Divergent Views in Goal-Driven Requirements Engineering

Axel van Lamsweerde
Université catholique de Louvain, Département d'Ingénierie Informatique
B-1348 Louvain-la-Neuve (Belgium)
avl@info.ucl.ac.be

1. Introduction

Requirements engineering is a complex process composed of several concurrent activities.

- Acquisition: raw, preliminary material is elicited about the domain, the existing system, and its deficiencies; goals on the target system are identified therefrom;
- Modelling and specification: alternative models for the target system are elaborated to meet such goals, and defined in a precise way to produce system requirements;
- Assesment: the specifications are analyzed against possible deficiencies (such as inadequacy, incompleteness, or inconsistency) and evaluated against feasibility (in terms of resources required, development costs, etc.);
- Negotiation and commitment: requirements from the alternative models are discussed among stakeholders and "best" compromises that receive mutual agreement are selected;
- Documentation: the various decisions made during the process are documented together with their assumptions and rationales;
- Evolution: requirements are modified to accommodate corrections, environmental changes, or new goals.

These various activities involve multiple parties --clients, users, domain specialists, requirements engineers, software developers, and so forth. Different parties in general have different objectives, concerns, perceptions, knowledge, skills, and expression means. Requirements completeness and adequacy requires that all relevant viewpoints be captured and eventually integrated. Requirements engineering methods should therefore provide support for *representing* multiple viewpoints and for *reasoning* about them with such purposes in mind as detecting and resolving inconsistencies, merging redundant information, propagating changes, etc.

Proceedings of the ACM SIGSOFT Workshop on Viewpoints in Software Development, San Francisco, Oct. 1996, 252-256.

Viewpoint capture has been advocated since the early days of requirements engineering [Ros77]. Preliminary proposals for viewpoint-oriented acquisition, negotiation, and cooperative modelling appeared only later [Fin89], [Rob89], [Pot94], [Boe95]. Two kinds of approaches have emerged.

- In a *multi-paradigm* approach, specifications in different viewpoints can be written in different notations. Multi-paradigm viewpoints are analyzed in a centralized or distributed way. Centralized viewpoints are translated into some logic-based "assembly" language for global analysis; viewpoint combination then amounts to some form of conjunction [Nis89], [Zav93]. Distributed viewpoints have specific inter-viewpoint consistency rules associated with them; consistency checking is made locally on pairs of viewpoints by evaluating the corresponding rules [Nus93]. Such rules typically capture "static semantics" constraints between pairs of (semi-formal) languages. [Nus93] goes further by proposing a construct for viewpoint representation that allows additional process-level information to be attached to a viewpoint, such as its owner, a workplan to be followed, or its development history.
- In a *single-paradigm* approach, specifications in different views are written in a same language; a same conceptual unit is manipulated from one view to another under different structures, different focus of concern, etc. Consistency may then be prescribed explicitly through inter-view invariants or implicitly through synchronization of operations [Jac95].

Related efforts for structuring information in terms of views exist in other areas, e.g., software development environments [Hab88] and databases [Ber92]. For example, the context mechanism proposed in [Myl95] allows different views of a same information unit to be associated under different names to different owners, time points, etc.; a transaction on a view may have its effect visible or not outside the view; change propagation rules can be defined from one view to the other.

This paper outlines our recent efforts in (i) modelling views associated with different actors involved in the requirements elaboration process, (ii) identifying different kinds of inconsistencies that may arise among such views, (iii) detecting inconsistencies of those various kinds, and (iv) resolving such inconsistencies. As in [Jac95], our specification language is formal; but it is a rich, multi-paradigm language for requirements specification. In particular, the explicit modelling and formalization of goals allow conflicts among requirements to be handled at their root, that is, at goal level [Rob89]. Views in our approach are also explicitly linked to their owners. Unlike [Nus93], the specification and assesment process models are kept separate from product-level views and defined at a meta level. In our approach, views may also contain formal specifications.

2. Modelling Views in KAOS

Our work has been done in the context of the KAOS methodology for goal-driven requirements elaboration [Dar93], [Lam95]; some of the principles and results discussed here are however independent of this methodology.

