

Automated Consistency Checking for Multiperspective Software Specifications

Thanwadee Sunetnanta
Department of Computer Science,
Faculty of Science, Mahidol University,
Rama 6 Road, Bangkok 10400,
Thailand
cctth@mahidol.ac.th

Anthony Finkelstein
Department of Computer Science,
University College London
Gower Street, London WC1E 6BT, England
A.Finkelstein@cs.ucl.ac.uk

Abstract

Multiperspectives naturally arise out of co-operative work in applying appropriate technologies to construct different parts of an application. The representation styles of various perspectives can be highly heterogeneous and open-ended since those perspectives should be presented in a form appropriate to each participant in the software development process. This makes it difficult to provide *consistency checking* and *integration mechanisms* for the perspectives. This paper presents an approach of dealing with automated consistency checking for multiperspective software specifications regardless of their heterogeneity of representation. We apply expressive graph notations called *Conceptual Graphs* (CGs) as *visual consistency checking* notations. The combination of underlying logical reasoning and graph-based reasoning of CGs provide a powerful consistency checking mechanism. Our framework is illustrated with excerpts of a case study using Unified Modeling Language (UML).

1. Introduction

Rapid improvements in computer technology have provided us with a variety of different technologies, i.e. development methods and tools, which can be applied to software construction. Multiperspectives naturally arise out of co-operative work in applying appropriate technologies to construct different parts of an application. A single system development

technique is often not adequate to analyze and design a large and complex software specification. Accordingly, it would be beneficial to enable a software specification to be developed by a team of developers and to allow various development methods and tools to be used in the development. To facilitate such a development process, a software specification can be constructed through multiple perspectives, or ‘multiperspectives’ for short. We term such a specification a *multiperspective software specification*. A perspective represents a way in which a software developer individually describes or models the artifacts of the system under consideration. By viewing a software specification in a multiperspective manner, we obtain a set of manageable subproblems to which appropriate methods and tools can be applied.

The recognition of viewpoints, such as those in terms of opinions, specifications and services [1], offers a multiperspective organizational framework to deal with complexity control and separation of concerns in constructing software specifications. In our approach, we consider, in particular, the partitioning and the organization of perspectives of software specifications based on the *ViewPoints* framework. The *ViewPoints* framework has been described in detail in [2] and initially realized as an implementation prototype in [3]. A *ViewPoint* is defined as a loosely coupled, locally managed, self-contained object. It represents an “agent” having a “role-in” and a “view-of” a problem domain. Accordingly, it may be associated with responsibilities, specialities, or roles of the developer.

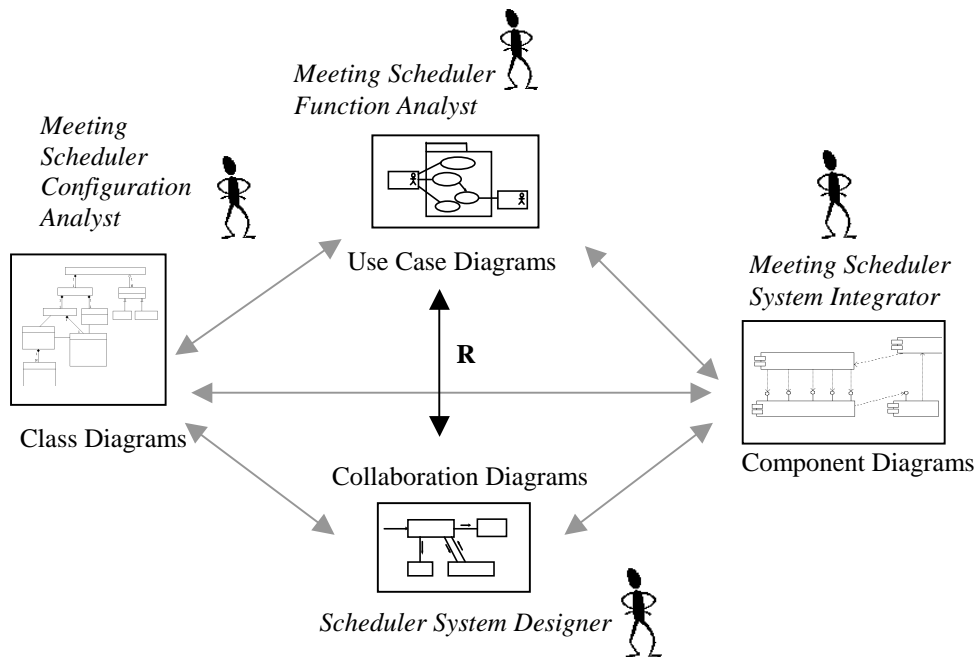


Figure 1. A Multiperspective Software Specification Through ViewPoints

Figure 1 illustrates different perspectives that are constructed in the development of our case study of the *Meeting Scheduler System* [4]. The system is an interactive system that supports decision making for a meeting plan. For each meeting request, the system determines a meeting date and location so that the potential participants will participate in the meeting effectively. In doing so, the meeting date and the location should be as convenient as possible to all participants.

In the example, we have a number of participants or software developers engaging in different development tasks for the scheduler system. The developers are allowed to select an appropriate representation to apply to their tasks. To demonstrate the practical use of our approach, we specifically employ the perspectives that are modeled using the *Unified Modeling Language* (UML) [5]. From the figure, the *Meeting Scheduler Configuration Analyst* uses class diagrams to represent his logical view of the system in terms of a structure of classes, while the *Meeting Scheduler Function Analyst* represents her view of system functions using use case diagrams. A use case diagram describes external *actors* and operational transactions or *use cases* in

which those actors participate to interact with the system. Similarly, the *Scheduler System Designer* applies collaboration diagrams to design the interaction of system objects, while the *Meeting Scheduler System Integrator* represents the view of system module dependencies using component diagrams.

