International Journal of Computing Academic Research (IJCAR)
ISSN 2305-9184 Volume 2, Number 2 (April 2013), pp. 49-59
© MEACSE Publications
http://www.meacse.org/ijcar

Requirements Engineering Through Viewpoints

Mohammed Messaoudi

Faculty of Sciences, Imam University, Riyadh, Saudi Arabia

Abstract

The use of Viewpoints in requirements engineering is an emerging area of research. This paper establishes the context for viewpoint-based requirements engineering and then gives a critical review of the existing methods. A viewpoint method is seen here as a requirements engineering process of identifying viewpoints, reasoning within a viewpoint, reasoning between different viewpoints, and revising a viewpoint. The paper then highlights the common issues related to viewpoint-based requirements engineering.

Keywords: Requirements Engineering, Elicitation, Validation, Viewpoint Analysis, Viewpoint Resolution.

Introduction

In many fields, it has been found necessary to take account of many ways of looking at some subject matter. Multiplicity appears in various guises in software engineering: view integration in software development environments [1], the multi-paradigm development [2] and N-version programming [3]. Multiplicity also appears in other areas such as data base design [4], distributed artificial intelligence [5], belief systems, and distributed problem-solving. Recently many research groups have being addressing the idea of multiple viewpoints with respect to requirements engineering. Mullery [6] declares that:"...The difficulties are often compounded by failure to recognize that what is needed is not one, but several expressions of requirements. The requirements expression must recognize several views of the system. Major aims must be: separation of different viewpoints, consistency and compatibility of the information in the overlap between viewpoints, and avoidance of unnecessary repetition in producing information common to more than one viewpoint".

Viewpoint Analysis Methods

A viewpoint can be informally defined as an angle from which a domain can be observed. An analyst trying to establish requirements for a high security, intensive care unit monitoring system may get very different accounts depending on whether he talks to a doctor or to a nurse. By taking account of both views the analyst gets a better picture of the domain than by considering only one.

There are three types of viewpoints:

- The agent responsible for the viewpoint, i.e. the person observing the problem domain. The agent could be a user, an analyst, a domain expert, a designer, etc. A viewpoint method is called agent-oriented if it is based on this type of viewpoint.

- The process by which that part of the domain perceived by the agent is modeled. The process could be a set of correctness-preserving transformations, a set of elaborations (an elaboration does not have to be correctness-preserving), etc. A viewpoint method is called process-oriented if it is concerned with the properties of the viewpoint modeling process.

- The representation scheme in which an agent's perception is described. A description of the perception is called a view. A view could be a data/control flow diagram or a Z schema [7]. A viewpoint method is classified as scheme-oriented if it operates on the characteristics of the representation scheme used to describe a view.

This broad categorization corresponds to the three areas of the single viewpoint requirements engineering: requirements acquisition, specification processes, and specification languages respectively. Multiple viewpoint approaches differ from the single viewpoint approaches in their explicit capture of alternative descriptions, whether it is a requirements specification, a system model, a domain model, or a cognitive model, and their support for resolving conflicts inherent in the process. The single viewpoint approach has been criticized by several authors [8].

The methods are analyzed along the following lines:

- The identification of a viewpoint.
- The reasoning within individual viewpoints, e.g., to check the internal consistency of a view.
- The reasoning between different viewpoints, e.g., comparison of disparate views, conflict analysis.
- The revision of a viewpoint, e.g., the modification of a view to restore consistency, fitting in new information, or the creation of a new viewpoint.

Process-Oriented Methods. This class includes the works of Feather [9] and Robinson [10]. Feather and Robinson take the view that a software specification process begins with a trivially simple specification, incrementally elaborates it in a number of parallel "Hues" of design, and merges the specifications that result from each of those divergent lines to achieve the fully detailed specification. A viewpoint is a line of design. An elaboration is a transformation that deliberately changes the meaning of the specification to which it is applied. A conventional transformation generally keeps the meaning of a specification constant, i.e. a correctness-preserving transformation.

