Proceedings of 15th International Conference on Software Engineering, Baltimore, USA, May 1993, IEE CS Press (to appear)

Expressing the Relationships Between Multiple Views in Requirements Specification

Bashar Nuseibeh Jeff Kramer Anthony Finkelstein

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

Abstract

Composite systems generally comprise heterogeneous components whose specifications are developed by many development participants. The requirements of such systems are invariably elicited from multiple perspectives which will overlap, complement and contradict each other. If these requirements are developed and specified using multiple methods and notations respectively, then it is necessary to express and check the relationships between the resultant specification fragments.

In this paper we deploy multiple "ViewPoints" that hold partial requirements specifications, described and developed using different representation schemes and development strategies. We discuss the notion of inter-ViewPoint communication in the context of this ViewPoints framework, and propose a general model for such communication. We elaborate on some of the requirements for expressing and enacting inter-ViewPoint relationships, and use fragments of the requirements specification method CORE to illustrate our model.

1. Introduction

1.1. Motivation

Heterogeneity is inevitable in most composite systems of significant size, and no single development process and representation will be sufficient for their development. This is particularly true of the requirements engineering phase of the software development life-cycle. Requirements engineering encompasses activities ranging from requirements analysis and elicitation to specification, conflict resolution and validation. Even a single activity such as requirements elicitation, is likely to deploy multiple development participants who will hold multiple perspectives of a single domain.

This heterogeneity of representations and processes poses challenging research problems of *integration:* (1) the integration of the methods used to specify system requirements, (2) the integration of the tools that support these methods, and (3) the integration of the multiple specification fragments produced by applying these methods and tools. By deploying "ViewPoints" that encapsulate partial specifications with the development techniques by which they are produced, a framework is in place within which the problems of integration outlined above may be addressed. Moreover, experience has shown that the difficulties of expressing and enacting the relationships between multiple ViewPoints need to be resolved, before integration in this setting may be achieved.

1.2. ViewPoints

We have used the term "multiple perspectives problem" [15] to describe the class of problems surrounding the development of composite systems [12] by many development participants who deploy sundry representation schemes, use a variety of development strategies and hold diverse domain knowledge. We have also proposed an object-based framework deploying ViewPoints within which the above problems may be tackled [15]. Issues relating to conflict resolution [10] negotiation and dialogue [14], configuration programming [24] and tool support [20, 21, 34] have already been examined within this framework.

The term "viewpoint" has been defined and deployed in a variety of settings in software engineering, particularly in the domain of requirements engineering; e.g., [23], [25], [31] and [38]. In our framework, we have attempted to generalise and formalise the definition of a viewpoint to facilitate its manipulation for composite system development. The definition draws together the notion of 'actor', 'knowledge source', 'role' or 'agent' with the notion of a 'view' or 'perspective' held by the former. Thus, we define a ViewPoint to be a loosely-coupled, locally managed object encapsulating representation knowledge, development process knowledge and partial specification knowledge about a system and its domain. This knowledge is assigned to five ViewPoint *slots* (figure 1):

- the ***style*** slot, in which the representation scheme used by the ViewPoint is described,
- the ***work plan*** slot, in which the development actions, process and strategy of the ViewPoint are described,
- the ***domain*** slot, which describes the area of concern of the ViewPoint with respect to the overall system under development,
- the ***specification*** slot, which describes the ViewPoint domain in the notation described in the style slot - and

developed using the strategy described in the work plan slot, and

- the ***work record*** slot, in which the development state and history of the ViewPoint specification is maintained (in terms of the work plan actions performed). It is the vehicle by which traceability (to and from requirements) may be achieved.

A *ViewPoint Template* is a ViewPoint 'type' in which only the style and work plan slots have been elaborated. A ViewPoint template, when *instantiated,* yields a ViewPoint - which can then be elaborated to produce a specification for a particular domain. A ViewPoint template is therefore a reusable description of a development technique (notation and process) which may be instantiated many times to produce many ViewPoints. A software engineering *method* in this context is then a collection or configuration of ViewPoint templates and their relationships, that together constitute the development techniques deployed by the method.

A ViewPoint *owner* is responsible for enacting the process model of a ViewPoint which appears in its work plan. ViewPoint owners are normally, but not always, human development participants. A non-human ViewPoint owner may be some form of 'intelligent' tool or expert system for example.

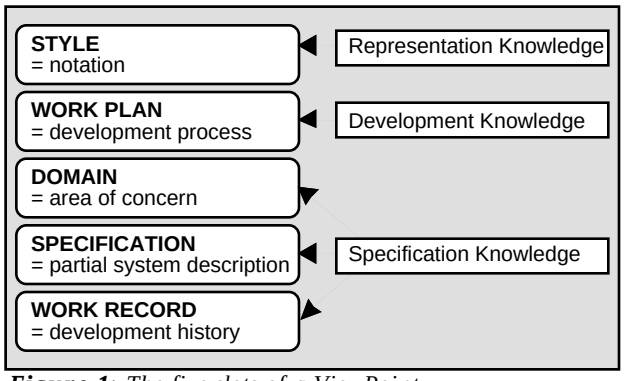

Figure-1: *The five slots of a ViewPoint.*

1.3. Scope of Paper

In attempting to integrate multiple requirements specification ViewPoints, overlaps must be identified and expressed, complementary participants made to interact and cooperate, and contradictions resolved. In this paper, we address the notion of inter-ViewPoint communication as a vehicle for ViewPoint integration. The communication model we present straddles both the method construction stage during which inter-ViewPoint relationships are expressed, and the method application stage during which these relationships are enacted (checked). We illustrate the model by constructing part of the requirements specification method CORE [30, 31, 42], and applying it to specify a simple problem.*

We argue that successful inter-ViewPoint communication - guided by a model of the development process - holds the key to achieving integration in a heterogeneous, possibly distributed, setting. Thus, there is a need to express relationships between ViewPoints, enact these relationships (e.g., check consistency and transfer information), and resolve conflicts (if and when it is necessary to do so).

Although we examine the application of ViewPoints for requirements specification, we further argue that requirements engineering from multiple perspectives, multiparadigm specification [46] and multiparadigm programming [29], are all facets of the same generic (multiple perspectives) problem.

