selected project's constituent ViewPoints and their current state of development (taken from their respective work records). It provides overall project monitoring, and is a step towards automatic report generation.

The Work Record Inspector for a selected ViewPoint provides a detailed view of that ViewPoint's work record. This allows for the ViewPoint's state, development history, rationale and annotations to be inspected by the development manager. Furthermore, since work record inspection may be performed while the ViewPoint is still under development, the Work Record Inspector provides up-to-date (read-only) monitoring of ViewPoint development progress.

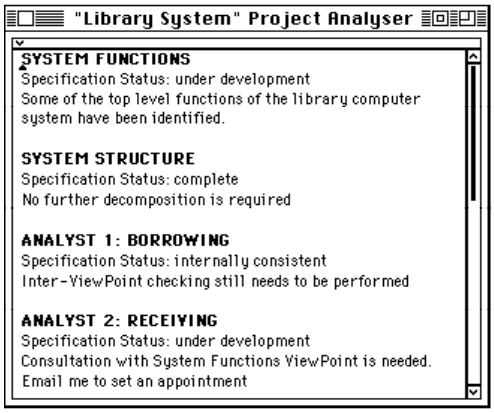


Figure-7: *A Project Analyser. This is a project management tool. It lists all ViewPoints created for a selected development project, and summarises their annotations and development status. It is a step towards automatic report generation.*

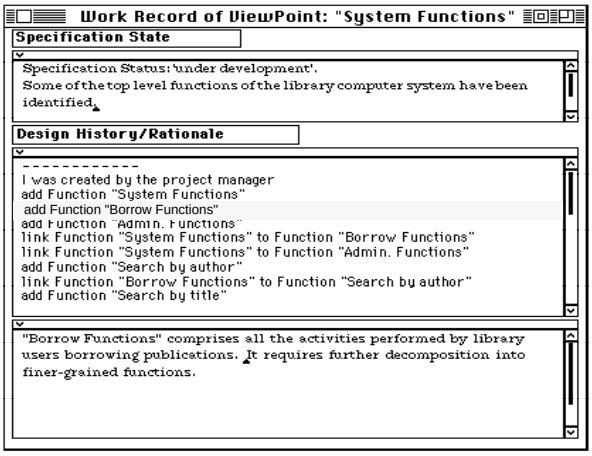


Figure-8: *A Work Record Inspector. It provides the project development manager with a view of the selected ViewPoint's work record. The development history for a selected ViewPoint is displayed, and annotations of individual development steps may be inspected. The Work Record Inspector is a useful development monitoring tool.*

ViewPoint Development: Actual project development occurs in projects' constituent ViewPoints. "Developing" a ViewPoint in the ViewPoint Configuration Diagram provides a ViewPoint Inspector on that ViewPoint (Figure-9). Several ViewPoint Inspectors may be active simultaneously, opening the possibility for distributed ViewPoint development, with each ViewPoint owner developing a different ViewPoint on a different workstation. Clearly however, a mechanism for concurrency control is also needed.

The ViewPoint Inspector provides the mechanisms for editing and checking ViewPoint specifications - both internally and across other ViewPoints. Editing commands appear under the "Assemble" button and are derived from the "Assembly Actions" description in the corresponding ViewPoint template (Figure-5b). Consistency checking may be performed by a Consistency Checker (Figure-10), and coarse-grain, context-sensitive method guidance is also available. The work record automatically keeps track of all actions performed on the specification, and the user may optionally annotate some or all of these actions to explain or provide a rationale for various design decisions.