Inevitably, the imposition of multiperspectives in software construction, such as in our example, raises crucial issues relating to consistency checking and integration. The relationships between various perspectives need to be rigorously maintained and validated to sustain co-ordination of the perspectives in order to ensure that the perspectives can work consistently together as an integrated whole. However, the representation styles of various perspectives can be highly heterogeneous and open-ended since those perspectives should be presented in a form appropriate to each participant in the software development process. As a consequence, it is rather difficult to provide consistency checking and integration mechanisms for those perspectives. Importantly, this difficulty needs to be tackled before we can implement an effective environment for the

development of multiperspective software specifications.

This paper presents an approach of dealing with automated consistency checking for multiperspective software specifications regardless of their heterogeneity of representation. One way to resolve such heterogeneity is simply to find a common way of understanding or describing different representational schemes. Our approach applies *conceptual graphs* (CGs) [6] as a *meta-representational and reasoning paradigm* to perform such automation. Section 2 extends the case study in Figure 1 onto the ViewPoints framework and describes our basic idea of meta-representation languages for automating consistency checking. The differences of our approach in comparison with the existing work are highlighted. Section 3 contains an overview of CGs. The details of our CGs meta-representation language are given in Section 4. Section 5 contains an illustrative example for the feasibility and the applicability of our approach in constructing multiperspective software specifications that are modeled using the UML. Section 6 summarizes this work and discusses its contributions.

2. The ViewPoints Framework and Meta-Representation Languages

As previously mentioned, we specifically apply ViewPoints to the organization and the partition of knowledge to manage different concerns and tools for a multiperspective software specification. A ViewPoint encapsulates *specification knowledge*, *representation knowledge* and *development knowledge* about a particular problem domain. These different types of knowledge are captured into five ViewPoint template slots, which are *domain*, *style*, *work plan*, *specification* and *work record* slots. A ViewPoint is created by instantiating a *ViewPoint template*. A ViewPoint template is a ViewPoint in which only the style and work plan slots have been elaborated. A single ViewPoint template is therefore a description of development techniques that can be used to produce a ViewPoint. As it is not our intention to describe the ViewPoints framework in detail in this paper, the descriptions of these ViewPoint template slots are elaborated in [2].

In Figure 1, bi-directional arrows depict the relationships between ViewPoints within and across

the domain. Such relationships enable explicit analysis of interaction and coordination among the ViewPoints so that the ViewPoints can be constructed independently, but managed jointly or interrelated through the relationships. For example, the UML specifies that the sender and the receiver of a message participate in a collaboration that defines the context of the interaction [7]. Accordingly, all actors defined in use case diagrams must correspond to those representing the collaborations that are defined in collaboration diagrams. As a result, there exists a relationship R in Figure 1 between ViewPoints represented in terms of use case diagrams and those represented in terms of collaboration diagrams. Further consistency rules of UML diagrams are formulated in [8].

A number of approaches have been proposed for consistency and integration mechanisms of multiple views or multiperspectives in software specifications. Nonetheless, there is no consensus about which is the most useful. Our approach is fundamentally different from the previous efforts that are the implementation of pair-wise translation rules and the application of canonical representations.

The use of translation rules is a common method to interpret perspectives in different representation styles. Simple translation can be done in a pairwise manner, i.e. as a one-to-one mapping from one ViewPoint to another. ViewPoints in n different representation styles would therefore require $n(n-1)$ translation rules for consistency checking and integration. Inconsistencies can then be detected and resolved at the level of a specific language or a representation style into which the ViewPoint is transformed. For example, the studies in [9][10] present the translation of some formal description techniques (FDTs), such as one between LOTOS and Z specification language. We argue that such usage of translation rules is not best suited to support consistency checking and integration of multiperspective software specifications such as in the ViewPoints framework, because the implementation of translation rules can be an intensive task, in particular when the representation styles of ViewPoints are so numerous and significantly different from one another.

A classic alternative to direct translation of perspectives is to integrate them through a *canonical representation*. A canonical representation captures

various perspectives into a common semantic model. In this respect, a ViewPoint is responsible for translating only from its representation style to a canonical representation and vice versa. Inconsistencies can then be detected and resolved at the level of that canonical representation. This reduces the number of translation rules required in consistency checking and integration mechanisms to $2n$ for ViewPoints in n different representation styles. A number of canonical representations have been employed for such mechanisms. Examples are first-order predicate logic [11], *Abstract Syntax Trees* [12][13], *Semantic Program Graphs* [14], *Conceptual Graphs* [15], *Telos* [16] and *Graph Grammar* [17]. As discussed in detail in [8], each of these canonical representations poses different degrees of limitation to translation and reasoning issues. Thus it is difficult to identify a single canonical representation that can interpret the entire semantic content of various representation styles. Furthermore, we contend that the deployment of a canonical representation in the ViewPoints framework restricts the extendibility of the framework, as all the ViewPoint representation styles would be tightly bounded to a particular canonical representation. If such a canonical representation is not expressive enough, it will be difficult to add a new representation style to the framework.