The next section of the paper describes the method construction process within the ViewPoint framework, which is followed by a description of how requirements methods are used to develop requirements specifications in this context. A model of inter-ViewPoint communication is then presented and illustrated using the examples introduced in the preceding two sections. Finally, overlapping and related research work is presented, some conclusions are drawn, and an agenda for further research is outlined.

2. ViewPoint-Oriented Method Construction

Like most methods, the requirements specification method CORE, comprises a number of development stages which deploy a number of different representation schemes. These stages are used to incrementally and iteratively produce a system requirements specification. In ViewPoints terminology, *CORE: the method* may be described using a number of ViewPoint templates. Since each stage in CORE deploys a single, simple representation scheme, one way to describe CORE would be to describe each stage as a single ViewPoint template. Figures 2 and 3 are sample, informal ViewPoint template descriptions of the agent structuring (AS) and the tabular collection (TC) stages of CORE, respectively. These stages support problem decomposition into an agent hierarchy and agent elaboration in a tabular form respectively (see §3 for a domain-specific example of each). For simplicity, the style slot of each template is described in terms of *objects* and *relations*, each having *attributes* with *types* and *values*. A BNF description may be more appropriate for text-based notations.

In describing the work plan slot, four generic categories of development actions are identified. The *assembly actions* are those basic actions required to build (assemble) a specification in the current representation style. *In-ViewPoint check actions* are those actions that can be performed to check that a specification is syntactically consistent. *Inter-ViewPoint check actions*, are those actions that can be performed to check the consistency between (overlapping or interacting) specifications residing in different ViewPoints. These actions may also be used to transfer information between ViewPoint

* Since CORE uses the term "viewpoint" as part of its terminology, we substitute the term "agent" in its place to avoid the clash in nomenclature.

STYLE

Object: Agent

Attributes	Types	Values
Name Identifier Icon	String Integer Bitmap	

Relation Part-of(Agent, Agent)

Attributes	Types	Values
Icon	Bitmap	

WORK PLAN

Assembly Actions: add(Agent), remove(Agent), connect(Agent, Agent, Part-of), disconnect(Agent, Agent, Part-of)

In-ViewPoint Check Actions: all-agents-connected, each-child-has-parent, no-name-clashes, ...

Inter-ViewPoint Check Actions: For each agent in the agent hierarchy there must be one ViewPoint instantiated from a TC template, ...

ViewPoint Trigger Actions: For each leaf agent create a new ViewPoint instantiated from a TC template, ...

Figure-2: *An informal description of CORE's agent structuring (AS) ViewPoint template.*

specifications. It is particularly challenging to describe inter-ViewPoint relationships in a generic manner, more so if the two ViewPoint specifications being related use representation styles with different underlying data models or schemas. *ViewPoint trigger actions* must be performed in order to create new ViewPoints (i.e., instantiate ViewPoint templates) 'on-the-fly'. These actions are normally performed as a consequence of one of the other development actions; e.g., adding an agent in an agent hierarchy should trigger the creation of a new ViewPoint for that agent, instantiated from the tabular collection template.

What the work plans in figures 2 and 3 do *not* show are the process models or process descriptions that may be used to guide a ViewPoint owner in building a ViewPoint specification. A "precondition $\rightarrow$ [Action] postcondition" notation [6] may be used to describe such process models; e.g.,

empty-spec $\rightarrow$ [Assembly Actions] spec.
spec $\rightarrow$ [Assembly Actions] spec.
spec $\rightarrow$ [In-ViewPoint Check Actions] (consistent-in-VP-spec $\vee$ spec).
consistent-in-VP-spec $\rightarrow$ [Inter-ViewPoint Check Actions] (consistent-inter-VP-spec $\vee$ spec).
consistent-inter-VP-spec $\rightarrow$ [ ] end.

Furthermore, each ViewPoint work plan may deploy its own particular process modelling or process programming [35] language to elaborate its individual specification development process. Other process modelling languages may also be used such as the visual software process language proposed in [40] or the RADs used by [36].

STYLE

Object: Source

Attributes	Types	Values
Name Icon	String Bitmap	

Object: Input

Attributes	Types	Values
Name Icon	String Bitmap	

Object: Action

Attributes	Types	Values
Name Icon	String Bitmap	

Object: Output

Attributes	Types	Values
Name Icon	String Bitmap	

Object: Destination

Attributes	Types	Values
Name Icon	String Bitmap	

Relation Connected-to(Object, Object)

Attributes	Types	Values
Icon	Bitmap	

WORK PLAN

Assembly Actions: add(Source), remove(Source), add(Input), remove(Input), ..., connect(Source, Input, Connected-to), disconnect(Source, Input, Connected-to), ...

In-ViewPoint Check Actions: all-sources-connected-to-inputs, all-inputs-connected-to-actions, no-actions-name-clashes, ...

Inter-ViewPoint Check Actions: every Source or Destination in a tabular collection diagram has a corresponding agent with the same name in the agent hierarchy, the Output produced by one agent in a tabular collection diagram must be consumed as an Input in the tabular collection diagram another agent, ...

ViewPoint Trigger Actions: Create a ViewPoint instantiated from an AS template if one does not already exist, ...

Figure-3: *An informal description of CORE's tabular collection (TC) ViewPoint template.*

The use of multiple ViewPoints also allows individual ViewPoint development processes to be modelled at different levels of *granularity*, to provide the appropriate level of *guidance* for different developers [16]. *Process integration* [27] however, which in our setting means the integration of multiple process models to produce an overall, coherent development process, remains a problematic research area. In [2], one technique for such integration is proposed, based on a concurrency control mechanism developed for a co-operative software development environment.

Finally, ViewPoint development process models may be partly described by precise inconsistency handling rules, that specify how to act in the presence of inconsistency [17]. These rules may therefore be used to *drive* the development process both within and between individual ViewPoints.