The KAOS approach is aimed at supporting the whole process of requirements elaboration --from the high-level goals that should be achieved by the composite system to the operations, objects and constraints to be implemented by the software part of it. (The term "composite system" is used here to denote the intended software together with its environment [Fea87].) KAOS provides a multi-paradigm specification language and a goal-directed elaboration method. The *language* combines semantic nets [Bra85] for the conceptual modelling of goals, constraints, agents, objects and operations in the system; a real-time temporal logic [Koy92] for the specification of goals, constraints and objects; and state-based specifications [Pot91] for the specification of operations. It supports the separation of requirements from domain descriptions [Jac93]. The *method* roughly consists of (i) identifying and refining goals progressively until constraints that are assignable to individual agents are obtained, (ii) identifying objects and operations progressively from goals, (iii) deriving requirements on the objects and operations to meet the constraints, and (iv) assigning the constraints, objects and operations to the agents. The formal assertion layer allows one to reason formally about goal refinements [Dar96] and goal-directed derivations of operations [Dar93].

We use the same modelling technique at the product and process levels, with some renaming of the underlying abstractions to avoid confusions. At the product level, the requirements model is captured in terms of goals, agents, objects, operations, etc. At the process level, the elaboration process model is captured in terms of objectives, actors, artefacts, operators, etc. An artifact is a portion of the whole requirements model defined at the product level; artefacts are produced by actors using elaboration operators to meet the objectives.

Consider a library system as an example. At the product level, one might identify a goal *Achieve* [CopiesEventuallyAvailable] whose satisfaction would require the cooperation of Staff, AutomatedMonitoring and Borrower agents; this goal might be produced at the process level by Staff and Specifier actors applying a RefineGoal operator to meet the objective *Achieve* [GoalsOperationalized].

Different actors will in general view domain descriptions and requirements under different *facets*. The latter may be complementary, overlapping, and/or partly inconsistent in a sense to be made precise.

Our view construct is introduced to (i) restrict the visibility of conceptual features to specific contexts associated with corresponding actors, and (ii) allow inconsistencies to be captured and recorded for later resolution.

A view is defined as a ternary relationship linking an actor, a master concept and a facet of it (see figure below).

We use the generic term "concept" here to denote a product-level artifact such as a goal, an object, an operation, an agent, etc. A master concept may thus be viewed under different facets by different actors. Note that a same concept facet can be shared among different actors (see the cardinality constraints in the figure above).

For example, one might define the (master) concept of proceedings copy as follows:

Entity ProceedingsCopy
    Has Title: **String**; Year: **Date**

Two different facets of this concept might then be associated with two different actors as follows:

Entity OrderableProceedings
    FacetOf ProceedingsCopy **SeenBy** Staff
    Has Subject: **SeqOf** [Keyword]
        Cost: **Integer**
    DomInvar $\forall$ pr: OrderableProceedings
            Invoiced (pr) $\Rightarrow$ $\blacklozenge$ Ordered (pr)

and

Entity BorrowableProceedings
    FacetOf ProceedingsCopy **SeenBy** Borrower
    Has Subject: **SeqOf** [Keyword]
        Papers: **SetOf** [Paper]
    DomInvar $\forall$ pr: BorrowableProceedings, p: Paper
        p $\in$ pr.Papers $\Rightarrow$ p.Topic **in** pr.Subject

An actor's view of a concept is defined by joining the features of the master concept and of the facet seen by the actor. A feature corresponds to an attribute of the concept or a link to other concepts in the semantic net; it is defined by a name, domain of values, and a value in some cases (e.g., the assertions above are values from the FormalText domain of the DomInvar attribute). For example, the features of the BorrowableProceedings view of the Borrower actor include the declarations of Title, Year, Subject, and Papers together with the corresponding domain invariant.

All features of a master concept thus belong to all views of that concept. A master has at least one feature, namely, the concept identifier; this mechanism allows different local names (e.g., BorrowableProceedings) to be associated with a unique master name (e.g., ProceedingsCopy).

Different views of a same concept are *complementary* or *overlapping* according as the sets of feature names in their respective facets are disjoint or intersecting. In the example above, the OrderableProceedings and BorrowableProceedings views are overlapping due to common Subject and DomInvar attributes.

