UNIVERSITY OF CALIFORNIA
Irvine

Viewpoint Resolution in Requirements Elicitation

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Information and Computer Science

by

Julio Cesar Sampaio do Prado Leite

Committee in charge:
Professor Peter Freeman, Chair
Professor Richard Selby
Professor Nicholas Vitalari

1988
The dissertation of Julio Cesar Sampaio do Prado Leite is approved, and is acceptable in quality and form for publication on microfilm:

[Signatures]

Peter Freeman
Committee Chair

University of California, Irvine

1988
Dedication

To the ones from there
my family
my parents
To the ones from here
my advisor
the Reuse – ASE project
the ICS department
who supported and encouraged me
through this amazing journey.
# Contents

List of Figures .............................................................. vi

Acknowledgements ......................................................... vii

Curriculum Vitae .......................................................... ix

Abstract ................................................................. xi

Chapter 1 Introduction .................................................. 1
  1.1 Requirements Analysis ........................................... 1
  1.2 Main Premises ..................................................... 2
  1.3 Software Validation .............................................. 7
  1.4 Viewpoints in Requirements Analysis ......................... 9
  1.5 Overview .......................................................... 11

Chapter 2 The Requirements Analysis Process .................... 13
  2.1 Current Views ................................................... 13
  2.2 Requirements Analysis Issues ................................ 15
  2.3 Requirements Analysis Definition ........................... 18
  2.4 Conclusion ...................................................... 33

Chapter 3 Strategies for Fact-Validation .......................... 34
  3.1 The Process of Fact-Validation ................................. 34
  3.2 Informal Checking .............................................. 38
  3.3 Prototypes ....................................................... 41
  3.4 Formalisms ...................................................... 42
  3.5 Domain-Specific Knowledge ................................... 43
  3.6 Conclusions .................................................... 45

Chapter 4 Viewpoint Resolution ...................................... 47
  4.1 The Existence of Viewpoints .................................. 47
  4.2 The Importance of Viewpoints ................................ 48
  4.3 The Lack of a Viewpoint Resolution Model .................. 49
  4.4 The Viewpoint Resolution Model and Its Research Problems 51
  4.5 Proposed Strategy ............................................. 54
  4.6 Conclusion ..................................................... 66
# List of Figures

1.1 Parts of the Requirements Analysis Process ........................................ 3  
1.2 Pushing the Validation Frontier ....................................................... 9  
1.3 Viewpoint Resolution ................................................................. 10  

2.1 A0 Requirements Analysis ............................................................ 20  
2.2 A-3 Elicitation ......................................................................... 21  
2.3 A-4 Modeling ............................................................................... 25  

3.1 Validation Loop ............................................................................. 35  
3.2 The Validation Problem ............................................................... 37  
3.3 A Framework for Comparing Validation Strategies ......................... 39  
3.4 Using Informal Checking .............................................................. 40  
3.5 Using Prototypes ........................................................................... 41  
3.6 Using Formalisms .......................................................................... 42  
3.7 Reusing Domain-Specific Knowledge .............................................. 44  

4.1 Viewpoint Resolution in Elicitation ............................................... 52  
4.2 The Proposed Strategy ................................................................. 55  
4.3 View Construction ......................................................................... 58  
4.4 The Structure of a Rule ................................................................... 62  

5.1 Comparing Rule Sets ..................................................................... 69  
5.2 Rule Base Comparison as a Search Problem ................................. 71  
5.3 Presentation of the Heuristics ......................................................... 77  

6.1 Using Viewpoint Resolution .......................................................... 124  

B.1 Overall Design of the Static Analyzer .............................................. 141  
B.2 Internal Form Tree .......................................................................... 144  
B.3 Score Tree ...................................................................................... 144  
B.4 Candidate Pairs Table ................................................................... 144  
B.5 Messages Table ............................................................................ 145
Acknowledgements

I am indebted to Professor Peter Freeman, my advisor, for his guidance and support. I would also like to thank my thesis committee, Professors Richard Selby and Nicholas Vitalari, for their support, valuable comments, and suggestions.

I consider myself lucky to have had the chance of being a member of the Reuse and ASE Projects along with Jim Neighbors, Rubén Prieto-Díaz, Ira Baxter, Guillermo Arango, Chris Pidgeon, and Y.V. Srinivas. I have learned a lot from each of them. My special thanks to Arango, Baxter and Srinivas for their support and friendship.

My thanks to my Embratel colleagues who supported me through the last months of this journey: Dias, Eduardo, Marco Aurelio, Esteves and Rocha who supported my work, and James, Alvaro, Roberto, and Luciano for their friendship. I also am grateful to José Carlos Silva, David Ham, and Eiichi Teratsuji, who contributed to the experimental validation of the work.

I would like to thank Catherine Pearce for her assistance in proofreading the text of this dissertation. Her assistance was invaluable in improving its clarity and readability.

I am deeply grateful for the financial support received from the Office of Research and Graduate Studies Graduate Studies and Research of the University
of California, through which I worked as a Teaching Assistant and later as a Research Assistant, and from the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq). Support from CNPq was essential and I am very grateful to have had this opportunity.
Curriculum Vitae

December 2, 1954  Born Rio de Janeiro, Brasil

1977  B.S. in Economics, Universidade Estadual do Rio de Janeiro, Brasil

1979  M.S. in Informatics, Pontifícia Universidade Católica do Rio de Janeiro, Brasil

1979  Systems Analyst, EMBRATEL S/A Rio de Janeiro, Brasil

1982–1983  Teaching Assistant, Department of Information and Computer Science, University of California, Irvine

Summer 1983  Visiting Lecturer, Department of Information and Computer Science, University of California, Irvine

1983–1984  Research Assistant, Reuse Project (Software Engineering), Department of Information and Computer Science, University of California, Irvine

1985  Assistant Professor, Department of Informatics, Universidade Estudal do Rio de Janeiro, Brasil

1986  M.S. in Information and Computer Science, University of California, Irvine

1985–1987  Research Assistant, Advanced Software Engineering Project Department of Information and Computer Science, University of California, Irvine

1988  Ph.D. in Information and Computer Science, University of California, Irvine

Dissertation: Viewpoint Resolution in Requirements Elicitation
Publications

Leite, J.C.S.P. *A Survey on Requirements Analysis*
Technical ASE Memo no. 71, Department of Information and Computer
Science, University of California, Irvine, June 1987.

Leite, J.C.S.P. Reutilização de Software, in *XVII Congresso Nacional

Report no. 230, Department of Information and Computer Science,
University of California, Irvine, 1984.

Analysis*, *XVI Congresso Nacional de Informática*, São Paulo – Brasil,
October 1983.

Leite, J.C.S.P. Análise de Sistemas, uma Perspectiva Social, *XVI Congresso

Leite, J.C.S.P. Contabilidade de Custos no Desenvolvimento de Software,
Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro,
Abstract of the Dissertation

Viewpoint Resolution in Requirements Elicitation

by

Julio Cesar Sampaio do Prado Leite

Doctor of Philosophy in Information and Computer Science

University of California, Irvine, 1988

Professor Peter Freeman, Chair

It is the thesis of this work that viewpoint resolution can be used as a means for very early validation in the process of requirements elicitation.

Requirements elicitation is the process in requirements analysis responsible for understanding, finding and gathering information. As a part of requirements elicitation, the objective of fact-validation is to make sure that the facts gathered reflect the original intent, as well as to help unfold the knowledge not yet recorded as facts.

Viewpoint resolution is a different approach to very early validation. It takes place in the process of fact-validation, and it is based on the acknowledgement that software requirements can be elicited from different viewpoints.