The approach to merging different views is to "replay" the evolutionary transformations of the separate lines of design in a serial order. If the parallel evolutions are completely independent then the result of merging them will be the same regardless of the serial order followed.

There are, however, cases where evolutions interfere with one another in some way. For example, one evolution renames a function F to G, while another evolution extends function F with an extra formal parameter. This interference can be resolved by applying the extension before the renaming. The merging process consists of interference detection and interference resolution.

The process of detecting interferences is based on the properties of the specification and the changes that affect those properties. The specification properties considered are limited to terminology (the set of signatures of the specification's constructs, e.g. a data structure) and usage (the use that constructs make of one another, e.g. a reference to a data structure). The detection of interferences consists of two stages:

- Determine the effects that each of the evolutionary transformations induces on each specification property. A possible change to terminology is ‘rename the parameter named *p* of construct *c* to *p*’ and a possible change to usage might be ‘add to construct *c* a use of construct *d*’.

- Make pairwise comparisons of changes within each property, and of changes between each property.

The following are examples of classes of interferences, between the terminology changes, considered together with their possible resolutions:

- Duplication: The two transformations make the same terminology change. Only one of the two transformations needs to be applied.

- Renaming interference: One transformation renames something that the other transformation refers to. A possible resolution is to apply the renaming transformation second.

- Duplication with renaming: For example, the transformations rename the same construct to different names. Only one transformation is applied.

- New name clash: The transformations introduce the same name for different purposes. For example, adding different constructs with the same name. This is resolved by replacing the new name introduced in one of the transformation with a different name.

- Remove and modify: One transformation removes a construct that the other modifies (e.g., adds a new parameter to). The interference is resolved by applying the modification transformation before the removal transformation.

- Contradiction: The transformations cannot both be performed. For example, the transformations change the type of the same parameter to different types. There is no resolution method for contradictions.

The interferences between the usage changes are classified similarly to the interferences between the terminology changes. For example:

- Duplication: The transformations duplicate the same usage change**.** Only one of the transformations needs to be applied.
- Add & Modify ordering dependency. For example, one transformation adds to c1 the use of d1 and the other removes all c2's uses of d2 while c l = c2 & d l = d2. This conflict can be resolved by applying the modification to the added use or by applying the addition only.

The incremental approach allows Feather to maintain a specification by altering elaborations (the process that we call the revision of a viewpoint) and then "replaying" them to create a new specification. Each elaboration is recorded i n terms of the changes it induces on the specification properties.

Robinson also uses the parallel elaboration approach but the starting point for the elaborations is a goal tree which stores different levels of domain goals. The attributes of a domain goal are instantiated to different perspectives. For example, in an academic library domain the proposed value attribute of the goal loan period can be set to '2 weeks' from a library staff's perspective and to '6 months' from a library user's perspective. Once a perspective is created, the specification construction process can take place in the same way as Feather's: the perspectives are operationalized by applying a sequence of elaborations to create specification components for each line of design. The goal perspectives, the resulting specifications, and the elaborations are recorded together with the links between the goal perspectives and the specification components that support them.

The integration of the resulting specifications involves the following steps:

1. *Correspondence Identification* to isolate equivalent specifications. Specifications can only be compared if there is some similarity between them. The analyst is relied on to carry out this process.

2. *Conflict Detection and Characterization*: Syntactic differences between specification components are mapped to differences of domain goal attribute evaluations from which the components were derived. This is also carried out by an intelligent agent.

3. *Conflict Resolution*: When a conflict is detected, the elaboration links are traced up to identify the domain goals from which the conflicting specifications were derived. So the resolution process concentrates on removing conflicts between domain goal perspectives. The conflict resolution method is based on the attributes utility theory. Attribute utility analysis provides a way for comparing alternatives with varying attribute values in order to pick the one that offers the maximum overall utility. Robinson uses the domain goal attributes for developing compromises.