3. ViewPoint-Oriented Requirements Specification

Once a requirements method has been constructed, it may then be deployed to specify system requirements. Problem-specific ViewPoints may be created by instantiating the appropriate ViewPoint templates, and their ViewPoint specifications developed by following the individual ViewPoint work plans. The result of this development process is a collection or configuration of ViewPoints which together form the total system requirements specification. At any point during development the different ViewPoint specifications may be overlapping and/or inconsistent with each other. This inconsistency is tolerated by the ViewPoints approach and not checked or corrected as a matter of course, but invoked on a "check-when-needed" basis. Integrity checking may only be appropriate at specific stages of the development life-cycle and detection of inconsistency may not require immediate resolution, but left for later action, or even not resolved at a particular stage at all. This approach to consistency checking is developed further in [18] who argue for "making inconsistency respectable", and develop a logic-based framework in which "INCONSISTENCY implies ACTION". In [17] we explore the applicability of this inconsistency handling technique in the context of the ViewPoint framework.

Example

The first graphical stage in CORE, agent structuring (AS), identifies the information processing entities (agents) in the problem domain, and arranges them in a hierarchy. The relation between child and parent in the hierarchy is that child is a "part-of" parent. In specifying a library system for example, the root of an agent hierarchy might be "Library World". This is then decomposed into its constituent agents, which may then be decomposed further and so on. Thus we may build a ViewPoint instantiated from the AS template for the domain "Library World" and with the specification shown in figure 4. The work record lists the work plan actions that were performed to produce the specification in its current state. The work record actions may be meaningfully annotated to provide a development rationale for the specification.

At this or any point in-ViewPoint actions may be performed to check that the specification of the ViewPoint in figure 4 follows the syntactic rules imposed on its representation style. Inter-ViewPoint actions may also be performed, but no other ViewPoints have been created in this example as yet. Performing ViewPoint trigger actions on the other hand causes the instantiation of the tabular collection template, once for each of the leaf agents in the agent hierarchy (as specified in the ViewPoint trigger

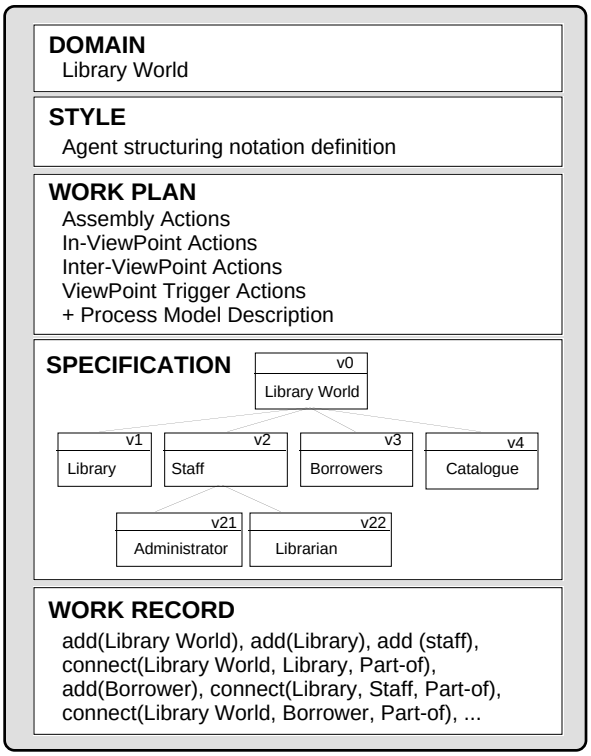

Figure-4: *A sample ViewPoint instantiated from an agent structuring ViewPoint template.*

actions section of the work plan of the AS template in figure 2). Thus from the agent hierarchy in figure 4, five further ViewPoints (one for each leaf agent in the hierarchy) containing blank specifications (tables) are created. Each may then be developed separately by their ViewPoint owner who enacts their individual work plans. One such tabular collection ViewPoint (for the 'Borrower' agent) in which some assembly actions have been performed is shown in figure 5.

It is again possible at this point to perform any of the ViewPoint's work plan actions. One of the inter-ViewPoint actions for example, checks that every source and destination in the tabular collection specification is a named agent shown in the agent hierarchy in the AS ViewPoint. This check was specified textually in the inter-ViewPoint check actions section of the work plan of the TC template in figure 5. If such a check fails, then some form of conflict resolution strategy must be employed in order for the check to succeed. Conflict resolution for this check in particular, implies that either a new agent must be added to the agent hierarchy specification in the AS ViewPoint, or the inconsistent source or destination must be renamed or removed from the specification of the TC ViewPoint. Approaches to conflict resolution in the ViewPoints context have been examined in [10], and a model of conflict resolution proposed.

Although it is possible, in principle, to perform any of the generic work plan actions at anytime during specification development, each ViewPoint process

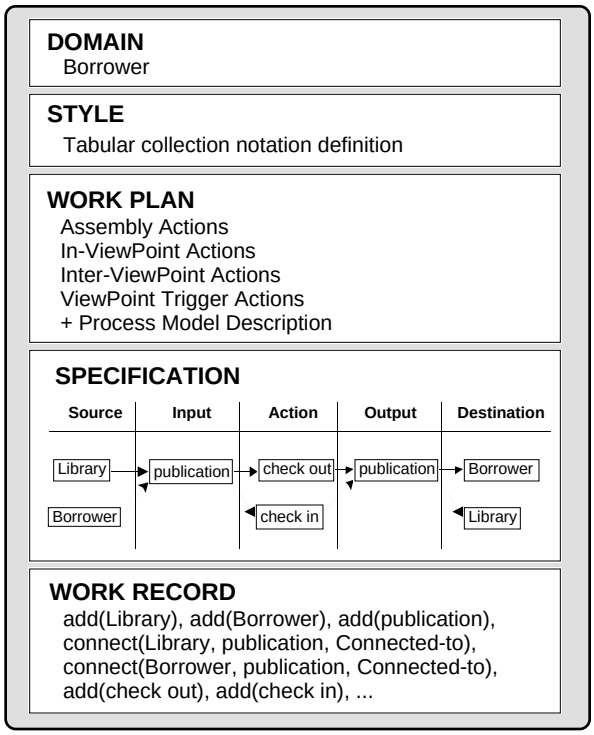

Figure-5: *A sample ViewPoint instantiated from a tabular collection ViewPoint template.*

model should prescribe when and under what circumstances it is appropriate to do so. For example, it would be unreasonable in most cases to perform inter-ViewPoint checks between two ViewPoint before the in-ViewPoint consistency of at least one of the two ViewPoints has been checked and established.