The thesis proposes a language for expressing views from different viewpoints and an automated static analyzer of views that is capable of differentiating between
missing information and conflicting information, thus providing support for viewpoint resolution. The thesis also presents the results of initial investigations into the effectiveness of these mechanisms for viewpoint resolution.
Chapter 1

Introduction

The objective of this chapter is to establish the context of our research. First, a brief description of requirements analysis is given. Second, the main premises used in the understanding of requirements analysis are presented. Third, the motivation for early validation is given, as well as the different stages where validation may take place. Fourth, the overall idea behind viewpoints is presented, together with the artificial intelligence techniques used in the automated viewpoint analyzer. Finally, an overview of the remaining chapters of this dissertation is provided.

1.1 Requirements Analysis

There is a wide belief that before doing something, the first step is to know and to understand what is to be done. This, however, is sometimes forgotten by people in their rush for solutions; therefore, several scholars have stressed the importance of this simple, but crucial fact. As in all disciplines that are oriented towards the production of a product, software engineering has as a precondition: the determination of what is to be produced. The process, in software engineering,
of determining "what is to be produced" is called requirements analysis (see Figure 1.1). The product of this process is called the requirements.

Requirements analysis, in particular, requirements elicitation, is a difficult task, and it is usually carefully avoided by most of the software engineering researchers [54]. On the other hand, research labeled as dealing with requirements, usually deals with specification, and this is the main reason for the lack of agreement on the definitions of requirements analysis and specification.

Requirements elicitation is the process within requirements analysis responsible for understanding, finding, and gathering information. As a part of requirements elicitation, fact-validation attempts to ensure that the facts gathered reflect the original intent, as well as helping to unfold the need for knowledge not yet recorded as facts.

1.2 Main Premises

The findings and conclusions of our research were bound by the following premises: 1) There is a general belief that before doing something, the first step is to know and to understand what is to be done. 2) Requirements analysis does not happen in a vacuum.

The process of knowing and understanding what is to be done requires that information about what is to be done available. The process of making this information available is called elicitation. To elicit is to extract $x$ from $y$. In this case, $x$
Figure 1.1: Parts of the Requirements Analysis Process
is the information and y is the source or sources of this information. The process of elicitation is necessary regardless of what software construction paradigm is used.

Using the system's point of view [9], software - a system itself - is a subsystem of a large system. It should be clear that, being a component of this larger system, software does not exist in a vacuum. The larger system establishes the universe of discourse\(^1\) for the software. This universe of discourse, delimited by the larger system, does not imply that the information necessary for what should be done by the software is available. It does, however, delimit the sources from which this information would be extracted. By assuming that there is a universe prior to the requirements analysis we are restricting our view of requirements analysis. This assumption matches what Davis [26] calls the second level of requirements information.

1.2.1 The Necessity of Elicitation

In problem solving, the first premise is taken to be true. Polya's [76] well-known advice questions are a clear example of this belief:

What is the unknown? Do you know a related problem? Could you restate the problem?

The first question clearly refers to what is to be solved. The second reflects the possibility that some previous knowledge related to the unknown does exist, while

---

\(^1\)Universe of discourse is the overall context in which the software will be developed. It includes all the sources of information and all the people related to the software. These people are referred to as the actors in the universe of discourse.
the third question functions like a proof that the unknown is clearly understood by the problem solver.

The basic idea behind Polya’s advice is to provide structure to reduce complexity, and to ease the task of problem solving. Other disciplines espouse this hypothesis as well, and provide structure to simplify tasks. Let us examine the disciplines of artificial intelligence, project management, and software engineering (SE).

<table>
<thead>
<tr>
<th>AI</th>
<th>PRJCT MNGNT</th>
<th>SE (iplct rqrmts)</th>
<th>SE (expl rqrmts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>goal</td>
<td>objective</td>
<td>specification</td>
<td>requirements</td>
</tr>
<tr>
<td>plan</td>
<td>plan</td>
<td>design</td>
<td>specification</td>
</tr>
<tr>
<td>solution</td>
<td>results</td>
<td>code</td>
<td>implementation</td>
</tr>
</tbody>
</table>

Artificial intelligence structures its tasks as: “goal”, “plan” and “solution”. Project management has “objective”, “plan” and “results”. Software engineering, however, may use a different terminology depending on the implicit or explicit description of design and requirements. Implicit requirements exist when the requirements are not presented as a separate document, that is they are included in the specification document. Explicit requirements exist when the requirements appear in a separate document from the specification. This definition is based on Freeman’s [33] description of implicit and explicit design. If we look closely, we will notice that the first element of each of these structures is related to Polya’s first question and to our hypothesis.

---

2In the table above, the combination of explicit requirements and explicit design is not presented, although we believe it to be a better SE breakdown.
Of interest is that each of these structures uses a strict order. Besides being a sequence, the elements inside the box have another important relationship, which exists between the first element and the last element. The first element provides a basis for measuring the degree of success of the last element. In project management, therefore, the results, when compared with the objectives, determine the extent to which the objectives are met.

Another important fact to be highlighted is the use of the idea of implicit design, implicit requirements, explicit design, and explicit requirements. On the structures shown above, design is made explicit and requirements are made implicit on the first column of SE, and requirements are made explicit and design is implicit on the second column. As we will describe later, this fact is one of the main reasons for the confusion between requirements and specification.

1.2.2 A Systems Point of View

The second premise, that requirements analysis does not happen in a vacuum, can be better described by an example of the development of an embedded software. That is, when a product that will use an embedded software is planned, software will be only one of its many components.

An example is the development of a new automobile. It usually takes more than five years [89] to develop an entirely new car. Several plans are made and several prototypes are built and tested. The design of an electrical system can take 1000 man years over a three-year period, and the design of a new engine is, in itself, a major effort. The structure of the vehicle requires careful planning and
numerous tests. Trying to reduce the number of parts is usually a goal, in order to make manufacturing more cost effective. In this process, the systems that control fuel injection and brake locking need software. Even though an entire universe of discourse already defined is at the designers’ disposal, there is still a need for determining exactly what the embedded software is to do. A requirements analysis is necessary to determine this.

Understanding that requirements analysis does not happen in a vacuum also is referred to as the “blank page fallacy” [5], that is, there is always an already existent universe of discourse, which defines the boundaries of the problem that need elicitation. Under the systems view, the Universe of Discourse is a subsystem of the general problem, which is already defined.

1.3 Software Validation

Barry Boehm in a recent note on IEEE Software [13] listed what he calls the “top 10 list of software metric relationships, in terms of their value in industrial situations.” On the top of the list he says:

Finding and fixing a software problem after delivery is 100 times more expensive than finding and fixing it during the requirements and early design phases.

The main issue here is where to test the software such that errors do not propagate far into the software life cycle. Testing often is associated with the terms: validation and verification.
Verification is the testing of an object against its specifications (the previous deliverable in the software life cycle), i.e., are we building the product right? Validation is the process of confirming that a deliverable matches the user's expectations, i.e., are we building the right product?

Since the user's expectations are not what we call a deliverable, there is no formally based description, - it is impossible to verify a requirement.

Validation traditionally has been done at the end of the life cycle. The acceptance by a user of the final product is when the predicate\(^3\) they used to validate their expectations is evaluated to be true (Figure 1.2 - Usual Validation).

Recently, several researchers in software engineering [2] have been proposing to push the validation frontier towards the specification phase. Among those researchers, the most usual trend is the use of executable specification languages (Figure 1.2 - Early Validation).

Most of this research, however, does not realize or promote the possibility of pushing this validation further, that is, towards the requirements analysis. Exceptions to this are Fickas [31], Arango [4], and Rich et al. [77], who have considered the usefulness of having this very early validation (Figure 1.2 - Very Early Validation).