4. *Resolution Implementation:* Changes made at the goal level, as a result of conflict resolution, should be mapped back to the specification level. Similarly to Feather's model, the elaborations are replayed with the new domain goal attributes to produce a new specification.

5. There is no evidence that the quality of the specification produced using Feather's model will be better than a specification produced by a single line of design. That is, the model does not allow for the validation of the specifications. Robinson's model improves on Feather's by incorporating domain modeling, so a specification component can be ‘justified’ in terms of the domain goals from which it originates.

Scheme-Oriented Methods. A typical example of a scheme-oriented method is the work reported in [11]. A viewpoint refers to a particular formalism that focuses on a particular aspect of a system description. For example, Data Flow Diagrams, Entity Relationship models and Petri Nets are better suited to describe functional, informational, and operational aspects of a system respectively. A viewpoint captures syntactic and semantic properties of a representation scheme. The syntactic properties are related to the correct combination of the scheme's primitive elements. Semantic properties capture the expected behaviors of the specified system.

A knowledge-based system called PRISMA is constructed to support the construction and integration of different views. Reasoning in PRISMA is based on a set of heuristics that use the properties of the schemes involved. Given a view, the following checks can be performed in the PRISMA environment:

- Agenda generation: The agenda mechanism is driven by the structuring heuristics. The structuring heuristics operate on the syntax properties of the representation used to characterize unsatisfactory situations in a view and to provide advice on how to overcome these problems.

- View Validation: The view validation is based on the validation heuristics that operate on the semantic properties of the representation scheme. The role of the view validation is to check the 'correctness' of view by paraphrasing, i.e. generate natured language descriptions of some aspects of the specification.

- Complementary Checking: The goal of Complementary checking is to ensure partial consistency of different views. Complementary checking is driven by the complementary heuristics which are pre-defined mappings, relating properties of one view to the corresponding properties of another view. For example, to each process representing a data transformation (Data-Flow Diagram) there is an associated event representing the occurrence of that transformation (Petri-net).

By using complementary heuristics the PRISMA approach suppresses conflicts, and does not, therefore, profit from using multiple viewpoints. In addition the authors avoided the 'correspondence' problem by selecting representation schemes that have correspondence. For example, it is difficult to integrate object-oriented models with data flow diagrams using PRISMA.

Finkelstein [12] considers the ‘multiple perspective problem’ in the wider context of "programming-in-the-large", an activity which involves many participants with different skills, roles, knowledge and expertise. Each participant has differing perspectives on, and about knowledge of, various aspects of software development and the application area. Further, the knowledge within each perspective may be represented in different ways and the development may be carried out concurrently by those involved using different development strategies at different stages of the development. Finkelstein uses viewpoints to partition the system specification, the development method and the formal representations used to express the system specification. A viewpoint is defined as a combination of the idea of an "actor", "knowledge source", "role" or "agent" in the development process and the idea of a "view" or "perspective" which an actor maintains. In software terms it is a loosely coupled, locally managed object which encapsulates partial knowledge about the system and domain, specified in a particular, suitable representation scheme, and partial knowledge of the process of design. Each viewpoint is composed of the following slots:

- a *representation style*, the scheme and notation by which the viewpoint expresses what it can see;
- a *domain*, which defines that part of the "world" delineated in the style;
- a *specification*, the statements expressed in the viewpoint's style describing particular domains;
- a *work plan*, describing the process by which the specification can be built;
- a *work record*, an account of the history and current state of the development.

The work plan is the most important and complex slot in a viewpoint. A work plan is composed of four 'sub-slots':

- The *assembly actions* slot which contains the actions available to the developer to build a specification;
- The *check actions* slot which contains the action available to the developer to check the consistency of the specification;
- The *viewpoint actions* slot which creates new viewpoints as development proceeds;
- The *guide action* slot which provides the developer with guidance on what to do and when.

