Requirements Validation Through Viewpoint Resolution

Julio Cesar Sampaio do Prado Leite, *Member, IEEE*, and Peter A. Freeman, *Senior Member, IEEE*

Abstract—**Requirements modeling depends on having available the knowledge of what should be modeled. Acquiring, or eliciting, this knowledge is recognized to be a hard problem, and different suggestions have been made to address it. In this paper we examine one such approach—viewpoint resolution. It is based on the fact that software requirements can and should be elicited from different viewpoints, and that examination of the differences resulting from them can be used as a way of assisting in the early validation of requirements. Our research proposes a language for expressing views from different viewpoints and a set of analogy heuristics to perform a syntacticly oriented analysis of views. This analysis of views is capable of differentiating between missing information and conflicting information, thus providing support for viewpoint resolution. We also present empirical evidence of the effectiveness of these mechanisms.**

Index Terms—**Requirements engineering, requirements analysis, validation, elicitation, viewpoint resolution, viewpoint analysis.**

I. Introduction

THIS research proposes a specific technique—viewpoint resolution—as a means of providing an early validation of the requirements for a complex system, and provides some initial empirical evidence of the effectiveness of a semi-automated implementation of the technique. While very preliminary, our research indicates that such a technique can indeed provide much-needed rigor and assistance to what is now an almost completely informal part of the critical front-end process of system definition.

While the concept of resolving conflicts between differing views of a system is not new, to our knowledge it has not been explored or utilized as a specific technique in requirements engineering. It not only appears to provide a specific requirements engineering technique, but more generally provides strength and assistance to the overall process of eliciting requirements from the environment in which a proposed system is to exist. It is this somewhat broader issue of requirements elicitation that provides the research context for the work reported here.

Manuscript received February 7, 1990; July 23, 1991. Recommended by P. A. Ng. This research, largely performed at the University of California, Irvine, is based on [18], and was supported in part by the U.S. National Science Foundation through Grant 85-21398, by CNPq (Brazil), and by the Virginia Center for Innovative Technology (through George Mason University).

J. C. S. P. Leite is with the Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Brazil.

P. A. Freeman is with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280.

IEEE Log Number 9104160.

As we will delineate in more detail below, this research must be understood as addressing only a very small portion of the overall process of establishing and recording the requirements for a complex system. Further, it is important to keep in perspective that our underlying research objective is to provide objective evidence of the value of a particular approach to viewpoint resolution (building rule-based models of views which are amenable to automated heuristic analysis), not to propose a final packaged approach to solving the entire problem of defining systems.

This paper is organized as follows. The next section introduces the context of requirements validation. In Section III is given the overall definition of viewpoint resolution. Section IV presents the proposed method for viewpoint analysis. Section V presents the language, VWPl. The description of the heuristics used in the automated analysis of viewpoints is the topic of Section VI. Section VII reports on some cases studies performed with viewpoint resolution. Section VIII deals with related work and future research, and we present some concluding remarks in Section IX.

II. The Context of Requirements Elicitation

Requirements engineering is a process which produces the *requirements*[1] for a software system. This process does not happen in a vacuum, since it depends on a previous process, the systems engineering process which defines the context and goals of the software artifact. Patient monitoring systems, navigation systems for airplanes, and medical information systems are examples of systems defined by a systems engineering process in which software will be one of the components. We call the context set by the systems engineering process the *universe of discourse*. Software requirements engineering takes place in the universe of discourse.

It is a fallacy in most cases to believe that the systems engineers will produce complete software requirements. The result of the requirements process at the system level is an overall definition of the software necessary for that specific system. It is mandatory that the software engineers **elicit** the details of the goals set for the software. Before building something we need to understand **what** is to be done. We call this fact the **necessity of elicitation**. Requirements elicitation is starting to attract attention of researchers [5].

Considering that requirements engineering happens in a universe of discourse and that elicitation is a necessity, we

[1] The requirements include both functional and nonfunctional requirements and is usually packaged as a software requirements specification

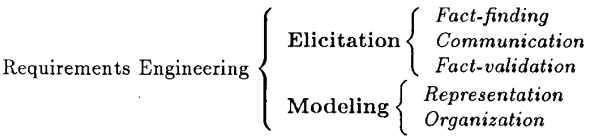

Fig. 1. Parts of the requirements engineering process.

have been working with the following characterization: requirements engineering is a process in which "what is to be done" is elicited, modeled, and validated. This process has to consider different viewpoints, and it uses a combination of methods, tools, and actors. The product of this process is a model, from which a document, the requirements, can be produced.

The Process of Requirements Validation

The whole process of requirements engineering is a web of subprocesses, and it is very difficult to make a clear distinction between them. Nonetheless, our characterization differentiates between two major disciplines. One is related to the task of eliciting or acquiring facts, and the other is modeling such findings. In between these two tasks there is the task of validation, which is embodied in the subprocesses of communication and fact validation (Fig. 1).

In the SADT [25] diagram pictured in Fig. 2, the "model" is the data involved in the feedback (validation) between the activities ELICIT FACTS and BUILD MODEL. It is important to notice that the diagram will iterate until there is a consensus between clients and developers.[2] The "model" in the SADT (Fig. 2) should be understood in a very broad sense. It can be composed of different sets of representations [5], [20] and be organized [10], [8] in different ways. We assume that at the end of this interaction a pretty-printed form of this "model," called the requirements or software requirements specification, can be produced. Requirements modeling is not dealt with in this paper.

Our objective is to work on the validation that occurs during the process of elicitation. Thus we have centered our attention on what we call *very early validation*. As we show in the SADT diagram, the validation loop of the requirements engineering process involves the activities ELICIT FACTS and BUILD MODEL; work, however, centers on the validation loop which occurs inside the activity ELICIT FACTS. In order to provide this very early validation scheme we propose a technique called viewpoint resolution. Viewpoint resolution compares different views of a given situation, and partially supports the negotiation process necessary to reconcile different opinions. Our empirical experiments indicate that viewpoint resolution can improve the understanding of the requirements.

It is important to observe that as we tackle new and more complex problems, it is harder to have a complete up-front understanding. New software development processes, like the one proposed by Boehm [1], are well aware of this. Our elicitation proposal, however, is not tied to any specific software development process. What we stress is that viewpoint resolution helps the elicitation of requirements by providing a very early validation inside the elicitation process. Viewpoint resolution could be applied in any situation where there is a need for better understanding of a given problem. In terms of requirements engineering, viewpoint resolution happens in the elicitation process, and its use will depend on the software development process being used in a given situation.

[2] Client identification [2] is a process bounded by the universe of discourse. Clients and developers may represent different sets of actors in the universe of discourse.

III. Viewpoint Resolution

Before presenting our definition of viewpoint resolution, we present the rationale for using viewpoints and describe how software engineering has been using them. Following the definition of viewpoint resolution, we describe its main problems and establish the terminology used in this paper.

A. Why Viewpoints?

The principle that more sources of information provide a better understanding of a subject has been used for centuries, in court investigation, for example. Different witnesses may have conflicting or complementary recollections. By using this principle in the process of elicitation, the chances of mastering correctness and completeness problems will be greater. To profit effectively from this principle, however, it is necessary to compare and analyze different views in a systematic manner.

In the task of modeling the users' expectations in the universe of discourse, a systems analyst usually encounters different opinions about the problem being addressed. Different systems analysts, when modeling the users' expectation in the **same** universe of discourse, may also produce different models. The same systems analyst, when modeling the same universe of discourse, may do so by using different perspectives (e.g., a data model or a process model).

All this is common knowledge. Some software engineering methods like SADT and CORE use viewpoints with the objective of producing a model that is closer to the reality being modeled, and it is this principle that we are using.

B. The Research Problem

The analysis and comparison of viewpoints as proposed by Ross' SADT and Mullery's CORE [19] are informal processes. They are similar to what we call "using informal checking" (see Section VIII-A) and rely heavily on being performed by a *good* systems analyst. Although CORE and SADT advocate the use of viewpoints, neither one provides much guidance on how to profit from doing so. Other than a reliance on inspection procedures and some general guidelines, no process is presented for the use of viewpoints. The lack of a proper representation for viewpoints in either method makes it hard to have a procedure for comparing and analyzing views resulting from different viewpoints.