This thesis concentrates on the process of fact-validation. Because the process of fact-validation is a part of requirements elicitation, the validation frontier is...
pushed towards the requirements analysis phase of the software life cycle (Figure 1.2 - Very Early Validation).

The different software life cycle stages, where validation may take place are pictured in Figure 1.2.

1.4 Viewpoints in Requirements Analysis

The use of different viewpoints, for broadening as well as deepening the understanding of a given situation, is a common practice in courts proceedings, where different witnesses give their testimony. These testimonies may complement one
another or may conflict with one another. We believe that requirements elicitation has a strong resemblance to such a process.

Viewpoint resolution is the process which identifies discrepancies between two different viewpoints, classifies, and evaluates those discrepancies and integrates the alternative solutions into a single representation\(^4\) (Figure 1.3).

In the field of software engineering, authors of some methods, e.g., – SADT [81], and CORE [64] – have claimed that viewpoints are important in describing a system. Although CORE and SADT advocate the use of viewpoints, neither

\(^4\)This work does not address the evaluation and integration aspects of viewpoint resolution.
one has an algorithm or a method to explicitly address the problem of viewpoint resolution. They rely on a walkthrough process among viewpoint holders as the sole way of dealing with viewpoints. One of the reasons why viewpoint resolution is not supported in CORE or SADT is that they do not have a formalism for representing viewpoints.

Although the concept of viewpoint resolution is valuable, as recognized by the literature, we believe it does not fulfill its potential if a proper representation does not exist. The importance of having a representation for viewpoints is that it makes possible the application of systematic methods towards the analysis of those viewpoints.

For representing viewpoints this work proposes the viewpoint language, VWPL, and defines its syntax and semantics. This language is derived from PRISM [52], a production system architecture. As such, the viewpoint resolution "process" is presented as a problem of finding and classifying discrepancies among two rule bases. The proposed solution to this problem is explained using a framework for analyzing artificial intelligence work on analogy [41].

1.5 Overview

Chapter 2 details the definition of requirements analysis and highlights the differences between elicitation and modeling.

^author/reader types of cycles
Chapter 2
The Requirements Analysis Process

In this chapter we present our understanding of the process of requirements analysis. This understanding is based on a survey [54] of requirements analysis techniques and languages. This chapter also points out the main findings of the survey, as well as, in our opinion, the main issues that surface in the process of producing a requirement.

2.1 Current Views

In a recent survey about requirements analysis [54] it was "learned" that:

Requirements analysis is a hard task.

There is no "silver bullet".

There is no agreement on the definition of requirements analysis, and there is no clear distinction between the process and the product of requirements analysis.
Chapter 3 explains, in more detail, the process of fact-validation as a step in the requirements elicitation task. It shows the evolution of the research from the hand waving scheme of "check consistency" and "check completeness" to ideas such as using domain-specific knowledge in fact-validation. The fact-validation research problem is detailed and the literature propositions are analyzed.

Chapter 4 analyzes the importance of considering different views in requirements elicitation and presents viewpoint resolution as a means of achieving very early validation. It identifies the basic research problems behind the implementation of viewpoint resolution as a support for fact-validation, and presents a strategy for identifying and classifying discrepancies between viewpoints. This strategy is based on a comparison of two rule bases which represent two different viewpoints.

Chapter 5 presents a solution for the viewpoint resolution problem. This solution is posed as a comparison between the two different rule bases, and uses an analogy-based framework for its presentation and explanation. The explanation derives from comments on the series of heuristics used, as well as on small examples that demonstrate the strategy.

Chapter 6 summarizes the contributions of the thesis, and point out the need for further investigation on both the strategy and on its application.

Appendix A presents the detailed description of the viewpoint language, VWPl.

Appendix B gives an overview of the Scheme functions used in the static analyzer of viewpoints.
Research in software engineering avoids dealing with requirements analysis.

Computer scientists, in general, overemphasize the importance of modeling.

There is an emerging consensus over issues in requirements analysis.

The first statement may seem obvious, as it should be to those of us interested in this area of research. The process of determining the intention of the future user or users, with respect to software to be produced, is not a trivial task. It is commonplace for textbooks to stress the importance of requirements analysis and its difficulty.

The second finding reflects the very low probability of some dramatic improvement to the way requirements analysis is being done. Brooks [18] made this clear in his statement that “there is no silver bullet” to the software process. Another important reference, for those who think that artificial intelligence is the solution to the software engineering problems, is the recent book by Winograd and Flores [104], which deals with the limitations of artificial intelligence.

The third finding reveals the confusion between process and product. It points out that the misuse of terminology and the lack of an agreement on the definition of requirements analysis are the main reasons for this confusion.

The fourth finding is intriguing. Although recognized by many as important and difficult, requirements analysis is, most of the time, avoided. Instead, researchers try to focus on specifications where the informality, incompleteness, and
inconsistency of problems can be forgotten. This trend, unfortunately, has a major drawback: the covering up of the elicitation problems in requirements analysis.

The fifth statement is a direct consequence of the fourth one. Researchers in computer science usually pay more attention to modeling than to the understanding of what they are modeling. Most of the progress in requirements analysis is really progress in specifications. The authors stress the expressiveness of their models and what they can represent, or how close to the real world they are. Some authors explicitly depart from pre-existing “users needs”. Others mention the user, but they fail to explain how they got the user’s intentions, how they got the users’ description of what is supposed to be done, and how they know that the description is really what the users want.

The last finding shows that, although there is a lack of consensus in the definition of requirements analysis, there is a common set of issues. These issues are usually pointed out, by different authors, as being important in the requirements analysis process.

### 2.2 Requirements Analysis Issues

In this section we list the main requirements analysis issues identified by our survey. These issues or concepts play important roles in the requirements process, as can be confirmed by their commonality on the literature.

We provide below, a list of issues and concepts that have attracted the attention of researchers. Some of the items were pointed out as being important in requirements analysis, and a few of them were referred to in the context of implicit
requirements. A brief description of each item's meaning is provided, but in general, they are self explanatory. Following each issue or concept is a list of the authors who refer to the term.

- Interviews - The old method of face to face interaction between systems analysts and users [25, 28, 73, 26].

- Questionnaires - Mass gathering of information, according to pre-defined set of questions [25, 28, 62, 73, 34].

- Reusability - The importance of reusing past experiences in the process of gathering information is widely recognized [26, 62, 73, 100, 67].

- Validation - Many authors understand that it is crucial that early validation be available [105, 45, 63, 12, 26, 28].

- Participation - The user involvement in the process of collecting and analyzing information [69, 60, 65, 22]. The systems analyst's participation in the user process, as an effective way of performing observation [100, 28].

- Presentation - The use of graphic tools to enhance the communication between users and systems analysts [60, 64, 90, 24, 32]

- Checklist - The list of verbs and nouns used in the application [48, 61, 21, 81].

- Application Vocabulary - The use of the same words in the model of the application as in the application itself. It is an approach using symbols (words) with an attached common understanding [62, 67, 43, 86, 44, 6].
Viewpoint - The idea of viewpoint is applicable not only to modeling different understandings of an application, but also to different forms of expressing the same understanding [81, 64, 65, 60].

Abstraction - One of the fundamental foundations of modeling. It addresses ways of presenting information in a more condensed form. The principles of decomposition, abstract data types, classification, and generalization are of fundamental importance [81, 40], [87, 20, 17, 66].

Components - In modeling, the idea that basic concepts should be represented as components (modules) [67, 96, 74, 62, 29].

Interfaces - In modeling, the idea of connecting components or modules [27, 38, 74, 29].

Pre/Post-Conditions - In representing a model's semantics, the use of Hoare's pre/post-conditions seems to be the most common [17, 8, 95, 86, 70].

State Machines - In representing a model's semantics, another very common method is the use of state machines [82, 62, 42, 86, 10].