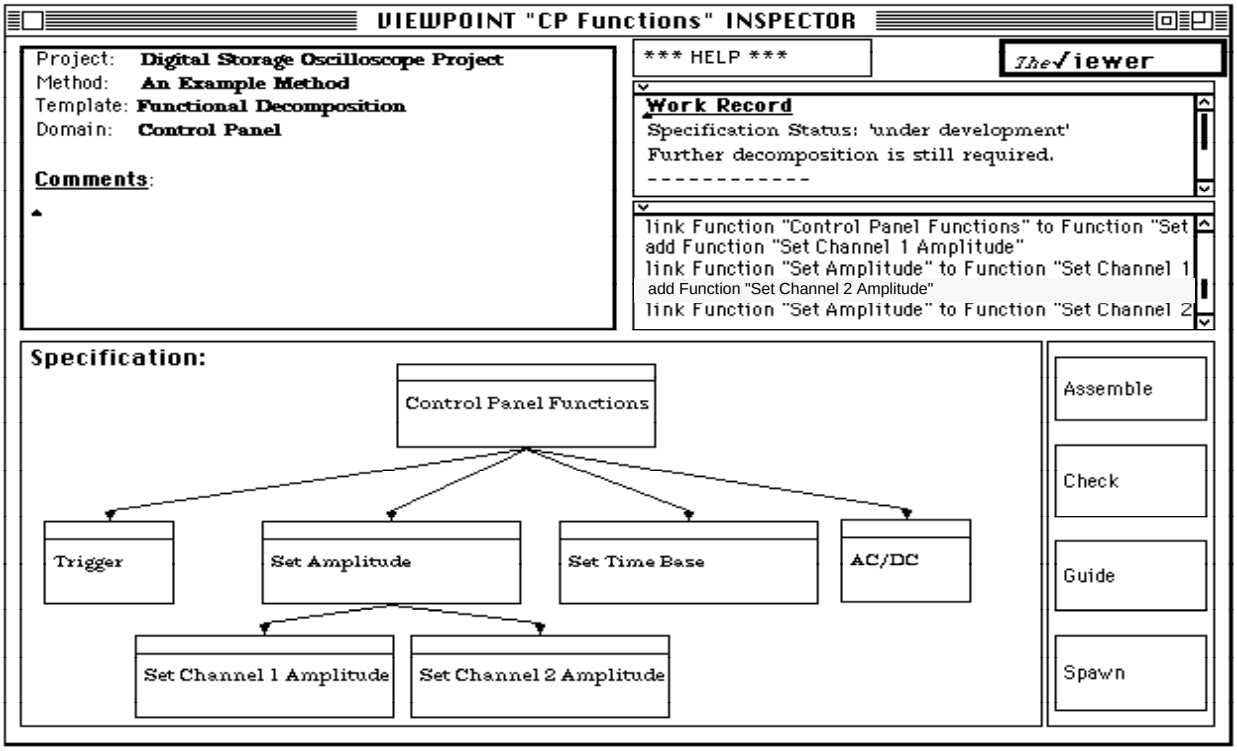


Figure-9: *A ViewPoint Inspector. This window provides a toolkit for the development of ViewPoint specifications. This toolkit includes tools for editing (assembling) specifications and checking their consistency. The figure shows a "typical" functional hierarchy (specification), with the work record shown in the two top right window panes.*

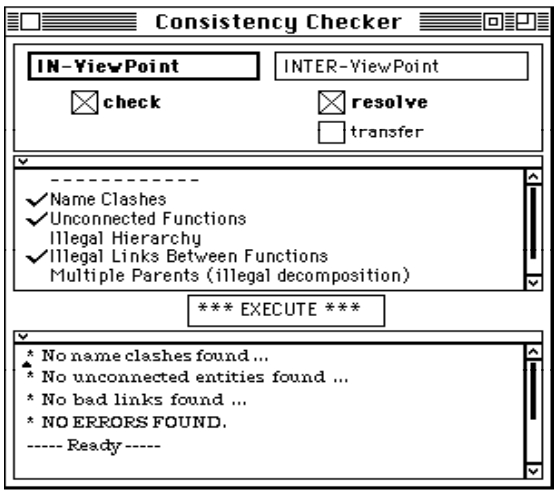


Figure-10: *A ViewPoint Consistency Checker. The scope of the checks is selected first: In- or Inter-ViewPoint. Inter-ViewPoint checks have two modes of application: (1) resolve - answers 'success' or 'fail' when checks are executed, (2) transfer - passes on the necessary information to and/or from other Viewpoints to maintain inter-ViewPoint consistency. The list of appropriate checks is displayed and may be selected and executed individually or in groups.*

4: Integration

Integration is central to the VOSE framework, where its scope extends to three activities:

(1) Method Integration
(2) ViewPoint Integration
(3) Tool Integration

Activities (1) and (2) are directly supported by *The√iewer*, while activity (3) is supported by the underlying smalltalk implementation.