4: Inter-ViewPoint Communication

Heterogeneity of notations, processes and specifications inevitably poses problems of integration. Within the ViewPoints framework, the relationships between ViewPoints need to be expressed, so that they may then be used to check consistency, transfer and transform information between ViewPoint specifications. Thus, there is a need to define *inter-ViewPoint rules* that describe these relationships, specify when they may be invoked and how they should be applied. These activities straddle the processes of ViewPoint-oriented method construction and ViewPoint-oriented requirements specification. They are generic in the sense that they make no comment about how inter-ViewPoint rules are represented or about the mechanisms for invoking and applying them. They are shown schematically in figure 6.

4.1. Step 1: Inter-ViewPoint Rule Definition

Inter-ViewPoint rules are defined in ViewPoint *template* work plans and thus describe relationships between ViewPoints that have not yet been created. They are of the general form:

$$\forall\ VP_S\ ,\ \exists\ VP_D \text{ such that } VP_S\ \Re\ VP_D$$

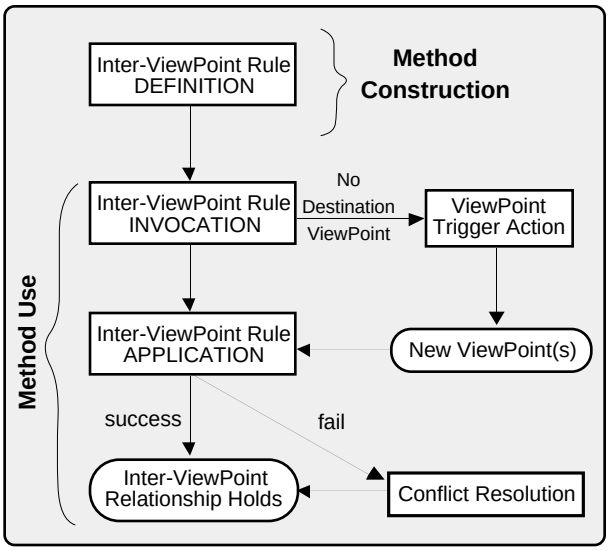

Figure-6: *A model of inter-ViewPoint communication activities. A labelled arrow indicates a precondition for the next step to be performed.*

where VP_S is the *source* ViewPoint in which the rule resides, and VP_D is the *destination* ViewPoint with which the relation $\Re$ holds. The broken lines in figure 7a illustrate the status of inter-ViewPoint rules at the definition stage of the model. They relate hypothetical ViewPoints, VP_S and VP_D, with a hypothetical relationship, $\Re$ - that is, they express what the method designer decides are the relationships between ViewPoints instantiated from particular ViewPoint templates. Their inclusion in the individual ViewPoint templates enhances the loose coupling and local management of each ViewPoint, which in turn facilitates the deployment of ViewPoints in a distributed environment.

Say for example we wish to write an inter-ViewPoint rule for the tabular collection stage of CORE which asserts that *every source in a tabular collection diagram must be a named agent in the agent hierarchy*. This rule makes a statement about *every* source in a tabular collection diagram, and can therefore be defined in the ViewPoint template describing tabular collection (TC). Furthermore, it requires information defined in the agent structuring (AS) ViewPoint template, and therefore will require information outside the boundaries of the ViewPoint in which it is defined in order to get this information. What is thus required is a means of identifying the ViewPoint from which this information will be obtained. Since there is no prior knowledge of what ViewPoints will be created during specification, the only way to identify a ViewPoint is by specifying the template from which it will be instantiated, and perhaps the domain with which it will be concerned. Thus, a ViewPoint can be identified at rule definition time by a tuple:

VP(t, d)

where t specifies the template from which the ViewPoint

will be instantiated,

and $d \in \{ D_p , D_a , D_s , D_d \}$

where,

D_p refers to a particular (named) domain,

D_a refers to any domain not known at template construction time,

D_s refers to the same domain as the current ViewPoint (if applicable),

D_d refers to a different domain from the current ViewPoint (if applicable).

Therefore, the general form of an inter-ViewPoint rule may be rewritten as:

partial_specification_1 ℜ VP(t, d): partial_specification_2

where the partial_specification_1 describes a partial specification in the ViewPoint, VP_S, created from the template in which the rule is defined, and which therefore does not require a ViewPoint identifier. The partial_specification_2 describes a partial specification in the ViewPoint with domain d and instantiated from template t. A rule of the above form asserts that for every partial_specification_1 there should exist at least one partial_specification_2 with which the relationship ℜ holds.

Returning to the CORE rule we wish to define, it may be written in the TC ViewPoint template work plan as follows:

Source.Name = VP(AS, D_d): Agent.Name

This rule states that the Name attribute of the Source object in the source ViewPoint, VP_S (instantiated from the TC template in which the rule resides), has the same value (=) as the Name attribute of the Agent object in the destination ViewPoint, VP_D (instantiated from the AS template and relating to a domain different from the source ViewPoint domain).

A similar rule may be written to assert that *every destination in a tabular collection diagram must be a named agent in the agent hierarchy:*

Destination.Name = VP(AS, D_d): Agent.Name

Rules expressing the relationships between ViewPoints instantiated from the same template may also be written in the same way. Take the rule in CORE which asserts that *every output from a tabular collection diagram must be an input in another tabular collection diagram for another agent (the destination agent for the original input).* This rule may be written as:

Connected-to(Output, Destination).Output.Name =
VP(TC, Destination.Name):
Connected-to(D_S, Input).Input.Name

In many cases, a converse of each rule must also be written in the destination ViewPoint template, so that the rule may be invoked and applied by either ViewPoint. The converse of the above rule in this case also applies. That is, *every input from a source in a tabular collection diagram must have been produced as an output by the tabular collection diagram of that source agent:*

Connected-to(Source, Input).Input.Name =
VP(TC, Source.Name):
Connected-to(Output, D_S).Output.Name