Relations - A fundamental concept in modeling. It is a way to express dependencies among elements in a model [23, 94, 20, 66, 39, 80].

Structures - In building a model, one has to have a structure. Some modeling methodologies and techniques pay particular attention to this fact [81, 35, 3, 44].

Contract - The requirements should carry the weight of a contractual relationship between the actors involved in the process [105, 22].
These above definitions of the established concepts in the area helped us to provide a process-oriented definition of requirements analysis. In this definition, these concepts were clustered into different subprocesses of the requirements analysis task. Next we present our definition of requirements analysis, which was used as a framework for our research on viewpoint resolution.

2.3 Requirements Analysis Definition

Requirements analysis is a process in which “what is to be done” is elicited and modeled. 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, is produced.

This definition focuses on what should be present in the requirements analysis process; not on how to perform the process. The definition emphasizes the process, since the product itself is a consequence of how this process is performed. The requirements, besides being a communication medium for the actors involved, are the elements used at the end of the software construction process to verify that the results obtained were the desired ones.
2.3.1 The Process

In order to clarify the requirements analysis process, our definition makes a division between elicitation and modeling. It is worth noting that such a distinction is difficult to make since both processes are very intermingled, however the distinction helps to separate the different sorts of issues present in the process of requirements analysis.

The process of elicitation is seen as being composed of: fact-finding, fact-validation and communication subprocesses. These subprocesses are conducted by actors. Actors have viewpoints and use tools and methods.

Modeling is the process responsible for portraying the results from the requirements elicitation. Modeling not only produces the final model, but also its prettyprinted form, the requirements.

The process of modeling is composed of two parts: representation and organization. In order to come up with a model of the elicited facts, actors use methods and tools. A model is composed of a representation scheme which should be arranged according to the organizational policies of the model. A model should be able to capture different viewpoints.

A SADT [81] model of this characterization of requirements analysis (Figure 2.1) is provided to help the clarification of the different sorts of relationships between the parts and the subprocess of requirements analysis. The viewpoint used for this SADT model is of identifying relationships that may exist in a requirements analysis process, it does address the operation of this process.
Figure 2.1: A0 Requirements Analysis

It should be kept in mind that the whole process of requirements production is very intermingled. Because of this, several levels of interaction occur before the final version of the model and the requirements are produced.

Elicitation

Elicitation (Figure 2.2) is a process that is further divided into fact-finding, fact-verification and communication subprocesses. Each of these subprocesses will be analyzed in detail, and the requirements analysis issues will be identified with each. A SADT is provided to detail the relationships between those subprocesses.
Note: (1) The result of validation should uncover missing facts, as well as wrong facts of the model

Figure 2.2: A-3 Elicitation
Fact-Finding The fact-finding embraces interviews, questionnaires, reusability, and checklists. This subprocess refers to all activities that have as an objective the gathering of information. It is usually performed by the systems analyst for purposes of better understanding the application. Techniques available are interviews; observation; participation in the application, reading of manuals, internal memos, policies, and related material; meetings; personal communications; and use of past experience. Using, or reusing past experience of the information gathering process can be a personal transfer of information, in the case of an analyst who has already worked on a similar application, or it can be, as has been advocated recently, a transfer of information between actors. This is is known as reusability of domain knowledge.

Interviewing is the most frequently cited method of gathering information. Different techniques are available. They do not greatly differ from each other and all of them stress the important point of communication. Questionnaires also are an important tool, and there exist several heuristics on how to produce a good questionnaire. These techniques presuppose the participation of human agents in the process.

Another important aspect of fact-finding is related to the personal capabilities (observation, for example) of the analysts, who have to capture, from different sources, a huge amount of information. An interesting tool recommended by several authors is the checklist.

The reusability of previous analysis and design is an issue that is getting more and more attention from the community. Mills [62] explicitly talks about an analysis library. Neighbors puts forward the idea of domain languages in the context
of developing software by components [67], which is a way of implementing the reusability of requirements analysis. The process of encapsulating this knowledge to promote reusability at a very high level of abstraction is yet an open research problem.

**Fact-Validation** The fact-validation process refers to the issue of validation and execution, where one is aiming to unfold tacit knowledge, i.e., the knowledge that exists but is not encoded. It is related to the general idea of early validation, which ideally should provide a way of testing the requirements before proceeding to the next phase of the software development process. Fact-Validation involves methods and techniques that enable the user and the systems analyst to obtain an idea of the potential behavior of the desired system before it is actually produced. Yeh, et al. [105] refer to it as “unfolding the tacit knowledge,” while Davis [26] calls it “discovering from experimentation”. This concept usually is associated with the idea of an early prototype [85], which assists in uncovering misinterpretations and missing or subassumed information, as well as false assumptions.

The use of scenarios [45, 63] is an important idea; it stresses the importance of using mockups for early validation of intent through the use of operational examples. The use of examples is somewhat related to the problems of software testing (usually done at the code level). The idea of “heuristic development” is described by Dickson [28]. The works of Wasserman [98], Blum [12], and Henderson [43] also deal with the idea of fact-validation through prototyping.

The use of very formal languages, such VDM [11], also provides a facility for fact-validation, although it is performed at a detailed level more akin to the
specification process than to the requirements analysis process. Fact-validation using a very formal language is done by the systems analyst who works as a theorem prover trying to identify inconsistencies on the model produced.

Another method for fact-validation is to rely on the ability and knowledge of the systems analyst, or to use a manual fact-validation scheme between the actors involved in the process, as proposed by the SADT author/reader cycle.

Communication Issues in communication are participation, presentation, and application vocabulary. Negotiation between users and systems analysts occurs in this subprocess. The fact that users are novices in the analysts' areas of knowledge, analysts are novices in the users' areas of knowledge, and each are experts in their respective areas [105] poses a great challenge for communication.

The idea of user involvement [69, 60, 65, 22] is fundamental, but it also is extremely important that analysts or software engineers involve themselves in the user process by using an anthropological approach [100]. Several techniques from organizational psychology could be employed in this process [26].

The use of graphic notation [81, 32, 60, 35] is important to enhancing communication. An interesting result from a survey done by Colter [24] is the user's preference of Hipo charts [90] and Structured English [35] over other languages or methods in terms of user communicability.

Another important fact is the consensus over the use of application vocabulary or keywords as an important strategy to enhance of requirements communication [48, 8, 43, 86, 67, 44].
Figure 2.3: A-4 Modeling

Modeling

In our definition, modeling (Figure 2.3) is seen as being a process composed of two subprocesses: representation and organization. Each of these subprocesses will be analyzed in detail, and the requirements analysis issues will be identified with each one of these sub-processes. A SADT diagram is provide to describe the relationships between representation and organization.
Representation  Representation is important because it establishes how the semantics of the model are to be defined. Issues in representation include: semantics (pre/post-conditions, state machines), abstractions, and relations. Most of the progress in software engineering has been in the area of representation, an area of very active research and controversy [99].

Most of the work done on representation, as pointed out earlier, is aimed at the specification and design of software. Several methods and techniques have been developed towards providing formalism together with expressiveness.

In terms of semantics, the underlying models that are used the most are:

1. Pre/post-conditions. Hoare’s formalism \( \{P\} S \{Q\} \) – where \( P \) is the input assertion, \( Q \) is the output assertion, and \( S \) is the list of statements that enforces the input assertion and satisfies the output assertion – is used in several representations as a way of portraying semantics [17, 8, 95, 86, 70]. Variants of Hoare’s formalism ultimately can be redefined in a first order calculus, as present in RML [39], Ina-jo [49], and Anna [59].

2. State Machine. The state machine model and its variations are the underlying model in the following representations: [82, 62, 42, 86, 10].