We propose an alternative approach using a *meta-representation language* to support consistency checking of multiperspective software specifications. Our approach aims to avoid the difficulties associated with the previously mentioned existing approaches. The basic features of the meta-representation language that we propose are:

- The language provides notations for describing a representation style. In other words, it allows a specialist, called the *method engineer*, to observe the definitions of representation styles on the basis of its constructs or notations. Such constructs are used to relate syntactic terms of a representation style to the corresponding terms in the language. Rather than using direct translation rules between ViewPoints, all ViewPoints will be described here in terms of a common meta-representation language. This will radically reduce the number of translation rules required to relate ViewPoints.
- Although in some sense a meta-representation language resembles an integrated form of representation, it is slightly different from a

canonical representation. Our meta-representation language aims to be a *translation aid*. To do so, it describes representational definitions more abstractly than a canonical representation. In other words, it does not set out to provide a complete mapping of semantic data models in the way that a canonical representation aims to do. A method designer may use the meta-representation language to express an abstraction of a representation style in any possible form that adequately facilitates an automatic generation of a representation style translator and a consistency checking procedure. Furthermore, such an abstraction is not required to preserve the semantic equivalence of representation styles. Accordingly, the definition of a representation style in terms of our meta-representation language is not necessarily a one-to-one mapping, which is the case when we translate a representation style into a canonical representation. The automatic generation is carried out via the representational types of the meta-representation language that are used to express the abstraction.

As mentioned earlier, we specifically employ *conceptual graphs* (CGs) [6] as a meta-representation language. The graphs are selected because of their intuitive, graphical notations and expressive power of reasoning.

3. An Overview of Conceptual Graphs

CGs were devised by John F. Sowa as a synthesis of several techniques for knowledge representation and originally applied as a representational language for knowledge acquisition in many aspects of artificial intelligence (AI) [18]. The fundamentals of CGs are based on the existential graphs of Charles Sanders Peirce [19] and the semantic networks [20]. CGs combine the expressive power of natural languages with the formality of symbolic logic. With direct mapping to and from natural languages, CGs can serve as an intermediate interpretation language between computer-oriented specifications and natural languages. With their graphical notations and expressive system of logic, the graphs can serve as a readable formal design and specification language. Although CGs were originally found mostly in AI applications, the graphs have also been used for various kinds of application, ranging from database

interfaces and text retrieval to natural language processing and reasoning.

The basic graphical notations of CGs are boxes and circles, representing concepts and conceptual relations respectively. A simple CG can be depicted below:



The direction of an arrow in the above CG indicates the subject and the object in a relation. The above CG expresses a relation called *RELATION* that occurs between two concepts, *CONCEPT_1* and *CONCEPT_2*. The CG can be read as "A *RELATION* of a *CONCEPT_1* is a *CONCEPT_2*".

The simplicity and expressiveness in transforming CGs into first-order predicate logic offer many benefits for applying CGs to facilitate consistency checking in our work. CGs can be mapped to and from predicate calculus and many forms of logic, such as propositional logic and modal logic [6][21]. Additionally, there have been some attempts to perform CG reasoning based on the graph models of CGs themselves [22][23]. For the sake of simplicity, we apply the transformation of CGs into first-order predicate logic to perform reasoning procedure in our framework.

As it is not our intention to explain the notion of CGs at length and we rather intend to demonstrate how the graphs are applied in our work, further details of CGs formalism and operations can be seen from [6]. In the section that follows, we elaborate the usage of CGs in our approach.

4. CGs as a Meta-Representational Language for Consistency Checking Automation

We enable the automation of the consistency checking process by providing a CG meta-representation language to describe various forms of ViewPoint representation. Section 5 will illustrate the examples of such meta-representation. To translate different representation styles, the notations of a ViewPoint representation style are mapped into the CG constructs, i.e. concepts and conceptual relations. Such a CG definition constitutes a metamodel, namely a *meta-representation binding*, of the style. A

meta-representation binding is therefore the visualization of syntactic and semantic descriptions of a ViewPoint representation style. It indicates a permissible structure of CG concepts and conceptual relations that can be expressed in the style. As the example in Figure 3, use cases and actors in use case diagrams are conceptualized as two concepts, *use cases* and *actors*, having a conceptual relation that indicates that they have interaction with each other.

When a representation style is used to describe a ViewPoint, the meta-representation binding of the style is used as a basis template to instantiate CG meta-representation of the ViewPoint. As a result, ViewPoint specifications are all abstractly transformed into CGs. The detailed procedure of such transformation has been demonstrated elsewhere in [24].

Since all ViewPoint specifications are transformed into CGs, we are able to enhance the interoperability among the ViewPoints. By interoperability we mean the ability of two or more ViewPoints to understand each other or co-operate to exchange information despite having been expressed in different representation styles. We enable this by allowing consistency rules that sustain co-ordination or relationships of ViewPoints to be expressed in terms of CGs. Such a consistency rule is defined by using the abstraction from the meta-representation binding of ViewPoint representation styles. Accordingly, we can construct an automated procedure for consistency checking between ViewPoints with heterogeneity of representation by abstracting them up one level to CGs. Figure 2 illustrates our framework for such an automated procedure.

5. An Illustrative Example: Multiperspectives Modeling Using UML

This section provides an example to illustrate the feasibility and the applicability of our CG meta-representation language in constructing multiperspective software specifications. To show the practical use of the language, the examples are constructed by using *UML* [5], which is a well-known object-oriented modeling representation for software analysis and design.