Our main goal is to help the elicitation process. Viewpoint resolution does it by comparing different views and providing a scheme for negotiation of conflicts. In order to compare the views, we faced a basic research problem—that is, how to have

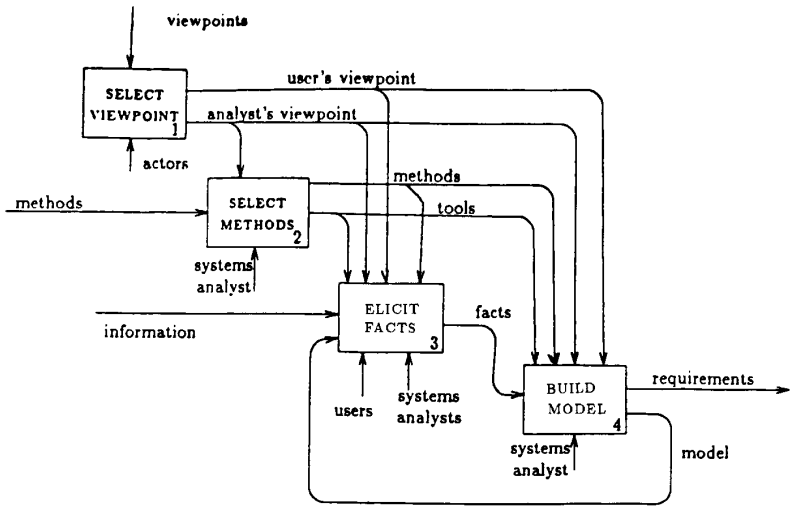

Fig. 2. A0 requirements engineering.

a representation that provides the basis for the application of a viewpoint resolution process. On the other hand, our major constraint was to provide a simple representation which would make the encoding process easy and quick. Our representation model is very simple, but sufficient to provide the basis for automatic analysis of different views. It is important to stress that we are not proposing a requirements modeling representation. Our representation is to be seen as a scratch paper. The main result of using viewpoint resolution is not the representation as proposed here, but the unfolding of knowledge that must afterwards be properly modeled.

C. Definitions and Terminology

Viewpoint resolution is a means for very early validation in the process of requirements elicitation. It assumes a particular definition of requirements engineering (Fig. 1): *viewpoint resolution* is a process which identifies discrepancies between two different viewpoints, classifies and evaluates those discrepancies, and integrates the alternative solutions into a single representation (Fig. 2).

By examining the model presented in Fig. 2 and the definition of requirements engineering (Fig. 1), it is easy to note that there are some viewpoint resolution tasks belonging to *fact-validation*, and others belonging to *communication*. The tasks related to fact-validation are called *viewpoint analysis*, and the tasks related to communication are called *viewpoint reconciliation.* Our work does not deal in detail with viewpoint reconciliation. We have focused on the problems of identification of discrepancies and the classification of these discrepancies, thus providing an *agenda* upon which the evaluation and integration of viewpoints can be based.

The terms and their definitions used hereafter in the paper are as follows.

The Universe of Discourse: The universe of discourse is the overall context in which the software will be developed. The universe of discourse includes all the sources of information and all the people related to the software. These people are referred to as the **actors** in this universe of discourse. It is the reality trimmed by the set of objectives established by those demanding a software solution.

Actors: Actors are the different people involved in the universe of discourse. Basically, we could divide actors into users on the demand side and software engineers on the supply side. Although that division does exist, it should not be forgotten that users are composed of a different set of persons: managers, buyers, operators, information suppliers, information users, and others. On the software engineering side, there are also differences: different levels of management, consultants, and different skill levels among analysts, among others.

Viewpoint: A viewpoint is a standing or mental position used by an individual when examining or observing a universe of discourse. It is identified by an individual, e.g., his name, and his role in the universe of discourse, e.g., a systems analyst, programmer, or manager.

Perspective: A perspective is a set of facts observed and modeled according to a particular modeling aspect and a viewpoint. An example of such a particular modeling aspect is what is known as "data modeling." In our method we use three modeling aspects: the data perspective, the actor perspective, and the process perspective. For example, the SADT technique has two different types of models: actigrams and datagrams. They impose two different modeling aspects—one in which the decomposition is centered on happenings, and another which is centered on things. We call these "perspectives." As such, an actigram has the happenings perspective independent of the chosen viewpoint.

View: A view is an integration of perspectives. This is achieved by a view-construction process.

Hierarchies: The is-a hierarchy of concepts in the universe of discourse and the parts-of hierarchy of concepts in the universe of discourse are used.

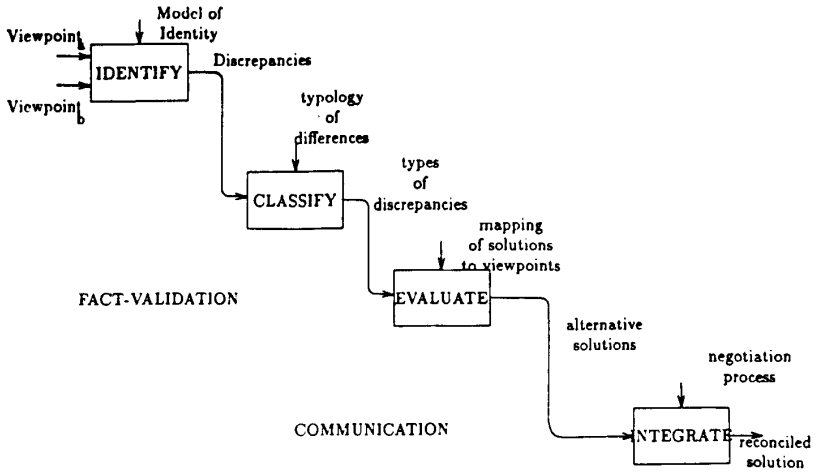

Fig. 3. Viewpoint resolution.

The proposed strategy for viewpoint analysis comprises procedures to formalize viewpoints (method), procedures to analyze the formalized viewpoints (static analyzer), and a special language, VWPl, to represent viewpoints. The language provides the representation which registers the formalism and makes possible its analysis.

The language is derived from PRISM [15], a production system architecture. As such, our viewpoint analysis strategy is basically a process for finding discrepancies between two rule bases, each one representing a different view or perspective according to a viewpoint.

The task of identifying which viewpoints to consider is one that depends on the identification of which actors are in the universe of discourse. Identifying actors should be one of the steps in defining the universe of discourse. Burstin [2] proposes a set of heuristics for composing what is called an *abstract user tree*. This tree will give an idea of all the abstract users, which we call actors, in the universe of discourse, as well as their levels of abstraction. Once actors are identified, some policy may be used to choose which ones are to be considered in a viewpoint resolution scheme.

The process of validating facts is dependent, of course, on the process of finding facts. It is assumed that the facts[3] are available before the application of the viewpoint resolution strategy.

In the following sections we present the overall method behind producing a viewpoint, the description of the VWPl, and an overall description of the procedures that analyze different viewpoints.

IV. Method

In Fig. 3 there is an overall description of our viewpoint resolution strategy. As illustration (Fig. 4), suppose John and Mary, both systems analysts, perform the task of modeling users' intentions. They both use the VWPl language to express their perception of the universe of discourse. They use different perspectives (process, data, and actor) and different hierarchies (is-a, and parts-of) to improve their own view. Once a series of critiques is provided, each analyst alone solves the internal conflicts and integrates their final perception into a view. This final view is expressed in the process perspective together with the hierarchies. After that, both viewpoints are compared and analyzed.

Thus in order to identify and classify discrepancies between different viewpoints, views are to be taken from the viewpoints. Views are produced by a process called "view construction." The construction of a view is based on the following assumptions:

- a set of viewpoints is identified
- a fact-finding method is defined
- a viewpoint holder with a specific role in the universe of discourse[4] uses different perspectives and hierarchies in modeling his viewpoints
- perspectives and hierarchies can be analyzed by a static analyzer
- a view is a model integrating the different perspectives and hierarchies taken from the same viewpoint.

After two views are available, it is possible to compare different viewpoints.

As noted above, it is common knowledge that systems analysts, when modeling the universe of discourse, may do so by using different perspectives. An example is the usual modeling of data and processes. This work, in addition to data and process modeling, uses actor modeling [18]. The idea behind actor modeling is to model using the perspective of those who are responsible for the processes; i.e., human agents or devices. The objective of using hierarchies is to try to attach some semantics to the information encoded in the viewpoint

[3] A fact is a relationship between entities and attributes.

[4] The use, in the examples, of systems analysts as viewpoint holders is for the sake of simplicity. Once viewpoint holders are identified, a system analyst will assume the actor role and construct a view reflecting the actor's viewpoint.