There are two types of check actions:

- *in-viewpoint checks***,** check the consistency of the specification within the viewpoint;
- *inter-viewpoint checks***,** check the consistency of the specification with those maintained by other viewpoints. Inter-viewpoint checks are, in turn, of two types: "transfer" and "resolve" corresponding to cooperation and competition respectively.

Agent-Oriented Methods. This category includes the viewpoint analysis method proposed in [14], The CORE method [15], the Viewpoint Oriented Approach [16], the dialogue model introduced in [17], and Easterbrook's 'multiple perspectives' model [8].

Finkelstein defines a viewpoint as a participant in the dialogue responsible for maintaining a particular perspective. A perspective can correspond to the participant's role in the application domain or to an area of concern to that participant. As an agent can hold several responsibilities he can hold several viewpoints. Requirements engineering through dialogue takes the form of a game, in which moves consist of speech acts, such as assertion, question, challenge, or withdrawal, and a set of rules to maintain a 'legal' dialogue. Viewpoints are committed to anything they state and to anything stated by other viewpoints. They are responsible for the maintenance of the validity and the internal consistency of their views and through the rules defined in the dialogue model, of the other views by consulting the chains of reasoning made by other viewpoints, and by requesting information which they need to verify the conclusions.

Easterbrook's work is based on the dialogue model. Easterbrook interprets the dialogue (between an analyst and an information source) transcripts into some formal language (first order predicate calculus). The first task is to break the textual information into chunks, where each chunk focuses on a particular area of knowledge (or a topic) called a perspective. Each chunk is identified by its source, where a source could be a person or a group of people, and then interpreted into a set of propositions which act as a formal representation of the information contained in that chunk. The formal representation of a perspective is called a viewpoint. If a viewpoint becomes inconsistent it is split into consistent sub-viewpoints creating new topics, i.e., conflicting statements are placed in separate descendants of the current viewpoint. The explorative nature of viewpoint decomposition is similar to the issue-based approach (e.g., [13]). In this way, the viewpoints descriptions are built up through the addition (assertion) or removal (retraction) of statements (called commitments). The commitment reasoning scheme allows a statement to be in one of four states: Uncommitted state is the default, indicating that the item has not been discussed yet. The true and false states indicate that a person has committed himself to one or the other. The inconsistent state is used when a person has contradicted himself. Conflicts detection is based on the detection of logical inconsistencies in the first order predicate calculus scheme, although the model allows other representations to be used given that inference rules are provided.

Part of Easterbrook's model is the *computer-supported negotiation* model supported by a tool that provides clerical support and some guidance for the participants, allowing them to compare their descriptions and negotiate options for resolution. Given two descriptions, within which particular statements are known to conflict, the participants should:

- Establish correspondences between the two descriptions by comparing the statements around the conflicting ones in order to establish a context for the conflict. The result is a list of correspondences between items in the viewpoints and a list of specific disparities between items.

- Identify the conflict issues (the points to be addressed). An example of an issue is the loan period of books and the fines policy issue.

- Agree resolution criteria by which to judge the possible resolutions with reference to the participants' satisfaction.

- Generate resolution options. The model is restricted to three types of conflicts: conflicts in terminology, e.g. the same terms used for different concepts, conflicting interpretations, and conflicting designs assuming that requirements contain some design information. The conflicts are given values reflecting their degree of severity: non-interference, partially-interfering, and mutually exclusive.

- Select a resolution from the options available.