Other underlying models for semantics, where formality is the foundation, are: functional-languages [106, 43], denotational semantics [11], petri-nets [19, 29], and axiomatic definition [40, 38].

Several other representational schemes, although not strongly formal, do provide a means of portraying the semantics of a problem. These tools or techniques
[81, 35, 48, 64, 60] usually focus the attention on data (objects) and procedures (operations) and their associated flows. They also focus on input/output (stimulus/response) descriptions and some of them focus on the conditions (constraints) existent in the application.

In addition to providing the semantics by which a model will be represented, representation must deal with abstraction, and relations between abstractions. Work on abstractions and relations [23, 94, 20, 66, 39, 80, 81, 87, 15, 17] has focused more attention on how to represent concepts which are closer to the real world.

The key points in representation, from the point of view of requirements analysis, are as follows.

1. Expressiveness - Concepts should be represented without relying on unstructured prose descriptions.

2. Formalism - The representation must have a formal underlying model.

3. Executability - The representation must allow the "execution" (i.e., show the behavior) of a requirements analysis model.

4. Organization and Redundancy - The representation also should be capable of having the organization and redundancy mandated by the requirements analysis model.

Organization	Organization is the part of the modeling process that is responsible for the overall structure of the model, as well as its prettyprinted form, the requirements. It includes the issues of abstraction, components, presentation, reusability,
interfaces, and structure. All models need an organization. This is extremely impor-
tant, because it is a procedure, though sometimes artificial, of reducing complexity.

Complexity is one of the problems constantly defeating systems analysts, and it is sometimes overlooked. The concept of abstraction, together with Systems Theory [9] has been of great help in providing some organization to the process of information analysis and design, where the concepts of modularization, decomposi-
tion, interface, and divide and conquer are important. Recently, artificial intel-
lligence [16] highlighted some other important concepts that help the organization of information\(^1\). These concepts are: classification, generalization/specialization, inheritance, and association.

SADT [81], A-7 [44], and the works of SARA [29], [74] MILs [27] are examples of the system's idea and the importance of interfaces.

Guttag [40] and Liskov [35] have presented works dealing with abstract data types, a concept which originated programming languages and is of great importance to software engineering. An application of this concept in organizing a model can be found in Belkhouche's thesis [8].

ACM/PCM [17], TAXIS [66], and RML [39] are organizations in which the ideas of artificial intelligence with respect to knowledge representation are present. Both works use the pioneer research of Smith [87], which applies the concept of semantic nets to data base organization. The importance of these works is the introduction of the possibility of inheritance by using generalizations/specializations.

\(^1\)For artificial intelligence this is known as knowledge representation.
Other fundamental concepts with respect to organization are: components [67, 96], data dictionaries [35] and decision tables [35]. Components are viewed as a foundation for reusability in a higher level of abstraction. In Draco [67], components are where the semantics of domain languages are defined. In Galileo [96] components are the basic building blocks of any design.

As in design [74, 75], being prepared to change is an important organizational characteristic that should be present in requirements analysis. Because of this, another aspect to consider is the notion of configuration management [84].

The resultant requirements analysis model, faces the problem that it has to serve different actors. First, it has to be readable by users and second, it should be the base for the designer’s understanding of the application. Therefore, the organization has to provide a prettyprinted version of the model, the requirements, in order to have an adequate medium for communication between the people involved in the requirements analysis process. An example of the distinction between model and requirements is the work of the Forest project [32], where a set of tables and graphs is employed for user communication (requirements) and a modal logic for future interface with designers is used.

Modeling with redundant information makes consistency checks, as well as localization of conflicting points in different viewpoints, possible. Salter [82], Nguyen et al. [68], SADT [81], PSL/PSA [94], CORE [64], SREM [3], and Paisley [106] are some of the works that present redundancy in modeling.
Viewpoint, Methods, Tools, and Actors

Viewpoint is a central issue in requirements analysis, since it affects not only elicitation but also modeling. It acknowledges that different views could exist in describing an application. It also reflects the ideology of the systems analyst's organization. This ideology determines which method or technique will be used. For instance, some data processing centers use Yourdon's structured analysis, and others use SADT. Some software contractors use SREM, and others use TAGS [84]. Each of these organizations is using a different viewpoint, reflecting the different ideologies of theirs managers. An example of a choice of different methods is given in Ripken [79].

Viewpoint is based on the fact that different views could exist to describe something, and, by describing the same thing with different views, enough redundancy would be provided such that checks for consistency of the models could be possible. Viewpoints can be simply a result of different professional backgrounds, such as engineers, managers and chemists, or they can be a result of a conflict of interest, or they can be derived from poor understanding (errors) or from incomplete information. Another aspect in which viewpoint is involved is making clear that there are different viewpoints, and that the model reflects them.

By being aware of the fact that different viewpoints do exist, the various actors involved in the requirements analysis process must achieve a consensus over a final model [37].
SADT [81] and CORE [64] are the major references regarding viewpoints. Others use viewpoints differently, such as differentiating between data and procedure [82], using a mix of frame-based and functional representation [15], differentiating between real flow and information flow [60], or differentiating connector, flow, process, and data as in Winchester’s thesis [101].

The process of requirements analysis has been shown to be composed of different components. Interacting within these components are actors. Actors, as previously defined, are composed of users² and systems analysts.³ Although a division between the two does exist, it should not be forgotten that users are composed of different sets of actors, with different skills, different interests, and different responsibilities. On the systems analyst side, there also are differences, such as different levels of management, consultants, and different skill levels among analysts.

Methods and tools will be used according to the viewpoint of that set of actors which has control over the requirements analysis process. Usually systems analysts are not only the ones who carry out the process, but also the ones in control. Sometimes, however, users have a more active participation in the control and selection of methods and tools.

2.3.2 The Product of Requirements Analysis

The resulting product of requirements analysis is clearly dependent on how the process is performed. Above all, the different viewpoints used in the process

²An user is an actor, whose role is to demand and use the software.

³A systems analyst is an actor who performs the task of modeling the user’s expectations.
determine what kind of model will be produced, and how the model will be represented and organized. It is important to clarify the difference between the model and the requirements.

The model is an internal representation for the process. It should capture as much information as possible in a machine-processable form. Because of the "completeness fallacy" however, an equilibrium point between the efforts invested and the quality of the model has to be reached. Although there is no golden rule for achieving such an equilibrium, a good heuristic is to rely on the negotiation process between users and analysts, using fact-validation and communication as effective ways for achieving a consensus. In its machine-processable form, the model is not suitable as a communication document between user and analysts. A solution is to have a prettyprinted form of this model, written in a subset of natural language and relying heavily on the concept of "application vocabulary". Balzer\(^4\) refers to this as a "paraphrase," which, in his case, takes Gist descriptions and generates English descriptions. The output of this prettyprint or paraphrase facility is called the requirements.

The requirements\(^5\) is a document that both users and analysts agree upon to represent the user's demand. As pointed out before, there is a consensus that such a document should bear legal implications. As a contract, the requirements is given law dimensions [103]. That is, the requirements is not only the document that

\(^4\)Opening presentation at the Fourth International Workshop of Software Specification and Design - Monterey - California, 1987

\(^5\)Again, it is worth pointing out that an implicit requirements point of view does not change the definition of requirements analysis. The contract will be a specification, instead of the requirements.
orients the development, but the predicate to be used at the end of the development to assure that the results are compatible with the user’s desires.

An important observation is that usually, in the business world, contracts do change in their life time. Changing contracts to meet new conditions or to reflect new regulations is a common practice for lawyers. Although there are changes in these contracts, they always are done through a negotiation process, between the parts involved, and the contract is held as the place where all the modifications are done, usually in the form of addenda. Similarly, in the requirements analysis the contract (or its underlying model) should be the place where all future changes are made and kept, in the same fashion as an addendum.