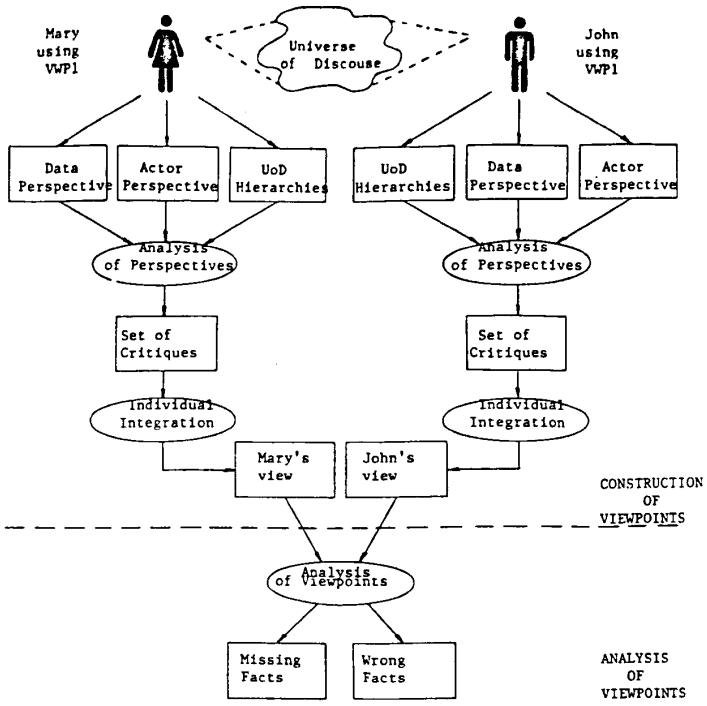

Fig. 4. The proposed strategy.

language. A specialization relationship between keywords is established, as well as a decomposition relationship.

In order to construct a view, a systems analyst describes the problem using the three perspectives and two hierarchies. The perspectives are the actor perspective, the data perspective, and the process perspective. The hierarchies are the is-a hierarchy and the parts-of hierarchy. The perspectives and hierarchies are compared, and the list of discrepancies and types of discrepancies are produced. A view is the integration of perspectives and hierarchies[5] achieved by the viewpoint holder with the help of an agenda, which is produced by the analysis of perspectives. When no more feedback is available from the analysis of perspectives, then an integration can occur.

The construction of a perspective is a process in which there is the assumption that the systems analyst uses the concept of *application vocabulary* [8], such that the keywords of the universe of discourse are carried into the viewpoint representation. The basic idea is that a systems analyst first analyzes a problem and writes down his findings using the VWPl actor perspective. Sometime later, without looking at the result of the actor perspective, the analyst does the same, using the data perspective and the process perspective. The same approach is used in acquiring the hierarchies. It is assumed that the comparison of those perspectives and hierarchies held by the same systems analyst will provide him or her with clues to produce a better process perspective and hierarchies which are then considered to be the representation of his or her viewpoint about the problem.

The general guidelines for acquiring hierarchies and perspectives are as follows. Provide is-a and parts-of hierarchies for the concepts presented in the universe of discourse. For each perspective:

- find the facts
- express the facts using keywords of the application domain
- classify the facts into object facts, action facts, and agent facts
- define the functionality of facts by coding them as productions.

Objects represent both the *objects* and *states* of these objects. That is, objects are the facts which could not be classified as actions or agents. There are no constraints on providing the hierarchies; an item is chosen to be in the hierarchy by the sole judgement of the viewpoint holder. The guidelines are purposely loose to make it simpler for the viewpoint holder to express his views.

Although it is not our objective to deal with requirements modeling [10], we have used a simple and shallow conceptual model in order to support our strategy. This conceptual model is composed of:

- three kinds of entities: objects, agents, and actions
- attributes
- hierarchies: is-a and parts-of

[5] It is important to note that in the construction of a view, there is no need for the tasks related to the communication process (see Fig. 2). Since the discrepancies in view construction are related to a single person, the systems analyst, the mapping of solution to viewpoints and the negotiation are not a problem, since they only depend on one individual.

- relations.

The relations are of two types:

- ⟨entity, attribute⟩, which constitutes facts, and
- ⟨facts, facts⟩, which defines functionality.

The conceptual model is instantiated by the viewpoint language, which is described below.

V. The Viewpoint Language, VWPL

A language was created for representing viewpoints, and its syntax and semantics were defined. This language is derived from PRISM [15], a production system architecture. As such, our viewpoint-resolution strategy is basically a process for finding discrepancies between two rule bases, each one representing a different perspective or viewpoint. This representation's main objective is to register early results of the fact-elicitation process in the requirements elicitation effort; it is not intended to be a requirements language, so that its usefulness is restricted to the fact-validation process. A complete description of VWPl is presented in the appendix.

Our research has been exploring the use of a rule-based language as a way of expressing functionality and constraints for the process of elicitation in requirements engineering. Our earlier conceptual development [18] and the empirical evidence available from our work, together with references from the literature [14], allow us to hypothesize that expressing functionality and functionality related constraints in production rules is simple and fairly easy to do.[6]

In a production system scheme, in order to express the functionality and show the behavior of the production rules, it is necessary to fill the production memory with rules, the working memory with facts, and to choose an adequate control strategy. A production memory is the database for rules, and the working memory is the global database of a production system, where the facts are kept and changed.

The overall idea of VWPl is to have a predefined structure for the construction of rules. The imposition of a further constraint on the usual scheme—of the left-hand side being conditions and the right-hand side being actions—makes it possible that some static semantic information is made available by the rule structure itself. The approach used in VWPl is similar to the one used in *case grammars* [22], where the structures produced by the grammar rules correspond to semantics relations rather than to strictly syntactic ones.

In the case of VWPl, the semantics relations are achieved by using what we call *types* and *classes* constraints. Types are the different sorts of facts used; that is, the object facts, the action facts, and the agent facts. Classes are the different roles each fact may have in a rule. In a rule, a fact can:

- be deleted from working memory (we call that the *input class*)
- be added to the working memory (the *output class*)
- remain in working memory (the *invariant class*).

A *fact* is a relationship between entities and attributes. The entity keywords used for expressing facts in VWPl are checked against a type list before being parsed. In other words, a set membership semantic check is performed on the keywords provided for describing the view.

The general structure of the rules is described by the VWPl grammar (see appendix). For each perspective there is a special combination of types and classes. In this paper we only use the process perspective, for which the rule structure is as follows:

- LHS—In this side of the rule there are **input** facts and **invariant** facts. **Input** facts may be of type action and of type object. **Invariant** facts can be of type agent and/or of type object
- RHS—In this side of the rule the **output** fact is of type object.

A fact is composed of a *fact-entity-keyword* and *fact-attribute*. An example is "(book =book-id =author =title)." In this case we have the fact-entity-keyword "book" and the attributes: "book-id," "author," and "title." By using a nondeterministic control, it is possible to use the process perspective as an early executable expression of the requirements [18].

Hierarchies are encoded as lists. The lists are organized by the kind of hierarchy (is-a, or parts-of) and the type of the facts. For each kind and type, the root in the hierarchy is the head of a list, followed by the leaves of that hierarchy.

In Table I we give an example of VWPl usage. The source of information for this example[7] is taken from the IWSSD[8] library problem.

VI. The Static Analyzer

This section details the static analyzer. The job of the static analyzer is to compare two different perspectives or two different views, and to provide an agenda for viewpoint resolution.

A. Structure of the Static Analyzer

The analysis of different perspectives and different views is achieved by a set of procedures which performs an analysis of two sets of perspectives or views. The perspectives and/or views are represented using VWPl. Because VWPl is a rule-based representation, the static analysis is really performed between two rule sets (Fig. 5).

Comparing two rule sets only makes sense when there is enough similarity between them. In our case, there is a set of factors that leads to that similarity, including:

- the fact that viewpoint holders are viewing the same universe of discourse
- the use of a method which stresses the importance of maintaining the concept of application vocabulary when modeling a view or a perspective
- the use of a special language which constrains how the rules are expressed.

[6] It takes no more than 2 h of training to have people, with computer science background, become familiar with the language and ready to encode their viewpoint. More than a dozen people have already used the language, and our experience is that the 2 h training is sufficient.

[7] The objective of the example is simply to give an idea of the representation.

[8] International Workshop on Software Specification and Design.

TABLE I

```
universe of discourse:
"Check out a copy of a book. This transaction is restricted to
staff users. There are two types of users; staff users and ordinary
borrowers. The following constraints have to be met: 1) all copies in
the library must be available for checkout or be checked out, 2) no
copy of the book may be both available and checked out at the same
time, 3) a borrower may not have more that a predefined number of
books checked out at one time."

Process Perspective:
(1 ((staff-users =staff-id) (book =book-id =author =title)
    (available-for-checkout =book-id =copy-n)
    (ordinary-borrowers =ordinary-borrower-id)
    (number-of-books-checked-out =ordinary-borrower-id =number)
    (not-have-more-predefined-number =predefined-n =ordinary-borrower-id))
   (($delete-from wm (check-out =book-id =copy-n =ordinary-borrower-id))
    ($add-to wm (checked-out =book-id =copy-n =ordinary-borrower-id))))

Hierarchies:
(is-a (agent (user staff-users ordinary-borrowers))
    (object (book available-for-check-out checked-out)))

(parts-of (agent (staff-users staff-id)
            (ordinary-borrowers ordinary-borrower-id))
        (object (book book-id author title)
            (number-of-books-checked-out ordinary-borrower-id number)))
```

The static analyzer as proposed and implemented has two major tasks: finding which rules are similar to each other, and, once rules are paired, identifying and classifying the discrepancies between them. Rules that are not paired are classified as missing information. The pairing of rules and their further analysis are basically syntacticly oriented.