CORE [6,15,18] was developed for requirements engineering with interactive elicitation from multiple requirements sources as its primary aim. Viewpoints are seen as agents that have interests to be supported/influenced by the proposed system and act as points where information elicitation takes place ("possessors of requirements"). By virtue of its support for both projection and decomposition CORE identifies two types of viewpoints: bounding viewpoints and defining viewpoints. Bounding viewpoints are the external agents that interact with the target system (called environmental agents) whereas defining viewpoints are the functional processes that make up the target system. For a patient-monitoring system, for instance, a *hospital staff* member such as a *ward nurse*, medical staff member such as a doctor, the central station, the bed and patient may be identified as bounding viewpoints and defining viewpoints are *analysis and monitoring*. CORE assists in meeting the following objectives:

- obtaining information from viewpoints (who have only as yet only half-formed ideas about the service required from the proposed system);

- detecting and illustrating differences in perception of the required service;

- getting decisions about whose view is to prevail or aiding the development of compromises;

- achieving completeness and consistency of the specified information, where possible, and a record of each instance where it is not achieved;

- recording it in a form understandable to the viewpoints and usable for developing a formal specification of the system requirements, suitable as a contract to develop the proposed system.

CORE comprises the following steps:

1. Viewpoint identification and structuring – classification of viewpoints.

2. Information gathering - interviewing each viewpoint to identify the actions performed by that viewpoint, the actions the proposed system is required to perform for the viewpoint, and their production and consumption of data flow from other viewpoints.

3. Data structuring - construct a diagram resembling a Jackson structure diagram [19] which shows the legal sequencing of the output data flows recorded during information gathering.

4. Actions structuring (isolated) - using the actions and their interfaces from information gathering and the order of derivation of the outputs from Data Structuring, actions

structuring (isolated) consists of establishing dependencies among the actions and producing a Single- Viewpoint Model similar to a data flow diagram, for each viewpoint.

5. Actions structuring (combined) - constructs Combined-Viewpoint Models. A combined viewpoint model (or a transaction) is typically a small set of interconnected actions, from different viewpoints, which interact closely to perform some specific sub-tasks of the system in its environment.

6. Constraints analysis - once the individual viewpoints have been completely reconciled transactions are 'animated' through 'what-if enquiries to discover anything that may cause a problem leading to a break-down or failure to provide the required service within the defined constraints. For example, the analyst might ask: If iteration is involved, could there be convergence problems or error build-up?

These activities are driven by a set of heuristics that are hints about checks that should be performed at each step. These heuristics can be seen as a special case of the heuristics built into PRISMA when using a data flow-based representation only and employing animation [18] instead of paraphrasing for specification validation.

In viewpoint analysis**,** proposed in [14], multiple analysts (viewpoints) describe their understanding of a problem in the same universe of discourse in the same language VWPL (VieWPoint Language) using a common vocabulary. Each analyst constructs his/her view using three perspectives corresponding to the modeling aspects: data modeling, process modeling, and actor modeling.

Actor modeling is related to the agents responsible for the processes. To attach some semantics to the information encoded in the viewpoint language, viewpoints use two hierarchies: the "is-a" hierarchy to represent specialization relationships between keywords; and the "parts-of hierarchy to represent decomposition relationships. Leite contends that the heavy use of redundancy will improve the chances of detecting problems related to consistency and completeness. Each viewpoint then integrates the perspectives into a view, resolving the internal conflicts. Once the views are completed they are compared, producing a list of 'discrepancies' that acts as part of an 'agenda' for negotiating resolutions to conflicts between viewpoints.

Kotonya and Sommerville [21] proposed a Viewpoint-based Object-oriented Approach to requirements analysis (VOA) in which a viewpoint is seen as an *external entity that interacts with the system being analyzed, but one that can exist without the presence of the system.* VOA is two layered: the viewpoint layer, concerned with the behavior of the environment of the proposed system, and the system layer, concerned with the system's responses to its environment. VOA includes four main stages:

- viewpoint identification;
- viewpoint structuring and decomposition;
- information collection;
- reconciliation of information across viewpoints;

There are close similarities between VOA and CORE. They share:

- Classification of the external entities: direct and indirect viewpoints (CORE), and active and passive viewpoints (VOA).

- Establishment of a viewpoint structure but use different structuring schemes. CORE employs functional decomposition (role/sub-role) whereas VOA provides for inheritance

(abstraction/specialization)

- Explicit capture of the interactions between entities in the environment and the target system, in terms of the services the system is required to provide and the constraints under which the services should be provided.