2.4 Conclusion

In this chapter we presented a process-oriented view of requirements analysis. We have made a distinction between modeling and elicitation based on concepts found in the literature, and have described subprocesses of both modeling and elicitation. In the next chapter we will focus our attention on fact-validation, and will examine the approach taken by others in addressing the fact-validation problem.
Chapter 3

Strategies for Fact-Validation

This chapter builds on the definition of fact-validation presented in Chapter 2, and reports and analyzes work by others in the realm of increasing the productivity of early validation.

3.1 The Process of Fact-Validation

Fact-validation, defined in the context of the process of elicitation, has the objective of confirming the facts already found in the process of fact-finding as well as discovering new facts not explicitly produced by the fact-finding process. It is assumed that previous feasibility has been determined, that is in the determination of the universe of discourse (see Chapter 1) a feasibility study is included.

Facts that are neither confirmed nor found in the model are said to be problems. Fact-validation is a process for the detection of problems. The correction of problems is the responsibility of the communication\(^1\) process (see Figure 3.1).

\(^1\)The communication process, as described in the previous Chapter, is the process where the analysts communicate with the users.
* from fact-finding to model, there is the process of modeling

Figure 3.1: Validation Loop
The elicited facts represent functional requirements and non-functional requirements such as performance requirements and quality requirements. We are primarily interested in functional requirements.

The ideal fact-validation strategy would be one that could detect all the problems of correctness, consistency, and completeness (see Figure 3.2) [4]. As mentioned before, this is impossible. The impossibility comes basically from the completeness fallacy [37, 54] as well as from the fact that there is no formal mapping between the universe of discourse and the model. discourse and the model.

Although formal and complete validation is not seen being as feasible, software has been developed over the years. Even with the great number of tales about errors or software misbehavior [see Risk bulletin board], there is no question that over the years considerable progress has been made since the previous reliance on testing the final code with a set of examples.

Different authors have been addressing the issue in the context of requirements analysis: Yeh [105] and Davis [26]. Both point out the usefulness of early prototyping as a way of fact-validation.

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

- Using Informal Checking
- Using Prototypes
- Using Formalisms
- Reusing Domain-Specific Knowledge
Figure 3.2: The Validation Problem
Although some of these strategies were not originally meant to be used in the stage of requirements analysis, there is no reason to believe that they could not be used. A comparison of these strategies assumes that they are applicable in fact-validation.

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 Delta, 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 the quality of the input provided to the Delta computation. To compare the above-listed strategies, a general schema, which separates the issues of fact-finding, fact-validation, and communication is used (Figure 3.3).

3.2 Informal Checking

Several authors [14, 88, 71] point out that 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 informal checking may differ, but they share the lack of automated support and the reliance on the analytical abilities of "good" systems analysts (Figure 3.4).
Figure 3.3: A Framework for Comparing Validation Strategies
Among the tasks proposed by the different methods are: reading and checking, interviews, reviews, checklists, and manual cross-referencing. Overall checklists are pointed out as being important. They are specialized lists based on experience, and are the most usual technique for detecting completeness problems. The informal checking provides a list of problems as a control for the Delta computation.

\[ \Delta = \text{list of problems (information, facts)} \]

An example of such an approach can be found in Ould [71] a report from members of the British Computer Society working group on testing.
3.3 Prototypes

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) [46], while others use an executable specification language [85]. The main idea is that by doing this, it is possible to validate the requirements/specification against the user's expectations (Figure 3.5). The delta computation is controlled by the behavior of the facts.

\[ \Delta = \text{behavior of facts} \text{ (user expectations, facts) } \]
3.4 Formalisms

The approach of using formal methods also can provide fact-validation because it provides the possibility of checking for inconsistencies (Figure 3.6), which consequently controls the delta computation.

\[ \Delta = \text{inconsistencies (facts, information)} \]
Several researchers in Software Engineering have proposed and applied formal methods in the process of specification [11, 49]. An example of that is the work by Wing [102] at Carnegie Mellon University on the specification language Larch.

### 3.5 Domain-Specific Knowledge

The approach of reusing domain-specific knowledge is a more recent one, which 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 analysis. The system 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 (Figure 3.7). In the case of reusing domain-specific knowledge in validation, the Delta computation is controlled by wrong facts and missing facts.

\[ \Delta = \begin{cases} 
\text{wrong facts (domain facts, facts)} \\
\text{missing facts (domain facts, facts)} 
\end{cases} \]

As ongoing research, this strategy is represented by two research groups, one at the University of Oregon [31] and the other at the Artificial Intelligence Laboratory at MIT [77]. Both are working with prototype systems with the capability of criticizing requirements.
Figure 3.7: Reusing Domain-Specific Knowledge
3.6 Conclusions

Although not solving the validation problem, each of these strategies has positive points as well as negative ones. The overall comparison between them is based on a series of their major advantages and disadvantages.

- Using Informal Checking
  
  (+) provides an overall guideline for requirements review

  (-) relies on “good” systems analysts

  (-) does not have a formal scheme to point out problems

  (-) lacks automated support

- Using Prototypes
  
  (+) prototyping capability

  (-) relies on the quality of scenarios

  (-) relies on the criticism of users and systems analysts

  (-) does not have a formal scheme to point out problems

- Using Formalisms
  
  (+) allows pinpointing of inconsistencies

  (-) lacks automatization

  (-) relies on the systems analyst’s ability to detect inconsistencies

  (-) the gap is too big between the problem and the formalism

- Reusing Application Domain-Specific Knowledge
(+) automatable

(+) distinguishes between missing facts and wrong facts

(-) high cost of encoding domain-specific knowledge

(-) depends on the existence of previously encoded domains

In the next chapter we shall present the overall idea of a viewpoint resolution strategy and its research problems. The comparison of the viewpoint resolution strategy with the strategies presented in this chapter will be subject of Chapter 6, which contains the thesis conclusions.
Chapter 4

Viewpoint Resolution

This chapter describes in more detail the idea of applying different viewpoints as a strategy for fact-validation. Although viewpoint has been used in some software engineering methods, as a way of improving the understanding of the universe of discourse, its potential has not been completely fulfilled because of the lack of formality in representing viewpoints. In the following sections we will identify the research problem in using viewpoint resolution as a fact-validation strategy, and will give an overview description of our solution.

4.1 The Existence of Viewpoints

In the task of modeling the users’ expectations in the universe of discourse, a systems analyst may encounter, and usually does, different opinions about the problem being addressed. Different systems analysts, when modeling the users’ expectation in the same universe of discourse, 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 x a process model).
All the above is common knowledge. The important point is that some software engineering methods use this fact with the objective of producing a model that "better" mirrors the users' expectations in the universe of discourse. An example of such a method is CORE [64]. Mullery says:

... The difficulties are often compounded by failure to recognize that what is needed is not one, but several expressions of requirement. The requirement 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.

4.2 The Importance of Viewpoints

As stated in Chapter 1, the principle that more sources of information provide a better understanding of a subject has been used for centuries in court investigation. Different witnesses may have conflicting or complementary recollections.

By using this principle in the process of elicitation, the "chances" of detecting correctness and completeness problems will be greater. To effectively profit from this principle it is necessary to compare and analyze different views.

The analysis and comparison of viewpoints as proposed by Ross' SADT and Mullery's CORE are informal tasks. They are similar to what was analyzed in
Chapter 3 under the classification of "Using Informal Checking." They rely heavily on the "good" systems analyst.

4.3 The Lack of a Viewpoint Resolution Model

Although CORE and SADT advocate the use of viewpoints, neither one has a structured model to explicitly state how to profit from doing so. That is, besides the reliance on inspection procedures and some general guidelines, no model is presented for the use of viewpoints with the results derived from their inspection.