Being based on the syntactic representation of terms, the analyzer depends on pattern matchers and partial matchers. Those matching procedures, which are applied between facts of the two different rule sets, have different scoring algorithms, depending on the semantic information available from the *types* and *classes* of each fact.

The classification of discrepancies, that is, determining which are the missing information discrepancies and which are the wrong information discrepancies, is done based on scores resulting from the matchers, and on the information available in the hierarchies. While designing the static analyzer we borrowed several ideas from the work on analogy in Artificial Intelligence and used a descriptive framework described by Hall [11].

Hall uses the following descriptive framework to examine computational approaches to analogy:

- recognition of an analogous source
- elaboration of an analogical mapping between source and target
- evaluation of the elaborated analogy
- consolidation of information generated while using the analogy.

Mapping the two rule sets using the analogy framework is a problem in which the processes of recognition, elaboration, and evaluation are subproblems. Since our static analysis does not address the problem of keeping information gathered from the mapping for future use, the consolidation process is not a subproblem.

The recognition problem is: **given** two sets of production rules in the viewpoint language, **produce** the most alike (most similar) rule pairs.

The elaboration problem is: **given** alike rule pairs produced by a recognition strategy, **identify** the most probable rule pairs, as well as the rules with no pair.

The evaluation problem is: **given** the most probable analogies between the rule bases as determined by an elaboration strategy, **identify** the wrong information, the missing information, and the inconsistencies for each pair.

In the process of building a mapping between the two rule sets, the solution of each subproblem narrows down the search space. By narrowing it down, the most probable rule pairs are identified, such that a close scrutiny is possible. For each pair, facts can be compared in order to find discrepancies.

The following are the types of discrepancies pointed out by the analyzer:

- Wrong Information
  - contradiction between the facts of the different rule sets
- Missing Information

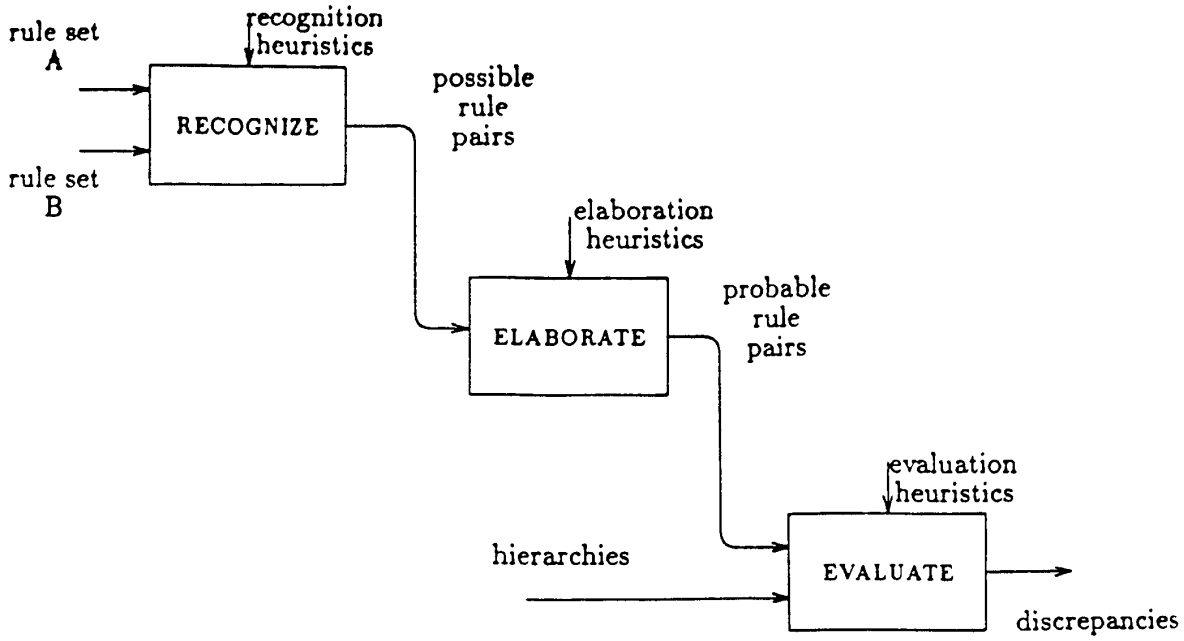

Fig. 5. The static analyzer heuristics.

- incomplete hierarchies with respect to rule facts
- missing rules
- missing facts

- Inconsistency
  - contradiction between a fact and the hierarchy
  - redundancy in the same rule set.[9]

The static analyzer was implemented as a series of Scheme functions. Scheme is a lexically scoped Lisp developed at MIT by Sussman and Steele [27].

B. An Example of the Static Analyzer

Suppose we have two actions dealing with ships:

The sailor has to take care of bringing up the sails, which may be of different types according to the class of the ship. Ships are recognized by a registration number.

The officer of a vessel has, as one of his responsibilities, the task of cleaning the vessel deck, and he or she may use different types of brushes.

A possible encoding of these descriptions in VWPI by two different viewpoint holders is presented in Table II. It is assumed that view A is produced by systems analyst A with one viewpoint, and that view B is produced by systems analyst B with a different viewpoint.

Given the two views in Table II, the static analyzer produced the following set of messages:

- A's rule 15 is missing the object "sail-ok" with respect to B's rule 25
- A's rule 10 is missing the object "brush" with respect to B's rule 20
- The attributes of A's "officer" and B's "officer" do not correspond
- The attributes of A's "deck-clean" and B's "deck-cleaned" do not correspond
- The attributes of A's "clean-the-deck-with-brush" do not correspond.

This example gives an idea as to what sorts of discrepancies are discovered by the static analysis. It is proper to note that the keywords "deck-clean" and "deck-cleaned" are understood as the same by the static analyzer, as well as the keywords "clean-the-deck" and "clean-the-deck-with-brush." The reason

[9]Not implemented.

TABLE II

```
                VIEW A
                        ; rules
(
 (10 ((officer =rank =name) (ship =reg-number)
      (clean-the-deck =reg-number))
     ->
     ((delete-from wm (clean-the-deck =reg-number))
      (add-to wm (deck-clean =reg-number))))
 (15 ((sailor =name) (boat =reg-number)
      (brings-up-the-sail =sail-type))
     ->
     (($delete-from wm (brings-up-the-sail =sail-type))
      ($add-to wm (sail-is-up =sail-type))))
)
                        ; hierarchies
(*table* (2 (ship deck sail))) ; parts-of-hrrchy of viewpoint-a
(*table* (1 (crew sailor officer))
         (2 (vessel ship boat))) ; is-a-hrrchy-viewpoint-a


                VIEW B
                        ; rules
(
 (20 ((officer =name) (boat =reg-number) (brush =type)
      (clean-the-deck-with-brush =type =reg-number))
     ->
     (($delete-from wm (brush =type)
              (clean-the-deck-with-brush =type =reg-number))
      ($add-to wm (deck-cleaned =reg-type))))
(25 ((sailor =name) (sail =sail-type) (vessel =reg-number)
      (pull-up-the-sail =sail-type))
     ->
     ((delete-from wm (sail =sail-type)
              (pull-up-the-sail =sail-type))
     ($add-to wm (sail-ok =sail-type))))
)
                        ; hierarchies
(*table* (2 (ship deck sail))) ; parts-of-hrrchy of viewpoint-b
(*table* (1 (crew sailor officer))
         (2 (ship vessel boat))) ; is-a-hrrchy-viewpoint-b
```

for having different syntactic entities as indentities is the application of the recognition heuristic, part II (see Section VI-C.2).

C. Overview of the Static Analysis Heuristics

In order to attain its goal, the static analyzer uses different sets of heuristics. Based on the syntactic representation of terms, the analyzer depends primarily on pattern and partial matchers. The matching procedures, which are applied between facts of the two different rule sets, use different sorts of heuristics. The selection of heuristics depends on the subproblem being addressed and on the type of analysis being performed; i.e., viewpoint or perspective analysis.