The following two views are overlapping as well:

Goal Achieve [CopiesEventuallyAvailable]
  FacetOf SatisfactoryAvailability **SeenBy** Staff
  FormalDef $\forall$m: Member, lib: Library, b: Book, bc: BookCopy
    Requesting (m, b) $\wedge$ Copy (bc, b) $\wedge$ InLib (bc, lib)
    $\Rightarrow \Diamond \exists$ bc: BookCpy
      Copy (bc, b) $\wedge$ InLib (bc, lib) $\wedge$ Available (bc, lib)

Goal Achieve [CopyKeptAsLongAsNeeded]
  FacetOf SatisfactoryAvailability **SeenBy** Borrower
  FormalDef $\forall$m: Member, b: Book, bc: BookCopy
    Borrowing (m, bc) $\wedge$ Copy (bc, b) $\wedge$ Needs(m, b)
    $\Rightarrow$ o Borrowing (m, bc)

As we will see, those views are in fact divergent.

Views associated with a same actor can be grouped into collections associated with the actor to form so-called *perspectives*.

The conventional *IsA* specialization mechanism [Bra85], also supported by our language, can be shown to have two weaknesses compared with our view mechanism: (i) it is impossible to associate specialized concept features with local contexts that are specific to some actor, and (ii) multiple inheritance conflicts must be resolved (e.g., by overriding of feature values in specializations) which makes it impossible to explicitly capture and record conflicts for later resolution.

3. Managing Inconsistent Views

Overlapping views elaborated by multiple actors are likely to be inconsistent. The word "inconsistency" has often been used with different meanings in the literature. We first try to bring some clarification by proposing a rough, preliminary taxonomy of possible inconsistencies among views. We will then concentrate on an interesting kind of inter-view inconsistency called divergence.

3.1 Classifying Inconsistencies

The following types of inconsistency may be distinguished.

(i) Syntactic unmatch: This type of inconsistency arises in a multi-paradigm framework when context-sensitive syntax rules are violated among views written in different languages. Consider, for example, the following rule constraining ERA and DFD views in an OMT model:

"Every entity in an ERA diagram must appear in at least one DFD diagram"

Violations of this rule result in syntactic unmatches. This type of inconsistency is the main focus of [Nus93]. In our framework, a domain-specific atomic predicate appearing in some formal assertion with no declaration of a corresponding relationship at semantic net level would correspond to a syntactic unmatch.

Resolving this type of inconsistency simply amounts to satisfying the prescribed rules sooner or later.

(ii) Type incompatibility: This type of inconsistency arises when a same feature is defined in overlapping views with incompatible domains, e.g.,

Entity OrderableProceedings
  FacetOf ProceedingsCopy **SeenBy** Staff
  Has Subject: ***SeqOf*** [Keyword]

and

Entity BorrowableProceedings
  FacetOf ProceedingsCopy **SeenBy** Borrower
  Has Subject: ***Tuple*** [Keyword]

Resolving such structure clashes can be done by application of restructuring operators [Dub87] or by addition of mapping invariants in the style advocated in [Jac95].

(iii) Divergence: This type of inconsistency arises when a same feature is defined in overlapping views with *potentially* incompatible values. The particular case where the values *are* incompatible is called ***conflict***.

One situation of special interest is when feature values are formal assertions. This corresponds to frequent cases where descriptions or requirements attached to a same concept are formulated in different, potentially incompatible ways from one view to the other. In the Staff's view of the SatisfactoryAvailability goal above, requested books must have a copy eventually available in the library; in the Borrower's view of that goal, borrowed copies should be kept as long as needed. These "interfering" views are intuitively divergent; it can however be easily shown that their formalizations are *not* logically inconsistent. To detect and resolve divergent views, we therefore need a technically more accurate characterization of what divergence is all about.

3.2 Handling Divergent Views

Assertions A1 and A2 are said to be *conflicting* in a specification S if they are logically inconsistent in S, that is, if

$$\{A1, A2\} \models_S \textbf{false}$$

Assertions A1 and A2 are said to be *divergent* in a specification S if there exists some additional, non-trivial boundary condition B that makes them logically inconsistent altogether in S, that is,

$$\{B, A1, A2\} \models_S \textbf{false}$$

with $\quad B \wedge A1 \;\; \textbf{false}, \; B \wedge A2 \;\; \textbf{false}$

For our SatisfactoryAvailability goal, a boundary condition to make the views CopiesEventuallyAvailable and CopyKeptAsLongAsNeeded logically inconsistent would be a book being requested by some member of the library with all copies of it being permanently needed by other members having borrowed them. It can indeed be shown that the three following assertions:

(B) $\exists$ m: Member, b: Book, bc: BookCopy, lib: Library
Requesting (m, b) $\wedge$ Copy (bc, b) $\wedge$ InLib (bc, lib)
$\wedge$ £ [($\forall$bc: BookCopy) (Copy (bc, b) $\wedge$ InLib (bc, lib)
$\Rightarrow$ ($\exists$ m: Member)(Borrowing (m, bc) $\wedge$ Needs (m, b))],

(A1) $\forall$m: Member, lib: Library, b: Book, bc: BookCopy
Requesting (m, b) $\wedge$ Copy (bc, b) $\wedge$ InLib (bc, lib)
$\Rightarrow$ $\Diamond$ $\exists$ bc: BookCpy
Copy (bc, b) $\wedge$ InLib (bc, lib) $\wedge$ Available (bc, lib)

(A2) $\forall$m: Member, b: Book, bc: BookCopy
Borrowing (m, bc) $\wedge$ Copy (bc, b) $\wedge$ Needs(m, b)
$\Rightarrow$ o Borrowing (m, bc)

are logically inconsistent in a specification that contains the following domain invariant:

($\exists$m: Member) Borrowing (m, bc) $\Rightarrow$ $\neg$ Available (bc, lib)

Divergence can be detected by formal derivation of a *smallest* boundary condition to make pairs of views conflicting. The principle is to start from the negation of one of the corresponding assertions, and to strengthen this negation recursively by replacing consequents in implications by antecedents from implications in the domain description whose consequent match such consequents. This principle is somewhat similar to goal regression in AI planning [Wal77].

Note that assertions need to be checked paiwise for divergence only if they appear in views of a same master concept . Views provide a structuring mechanism that prevents exhaustive divergence checking among all possible assertions.

Divergence can then be resolved in this framework by (i) weakening assertion A1, or (ii) weakening assertion A2, or (iii) *inhibiting the smallest boundary condition.* In our library example, the latter alternative for resolving the divergence between the views CopiesEventuallyAvailable and CopyKeptAsLongAsNeeded could be operationalized by keeping some copies permanently available in the library for popular books.

4. Discussion

A powerful view mechanism should be helpful at every step of the requirements lifecycle --during requirements acquisition, it should allow all relevant sources of material to be taken into account thus providing more guarantee for requirements completeness and adequacy; during requirements specification, it should facilitate the precise formulation of requirements [Jac95]; it should provide a basis for requirements assessment and negotiation; it should allow the requirements documentation to be structured by views to facilitate requirements understanding, evolution and reengineering. There seems to be no view mechanism to date that would fulfill all these expectations. It is however encouraging to see different attempts emerging.

View integration and requirements evolution often require negotiation among the owners of the different views; explicit links between views and actors should therefore be captured.

Conflicting requirements are often rooted at goal level; it is therefore important that views capture goal information as well.

The price to pay for completeness and adequacy is temporary inconsistency. Multiple views tend to be inconsistent in some sense or another, and it is important to recognize which kind of inconsistency arises before taking appropriate action. The classification proposed here is an initial attempt in that direction. Divergence is probably the most typical kind of inconsistency that may arise in multi-party requirements elaboration.

View formalization provides opportunities for formal methods to support inconsistency management and view integration. We gave some hints here on how divergent views could be detected precisely on a formal basis for subsequent resolution. These are just preliminary attempts; much work remains to be done to make formal methods effective for view management. Another interesting direction is to allow useful deductions to be carried out further in spite of inconsistency [Hun97].

The techniques suggested here should be complemented with a rich set of tactics for determining when and how to resolve divergences. Domain-independent tactics might be identified (e.g., “favour views subsuming higher priority goals”) and could be made more specific through a rich taxonomy of goals, divergences, actors, etc. Besides, domain-specific tactics could exploit knowledge about the application domain under consideration. Divergence patterns would be worth being studied in that prospect.

Acknowledgement. Some of the ideas in this paper emerged from joint work with Robert Darimont. The work reported herein was partially supported by ESPRIT Project Nr. 2537 (ICARUS) and the “Communauté Française de Belgique” (FRISCO project, Actions de Recherche Concertées Nr. 95/00-187 - Direction générale de la Recherche).

References

[Ber92] E. Bertino, “A View Mechanism for Object-Oriented Databases”, *Proc. EDBT’92 - Intl. Conf. on Extending Database Technologies,* LNCS, 1992.