The two approaches differ in the following:

- CORE's viewpoint structure is a mixture of external and internal viewpoints with the top level defined as 'system + environment' while VOA treats them separately;

- CORE captures the interactions between the external viewpoints for a fuller model of the environment;

- VOA distributes the non-functional constraints across the viewpoint structure and reconciles them across the viewpoints while CORE treats them in a separate activity (constraints analysis) after the full viewpoints have been integrated;

- CORE provides heuristics for detecting structural inconsistencies. VOA does not provide a firm mechanism for 'information reconciliation across viewpoints'.

Discussion

A viewpoint method has been identified as agent-oriented, scheme-oriented, or process-oriented depending on the type of viewpoint it adopts (agent, process, or formalism). The following issues are addressed by viewpoint methods:

Viewpoint identification. There is a high number of angles from which a domain can be observed. A viewpoint method should have clear criteria for distinguishing viewpoints, i.e., an unambiguous definition of a viewpoint. Some methods fix a set of pre-defined viewpoints (e.g., [14], [6]) while others start with a set of viewpoints then identify others during the analysis process through decomposition, refinements, etc. (e.g., Easterbrook).

View Modeling. Once a relevant viewpoint has been identified it can be applied to the domain under analysis to produce a description of the domain from that viewpoint, i.e., a view. A method can either model views independently (competitive) or derive one from another (cooperative).

Comparing Disparate Views. Some methods compare the views in parallel with the view modeling process, others compare them only when they are 'final' (most of the methods). The result is a list of 'discrepancies'. Comparison makes sense only when the viewpoints correspond (have something in common). Establishing correspondence is a tough problem. Leite, for example, imposes several restrictions: viewpoints should consider the same topic, use a common vocabulary, and use the same language to constrain how the facts should be expressed. Easterbrook assumes that viewpoints will not be wholly unfamiliar with other viewpoints' knowledge, so that they will be able to suggest correspondences between their views.

Conflicts Characterization. The discrepancies resulting from the comparison are, often, syntactical differences, structural differences, or differences of terminology (viewpoints use different terminology to describe the same thing); Conflicts characterization establishes an agenda to be used as input to the negotiation. Part of the negotiation process is to distinguish between real and apparent conflicts by exploring the context of the differences and gathering more information necessary to identify misunderstandings, differences in terminology, etc. [8].

Conflicts resolution. Once the different options about an issue have been identified a conflict resolution process is launched. Easterbrook and Robinson have attempted to address the conflict resolution problem. They both employ an iterative strategy of generation followed by evaluation. Sycara [20] uses this approach in her model of an automated labor mediator. Options for a resolution are suggested then evaluated against the satisfaction of the participants. The process is repeated until reconciliation is achieved.

In a study of conflict behavior in the requirements engineering phase of large systems development [16], it has been shown that conflicts or interpersonal disagreements increase the quality of group decision making by stimulating critical thinking, increasing group involvement, and widening the search for alternatives. The conflicts resolution process has been characterized by [20,22]:

1. The iterative nature: Negotiators often employ an iterative strategy of generation followed by evaluation.

2. The participative nature: all the viewpoints should be involved in the reconciliation process.

3. The learning process involved. A participative framework intends to encourage a pooling of knowledge and insight, and the decision-makers become engaged in a process of learning and understanding.