There are four general heuristics used in the static analyzer:

- partial matching heuristics
- scoring heuristics
- scoring evaluation heuristics
- classification of discrepancies heuristics.

The *partial matching* heuristics are used in finding out similarities between facts from the different rule sets; there are different subsets of these heuristics depending on where they are used. The *scoring* heuristics have the objective of compounding weights to rule pairs in such a way that it is possible to find out the candidate pairs. The *scoring evaluation* heuristics use a series of heuristics to identify the best pairings between rules, which in some cases are just a one-to-one pair, but which in general are a one-to-two pairing. In *classification of discrepancies* heuristics the static analyzer uses the "semantic" information of the hierarchies; these heuristics determine the type of message produced by the analyzer. In general, the first two sets of heuristics are

applied to the recognition subproblem, the third set to the elaboration subproblem, and the fourth set to the evaluation subproblem.

Since the heuristics depend on the subproblem being addressed—i.e., recognition, elaboration, evaluation—we present them separately. The subproblems are further divided and numbered accordingly. For each sub-subproblem we show the heuristics for perspective analysis and viewpoint analysis. The formal description of the heuristics as well as their justification, rationale, and basis are presented in [18].

1) The Recognition Heuristic, Part I: The recognition strategy, part I addresses the following problem:

How to avoid the comparison of each fact of each rule with all of the facts from all of the other rules of the other rule set.

Heuristic: The problem is avoided by using the rule set horizontal structure imposed by the language. Only facts of the same type are compared.

2) The Recognition Heuristic, Part II: The recognition strategy, part II addresses the following problem:

How to establish the analogy between facts of the different rule sets.

Heuristic: A pattern match is used for the facts of the two perspectives. The match is between the verb phrases which represent each of the fact keywords. For example, the pattern matcher would try to match "clean-the-deck" with "clean-deck-with-brush."

The matcher basically determines if each word of the first fact is a member of the second fact, adds 1 if the function membership is true, and finally divides the sum by the number of words in the longer keyword.[10]

The result of the match is a score, the score interval is [1,0].

When comparing viewpoints there is a great chance of variation in the *application vocabulary* keywords used. Consequently, a different matcher from the one used in the perspective strategy is used. As such, the same input to the pattern matcher is given to a fine-pattern matcher, but the result is completely different.

This fine matcher differs from the one described above, because when checking for membership: a) it does a very simple morphological analysis for each word when considering membership, and b) if the function membership returns true a number of times equal to the length of the fact-entity-keyword, then the match score is 1. The less restrictive matcher considers the keywords "clean-the-deck" and "clean-the-deck-with-brush" to represent the same fact in the universe of discourse.[11]

[10] In the example above the result is 0.5, since there are two words that are member (clean and deck), and four words in the longer fact-entity-keyword.

[11] The words "clean," "the," and "deck" when checked for membership return true, so the maximum score, 1, is assigned, turning it into a perfect match.

3) The Recognition Heuristic, Part III: The recognition strategy, part III addresses the following problem:

How to establish a scoring scheme of each rule combination to produce the most alike rule pairs.

Heuristic: This heuristic uses a different weight for each type of fact, multiplies the weight type by the pattern match score, and divides the total by the maximum number of facts of the type. The maximum is chosen from the number of facts of the same type in each perspective. The scores for a fact type are added together to establish the total score for the rule combination. This total score is then passed through a descending sort to mount a list of score combinations. The list in descending order exists for each rule of the actor perspective.

The strategy for viewpoints is similar to the perspective strategy. The major differences are the introduction of a weight multiplier, the consideration of the rule structure, and the consultation of a dictionary for action matches.

The dictionary, although produced by the matcher for fact-keywords of type action, needs to be analyzed by a human agent. The intervention of this human agent will sort out unreasonable matches and add important missing ones.

4) Heuristics Used for the Elaboration Problem: Picking the first pair on the list of scores can lead to undesired rule matches due to misleading scores. Misleading scores can be a result of the following:

- different cardinality of the rule sets
- rule scores not significantly different from each other
- scores that are too low to be considered
- rules that may condense matches, thus generating false similarities.

Heuristic: Identification of rules with no pairs is based on the minimum allowed score as well as on the nonexistence of a score at all.

An *actor perspective* list is the set of rules in the data perspective sorted by higher scores. For each actor perspective list select the first two scores. Submit both pairs to a set of criteria that examines the two rules and verifies if any of the problems related to misleading scores are present. If an anomaly is detected, then other data rules may be chosen as best candidates for the actor rule/data rule pairs, or a critique concerning missing information may be issued.

For those pairs selected so far a further analysis is done. The selected pairs are scrutinized and facts are matched by type. The final selection is based on whether the two prospective data rules have the same perfectly matched action, and on the number of misses of the fact type comparison for each pair.

5) The Evaluation Heuristic, Part I: The evaluation strategy, part I addresses the following problem:

How to separate the facts from the two rule sets into facts that are in perfect match and facts that are not matched.

Heuristic: The set of heuristics described below relies on the horizontal structure of the rule base. Thus to decide which facts are in perfect match or are not matched, a comparison is made using the type and class structure of the facts. The combination of types and classes are the *agent-invariant*, the *agent-input*, the *object-invariant*, the *object-input*, and the *action-output*.

The comparison is made using the pattern matcher already described, and a list of perfect matches as well as lists of nonmatched facts are produced. The heuristics also examine the attributes of facts that have a perfect match and compare them.

6) The Evaluation Heuristic, Part II: The evaluation strategy, part II addresses the following problem:

How to choose the most probable matches between candidate facts from the two rule sets.

Heuristic: The set of heuristics for this strategy is similar to the one used for the list of rules in the recognition strategy, the difference being that in this case there are facts instead of rules. In addition, a more complex matcher, using the information present in the attributes of each fact, is used.

The heuristics differentiate, according to the matcher results, between *candidate-pairs*, which have some similarity, and *missing-information*, consisting of facts that did not find a pair.

In the case of viewpoint analysis, the only differences are the use of the finer pattern matcher, and the use of the action dictionary.

7) The Evaluation Strategy, Part III: The evaluation strategy, part III addresses the following problem:

How to differentiate among inconsistencies, wrong information, and missing information.

Heuristic: The heuristics for this strategy are the ones that ultimately define if the lack of similarity between facts is determined by a missing information or wrong information. The heuristics also detect inconsistencies. The final differentiation among candidate-pairs and missing-information is done based on the hierarchies, on the combination of class and type of facts in the rule, and on the final score produced by the matchers. For instance, in a candidate pair (x,y):

If a fact in rule x from actor perspective
    is a leave in a parts-of hierarchy
And a fact in rule y from data perspective
    is a leave in a parts-of hierarchy
And the hierarchy root is the same.
        (leaves of the same hierarchy)
—>
the facts are in contradiction.
        (one of the rules has wrong information)

On average, we have a set of 12 to 15 different heuristics for each combination of perspective comparison.

For viewpoints as described in the perspective case, the hierarchies are used to avoid conclusions based solely on scores. The significant difference from the perspective case is that a pair of facts which are a nonmatch can be considered to be a match if the hierarchies lead to this conclusion.

VII. Case Studies Performed

Studies to validate our strategy showed that the static heuristics are effective in finding inconsistencies, missing information, and wrong information. These studies were detailed in Leite's thesis [18] and in a case study paper [17]. The case studies confirm the effectiveness of the static analyzer in finding discrepancies and classifying them. Our results help to show that comparing different perceptions about a problem helps the understanding of the problem being addressed.

In [18], two different sorts of problems and four different people were used. In [17], the seemingly simple IWSSD library problem was used to show how an early validation scheme, based on viewpoints, was capable of identifying and classifying problems related to correctness and completeness. Below, we provide more detail on each of these cases.

A. Subject D's Perspective Analysis

This case study was performed by Subject D, a system engineer, with extensive training in data processing and computer science. Subject D used as the universe of discourse a part of an English description of a book inventory problem used in Gane's book [9]. Using a manual for using VWPI, Subject D first found out the keywords of the problem and then coded the three perspectives. In addition, Subject D had some interaction with Leite in order to clarify the procedures in the manual as well as on the formalism. The perspectives were analyzed and a series of discrepancies was found.

The final version of Subject D's view was considerably different from the previous perspectives, and he acknowledged the usefulness of the critiques. A problem observed by D was the repetition of the message about missing "order-information" for each of the rules' pairs. This problem is due to a lack of discernment from the analyzer in distinguishing useful messages from nonuseful ones.