[Boe95] B. Boehm, P. Bose, E. Horowitz, M.J. Lee, “Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach”, *Proc. ICSE-17 - 17th Intl. Conf. on Software Engineering*, Seattle, 1995, pp. 243-253.

[Bra85] R.J. Brachman and H.J. Levesque (eds.), *Readings in Knowledge Representation*, Morgan Kaufmann, 1985.

[Dar93] A. Dardenne, A. van Lamsweerde and S. Fickas, “Goal-Directed Requirements Acquisition”, *Science of Computer Programming*, Vol. 20, 1993, 3-50.

[Dar96] R. Darimont and A. van Lamsweerde, “Formal Refinement Patterns for Goal-Driven Requirements Elaboration”, *Proc. FSE-4 - 4th ACM SIGSOFT Symp. on the Foundations of Software Engineering*, October 1996.

[Dub87] E. Dubois, N. Levy, and J. Souquières, “Formalising Restructuring Operators in a Specification Process”, *Proc. ESEC-87 - 1st European Software Engineering Conference*, LNCS 289, September 1987, 161-171.

[Fea87] M. Feather, “Language Support for the Specification and Development of Composite Systems”, *ACM Trans. on Pro-*

gramming Languages and Systems 9(2), Apr. 87, 198-234.

[Fin89] A. Finkelstein and H. Fuks, "Multi-party Specification", *Proceedings 5th International Workshop on Software Specification and Design* (Pittsburgh, Pa.), IEEE, May 1989, 185-195.

[Hab88] A.N. Habermann, C. Krueger, B. Pierce, B. Staudt, and J. Wenn, Programming with Views, Report CMU-CS-87)177, School of Computer Science, Carnegie-Mellon Univ., 1988.

[Hun97] A. Hunter and B. Nuseibeh, "Analyzing Inconsistent Specifications", to appear in *Proc. RE'97 - 3rd Intl. Symposium on Requirements Engineering*, IEEE, 1997.

[Jac93] M. Jackson and P. Zave, "Domain Descriptions", *Proc. RE'93 - 1st Intl. IEEE Symp. on Requirements Engineering*, Jan. 1993, 56-64.

[Jac95] D . Jackson "Structuring Z Specifications with Views", *ACM Transactions on Software Engineering and Methodology* Vol. 4 No. 4, October 1995, 365-389.

[Koy92] R. Koymans, *Specifying message passing and time-critical systems with temporal logic,* LNCS 651, Springer-Verlag, 1992.

[Lam95] A. van Lamsweerde, R. Darimont and P. Massonet, "Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learned", *Proc. RE'95 - 2nd Int. Symp. on Requirements Engineering*, York, IEEE, 1995.

[Myl95] J. Mylopoulos and R. Motschnig-Pitrik, "Partitioning Information Bases with Contexts", *Proc. 3rd Intl Conf. on Cooperative Information Systems,* Vienna, May 1995.

[Nis89] C. Niskier, T. Maibaum and D. Schwabe, "A Pluralistic Knowledge-Based Approach to Software Specification", *Proc. ESEC-89 - 2nd European Software Engineering Conference*, LNCS 387, September 1989, 411-423.

[Nus93] B. Nuseibeh, J. Kramer and A. Finkelstein, "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specifications", IEEE Transactions on Software Engineering, Vol. 20 No. 10, October 1994, 760-773.

[Pot91] B. Potter, J. Sinclair and D. Till, "An Introduction to Formal Specification and Z". Prentice Hall, 1991.

[Pot94] C. Potts, K. Takahashi and A.I. Anton, "Inquiry-Based Requirements Analysis", IEEE Software, March 1994, 21-32.

[Rob89] W.N Robinson, "Integrating Multiple Specifications Using Domain Goals", *Proc. IWSSD-5 - 5th Intl. Workshop on Software Specification and Design*, IEEE, 1989, 219-225.

[Ros77] D.T. Ross and K.E. Schoman, "Structured Analysis for Requirements Definition", IEEE Transactions on Software Engineering, Vol. 3, No. 1, 1977, 6-15.

[Wal77] R. Waldinger, "Achieving Several Goals Simultaneously", in *Machine Intelligence*, Vol. 8, E. Elcock and D. Michie (Eds.), Ellis Horwood, 1977.

[Zav93] P. Zave and M. Jackson, "Conjunction as Composition", *ACM Transactions on Software Engineering and Methodology* Vol. 2 No. 4, 379-411.