4. The amount of information to be handled in order to make a reasonable decision

References

[1] Meyers, S. : Difficulties in Integrating Multiview Development Systems, IEEE Software, Vol. 8, No. 1, 1991, pp. 49-57.
[2] Zave, P. : A Compositional Approach to Multi-Paradigm Programming, IEEE Software, Vol. 6, No. 5, 1989, pp. 15-25.
[3] Chen, L. and Avizienis, A. : N-version Programming: a Fault-tolerance Approach to Reliability of Software Operation**,** in 8th Ann. Int. Conf. on Fault Tolerance Computing, Toulouse, France, 1978, pp. 3-9.
[4] Batini, C, Cenzerini, M., and Navathe, S.B. : A Comparative Analysis of Methodologies for Database Scheme Integration, ACM Computing Surveys, Vol. 18, No 4, 1986.
[5] Smith and Davis : Frameworks for Cooperation in Distributed Problem Solving, IEEE Trans. Systems, Man & Cybernetics, Vol. 11, No. 1, 1981, pp. 61-69.
[6] Mullery, G. : CORE - A method for Controlled Requirements Expression, *Proc.* of Fourth IEEE Int. Conf. on Soft. Eng., Munich, Germany, 1979.
[7] Ainsworth, M., Cruikshank, A.H., WaUis, P.J.L., and Groves, L.J. : Viewpoint Specifications in Z, Information and Software Technology, Vol. 36, No. 1, pp. 43-51, 1994.
[8] Easterbrook, S. : Elicitation of Requirements from Multiple Perspectives, Ph.D. thesis. Department of Computing, Imperial College of Science, Technology, & Medicine, University of London, 1991.
[9] Feather, M.S. : Detecting Interference when Merging Specification Evolution, ACM Sigsoft, Software Engineering Notes, Vol 14, No 3, 1989, pp. 169-176.
[10]Robinson, W.N. : Integrating Multiple Specifications using Domain Goals, ACM SIGSOFT Engineering Notes, Volume 14, Number 3, 1989, pp. 219-226.
[11]Niskier, C , Maibaum, T., and Schawbe, D. : A Look Through PRISMA: Towards Pluralistic Knowledge-based Environments for Software Specification Acquisition, ACM Sigsoft, Software Engineering Notes, Vol. 14, No 3, 1989, pp. 128-136.

[12]Finkelstein, A., Kramer, J., and Goedicke, M . : Viewpoint Oriented Software Development, 3rd International Workshop on Software Engineering and its Applications, Toulouse, France, (IEEE Computer Society), 1990, pp. 337-351.

[13]Wing, L. : A n issue-Based Framework for Requirements Elicitation, Description and Validation**,** Technical report No. 92/02, Department of Computing, King's college London (University of London), 1992.

[14]Leite, J. and Freeman, P. : Requirements Validation Through Viewpoint Resolution, IEEE transactions on Software Engineering Vol. 17, No. 12, 1991..

[15]Mullery, P. : Acquisition-Environment**,** in Distributed systems: Methods and tools for *specification,* Spriner-Verlag, 1985.

[16]Kotonya, G. and Sommerville, I . : Viewpoints for Requirements Definition, Software Engineering Journal, 1992, pp. 375-387.

[17]Finkelstein, A. and Fuks, H. : Multi-party Specification**,** ACM SIGSOFT Engineering Notes, Volume 14, Number 3, 1989, pp. 185-195.

[18]Kramer, J., Chinnick, S., and Finkelstein, A. : TARA: Tool Assisted Requirements Analysis, Final Report September 1987, Research Report 87/18, Imperial College, London and Systems Designers pic, Camberley, Surrey, U.K.

[19]Jackson, M.A. : Jackson System Development, Academic Press, 1975.

[20]Sycara, K. : Resolving Adversarial Conflicts: an Approach Integrating Ceisaebased and Analytic Methods**.** Ph.D. thesis, Georgia Institute of Technology, 1987.

[21]Elam, J.J., Diane B. Walz, D.B., Krasner, H., and Curtis, B. : A Methodology for Studying Software Design Teams: An Investigation of Conflict Behaviors in the Requirements definition Phase, 2nd Workshop on Empirical Studies on Programmers, 1987, pp. 83-99.

[22]Efstathiou, J.H.: A Practical Development of Multi-attribute Decision Making using Fuzzy Set Theory**,** Ph.D. thesis. Department of Computing, University of Durham, 1979.