Consistency Checking Between Use Case Diagrams and Collaboration Diagrams

In our case study, we construct a multiperspective software specification using two types of UML diagrams, which are use case diagrams and collaboration diagrams (c.f. Figure 1). Use case models were originally used for requirement analysis in the Objectory [25]. Figure 3 shows the meta-representation binding of use case diagrams in terms of CGs. From the figure, the concept *use case* represents a use case, which is identified by its name describing the functionality of that use case.

Similarly, the concept *actor* represents an actor, which is identified by the role by which it participates in the system. The relation *association* in the figure represents an interface between an actor and a use case. The concept *use* represents a *use* relationship from one use case to another. Such a *use* relationship indicates that an instance of the source use case will also include the behavior specified by the destination use case. The concept *extends* represents an extend relationship from a use case to another. Such an *extends* relationship indicates that an instance of the destination use case may include the behavior specified by the source use case.

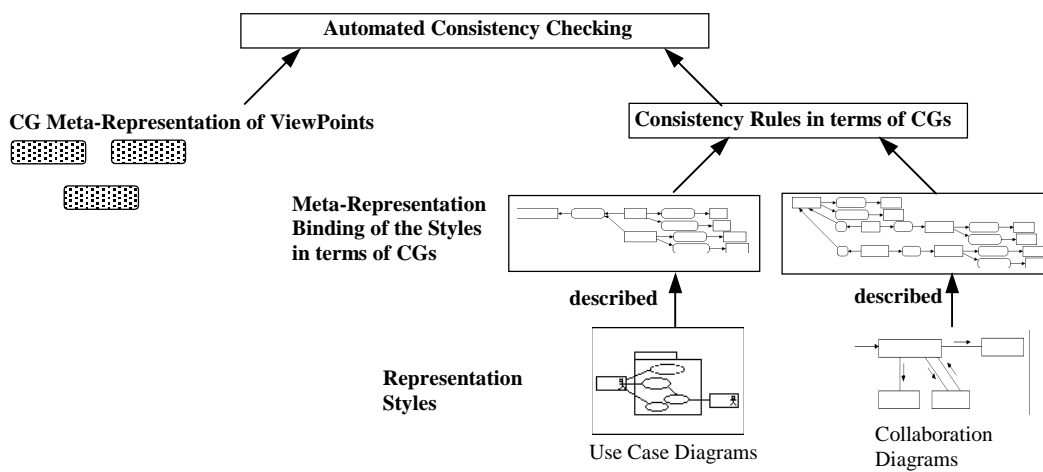


Figure 2. Automated Consistency Checking Using the CG Meta-Representation Language

A collaboration diagram describes a sequence of object interactions in terms of a thread of execution. In other words, the diagram shows the objects and their collaborations for participating in an operation.

As opposed to a sequence diagram, a *collaboration diagram* shows such interactions without timing constraints [5].