B. Subject E's Perspective Analysis

Subject E, a senior computer science student at UCI with minor experience in data processing programming, used the IWSSD Lift Problem, an English description of a controller for lifts. As with Subject D, Leite provided explanation of the contents of the VWPI manual to Subject E. Subject E produced three perspectives and received a set of critiques. These critiques were produced manually and then were later repeated by the automated implementation. Subject E found that the critiques were useful in his understanding of the problem, and produced a final view that is different from the previous perspectives. Although they are different, they are not as different as in Subject D's case. Subject E kept the same actions and only changed a few objects and agents.

C. Subject D's and Leite's Viewpoint Analysis

Subject D's viewpoint, using the view determined by the perspective analysis referred to above, and Leite's viewpoint toward the Gane description were analyzed. The first set of critiques was found by the manual application of the method, which was then repeated by the automated analyzer. Most of the messages produced were with respect to naming conflicts, and some of them were wrongly classified; that is, a conflict was mistakenly taken to be missing information.

In these messages we had the opportunity of observing an interesting aspect of the viewpoint resolution. Subject D was firm in his opinion that the supervisor had no role in the actions in the description. Subject J was firm in the opposite opinion. Although provided with the agenda, they were not able to negotiate a possible reconciled solution. This may indicate the need for a referee in the process.

D. Subject E's and Subject T's Viewpoint Analysis

This case study was performed with the final view produced by Subject E on the lift problem and by a view produced by Subject T. Subject T is an electronic engineer with a Master's degree in computer science and extensive industrial experience. Subject T followed the same manual as Subject D and Subject E, but with less tutorial explanation from Leite. The static analysis produced by the automated analyzer indicated several discrepancies. Subject T produced 16 rules, while Subject E had only 6 rules. The static analyzer correctly found most of the pairs, and identified the missing rules in E's case. Although the messages reflect the contradictions between the two views, Subject T did not think that they helped him in understanding the problem.

Most of the discrepancies were originated by the actor perspective taken by each one. Subject E identified the agents' "user" and "site-manager," while Subject E used "states" as agents like "floor-request-button-pressed." Although violating the rules of the manual in which states have to be objects, the case study provided useful data. When asked for his overall opinion about the method, Subject T observed that it would be more useful if a negotiation process took place before the analysis of the rules; i.e., that the keywords be analyzed before the writing of the rules by each viewpoint holder.

E. The IWSSD Library Problem

The case study undertaken in [17] is based on four articles presented at the Fourth IWSSD: *A Larch Specification of the Library Problem* [28], *What Does it Mean to Say that a Specification is Complete* [30], *The Requirements Apprentice* [23], and *SXL: An Executable Language* [16]. Each of these articles has a different approach—Wing's work [28] uses a formal language, Yue's [30] is based on the notions of causation and conditionals in logic, Rich *et al.*'s work [23] is based on the notions of domain knowledge, and Lee and Sluizer's [16] is based on an executable language. All the papers deal with the library problem. Three viewpoint analyses were performed, each taking a pair of VWPl descriptions. The following comparisons were made: Rich *et al.*'s rules with Yue's rules, Yue's rules with Wing's rules, and Wing's rules with Lee and Sluizer's rules.

This study is very similar to the study done by Wing [29] on the 12 articles addressing the library problem presented at the Fourth IWSSD. Wing compares the different approaches and how they reveal problems in the description of the library example. Wing's comparison produced a list of problems of ambiguities and incompletenesses of the library description. She correctly pointed out that "the interesting result of the specification exercise is not the specification itself, but the insight gained about the specificand." From our viewpoint, Wing performed a viewpoint resolution over the 12 cases analyzed. Our automated static analysis of four of these articles produced results very similar to Wing's, thus helping the empirical validation of the static analyzer.

VIII. Related Work and Future Research

Our work has to be understood in the context of very early validation in the process of requirements elicitation. It is our thesis that viewpoint resolution, as defined above, is an important strategy to support validation during the process of elicitation. Work related to Viewpoint Resolution is analyzed from the very early validation perspective, as well as from the perspective of multiple viewpoints usage. We conclude this section by dealing with the limitations of our approach, as well as what we consider to be promising directions for future research.

A. Related Work in Very Early Validation

If we broaden our perspective, we will see that different strategies have been advocated to deal with early validation. The best known are:

- using Informal Checking
- using Prototypes
- using Formalisms
- reusing Domain-Specific Knowledge.

From our characterization of very early validation we could draw a line between the first two and the last two strategies shown above. The first set does not require extensive use of modeling, whereas the last set must have a well-defined model in order to be effective. This distinction is important in the comparison of existing strategies with the viewpoint resolution strategy that we are advocating.

All of these strategies are capable of finding out some difference (Delta Δ) between the elicited facts and the universe of discourse. The identification of these Deltas, called *Delta computation*, is part of the process of communication. That is, by communicating with each other, users and analysts try to find out the differences between elicited facts and the universe of discourse. The fact-validation process provides the inputs to the Delta computation. The differences between validation strategies are the kind and quality of the input provided to the Delta computation.

In *Informal Checking* [21] the validation or requirements review is basically a task of reading natural language descriptions and using checklists to detect problems in the requirements expression. The methods proposed for this infor-

mal checking may differ, but they share the lack of automated support and reliance on the analytical abilities of good systems analysts. The Delta computation is controlled by the list of problems:

$$\Delta = \textbf{list of problems (information, facts)}.$$

In *Prototyping*, different kinds of prototypes have been proposed as a way of getting feedback from users. Some of them use a very high-level language, application generators' kinds of languages [12], while others use an executable specification language [26]. The main idea is that by doing this, it is possible to validate the requirements/specification against the user's expectations. The Delta computation is controlled by the behavior of the facts:

$$\Delta = \textbf{behavior of facts (user expectations, facts)}.$$

In *Formalisms* [13], where the software engineer works as a theorem prover, the approach of using formal methods also can provide fact-validation, because it provides the possibility of checking for inconsistencies, which consequently controls the delta computation:

$$\Delta = \textbf{inconsistencies (facts, information)}.$$

The *Domain-Specific Knowledge* [23], [6] approach of reusing domain-specific knowledge is a more recent one that tries to apply some results and insights from artificial intelligence. It advocates the use of pre-encoded facts from previous similar systems in the process of requirements engineering. The systems analyst can then use a mechanical assistant in criticizing his own description of the problem. Given the availability of a domain, it is possible to determine wrong facts as well as missing facts. Although not explicitly addressing consistency, it is assumed that a domain is free of internal conflicts, since the correctness predicate of a domain should be guaranteed. In the case of reusing domain-specific knowledge in validation, the Delta computation is controlled by wrong facts and missing facts:

$$\Delta = \begin{cases} \textbf{wrong facts (domain facts, facts)} \\ \textbf{missing facts (domain facts, facts).} \end{cases}$$

The *Viewpoint Resolution* strategy is able to distinguish between inconsistencies, wrong facts, and missing facts:

$$\Delta = \begin{cases} \textbf{inconsistencies } (facts_a, facts_b) \\ \textbf{wrong facts } (facts_a, facts_b) \\ \textbf{missing facts } (facts_a, facts_b). \end{cases}$$

When compared with the four strategies presented earlier, these are the advantages (+) and disadvantages (-) of the proposed viewpoint strategies:

(+) automated
(+) distinguishes between inconsistencies, wrong facts and missing facts
(+) domain independent
(–) depends on the quality of viewpoints
(–) cost of redundancy.

Although the quality of a critique from a domain validation should be better, given the quality of the domain, the quality of a viewpoint analysis could improve with successive applications. The major point to be highlighted is the input quality produced by the viewpoint analysis to the Delta computation. By having an organized set of critiques, classified according to correctness, completeness, and conflicts, the chances of exposing tacit knowledge are improved. There is less dependency on observers and readers.

B. Related Work in Multiple Viewpoints

Recently, different research groups have being tackling the idea of multiple viewpoints with respect to software construction, especially with respect to the specification process.

Burstin [2] approaches the requirements problem in a manner very similar to our work. This work focuses on the triad: abstract users, abstract requirements, and abstract data types. Abstract users are what we call *actors*; they are identified by a set of heuristics based on the idea of functional decomposition up to the point where an abstract user tree is formed. Abstract requirements are identified bottom-up by using the abstract user tree as a guide. A requirement is defined as "an imperative sentence with a function as the verb of the sentence and abstract object as the direct object of the sentence, being the function and the object the natural language words that have a meaning to that application." Composition of requirements is done by climbing the abstract user tree and composing elementary users requirements, the leaves of the tree. During the composition process, different elementary user requirements are compared and, if conflicting, resolved, before yielding abstract requirements for the parent node. This process is done until the root of the tree has the overall abstract requirements. Burstin's method for composing requirements, although comparing different users' requirements and viewpoints in an orderly manner, relies mostly on *Informal Checking* (see Section VIII-A). The major difference from our work is that we have focused on an explicit elicitation strategy supported by an automated scheme.