4.1: Method Integration

In VOSE, a method is treated as the union of the techniques that make up that method. In other words, it is composed of the set of ViewPoint templates that describe those techniques (Figure-11). The inter-dependencies between the techniques (which also form part of the method) are specified by the method designer, and are described in the local ViewPoint work plans as "Inter-ViewPoint Checks".

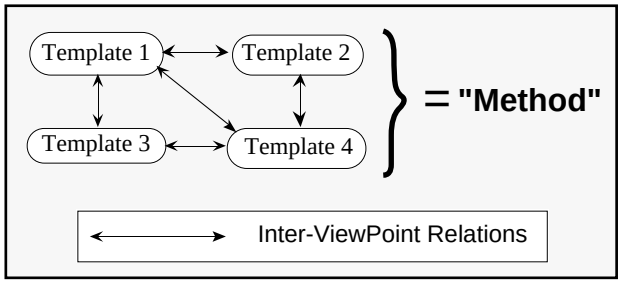


Figure-11: *A software engineering method is a configuration of ViewPoint templates, integrated via a series of inter-ViewPoint relations.*

Therefore, to "package" a method, its constituent templates are described using the Template Browser, and then collectively stored and distributed as "a method".

Alternatively, single templates may be described and stored individually in a Templates Database. These templates may then be used by different method designers, who must elaborate their inter-ViewPoint rules in order to integrate them into their own particular methods. The templates themselves may also be customised to fit designers' own stylistic and strategical requirements.

4.2: ViewPoint Integration

A project in VOSE is a configuration of ViewPoints. These ViewPoints are linked by relationships defined in the respective templates from which they were instantiated. While templates describe general inter-ViewPoint relationships, the actual ViewPoints that constitute projects are related via instantiations of these relationships. For example, a general inter-ViewPoint rule that relates a 'Function' in a Functional hierarchy to an 'Action' in an Action Table, would, on instantiation, relate an actual function in the specification of one ViewPoint to an actual action in the specification of another. Figure-12 shows two ViewPoint specifications produced by *The√iewer*, in which the function 'search by author' and the action 'search by author' are related by an identity relationship.