> ***Note on notation:*** *A dot (.) in the rules above separates (from left to right) relations, objects, attributes and values. For example, the term*
>
> Object1.Attribute1
>
> *reads "the value of Attribute1 of Object1". While:*
>
> Relation(Object1, Object2).Object1.Attribute1
>
> *reads "the value of Attribute1 of Object1 in the Relation(Object1, Object2)". Other terms include:*
>
> Relation1(Object1, Object2).Attribute1
> *(the value of Attribute1 of Relation1)*
>
> *and:*
>
> Object1.Attribute1.25
> *(the value '25' for the Attribute1 of Object1)*

Not every rule in CORE however has a valid converse; e.g., *every agent in an agent hierarchy does NOT necessarily have to be a named source or destination in a tabular collection diagram*. CORE *does* require however, that the AS ViewPoint template contain a rule which asserts that *every agent in an agent hierarchy must have a tabular collection diagram associated with it.* This may be written as:

Agent → VP(TC, Agent.Name)

The above rule simply states that every Agent object triggers (→) the creation of a new ViewPoint instantiated from a tabular collection template, and concerned with the domain D_p = Agent.Name.

4.2. Step 2: Inter-ViewPoint Rule Invocation

Inter-ViewPoint rules are invoked by the owner of the ViewPoint in which they reside. At invocation time (figure 7b), an inter-ViewPoint rule asserts that for the ViewPoint VP_S (which now exists because the rule was invoked from it), there should be at least one ViewPoint VP_D, such that VP_S ℜ VP_D. If VP_D does not exist, then a ViewPoint trigger action to create it must be performed before rule application (step 3) may be performed. The inter-ViewPoint rule invocation step is required for establishing that the two ViewPoints, between which consistency needs to be checked or information transferred, are identified. The ViewPoint process model may define when inter-ViewPoint rules should be invoked; e.g., "if condition X holds in VP_S, then check that VP_Sℜ VP_D". In

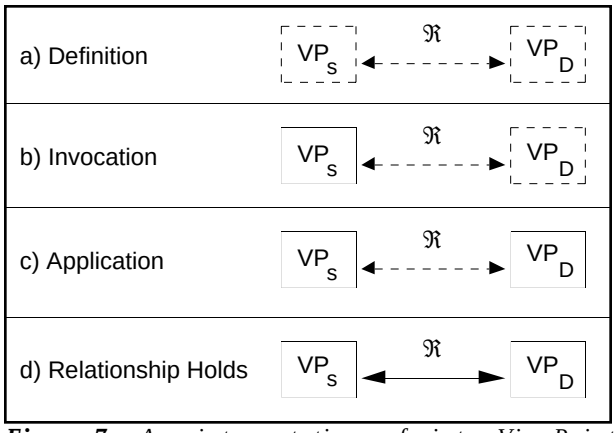

Figure-7: *An interpretation of inter-ViewPoint communication at various stages of the model. A broken line indicates that a ViewPoint or relation can exist or hold, but has not necessarily been established yet.*

[16] we discuss three approaches to rules invocation: the "stupid", in which rules are constantly invoked; the "pragmatic", in which rule invocation may be turned on and off by the user; and the "problematic", in which the process model guides rules invocation.

4.3. Step 3: Inter-ViewPoint Rule Application

The inter-ViewPoint rules defined in step 1 express the relationships between partial specifications residing in different ViewPoints. It is a relatively simple task for the human observer to parse and interpret the rules mentally, but computer-support for inter-ViewPoint communication requires an explicit step to describe the process and mechanism of rule application. While a full description of such a mechanism is beyond the scope of this paper, an outline of the issues involved and the techniques that need to be deployed is given below.

Most inter-ViewPoint rules that traditional software engineering methods deploy require some form of pattern matching to check that values of certain types of objects are related by simple binary relations (e.g., =, <, >). For example, it is frequently necessary to check that the string values of various named objects have been preserved or that integer values are within certain numerical limits. Other rules are more complex in that the relationships between the partial specifications are not simply a comparison between typed values. Instead the rules express a correspondence between different types of objects in different specifications. To avoid having to define all the rules during method definition, it should also be possible to define the relationships separately in the form of, say, a computer algorithm or program for example. Ideally, a method designer would be provided with a predefined library of relationships at his/her disposal, with the option of defining any further relationships if required.

Inter-ViewPoint rules at this stage of the model pass through two states. On application of an inter-ViewPoint rule, the two ViewPoints VP_S and VP_D exist, but it is not yet known whether or not the relationship $\Re$ holds between them (figure 7c). Successful application of the rule, directly or after some conflict resolution, results in a valid relationship $\Re$ that holds between these two specific ViewPoints (figure 7d). To pass through these states, ViewPoints need to exchange information. VP_S needs to obtain a partial specification from VP_D, and if necessary transform it into a form it can understand and manipulate (so that pattern matching, for example, may be performed). If the relationship $\Re$ fails to hold, then VP_Dneeds to be made aware of this failure (i.e., another transfer), and some form of conflict resolution needs to be performed. In a typical software engineering setting, time constraints on such transfers may be insignificant, but if the ViewPoints are deployed in a real-time distributed environment (following a client-server model for example), then traditional problems such as communication load overhead or high rate of change of fetched server information may become much more significant [39].

There are two modes of application of an inter-ViewPoint rule:

Check Mode - in which question $?\Re$ is asked, that is, does the relation $\Re$ hold between VP_S and VP_D. Consequently, either $\Re$ holds or conflict resolution must be performed to make it hold.

Transfer Mode - in which the function $f(\Re, VP_S, VP_D)$ is applied to transfer and transform information from one ViewPoint to another so that the relation $\Re$ will hold between them.

An invoked inter-ViewPoint rule is normally applied in 'check mode' to begin with, after which a 'transfer mode' may be entered into if the rule fails. Information transfers between ViewPoint specifications may therefore be regarded as a form of conflict resolution, although the effectiveness of the resolution will depend on the granularity of the transferred information. Inconsistency handling in this setting is discussed in more detail in [17].