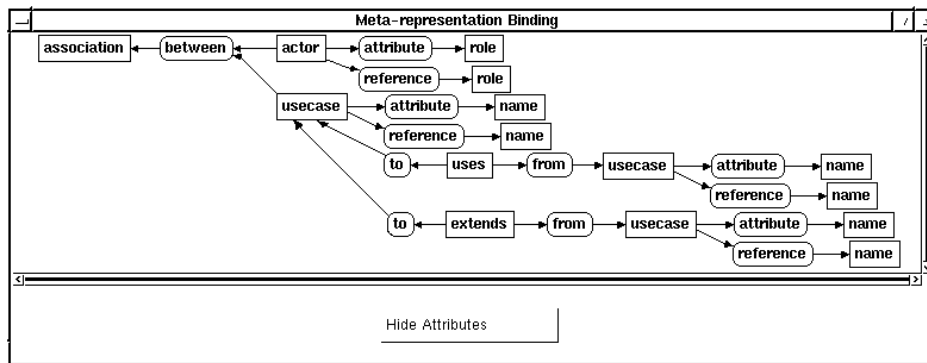


Figure 3. Meta-Representation Binding of Use Case Diagrams

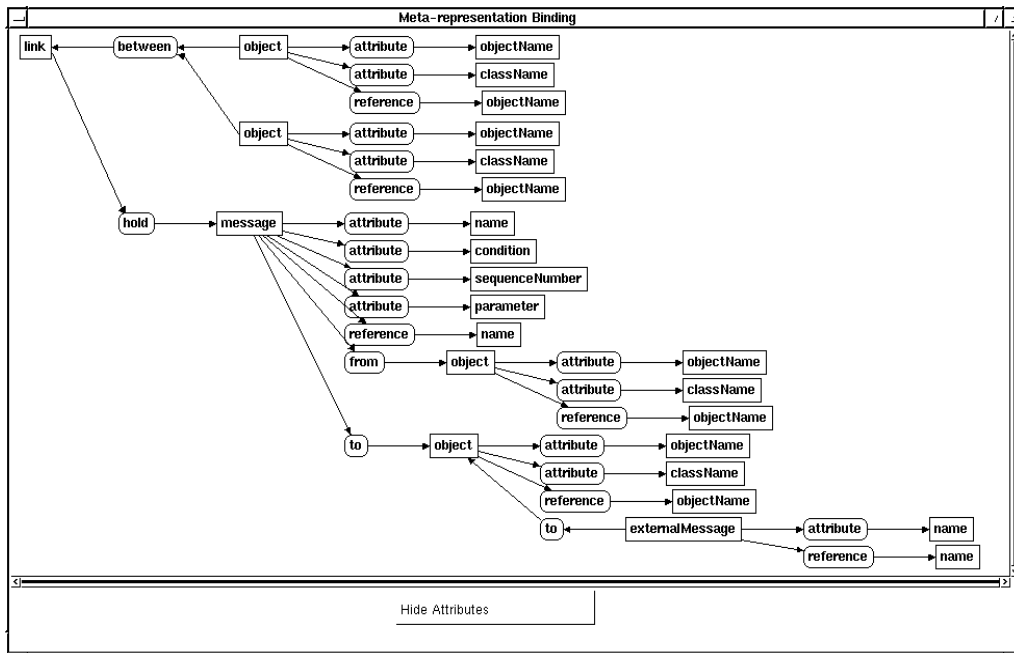


Figure 4. Meta-Representation Binding of Collaboration Diagrams

Figure 4 shows the meta-representation binding of UML collaboration diagrams. The concept *object* in the figure represents an object that is identified by its name and the class from which it is instantiated. The concept *link* represents an interaction, i.e. a set of messages, between objects. Likewise, the concept *message* represents a message that is sent from an object to another. A message is indicated by its name, describing a message expression. It may be specified with *condition*, *sequence number* and *parameter*. The concept *external message* represents a call from the entities outside the scope of the operation.

As mentioned earlier, the following rule, *R*, specifies the relationship between the model elements of use

case diagrams and those of collaboration diagrams:

R: *The sender and the receiver of a message must participate in a collaboration that defines the context of their interaction.*

Accordingly, all *actors* defined in use case diagrams must correspond to those representing the collaborations that are defined in the interaction type of diagrams. This rule is illustrated as CG in Figure 5. The CG states that if there exists a ViewPoint which is represented the domain *Y* in terms of *use case diagrams* and that ViewPoint contains any actor with role *X*, then there must exist a ViewPoint which is represented in terms of *collaboration diagrams*

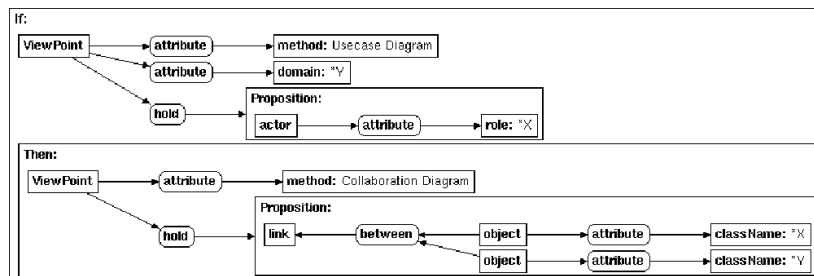


Figure 5. CG Consistency Rule for the Relationship Between Use Cases and Their Collaboration

and that ViewPoint contains a link between an object of class *X* and an object of class *Y*. Such a link in collaboration diagrams represents the interface of an actor with role *X* and a use case of the domain *Y*.

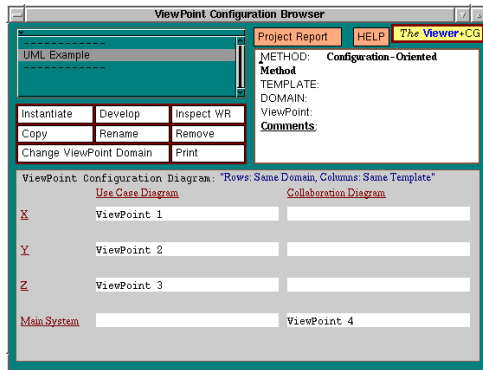


Figure 6. Project Browser of Our Example

To verify the relationship between use cases and their collaboration, assume that we construct two ViewPoints, "ViewPoint 1" and "ViewPoint 4" in terms of use case diagrams and collaboration diagrams respectively. Figure 6 shows the project

browser for the construction of our example specification.

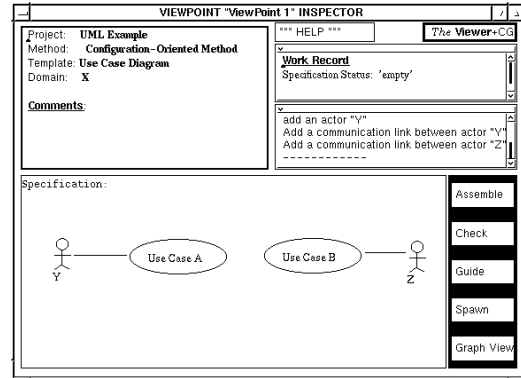


Figure 7. Use Case ViewPoint of *X*

Figure 7 shows the specification of "ViewPoint 1" of Domain *X*. The specification contains two use cases *A* and *B*. The use case *A* is associated with the actor *Y* while the use case *B* is interacted with the actor *Z*.

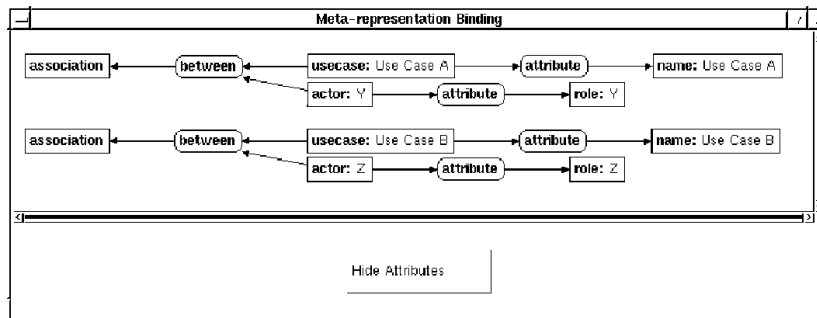


Figure 8. Transformation of Use Case ViewPoint into CGs

Figure 8 shows the transformation of the use case ViewPoint "ViewPoint 1" into CGs. The transformation is carried out using or instantiating the meta-representation binding of the use case diagrams as defined in Figure 3. The graph in Figure 8 is semantically equivalent to the specification of the ViewPoint in Figure 7.