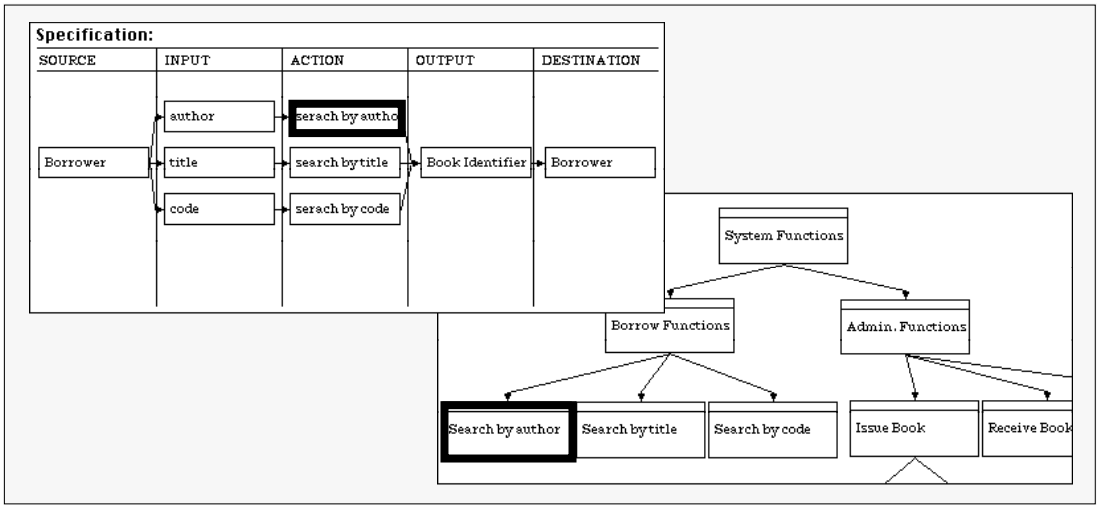


Figure-12: *Two simple ViewPoint specifications produced by The√iewer. A Functional Hierarchy and an Action Table. A general Inter-ViewPoint rule might require a Function in the Functional Hierarchy to have a corresponding Action in an Action Table. An instance of this general rule would (1) select the Function 'search by author' and compare it with all the Actions in the Action Table, (2) detect the Action 'search by author', (3) succeed, then (4) check the next Function in the hierarchy.*

Currently, a project is an integrated collection of ViewPoints arranged in a rectangular lattice. This collection represents the project's system specification, consisting of the individual ViewPoint specifications and their instantiated inter-ViewPoint relations.

A multi-layered approach to ViewPoint organisation and integration is also being investigated [10]. This is based on configuration programming [8], which treats ViewPoints as atomic or as configurations of yet more ViewPoints. *The√iewer* does not currently support this model, but it is clear that some form of ViewPoints' structuring is needed to manage large ViewPoint lattices.

4.3: Tool Integration

The VOSE framework was constructed with tool support and integration in mind. An emphasis was made on providing a framework where effective tools could be constructed to support methods' constituent techniques, and to ensure that these tools were both integrated and reusable. To achieve this, the problem of tool construction and integration was treated as a subset of the wider problem of method development and integration. The framework, and *The√iewer,* were constructed to allow the addition of individual tools to support individual development techniques, which could then be integrated and reused in a manner analogous to the integration and reuse of ViewPoint templates. This was achieved by recognising the need to separate declarative from algorithmic information [3].

A ViewPoint template encapsulates within its two slots (style and work plan) declarative information about the development technique which it describes. How this is supported by a CASE tool is left to the tool implementor. Different tool implementors may choose different algorithms, indeed different implementation languages, to support the same template. What these tools must conform to however, is the declarative information supplied to them by the templates they support. In particular, the inter-ViewPoint, and hence inter-Tool, rules must be picked up from the appropriate templates. These rules, which provided the mechanism for method integration, now also act as the tool integration mechanism. Method tool support is then treated as the union of the tools supporting the method's constituent templates (Figure-13).

Note: Currently, the mechanisms for tool integration and reuse depend on the smalltalk architecture and implementation of The√iewer and its tools.

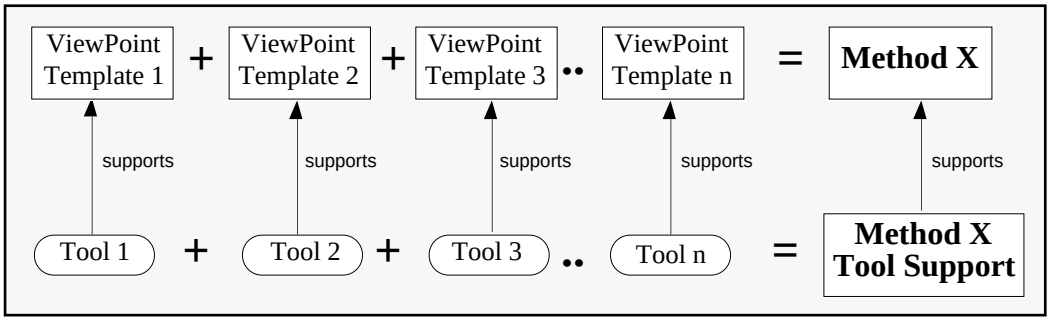


Figure-13: *A "method" is the union of its constituent development techniques (ViewPoint Templates). A "tool" supports an individual ViewPoint Template. "Method Tool Support" is the union of all the tools supporting the method's constituent ViewPoint Templates.*