Feather [3] and Robinson [24] are dealing with a concept which they call "parallel elaboration in specification construction." This concept is based on a transformational view of specification construction that, although similar to the program transformation paradigm, uses the idea of evolutionary transformations; that is, they change the meaning of the specifications to which they are applied. Parallel elaborations admit that separate lines of specification design can be done independently and that at some point must merge their results. Feather approaches the merging problem based on specification properties, and Robinson uses a planning-based approach in which a goal tree stores different levels of domain goals.

Although Feather and Robinson are dealing with specification construction and we are dealing with requirements elicitation, they face some of the same kind of problems as we do. Most of our work was done in what Feather calls "detection of interference" and Robinson calls "correspondence identification and conflict detection and characterization." Feather's approach to detection is based on terminology, usage, and on classification of interference. His approach is very similar to ours, since terminology is basically what we call *facts*,

usage is similar to our concept of *type*, and we also classify discrepancies. Feather has not automated his method yet, but recognizes that a large number of trivial comparisons is necessary. In our case, as reported, the static analyzer handles those trivial comparisons by intense use of partial matchers. Robinson, on the other hand, centers his attention on resolution and does not elaborate on the problem of identification, which is carried out by an intelligent agent. Robinson's approach of planning and use of domain goals is certainly a good candidate for what we call *Viewpoint Reconciliation*, which we defer as a harder problem, since it depends on social interactions between actors in the realm of requirements elicitation.

Niskier *et al.* [20] use the viewpoint idea to support what they call the "multiple view specification process." In their case a specification is constructed by using different formalisms, and a knowledge-based system (PRISMA) supports this process. PRISMA has several heuristics to help the proper formation of DFD, ER, and Petri-Nets diagrams. Validation is based on paraphrasing heuristics. Although they mention what they call "complementary heuristics," they really do not support the process of merging specifications.

Finkelstein and Fuchs [4] propose a scheme where the specification is constructed as a dialogue between two parties. In this scheme, different viewpoints negotiate, establish responsibilities, and cooperatively achieve a mutual understanding. Using dialogue logics as its basis, they propose an assistant, with a set of heuristics, that helps the interaction between the viewpoints. We understand their work as similar to ours in the sense that it can support what we believe is a crucial point in requirements elicitation; that is, acquisition validation. Their strategy of dialogues is being explored to support N-party dialogues, and we consider it a very promising candidate for Viewpoint Resolution, since it could be the basis for a real time assistant, an ideal setting to handle requirements elicitation.

C. Limitations of our Method and Future Work

We believe our strategy, differing from the proposals of Feather, Robinson, and Finkelstein, has the possibility of being engineered and transferred to practical usage in a relatively short term. Although there are certainly some aspects that will require additional research, there are already some that could be engineered in a relatively short term.

In terms of engineering the method, we see three areas where improvements are mandatory. First, it will be necessary to adjust our simple representation scheme to better handle large requirements; that is, it is necessary to add structuring facilities to our scheme. Second, it would be necessary to construct an user interface to the analyzer, both for entering data (a rule editor) as well as to retrieve the results (prettyprinter for the messages). Third, it would be necessary to package our method with manuals, courses, and examples. Although engineering the viewpoint analysis approach is not trivial, we believe it could be a short-term project.

Being based on small-scale experiments the experiences we have conducted so far, it is possible to foresee the utility of using viewpoint resolution in the requirements elicitation. Viewpoint resolution is basically a cooperative work method, where different people can check their understanding about a subject and discuss the issues that were brought up by checking their differences. Once the method is packaged, it would be ready for pilot experiments in industry.

In terms of research, there are two important aspects that need our attention. One is with regard to improving the heuristics used; that is, we need better heuristics to drive the process of viewpoint analysis, perhaps using artificial intelligence learning capabilities. Another has to do with viewpoint reconciliation automated support to help the negotiation process in discussing different views. As we noted before, the work on commitments [4] could be very useful for viewpoint reconciliation.

Finally, we make some concluding remarks, stressing the contributions of this study and focusing on the processes of evaluation and integration which follow the process of viewpoint analysis.

IX. Concluding Remarks

The case studies helped to show that it is highly desirable to have early validation in the software construction process. They also showed the role of a viewpoint analysis strategy in pointing out problems in the understanding of a situation which we are trying to model. Although the agenda produced by the viewpoint analysis distinguishes between completeness and consistency problems between views, it raises a series of problems that may show ambiguities or incompleteness of the source of information in the universe of discourse. By examining different views of the same universe of discourse, we are obtaining a meta-insight about what Wing [29] calls the "specificand."

Our strategy to difference detection between views is supported by a running program, the static analyzer, which given two different views of two different viewpoint holders produces an agenda with discrepancies between the two views. The agenda produced improves the understanding of the problem at hand, as well as serves as a guide to the reconciliation process, which is dependent on a social negotiation between the actors involved.

While preliminary, we are encouraged by the results produced so far in our research. Subject to additional needed research, we believe that some aspects of this research could be put into practical use with a moderate amount of effort.

Appendix
A VWPI Grammar

Here we describe the VWPI grammar, using standard BNF and notations derived from semantic grammar [22]. The language description is similar to case grammars, where the syntax form conveys some semantics of the application. The semantics of the problem encoded in VWPI are reflected on the abstract forms produced by the VWPI parser.

The semantics of the language VWPI is a subset of PRISM [15] semantics, since in our case the control strategy, nondeterministic firing, is fixed.

The VWPI grammar is as follows:

```
Viewpoint := "(" FactSet RuleSet ")"

FactSet := "(" Objects ")" "(" Agents ")"
"(" Actions ")" "(" Hierarchies ")"
Objects := Object | Object Space Objects
Agents := Agent | Agent Space Agents
Actions := Action | Action Space Actions
Hierarchies := "(" PartsofHierarchies ")" "(" IsaHierachies ")"
PartsofHierarchies:= "(parts-of" Action Space Actions ")" |
"(parts-of" Agent Space Agents ")" |
"(parts-of" Object Space Objects ")"
IsaHierachies := "(Is-a" Action Space Actions ")" |
"(Is-a" Agent Space Agents ")" |
"(Is-a" Object Space Objects ")"

RuleSet := "(" RuleSeries ")"
RuleSeries := Rule | RuleSeries Rule
Rule := "(" RuleNumber Space RuleBody ")"
RuleBody := Lhs "->" Rhs
Lhs := ActorsLhs | ProcessLhs | DataLhs
Rhs := ActorsRhs | ProcessRhs | DataRhs


ActorsLhs := "((" AgentInput ")" Space
          "(" AgentInvariantSeries ")" Space
"(" ObjectInvariantSeries "))"
ActorsRhs := "((delete-from" Space "wm" Space "(" AgentInput "))"
   "(add-to" Space "wm" Space "(" ActionOutput ")))"
| ConstraintRhs
ProcessLhs := "((" ActionInput ")" Space
[ "(" ObjectInput ")" ] Space
"(" ObjectInvariantSeries ")" Space
"(" AgentInvariantSeries "))"
ProcessRhs := "((delete-from" Space "wm" Space
"(" ActionInput ")"
[ "(" ObjectInput ")" ] ")" Space
"(add-to" Space "wm" Space "(" ObjectOutput ")))"
| ConstraintRhs
DataLhs := "((" ObjectInput ")" Space
"(" AgentInvariantSeries ")" Space
[ "(" ObjectInvariantSeries ")" ] "))"
DataRhs := "((delete-from" Space "wm" Space
"(" ObjectInput "))" Space
   "(add-to" Space "wm" Space "(" ActionOutput ")))"
| ConstraintRhs

ConstraintRhs := "((add-to" Space "wm" Space "(error" Object ")))"
ObjectInvariantSeries := "(" ObjectInvariant ")" |
ObjectInvariantSeries "(" ObjectInvariant ")"
AgentInvariantSeries := "(" AgentInvariant ")" |
AgentInvariantSeries "(" AgentInvariant ")"

Object := String in Fact-keywordObj
Agent := String in Fact-keywordAgt
Action := String in Fact-keywordAct

Fact-keyword := Fact-keywordAgt | Fact-keywordAct | Fact-keywordObj
RuleNumber := digits
```

```
ObjectInput := Fact-keyword Space Attributes |
"< not>" "(" Fact-keyword Space Attributes ")"
ObjectOutput := Fact-keyword Space Attributes
ObjectInvariant := Fact-keyword Space Attributes |
"< not>" "(" Fact-keyword Space Attributes ")"
AgentInvariant := Fact-keyword Space Attributes |
"< not>" "(" Fact-keyword Space Attributes ")"
AgentInput := Fact-keyword Space Attributes |
"< not>" "(" Fact-keyword Space Attributes ")"
ActionInput := Fact-keyword Space Attributes |
"< not>" "(" Fact-keyword Space Attributes ")"
ActionOutput := Fact-keyword Space Attributes

Attributes := Variable | Attributes Space Variable
Variable := "=" StringL
StringL := lowercaseletter | StringL Hyphen StringL |
lowercaseletter StringL
Space := space
Hyphen := "-"
```

The terminals `Object`, `Agent`, and `Action` are defined as included, respectively, in the rules `Fact-keywordObj`, `Fact-keywordAgt`, and `Fact-keywordAct` of the following semantic grammar:

```
Fact-keywordObj -> obj1 | obj2 | ... | objn
Fact-keywordAgt -> agt1 | agt2 | ... | agtn
Fact-keywordAct -> act1 | act2 | ... | actn
```

REFERENCES

[1] B. W. Boehm, "A spiral model of software development," *IEEE Computer*, pp. 61–72, May 1988.