The lack of a proper representation for viewpoints makes it harder to have a procedure for comparing and analyzing views resulting from different viewpoints. In the field of data base, the problem of view integration has some resemblance to the problem of viewpoint resolution. View integration [7] produces a global, conceptual description of a proposed data base. This global description is sought in order to eliminate duplication, avoid problems of multiple updates, and minimize inconsistencies across applications. Since view integration is done after or during the conceptual data base design, the representation used is the conceptual schema. The input to view integration is made of the different schemas and the output is the integrated schema.¹

In the case of data base, view integration departs from an already available model, the conceptual schema. In our case, in the process of fact-validation, we are attempting to validate the facts that should be in the model. With respect to this,

¹Batini et al. note: "The form in which the inputs and outputs exist in an integration system (which may be partly automated) is not stated explicitly by any of the authors considered."
there is a need for a representation that is able to cast different views. These views will be used to validate the facts such that, at the end of the elicitation process, the model "better" reflects the universe of discourse.

Fact-validation is a process that is intermingled with the process of building a model. Viewpoint resolution plays a role in helping to validate the facts that should be present in the model. A representation used for this task is not the same as a representation for the resulting model produced by requirements analysis. So, the basic problem is how to have a representation for viewpoints given the constraints above, such that the viewpoint resolution model (Figure 1.3) could be applied. In the next section we examine what research problems have to be addressed so that a viewpoint resolution strategy can be used in fact-validation. The terms and their definitions used thereafter in the text are as follows.

**Universe of discourse** – 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 the ones demanding a software.

**Viewpoint** – A viewpoint is a standing or mental position used by an individual when examining or observing a universe of discourse. A viewpoint 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 of reality 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.

View - A view is an integration of perspectives. This integration 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.

4.4 The Viewpoint Resolution Model and Its Research Problems

By examining the model presented in Chapter 1 (Figure 1.3) with the definition of elicitation put forward in Chapter 2, it is easy to note that there are some viewpoint resolution tasks belonging to fact-validation and others belonging to communication (Figure 4.1).

Comparing viewpoint resolution with the schema used to compare validation strategies (Figure 3.3) we will note that “identify discrepancies” and “classify discrepancies” are related to the middle box (Validate Facts) and that “evaluation of discrepancies” and “integration of solutions” are related to the Delta computation (Communicate).

Our work does not deal with the communication problems of viewpoint resolution. Our work has focused on the problems of identification of discrepancies and the classification of these discrepancies. By identifying and classifying discrepancies between different viewpoints, we are able to provide an agenda for the process of
Figure 4.1: Viewpoint Resolution in Elicitation

Delta computation. This agenda provides the basic data to be used by the processes of evaluation and integration of viewpoints. The processes of evaluating the differences and negotiating a reconciled solution have not been explicitly addressed by our work.²

²Although not directly addressed, we had the chance of observing the agenda role in the initial investigations into the effectiveness of our proposed strategy.
4.4.1 Research Problems

In the model presented so far, there is no mention of the task which is the prerequisite for a structure viewpoint resolution, that is the organization and representation of different views such that a method of viewpoint resolution could be applied. This characterizes the first problem.

How to formalize viewpoints?

In Figure 4.1, box 1, "identification of discrepancies", is clearly dependent on the proposed formalism and poses the following question:

How to formally compare viewpoints?

Box 2 of figure 4.1, "classify discrepancies", poses the question:

To what extent is it possible to differentiate between wrong facts and missing facts among viewpoints, and how to classify the types of differences among viewpoints?

We could restate our main research problem as the following:

In the process of requirements analysis, the problem involves how to provide an early validation scheme capable of distinguishing between wrong information and missing information, without relying on a previous encoded domain.

In order to be able to address this sort of question, it is necessary that the first two questions above be answered. The next section provides an overview description of the representation proposed, as well as how it is used in the context of fact-validation.
4.5 Proposed Strategy

The proposed strategy comprises a method and a special language, VWPl, to represent viewpoints. The method has procedures to formalize viewpoints as well as procedures to analyze the formalized viewpoints. The language provides the representation which registers the formalism, and makes possible its analysis.

The language is derived from PRISM [52], 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 view or perspective according to a viewpoint.

As observed in Figure 3.3, the process of validating facts is dependent on the process of finding facts. It is assumed that the facts (keywords) are available before the application of the viewpoint resolution strategy. Although a detailed method to perform the fact-finding is beyond the scope of this manuscript, we provide a general description of how this is done. In a previous work [53], we described in detail a fact-finding method oriented towards the construction of productions. The manual for using the VWPl language [58] also provides heuristics in how to cast facts into the proposed language. Next 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.

4.5.1 Method
Figure 4.2: The Proposed Strategy
In Figure 4.2 there is an overall description of the strategy. John and Mary, both systems analysts, perform the task of modeling users' intentions. They both use the VWPI 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.

- A fact-finding method is defined,
- a viewpoint holder with a specific role in the universe of discourse,\(^3\) uses different perspectives and hierarchies in modeling his viewpoints,
- perspectives and hierarchies are analyzed by a static analyzer, and
- 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

\(^3\)Our work used a fixed role of the viewpoint holder, that is the role of a systems analyst.
modeling, uses actor modeling [56]. 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 [93] is to try to attach some “semantics” to the information encoded on the viewpoint language. A specialization relationship between keywords is established as well as a decomposition relationship.

In order to construct a view (see Figure 4.3) 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, 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.

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 Figure 4.1). 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 single individual.

The construction of a perspective is a process in which there is the assumption that the systems analyst uses the concept of “application vocabulary” as described in Chapter 2, 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
Figure 4.3: View Construction
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 with clues to produce a “better” process perspective and hierarchies which are then considered to be the representation of his viewpoint about the problem.

The general guidelines for acquiring heuristics 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: objects facts, actions facts, and agents facts; and
  - define functionality of facts by coding them as productions.

- Objects represent both the “objects” and the *states* of these objects.

- 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.

An important aspect to highlight is the meta use of the viewpoint resolution strategy. The strategy is itself applied in the construction of a viewpoint. That is,
the strategy for validating viewpoints also can be used in checking the perspective models, i.e., the data model, the process model, and the actor model.

In order to use the method as described above a representation in which this information will be cast is necessary. The description of such representation is given below.

4.5.2 The Viewpoint Language

A language was created for representing viewpoints, and its syntax and semantics were defined. This language is derived from PRISM [52], 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. Its usefulness is restricted to the fact-validation process.

Our research has been exploring the use of a rule base language as a way of expressing functionality and constraints for the process of elicitation in requirements analysis. A claim made earlier [53] and the empirical evidence available from our work [53], together with references from the literature [51], allows us to hypothesize that expressing functionality and functionality-related constraints in production rules is simple and fairly easy to use.

The overall idea of VWPI 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 [78], 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 (we call that the output class), or
- remain in working memory (we call that the invariant class).

A fact is a relationship between keywords. The 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 are as given in Figure 4.4. Thus, for each
perspective there is a special combination of types and classes. A fact is composed
of a fact-keyword and fact-attribute. An example is "(book =book-id =author
=title)." In this case we have the fact-keyword "book" and the attributes: "book-
id", "author", and "title."
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.

Below we give an example of VWPl usage. The source of information for this example\(^4\) is taken from the IWSSD library problem [47]. More detail, including the grammar for VWPl is provided in Appendix A.

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."

\(^4\)The objective of the example is just to give an idea of the proposed representation.
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)))

4.5.3 The Static Analyzer

The analysis of different perspectives and different views is achieved by a set of procedures, which perform 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.
Comparing two rule sets only makes sense when there is enough similarity between them. In our case there is a series of factors that leads to that similarity; below we list the most important ones.