Clearly, the infrastructure of ViewPoints needs to be extended to handle the various transfers and transformations that will occur during typical inter-ViewPoint communication. One such modification might be the addition of ViewPoint *interfaces* to provide information hiding and other transformation services. These interfaces may also provide 'mailboxes' to which information from other ViewPoints may be 'posted' rather than forcibly transferred into destination ViewPoint specifications. It is then left to the discretion of individual ViewPoint owners to incorporate information and/or guidance residing in their ViewPoint mailboxes into their local ViewPoint specifications.

4.4. Structural Consequences

The above three inter-ViewPoint communication steps may be used to provide interesting structural information about ViewPoint-oriented methods, processes and

specifications. From the ViewPoint templates and the inter-ViewPoint rules defined within them (step 1), the structure of a ViewPoint-oriented *method* may be observed (figure 8).

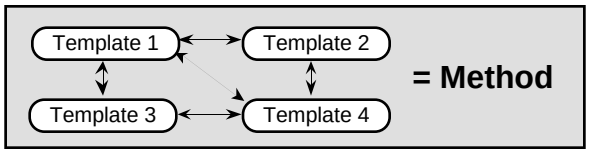

Figure-8: *Method structure: a method is a configuration of ViewPoint templates, related by inter-ViewPoint rules. Connecting arrows indicate inter-ViewPoint rules.*

A snapshot of a project at step 2 on the other hand shows the ViewPoints that have already been created for a project so far, and indicates what ViewPoints may be created from this particular configuration of ViewPoints. The snapshot therefore provides a more method-specific structural view of the ViewPoint-oriented development *process* (figure 9).

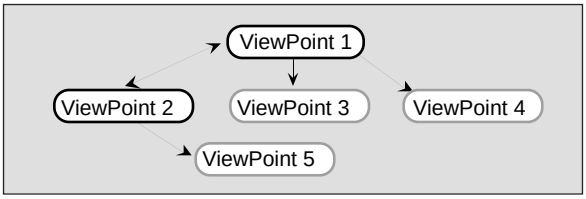

Figure-9: *ViewPoint-oriented development process: at any point during a system's development a number of ViewPoints will be under development, with further ViewPoints that need to be created from that point. Broken arrows point to ViewPoints not yet created, but reachable from the source ViewPoint.*

Finally, and by the end step 3, a configuration of ViewPoints has been created and the relationships between them have been checked and established. The configuration of ViewPoints at this stage is therefore a structural view the *system specification* at a particular point in time (figure 10). Figure-10 also illustrates the potential practical problems of scaling-up the ViewPoints framework to cope with large numbers of ViewPoints. Kramer and Finkelstein [24] propose the use of structural configurations to cope with this inevitable complexity. We thus envisage the use of "configuration ViewPoints" to act as organisational tools for grouping together closely related ViewPoints.

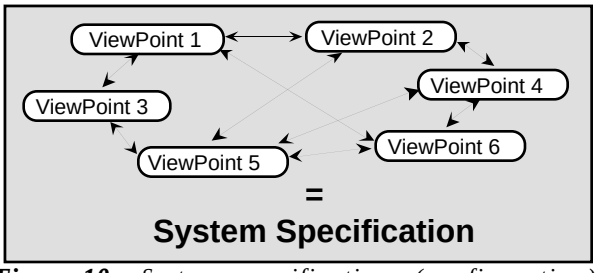

Figure-10: *System specification (configuration) structure. Arrows indicate inter-ViewPoint relationships that hold between the two connected ViewPoints.*

5. Tool Support

A generic, computer-based prototype environment called *The√iewer* [34] has been built in Objectworks/ Smalltalk, to support the ViewPoints framework. *The√iewer* (Figure-11) runs on a variety of platforms (e.g., Apple Macintosh, PC/MS-Windows and Unix/X-Windows), and provides tools for method construction and deployment as outlined in sections 3 and 4 of this paper. A number of simple graphical development techniques (such as hierarchical structuring and tabular data flow diagramming) have been described in ViewPoint templates and supported by CASE tools. These tools are partially generated from ViewPoint template descriptions using *The√iewer's* meta-CASE capabilities.

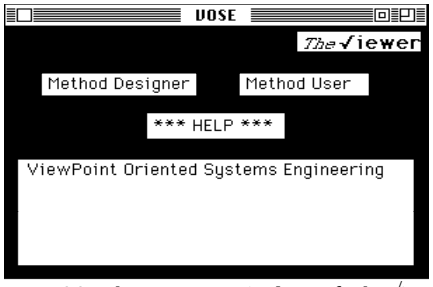

Figure-11: *The startup window of The√iewer.*

The model of inter-ViewPoint communication described in this paper is still under development and has not been incorporated into *The√iewer*. Nevertheless, a number of sub-projects have studied various implementations of inter-ViewPoint communication protocols. In [4], a model of inter-ViewPoint communication as dialogue was implemented in a Smalltalk-based tool called ICDC. In [33], we constructed a simple toolset (called CoreDemo) to support part of the CORE method, and investigated several types of consistency checks and information transfers between CORE stages. In [20, 21], a tightly integrated toolset was constructed to support ViewPoint templates describing a variant of Petri Nets [19]. ViewPoints developed by this toolset are managed by a hypertext-based environment called HyperView [20].

The use of ViewPoints as a vehicle for method and tool integration has also been investigated in the context of the REX [11] project, the objectives of which encompassed the development of reconfigurable and extensible distributed systems [24].

Continued work on a variety of communication models and their implementations is providing us with valuable experience in the expression and enactment of consistency checks and information transfers between many partial specifications. Thus for example, we derived the general form of the rules described in §4.1 by reverse-engineering hard-coded checks. We hope to extend *The√iewer* to support the model of inter-ViewPoint communication described in this paper, and use it as a vehicle to further demonstrate the ViewPoints approach.

6. Related Work

Work in a number of software engineering fields has made its mark on our ViewPoints framework. Analogies of ViewPoints may be found in multidatabases [3], where much of the research on interoperable, heterogeneous, multidatabase systems [1, 26] is relevant. Multidatabases deploy many, heterogeneous - possibly distributed - databases, based on more than one data model or schema. Many of the problems of checking consistency between such databases are therefore identical to the problems of checking and integrating multiple ViewPoint representation styles and specifications developed in those styles. Furthermore, research in the areas of method and tool integration and integrated project support environments [e.g., 5, 22, 28, 43] tackles many of the issues surrounding integration in the ViewPoints setting such as process modelling and tool support. A few integration models rely on the controlled transfer of information between a number of databases [41] in which objects are related via inter-database relationships [8].