Figure 9 shows the specification of "ViewPoint 4" of Domain *Z*. The specification shows the collaboration between three objects, *Object1* of class *X*, *Object2* of class *Y*, and *Object3* of class *Z*. In the specification,

there is a message *m1* representing an interaction from *Object2* to *Object1*.

Figure 10 shows the result of the transformation of collaboration ViewPoint into CGs. The transformation is carried out using or instantiating the meta-representation binding of the collaboration diagrams as defined in Figure 5. The graph in Figure 10 is semantically equivalent to the specification of the ViewPoint in Figure 9.

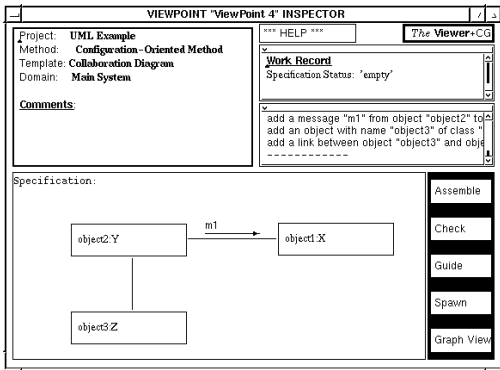


Figure 9. Collaboration ViewPoint of the Main System of Domain X, Y, Z

Automated Consistency Checking Process

As discussed earlier, we automate the process for ViewPoint consistency checking by using the abstract forms of ViewPoints, i.e. CG meta-representation, and the consistency rules that are formed in terms of CGs (c.f. Figure 3). To verify the relationship between use case diagrams and collaboration diagrams in our example, the CG consistency rule represented such relationship (c.f.

Figure 5) are checked with the CG meta-representation of "ViewPoint 1" (c.f. Figure 8) and the CG meta-representation of "ViewPoint 4" (c.f. Figure 10). Figure 11 shows how the CGs for both the rule and the specifications that are executed in the checking process.

Figure 12 presents the output window from applying the CG consistency rule in Figure 5 for the relationship between use cases and their collaborations. The bottom left window in the figure indicates that there is an error encountered in the interaction checking between use cases and their collaborations. In the figure, the messages in the top right window indicates that there is a missing link that represents the collaboration of the actor Z with the use cases in the ViewPoint 'ViewPoint 1'. Such an inconsistency is derived from CGs in the automated checking process in Figure 11. The CG predicates are implemented by adapting CG-editor [26] and prolog transformation. We are currently dealing with an automated process to interpret the resultant CGs from our consistency checking mechanism into meaningful actions for inconsistency handling.