5: Implementation Architecture

The√iewer uses a very basic approach to tool implementation and integration. Currently all the ViewPoint template support tools are implemented in, and depend on, smalltalk - with the architecture allowing for their "easy" addition, modification and extension. Standard, abstract classes are available to get skeleton tools operational quickly, with mechanisms implemented for picking up textual and graphical information from template descriptions provided by the method designer. For example, the graphical descriptions created in the template style slots using the Graphics Editor may be picked up as icons by the tools, while the assembly actions are treated as items of pop-up menus by ViewPoint Inspectors. We are currently examining ways of improving tool construction using *The√iewer* by providing better meta-CASE utilities, in order to make tool-building more "implementor-friendly" [1].

In many ways, *The√iewer* is based on a "traditional" smalltalk architecture. Most of the Browsers, Inspectors and other tools employ Model-View-Controller (MVC) triads [11] to model their behaviour, presentation and user-interfaces. Of greater interest are the objects which the various tools manipulate. A class Template was implemented as a subclass of Object, with two principle instance variables 'style' and 'workPlan'. Class ViewPoint was implemented as a subclass of Template with the addition of three more instance variables 'domain', 'specification' and 'workRecord' (Figure-14). Clearly, these classes implemented in smalltalk directly model the VOSE framework - an acknowledged benefit of object-oriented programming. Moreover, the direct mapping of ViewPoints onto smalltalk objects facilitates the incorporation of exploratory modifications of the VOSE framework into *The√iewer* environment.

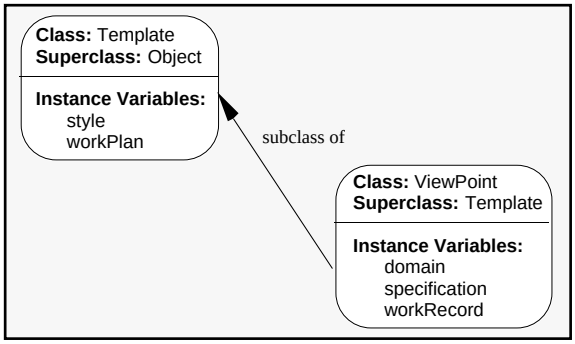


Figure-14: *The√iewer implements the class "Template" and its subclass "ViewPoint" which directly reflect the objects manipulated in the model of the VOSE framework.*

Templates and ViewPoints are stored in separate smalltalk databases which may be selectively accessed by the different tools of *The√iewer.* These databases may also be saved to files and exported across smalltalk images.

The√iewer is a large prototype smalltalk application implementing over 45 classes. It employs a number of different clusters of classes and MVC triads - the most common being a single model and controller with multiple views. Frequently however, views also carry a substantial part of the model functionality, to allow many views to share a single model. The controller is sometimes also combined with its view, particularly when direct interaction with the view is required; eg, in the case of ViewPoint Inspectors which allow users to manipulate specifications directly using the mouse. Figure-15 shows the top level architecture of *The√iewer.*

The√iewer is a prototype VOSE support environment under development. Work is underway to improve consistency checking and method guidance, and to provide "hooks" to ViewPoint support tools written in languages other than smalltalk.