Furthermore, system specification from multiple perspectives has been researched in various guises by a number of authors. Doerry *et al* [9] propose a model for composite system design based on multiple cooperating/interacting agents with individual behaviours and goals. In Dardenne *et al* [7] a goal-directed approach to composite system development is described, while Feather [13] suggests using many, parallel "evolutionary transformations", which may then be merged by replaying them sequentially. Leite [25] demonstrates a viewpoint analysis strategy for early validation of the requirements elicitation process (so-called viewpoint resolution).

Work on program transformation [44, 45] provides an additional vehicle for tackling consistency checks and information transfers between different ViewPoints. Robinson [37] proposes an multiple perspectives integration architecture as part of a model of specification design. Meyers and Reiss [29] study inter-perspective (*cf.* inter-ViewPoint) communication, and propose the development of a single canonical representation for software specification. Finally, Niskier *et al* [32] propose a pluralistic knowledge-based approach to software specification in the style we favour - using multiple overlapping views elaborated using multiple representation schemes. Their implementation (PRISMA) of this however, tightly couples the fixed views and uses a common, centralised (bottle-necked) data structure to express consistency checks.

7. Conclusions and Further Work

ViewPoints facilitate the partitioning of a problem domain into loosely-coupled, distributable objects that encapsulate partial specifications described in different notations and locally developed and managed according to different work plans. Although representation, development and specification knowledge are all bundled into the same object, they are separated within a single ViewPoint into slots, to facilitate their individual manipulation and enhance their tailorability and reusability. Tolerating the coexistence of multiple, heterogeneous ViewPoints to specify system requirements brings to the fore the problems of integration - these include the integration of specification fragments described using different notations, *and* the integration of methods and tools used to develop such descriptions.

In this paper we have explored the use of inter-ViewPoint rules to express the relationships between different ViewPoints. These rules are defined during method construction, and invoked and applied during specification development. They define the "regions of overlap" between pairs of ViewPoints, and thus identify "redundant" (but perhaps desirable) information. Moreover, while these rules describe syntactic relations between partial specifications in different ViewPoints, we may also view these same rules as definitions of semantic relations between these partial specifications. Further work is still needed however to describe more domain-specific knowledge and rules. One avenue of investigation may be to develop the role of ViewPoint owners in providing this domain knowledge.

Inter-ViewPoint rules themselves play a number of important roles in ViewPoint-oriented requirements engineering. First, they describe the relationships between different development techniques that form methods. In this context they are a vehicle for method integration. Second, they describe the relationships between the different tools that support the constituent development techniques that form methods. In this context they are a vehicle for tool integration. Third, they describe the relationships between various specification fragments found in different ViewPoint specifications. In this context they may be used to check consistency between partial specifications, or to transfer and transform information in one ViewPoint specification to another. Finally, ViewPoints may be used to represent requirements specification development participants, and therefore inter-ViewPoint rules describe protocols of interaction and behaviour between such participants. In this context, they are a useful vehicle for computer-supported cooperative work (CSCW).

In this paper we have concentrated on the problem of expressing these inter-ViewPoint rules for the purposes of inter-ViewPoint consistency checking. We have tried to describe these rules, independent of the mechanisms or communication protocols that will be deployed to invoke and apply them. In fact we have also said very little about the notation for describing the actual relations, $\Re$, between ViewPoints. These need to be explored further by looking at more complex relations than those demonstrated by our examples (namely, = and $\rightarrow$, which we nevertheless believe are typical of software engineering methods). We further believe that these rules may have an alternative mode of application to consistency checking, namely, a transfer mode. This is analogous to Prolog rules which may succeed, fail or generate the solutions that satisfy a rule. The mechanisms

for using these modes of application in the ViewPoints setting are currently being investigated. We believe that the transfer mode of inter-ViewPoint rule application deals with the issue of language conversion in our framework - an area where more work is needed.

Acknowledgements

The authors would like to thank the anonymous reviewers for their constructive comments on an earlier version of the paper. This work was partly funded by the UK SERC SEED project, and the Department of Trade and Industry (DTI) ESF project.

References

[1] R. Ahmed, P. De Smedt, W. Du, W. Kent, M. Ketabchi, W. Litwin, A. Rafii & M. Shan, "The Pegasus Heterogeneous Multidatabase System", IEEE Computer, 24(12), December 1991, 19-27.

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

[3] M.W. Bright, A.R. Hurson & S.H. Pakzad, "A Taxonomy and Current Issues in Multidatabase Systems", IEEE Computer, 25(3), March 1992, 50-60.

[4] W. Butcher, "ICDC - An Implementation of Dialogue in Smalltalk-80", M.Sc. Thesis, Department of Computing, Imperial College, London, September 1988.

[5] G. Clemm & L. Osterweil, "A Mechanism for Environment Integration", ACM Trans. on Prog. Languages and Sys., 12(1), January 1990, 1-25.

[6] R. Cunningham, A. Finkelstein, S. Goldsack, T. Maibaum & C. Potts, "Formal Requirements Specification - The FOREST Project", Proc. of 3rd Int. Work. on Spec. and Design, IEEE CS Press, 1985.

[7] A. Dardenne, S. Fickas & A. van Lamsweerde, "Goal-directed Concept Acquisition in Requirements Elicitation", Proc. of 6th Int. Work. on Soft. Spec. and Design, Como, Italy, October 1991, IEEE CS Press, 14-21.

[8] K.R. Dittrich, "The DAMOLKLES Database System for Design Applications: Its Past, its Present, and its Future", In "Software Engineering Environments: research and practice", K.H. Bennett (ed.), Ellis Horwood Ltd., Chichester, 1989, 151-171.

[9] E. Doerry, S. Fickas, R. Helm & M. Feather, "A Model for Composite System Design", Proc. of 6th Int. Work. on Soft. Spec. and Design, Como, Oct. 1991, IEEE CS Press, 216-219.