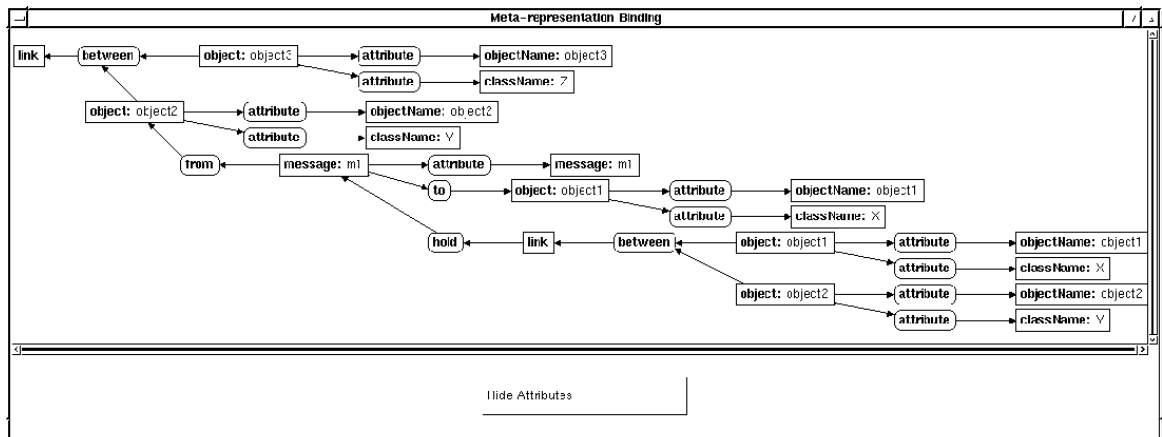


Figure 10. Transformation of Collaboration ViewPoint into CGs

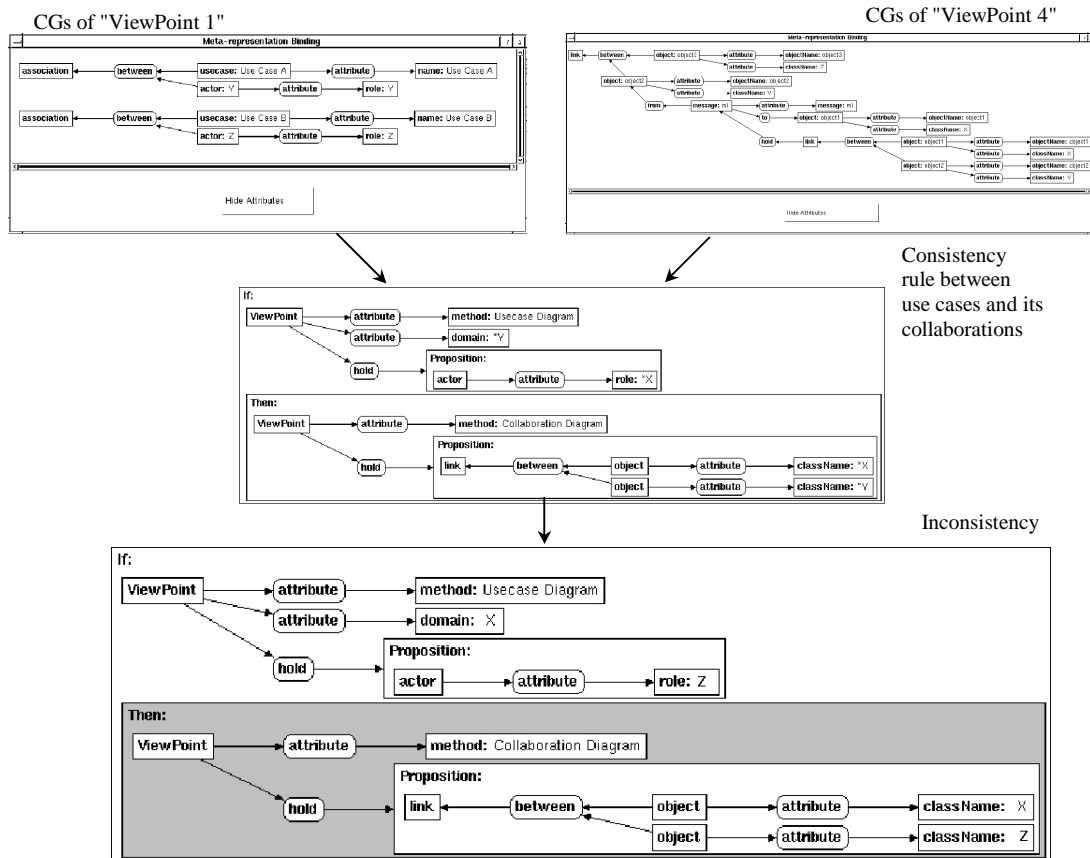


Figure 11. An Example of CG-Based Automated Consistency Checking Process

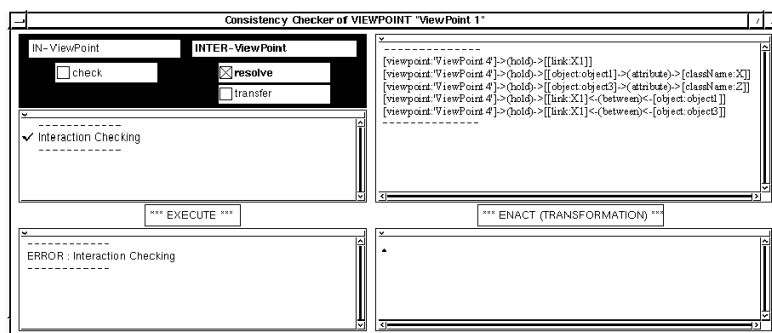


Figure 12. Consistency Checking Result

6. Conclusion

Multiperspective development, such as the ViewPoints framework, has its pros and cons. Supporting multiperspectives in software

development is a way of dealing with complexity in the analysis and design process. It also contributes to an effective improvement, including the benefits from the potential of decentralization and distribution in construction process, of software production. As discussed in the introductory part, the integration of

the multiperspectives is *however* problematic due to the heterogeneity of representation.

By abstracting a ViewPoint specification up one level to CGs, we are able to augment a ViewPoints-based environment with an automated consistency checking procedure that is independent of representation styles. CGs offers intuitive, but expressive, notations to describe ViewPoint representation styles. The visualization of a representation style in CG form helps software developers to understand, and enable the developers to express clearly, the syntax and semantics of ViewPoint-based development techniques, including and consistency rules. The graphs also serve as *visual* consistency checking notations. The combination of the underlying logical reasoning and graph-based reasoning of CGs provides a powerful consistency checking mechanism. In addition, the mapping of descriptive definitions of the rules in natural languages (English) to CGs is more readable than the mapping to logic predicates directly.

The consistency rules that we implement are in some sense similar to the well-formedness rules given by using the *Object Constraint Language* (OCL) [27] in the UML semantics document. As a publicly available standard, the enhancement of OCL is in the process of revision at Object Management Group (OMG). Although, OCL offers significant benefits by reason of their formal reasoning techniques, its notations are typically difficult to learn for those without a formal method background. A mapping between OCL and our CG-based consistency rules can be developed as future work.