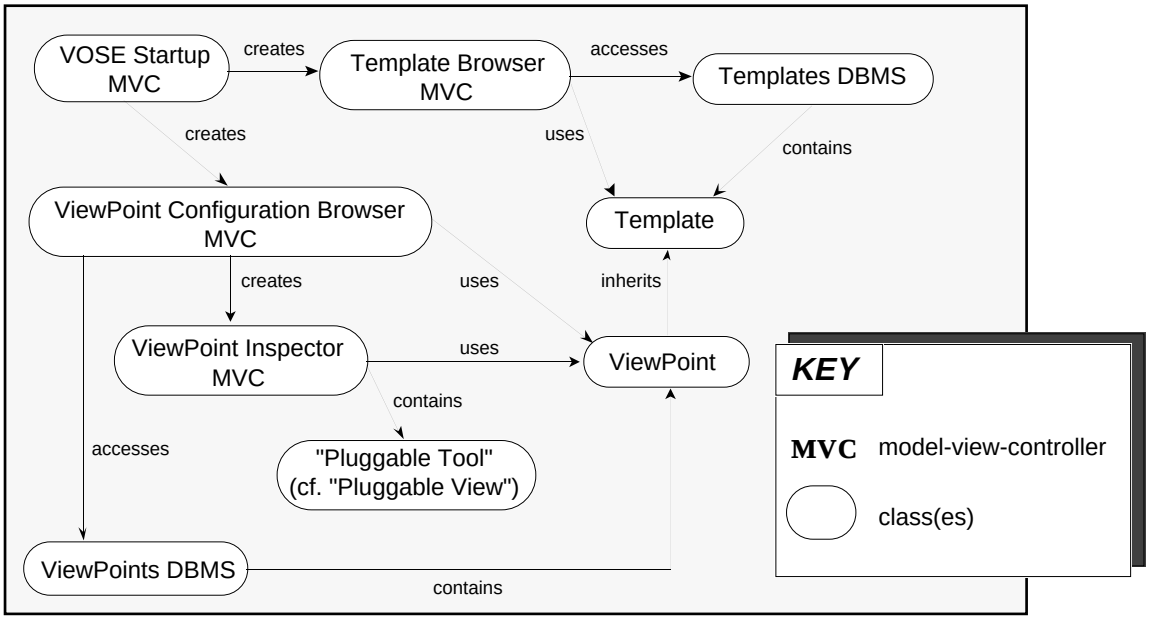


Figure-15: *The top level architecture of The√iewer.*

6: Conclusions and Future Work

The ViewPoints approach to software engineering acknowledges the inevitable role played by a multitude of development techniques in a single development project. ViewPoints encapsulate development and specification knowledge within a single object. Further modularity is achieved by selecting ViewPoints that cover different problem domains.

While this “separation of concerns” is clearly intuitive and logical, the additional attraction of VOSE is its support for integration. Integration is supported across the framework, from method and tool design to method use.

The VOSE object-based architecture offers considerable scope for reuse. Again, reuse is supported at the levels of both method design and method use. ViewPoint templates (classes) are the reusable components of representation and development knowledge during method design and construction. ViewPoints (instances) are the reusable components of specification knowledge during method use.

Methods are treated as a union of ViewPoint templates, and method tool support as the union of the individual tools supporting the methods’ constituent templates. Tool integration is a natural consequence of the method integration provided by the VOSE framework via inter-ViewPoint consistency checks and transformations.

VOSE is a proposed integration mechanism for the ESPRIT II project REX [15], and is under development at a number of European industrial and academic sites. Several case studies [14] have been performed using methods such as CORE [12] and the Constructive Design Approach [9] from which valuable feedback has been gained. Individual development techniques, including an extended form of Petri Nets [5], have also been described using ViewPoint templates, and special-purpose tools constructed to support them [6, 7].

Our current research is focused on two areas: notation and process modelling. We are particularly interested in exploring suitable notations for expressing in- and inter-ViewPoint consistency checking rules. One possibility is to use an extended form of Prolog - extended because of the need to handle distributed knowledge sources. Prolog rules lend themselves to two “modes of application”: a success/fail mode or solution generation (information transfer) mode. These two modes are particularly appropriate for inter-ViewPoint checking.

Our process modelling research is investigating a number of issues, one of which is also notation. At one level, we are interested in modelling the overall, method-driven, systems development process. At another level - that of individual ViewPoints - we are using a “precondition-action-postcondition” notation to describe our individual process models; however, a richer language may be required for the finer-grained modelling of the ViewPoint development process. In particular, we need to provide support for method guidance at the level where process meets representation.

ViewPoint Oriented Systems Engineering has taken a multiple perspectives approach to the development of heterogeneous, composite systems. Problems and

processes are divided into simpler, communicating units of knowledge. Although the concepts of modularity, encapsulation and cooperative work are not new, the novelty lies in their realisation through ViewPoints which provide a means for both heterogeneity and integration.