[10] S.M. Easterbrook, "Elicitation of Requirements from Multiple Perspectives", Ph.D. Thesis, Department of Computing, Imperial College, London, June 1991.

[11] REX Technical Annex, ESPRIT Project 2080, European Economic Commission, March 1989.

[12] M. Feather, "Language Support for the Specification and Development of Composite Systems", ACM Trans. on Prog. Languages and Systems, 9(2), April 1987, 198-234.

[13] M.S. Feather, "Detecting Interference when Merging Specification Evolutions", Proc. of 5th Int. Work. on Soft. Spec. and Design, USA, May 1989, IEEE CS Press, 169-176.

[14] A. Finkelstein & H. Fuks, "Multi-party Specification", Proc. of 5th Int. Work. on Soft. Spec. and Design, Pittsburgh, USA, May 1989, IEEE CS Press, 185-195.

[15] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein & M. Goedicke, "Viewpoints: A Framework for Integrating Multiple Perspectives in System Development", Int. J. of Soft. Eng. and Know. Eng., 2(1), March 1992, World Scientific Publishing Company, 31-58.

[16] A. Finkelstein & B. Nuseibeh, "Fine-Grain Process Modelling", Draft Proposal, 1993.

[17] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer & B. Nuseibeh, "Inconsistency Handling in Multi-Perspective Specifications", Tech. Rep. DoC 93/2, Imperial College, 1993

[18] D. Gabbay & A. Hunter, "Making Inconsistency Respectable: A Logical Framework for Inconsistency in Reasoning, Part 1 - A Position Paper", Proc. of Fundamentals of Artificial Intelligence Research '91, LNCS 535, Springer-Verlag, 19-32.

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

[20] P. Graubmann, "The HyperView Tool Standard Methods", REX tech. rep. REX-WP3-SIE-021-V1.0, Siemens, Munich, Germany, January '92.

[21] 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.

[22] M. Jarke, "Strategies for Integrating CASE Environments", IEEE Software, March 1992, 54-61.

[23] G. Kotonya & I. Sommerville, "Viewpoints For Requirements Definition", Soft. Eng. J., 7(6), November 1992, IEE, 375-387.

[24] J. Kramer & A. Finkelstein, "A Configurable Framework for Method and Tool Integration", Proc. of European Symp. on Soft. Development Environments and CASE Technology, Konigswinter, Germany, June 1991, Springer-Verlag.

[25] J.C.S.P. Leite, "Viewpoint Analysis: A Case Study", Proc. of 5th Int. Work. on Soft. Spec. and Design, Pittsburgh, Pennsylvania, May 1989, USA, IEEE CS Press, 111-119.

[26] W. Litwin, L. Mark & N. Roussopoulos, "Interoperability of Multiple Autonomous Databases", ACM Computing Surveys, 22(3), September 1990, 267-293.

[27] P. Mi & W. Scacchi, "Process Integration in CASE Environments", IEEE Software, March 1992, 45-53.

[28] S. Meyers, "Difficulties in Integrating Multiview Development Systems", IEEE Software, January 1991, 49-57.

[29] S. Meyers & S.P. Reiss, "A System for Multiparadigm Software Systems", Proc. of 6th Int. Work. on Soft. Spec. and Design, Como, Italy, Oct. 1991, IEEE CS Press, 202-209.

[30] G. Mullery, "CORE - a method for controlled requirements specification", Proc. 4th Int. Conf. Soft. Eng., 126-135, IEEE CS Press, 1979.

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

[32] C. Niskier, T. Maibaum & D. Schwabe, "A Pluralistic Knowledge-Based Approach to Software Specification", Proc. of European Soft. Eng. Conf. (ESEC 89), 1989.

[33] B. 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.

[34] B. Nuseibeh & A. Finkelstein, "ViewPoints: A Vehicle for Method and Tool Integration", Proc. of 5th Int. Work. on Computer-Aided Soft. Eng. (CASE '92), Montreal, Canada, 6-10th July 1992, IEEE CS Press, 50-60.

[35] L. Osterweil, "Software processes are software too", Proc. of 9th Int. Conf. on Soft. Eng. (ICSE-9), March-April 1987, IEEE Computer Society Press, 2-14.

[36] R. Wharton, "Doing it with RADs", IOPener, 1(3), February 1992, IOPT Meeting Report: Modelling Processes in Organisations, Praxis Systems plc., UK, 2-4.

[37] W.N. Robinson, "Integrating Multiple Specifications Using Domain Goals", Proc. of 5th Int. Work. on Soft. Spec. and Design, Pittsburgh, USA, May 1989, IEEE CS Press, 219-226.

[38] D.T. Ross, "Structured Analysis (SA): A Language for Communicating Ideas", IEEE Trans. on Soft. Eng., 3(1), January 1977, 16-34.

[39] M. Satyanarayanan, "An Agenda for Research in Large-Scale Distributed Data Repositories", Proc. of Int. Work. on Operating Systems of the 90s and Beyond, July 1991, LNCS 563, Springer-Verlag, 2-12.

[40] T. Shepard, S. Sibbald & C. Wortley, "A Visual Software Process Language", Communications of the ACM, 35(4), April 1992, 37-44.

[41] M. Stanley, "Typing in an Object Management System (OMS)", Proc. of Int. Work. on Env., Chinon, France, Sept. 1989, F. Long (ed.), LNCS 457, Springer-Verlag, 1990, 235-250.

[42] System Designers, "CORE - the Method", System Designers Plc., Issue 1.0, November 1985.

[43] A.I. Wasserman, "Tool Integration in Software Engineering Environments", Proc. of Int. Work. on Env., Chinon, Sept. 1989, F. Long (ed.), LNCS 457, Springer-Verlag, 1990, 137-149.

[44] D.S. Wile, "Local Formalisms: Widening the Spectrum of Wide-Spectrum Languages", Prog. Spec. & Transf., L. Meertens (ed.), April 1986, Elsevier Science Pub., 459-482.

[45] D.S. Wile, "Integrating Syntaxes & their Associated Semantics", USC/ISI Technical Report, 1991.

[46] P. Zave & M. Jackson, "Conjunction as Composition", ACM Trans. on Soft. Eng. and Methodology (to appear), 1993.