7. References

- [1] Leite J. C. S. P., "Viewpoints on Viewpoints", International Workshop on Multiple Perspectives in Software Development, SIGSOFT'96 Workshops, Vidal L., Finkelstein A., Spanoudakis G., and Wolf A. L. (Eds.), ACM Press, San Francisco, USA, 1996, pp. 285-288.
- [2] Finkelstein A., Kramer J., Nuseibeh B., Finkelstein L., and Goedicke M., "Viewpoints: A Framework for Integrating Multiple Perspectives in System Development", International Journal of Software Engineering and Knowledge Engineering, Vol. 2, No. 1, World Scientific Publishing Co., March, 1992, pp. 31-58.
- [3] Nuseibeh B., "A Multi-Perspective Framework for Method Integration", PhD. Thesis, Department of Computing, Imperial College, University of London, London, UK, 1994.
- [4] Feather M. S., Fickas S., Finkelstein A., and Lamsweerde A. V., "Requirements and Specification Exemplars", Automated Software Engineering Journal, No. 4, 1997, pp. 419-438.
- [5] Booch G., Jacobson I., and Rumbaugh J., "UML Notations", UML Documentation Set Version 1.1, Rational Software Cooperation, September, 1997.
- [6] Sowa J. F., "Conceptual Structures: Information Processing in Mind and Machine", Addison-Wesley, Reading, MA, 1984.
- [7] Booch G., Jacobson I., and Rumbaugh J., "UML Semantics", UML Documentation Set Version 1.1, Rational Software Cooperation, September, 1997.
- [8] Thanitsukkarn T., "Multiperspective Development Environment for Configurable Distributed Applications", Ph.D. Thesis, Department of Computing, Imperial College, University of London, UK, 1999.
- [9] Boiten E., Bowman H., Derrick J., and Steen M., "Cross Viewpoint Consistency in Open Distributed Processing (Intra Language Consistency)", Technical Report 8-95, Computing Laboratory, University of Kent, Canterbury, UK, June, 1995.
- [10] Bowman H., Derrick J., and Maarten Steen M., "Some Results on Cross Viewpoint Consistency Checking", IFIP TC6 International Conference on Open Distributed Processing, Brisbane, Australia, Chapman and Hall, 1995, pp. 399-412.
- [11] Zave P., and Jackson M., "Conjunction as Composition", ACM Transactions on Software Engineering and Methodology, Vol. 2, No. 4, October, 1993, pp. 379-411.
- [12] Habermann A. N., and Notkin D., "Gandalf: Software Development Environments", IEEE Transactions on Software Engineering, Vol. SE-12, No. 12, December, 1986, pp. 1117-1127.
- [13] Reiss S. P., "PECAN: Program Development Systems that Support Multiple Views", IEEE Transactions on Software Engineering, Vol. SE-11, No. 3, March, 1985, pp. 276-285.
- [14] Meyers S. D., "Representing Software Systems in Multiple-View Development Environments", PhD.

Thesis, Department of Computer Science, Brown University, May, 1993.

[15] Delugach H. S., "Specifying Multiple-Viewed Software Requirements with Conceptual Graphs", *Journal of System Software*, Vol. 19, 1992, pp. 207-224.

[16] Spanoudakis G., and Finkelstein A. "Reconciling Requirements: a Method for Managing Interference, Inconsistency and Conflict", *Annals of Software Engineering, Special Issue on Software Requirement Engineering*, No. 3, 1997.

[17] Nagl M., and Schürr A., "Software Integration Problems and Coupling of Graph Grammar Specifications", In Ehrig H., Cuny J., Engels G., and Rozenberg G. (Eds.), *Proceedings of the 5th International Workshop on Graph Grammars and their Applications to Computer Science*, Lecture Notes in Computer Science 1073, Williamsburg, VA, USA, November, Springer-Verlag, 1994, pp. 155-169.

[18] Sowa J. F., "Conceptual Graphs as a Universal Knowledge Representation, Computer and Mathematics with Applications", Vol. 23, Part 2-5, 1992, pp. 75-93.

[19] Robert D. D., "The Existential Graphs of Charles S. Peirce", In Sebeok T. A. (Ed.), *Approach to Semiotics*, Mouton & Co., NY, 1973.

[20] Shastri L., "Semantic Networks: An Evidential Formalization and Its Connectionist Realization", *Research notes in artificial intelligence*, Pitman, London, 1988.

[21] Berg H. van den, "Modal Logics for Conceptual Graphs", *Proceedings of the 1st International*

Conference on Conceptual Structures, ICCS'93, In Mineau G. W., Moulin B., and Sowa J. F. (Eds.), *Conceptual Graphs for Knowledge Representation*, Quebec City, Canada, August 4-7, Springer-Verlag, 1993, pp. 411-429.

[22] Chein M., and Mugnier M. L., "Conceptual Graphs are Also Graphs", LIRMM (CNRS and Université Montpellier II), France, Report Number: RRI CHIEN 95, 1995.

[23] Guinaldo O., "Conceptual Graphs Isomorphism: Algorithm and Use, Proceedings of the 4th International Conference on Conceptual Structures, ICCS'96, In Eklund P., Ellis G., and Mann G. (Eds.), *Conceptual Structures: Knowledge Representation as Interlingua*, Sydney, Australia, Springer-Verlag, 1996, pp. 160-174.

[24] Thanitsukkarn T., and Finkelstein A., "A Conceptual Graph Approach to Support Multiperspective Development Environments", *Proceedings of the 11th Knowledge Acquisition For Knowledge-based Systems Workshop*, Gaines B. R. and Musen M. (Eds.), Vol. 1, Banff, Canada, 1998.

[25] Jacobson I., "Object-Oriented Software Engineering: A Use Case Driven Approach", Addison-Wesley, 1992.

[26] Petermann H., Möller J.-U., and Wiese D., "CG-Editor User's Guide", University of Hamburg, 1995.

[27] Booch G., Jacobson I., and Rumbaugh J., "Object Constraint Language Specification", UML Documentation Set Version 1.1, Rational Software Cooperation, September, 1997.