[2] M. Burstin, "Requirements analysis of large software systems," Ph.D. diss., Tel-Aviv Univ., Israel, 1984.

[3] M. Feather, "Detecting interference when merging specification evolutions," in *Proc. 5th Int. Workshop on Software Specification and Design* (Pittsburgh, PA). Los Alamitos, CA: IEEE Computer Soc. Press, 1989, pp. 169–176.

[4] A. Finkelstein and H. Fuchs, "Multiparty specification," in *5th Int. Workshop on Software Specification and Design* (Pittsburgh, PA). Los Alamitos, CA: IEEE Computer Soc. Press, 1989, pp. 185–195.

[5] A. Finkelstein and R. Waters, "Summary of the requirements elicitation, analysis and formalization track," *ACM SIGSOFT*, vol. 14, no. 5, p. 40, July 1989.

[6] S. Fickas, "Automating the analysis process: an example," in *4th Int. Workshop on Software Specification and Design* (Monterey, CA). Los Alamitos, CA: IEEE Computer Soc. Press, 1987, pp. 58–67.

[7] P. Freeman, *Software Perspectives: The System is the Message*. Reading, MA: Addison Wesley, 1987.

[8] P. Freeman and J. C. S. P. Leite, "Requirements techniques and languages: a survey," Dept. Computer Sci., Univ. California, Irvine, 1988.

[9] C. Gane and T. Sarson, *Structured Systems Analysis: Tools and Techniques*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

[10] S. Greenspan, "Requirements modeling: a knowledge representation approach to software requirements definition Ph.D. thesis," Computer Res. Group, Univ. Toronto, Mar. 1984.

[11] R. Hall, "Computational approaches to analogical reasoning: a comparative analysis," *Artificial Intell.*, vol. 21, no. 1, pp. 241–250, Jan. 1988.

[12] E. Horowitz, A. Kemper, and B. A. Narasimhan, "Survey of application generators," *IEEE Computer*, vol. 18, no. 1, pp. 40–54, Jan. 1985.

[13] C. Jones, *Systematic Software Development Using VDM*. Englewood Cliffs, NJ: Prentice-Hall, 1986.

[14] R. Kowalski, "Software engineering and artificial intelligence in new generation computing," presented at an Award Lecture in London, May 15, 1984.

[15] S. Ohlsson and P. Langley, "PRISM: tutorial and manual," Dept. Computer Sci., Univ. California, Irvine, Tech. Rep. 86–02, Feb. 1986.

[16] S. Lee and S. Sluizer, "SXL: an executable language," in *Proc. 4th Int. Workshop on Software Specification and Design* (Monterey, CA). Los Alamitos, CA: IEEE Computer Soc. Press, 1987, pp. 231–235.

[17] J. Leite, "Viewpoint analysis: a case study," in *Proc. 5th Int. Workshop on Software Specification and Design* (Pittsburgh, PA). Los Alamitos, CA: IEEE Computer Soc. Press, 1989, pp. 111–119.

[18] J. Leite, "Viewpoint resolution in requirements elicitation," Ph.D. thesis, Dept. Computer Sci., Univ. California, Irvine, 1988.

[19] G. Mullery, "CORE—a method for controlled requirement specification," in *Proc. 4th Int. Conf. on Software Eng.* Los Alamitos, CA: IEEE Computer Soc. Press, 1979, pp. 126–135.

[20] C. Niskier, T. Maibaum, and D. A. Schwabe, "Look through PRISMA: toward pluralistic knowledge-based environments," in *Proc. 5th Int. Workshop on Software Specification and Design* (Pittsburgh, PA). Los Alamitos, CA: IEEE Computer Soc. Press, 1989, pp. 128–136.

[21] M. Ould and C. Vuwin, *Testing in Software Development*. Cambridge, UK: British Computer Soc. and Cambridge Univ. Press, 1986; see also, "Requirement specification," in *Proc. 4th Int. Conf. on Software Eng.* Los Alamitos, CA: IEEE Computer Soc. Press, 1979, pp. 126–135.

[22] E. Rich, *Artificial Intelligence*. New York: McGraw-Hill, 1983.

[23] C. Rich, R. Waters, and H. Reubenstein, "Towards a requirements apprentice," in *Proc. 4th Int. Workshop on Software Specification and Design* (Monterey, CA). Los Alamitos, CA: IEEE Computer Soc. Press, 1987, 79–86.

[24] W. Robinson, "Integrating multiple specifications using domain goals," in *Proc. 5th Int. Workshop on Software Specification and Design* (Pittsburgh, PA). Los Alamitos, CA: IEEE Computer Soc. Press, 1989, pp. 219–226.

[25] D. Ross, "Structured Analysis (SA): a language for communicating ideas," in *Tutorial on Design Techniques*, P. Freeman and M. Wasserman, Eds. Long Beach, CA: IEEE Computer Soc. Press, 1980, pp. 107–125.

[26] "Special issue on rapid prototyping," *SIGSOFT*, Dec. 1982.

[27] G. Steele and G. Sussman, "The revised report on Scheme, a Dialect of LISP," MIT, Cambridge, Tech. Rep. AI Memo. No. 452, Jan. 1978.

[28] J. Wing, "A Larch specification of the library problem," in *Proc. 4th Int. Workshop on Software Specification and Design* (Monterey, CA). Los Alamitos, CA: IEEE Computer Soc. Press, 1987, pp. 34–41.

[29] J. Wing, "A study of 12 specifications of the library problem," *IEEE Software*, vol. 5, pp. 66–76, July 1988.

[30] K. Yue, "What does it mean to say that a specification is complete?," in *Proc. 4th Int. Workshop on Software Specification and Design* (Monterey, CA). Los Alamitos, CA: IEEE Computer Soc. Press, 1987, pp. 42–49.

Julio Cesar Sampaio do Prado Leite (S'87–M'88) received the B.A. degree in economics from the State University of Rio de Janeiro in 1976, the M.S. degree in informatics from the Pontifícia Universidade Católica do Rio de Janeiro (PUC) in 1979, the M.S. degree in computer science from the University of California, Irvine in 1986, and the Ph.D. degree in computer science, also from the University of California, Irvine, in 1988.

He is an Assistant Professor of Computer Science at the PUC, where he joined the faculty in 1989. He was a member of the technical staff of Embratel (the Brazilian Telecommunications Company) for nine years before joining the PUC. His research interests include requirements analysis, conceptual modeling, and domain-oriented reusability.

Peter A. Freeman (S'62–M'87–SM'88) received the Ph.D. degree in computer science from Carnegie-Mellon University, Pittsburgh, PA, in 1970.

He is the Dean of the newly created College of Computing at the Georgia Institute of Technology, Atlanta. During 1989–1990 he was Visiting Distinguished Professor of Information Technology at George Mason University, and from 1987 to 1989 he served as Division Director for Computer and Computation Research at the National Science Foundation. He served on the Faculty of the Department of Information and Computer Science at the University of California, Irvine for almost 20 years. He is the author of *Software Perspectives: The System is the Message* (Addison-Wesley, 1987), *Software System Principles* (SRA, 1975), and numerous technical papers. In addition, he has edited or coedited four books, including *Software Design Techniques* and *Software Reusability*. He was the founding editor of the McGraw-Hill Series in Software Engineering and Technology and serves on several editorial boards. He is an active Consultant to industry and government.

Dr. Freeman is Vice Chair of the Computing Research Board, was co-Chair of the technical program for the 1990 International Conference on Software Engineering, and is a member of the Board of Visitors of the Computer Science Division of the Office of Naval Research. He is past-Chair of the IEEE/CS Technical Committee on Software Engineering.