Acknowledgements

The authors would like to gratefully acknowledge the contributions of their colleagues at the Distributed Software Engineering Group at Imperial College. In particular, Jonathan Moffett and Diomidis Spinellis provided constructive feedback on earlier drafts of the paper. Special thanks to Jeff Kramer, Michael Goedicke and Peter Graubmann for their work on ViewPoints. This work was funded by the UK Science and Engineering Research Council (SERC) and the Commission of European Communities (CEC), as part of the SEED (Software Engineering and Engineering Design) and REX (Reconfigurable and Extensible Parallel and Distributed Systems) projects, respectively.

References

[1] A. Alderson, “Meta-CASE Technology”, Proceedings of European Symposium on Software Development Environments and CASE Technology, Konigswinter, Germany, June 1991, Published by Springer-Verlag.

[2] G. Booch, “Object-Oriented Design with Applications”, Benjamin Cummings, Redwood City, CA, USA, 1991.

[3] G. Clemm and L. Osterweil, “A Mechanism for Environment Integration”, ACM Transactions on Programming Languages and Systems, Volume 12, Number. 1, January 1990, pp. 1-25.

[4] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein and M. Goedicke, “Viewpoints: A Framework for Integrating Multiple Perspectives in System Development”, To appear in the International Journal of Software Engineering and Knowledge Engineering, Special issue on “Trends and Future Research Directions in SEE”, World Scientific Publishing Company, 1992.

[5] P. Graubmann, “Definition of SPEC Nets”, REX technical report REX-WP3-SIE-008-V1.0, Siemens, Munich, Germany, July '90.

[6] P. Graubmann, “The HyperView Tool Standard Methods”, REX technical report REX-WP3-SIE-021-V1.0, Siemens, Munich, Germany, January '92.

[7] P. Graubmann, “The Petri Net Method ViewPoints in the HyperView Tool”, REX technical report REX-WP3-SIE-023-V1.0, Siemens, Munich, Germany, January '92.

[8] J. Kramer, “Configuration Programming - A Framework for the Development of Distributable Systems”, Proceedings of IEEE International Conference on Computer Systems and Software Engineering (CompEuro 90), May 1990.

[9] J. Kramer, J. Magee and A. Finkelstein, “A Constructive Approach to the Design of Distributed Systems”, Proceedings of the 10th International Conference on Distributed Computing Systems, Paris, France, 28th May-1st June 1990.

[10] J. Kramer and A. Finkelstein, “A Configurable Framework for Method and Tool Integration”, Proceedings of European Symposium on Software Development Environments and CASE Technology, Konigswinter, Germany, June 1991, Springer-Verlag.

[11] G.E. Krasner and S.T. Pope, “A Cookbook for using the Model-View-Controller User Interface Paradigm in Smalltalk-80”, ParcPlace Systems, CA, USA, 1988.

[12] G. Mullery, “Acquisition - Environment”, (In) M. Paul & H. Siegert (editors), Distributed Systems: Methods and Tools for Specification, LNCS 190, Springer-Verlag, 1985.

[13] B.A. Nuseibeh, “CoreDemo: An Investigation into the use of Object-Oriented Techniques for the Construction of CASE Tools”, M.Sc. Thesis, Department of Computing, Imperial College, London, September 1989.

[14] B.A. Nuseibeh, “VOSE: An Interim Report and Case Study”, Internal Report, Department of Computing, Imperial College, London, March 1991.

[15] REX Technical Annex, “ESPRIT Project 2080”, European Economic Commission, March 1989.

[16] F. Tontsch, “Methods and Tools”, pp. 181-199, Chapter 10, Managing Complexity in Software Engineering, R.J.Mitchell (editor), Peter Peregrinus Ltd. on behalf of the IEE, London, 1990.

[17] P. Wegner, “Dimensions of Object-Based Language Design”, Proceedings of OOPSLA ‘87, ACM, New York, pp. 168-182.