- The fact that viewpoint holders are viewing the same universe of discourse.
- The use of a method which stress 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.

The static analyzer proposed and implemented has two major tasks: finding which rules are similar between 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 syntactic oriented. Simple “semantics” hints, such as: using hierarchies and case grammars, are used to enhance the performance of the analyzer.

Being based on the syntactic representation of terms, the analyzer depends basically 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.\(^5\) It is obvious that the static analyzer can not say anything about

\(^5\) The static analyzer also is able to point out inconsistencies inside a view.
two rules that do not have discrepancies but which are not correct with respect to the universe of discourse.

At this point it is interesting to observe the difference between our work and the idea of N-version programming. In both cases different actors encode their view of a problem. In the viewpoint analysis, the problem is the universe of discourse, while in N-version programming the problem is the specification.

The analogy can be stated as follows:

N analysts
producing
N rule bases that capture their views of a universe of discourse.

N programmers
producing
N programs that implement a particular specification.

Concerns that N-version programming is not the best approach to fault tolerance were substantiated by experiments [50] which have found that, in the cases studied, there is no independence in multiversion programming. At first it would appear that the result could be transferred to viewpoint analysis. This is not the case. The supposed analogy does not hold because in the N-version case the dependency is between a combination of inputs and outputs, and in viewpoint analysis we are not interested in results but on the differences between views.
4.6 Conclusion

This Chapter presented the main research problems involved in applying a viewpoint strategy to fact-validation, and gave an overview of the approach taken to face these problems. The objective of our research with respect to the viewpoint resolution model is one of providing an "agenda" such that the integration of different viewpoints has the basic data to work on. We presented an overall method to formalize viewpoints by using a formal language, and we sketched the overall strategy by the automatic analyzer, which identifies and classifies discrepancies. In the next Chapter we provide the detail of this analyzer as well as examples which demonstrate its use.
Chapter 5

Static Analysis of Viewpoints

The objective of this chapter is twofold. First the heuristics used in the perspective static analyzer as well as on the viewpoint static analyzer are described in detail. The second objective is to give an overall account of the case studies performed to assess the effectiveness of the static analyzer. Before plunging into the details of the heuristics used and how the case studies were performed, however, we provide an overview description of the static analyzer as well as an overview of how the detailed information on the heuristics are presented.

5.1 Static Analysis as a Rule Base Comparison Problem

5.1.1 Overview

The analysis of different perspectives and different views is achieved by a set of procedures which perform 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.
The formulation of the problems of finding and classifying discrepancies between different views or perspectives as a rule based comparison problem is possible because it is assumed that there is enough similarity between the rule sets.

In order to find and classify discrepancies between two separate rule sets, it is necessary to establish that a mapping from one rule set to the other. Once a mapping is established, it is possible to evaluate the mapping and then find and classify discrepancies.

In artificial intelligence, the problem of "defining a representational mapping from a known source domain into a novel target domain" [41] is classified as analogy. In our case, although not dealing with a known source domain, we are trying to establish a mapping in the same sense as in analogy. Our mapping, although consisting of two representations in the same general domain (the universe of discourse) does not benefit from a previously encoded domain. On the other hand, our approach to similarity relies more on syntactic analysis than on semantic analysis.

Since analogy deals with establishing mappings between two representations, and our approach tries to establish a mapping between two rule sets, we borrowed a descriptive framework from analogy [41]. This descriptive framework is used to help the structuring and understanding of the problems which need to be addressed in comparing two rule sets.

Hall [41] 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,
Figure 5.1: Comparing Rule Sets

- evaluation of the elaborated analogy, and
- 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 sub-problems (Figure 5-1). 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 likable (most similar) rule pairs.
The elaboration problem is: **given** likable 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 down the search space, 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 out discrepancies (Figure 5.2).

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**
  - incomplete hierarchies with respect to rule facts,
  - missing rules, and
  - missing facts.

- **Inconsistency**
  - contradiction between a fact and the hierarchy,
  - redundancy in the same rule set.\(^1\)

\(^1\)Not implemented.
The static analyzer was implemented as a series of Scheme functions. Scheme is a lexical scoped Lisp developed at MIT by Sussman and Steele [91]. The actual Scheme used in the static analyzer prototype is a Texas Instruments [92] version for the IBM-PC class of machines. The data structures naturally are list based, the most used ones are trees and tables. In Appendix B we give an overview idea of how the prototype is implemented. A technical report [57] gives the detail design of the static analyzer.

### 5.1.2 An Example

Suppose I 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 may use different types of brushes.

A possible encoding of these descriptions in VWPI by two different viewpoint holders is presented below. It is assumed that viewpoint A is produced by systems analyst A and that viewpoint B is produced by systems analyst B.

**VIEW A**

```plaintext
; 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**
(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

Given the two views, the static analyzer produced the following set of messages.

(*TABLE*

(16 (1525 ((SAIL-OK) . 20)) (1020 ((BRUSH) . 21)))
(1 (1020 1 ((OFFICER
   ) OFFICER) (=RANK =NAME) =NAME 2 ((DECK-CLEAN) DECK-CLEANED) (=REG-NUMBER)
   =REG-NUMBER))))

These messages are encoded in the internal form of the tool, and as such should be explained. The general idea is that the messages have a number and that this
number is the head of a list containing the body of facts to which the message applies.

In the above table the message 16, which means that the first viewpoint is missing information, is applied to the rule pairs (15 and 25) and (10 and 20). In the first case the decoding of the message is: “Rule 15 is missing the object sail-ok.” In the second case the decoding is: “Rule 10 is missing the object brush.” The numbers 20 and 21 are the types and classes of “sail-ok’ and “brush” respectively.

The message 1 means that the attributes of facts, perceived as the same, are in disagreement. In our example, the agents “officer” and “officer”, the actions “clean-the-deck” and “clean-the-deck-with-brush”, and the objects “deck-clean” and “deck-cleaned” have different attributes in each viewpoint, although they share the same meaning pairwise. Message 1 identifies contradictory or wrong information between the two viewpoints.

This example, which is explained in more detail in Section 5-6 gives an idea as to what sorts of discrepancies are discovered by this static analysis approach.

5.2 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, and
- 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 in separate Sections. Each heuristic Section (5.3, 5.4, 5.5) deals with the set of heuristics applied to the specific subproblem. In Sections 5.3 and 5.5, the subproblems are further divided and numbered accordingly. For each sub-subproblem in Sections 5.3, 5.5 and for the subproblem of 5.4, the general procedure where the heuristics are applied is presented, followed
by an example and the justification for the heuristic (Figure 5.3). The justification provides the rationale, as well as the basis, for the heuristic.

Within each Section the perspective analysis heuristics are presented first, followed by the viewpoint heuristics, if applicable. In some cases the same set of heuristic applied for perspective analysis is also valid for viewpoint analysis. As a footnote to each heuristic presented, there is an identification of the group of Scheme functions used to implement the heuristic.

5.3 Heuristics Used for the Recognition Problem

The recognition problem is further divided into three subproblems. They are named recognition problem part I, part II and part III. Each one is a different sub-subsection where the problem is stated and the heuristic, the example, and the justification are presented.

5.3.1 The Recognition Strategy, 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. That is, given two sets of production rules, 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.
Figure 5.3: Presentation of the Heuristics
Heuristic

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

Example

Suppose I 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 may use different types of brushes.

Two possible representations of both actions, seen from the actor perspective and from the data perspective are as follows.

\[
\text{ACTOR Perspective Rules}
\]

\[
(10 \ ((\text{agent-officer} = \text{rank} = \text{name}) \ (\text{object-ship} = \text{reg-number}))
\]

\[
\rightarrow
\]

\[
((\text{delete-from} \ \text{wm} (\text{agent-officer} = \text{rank} = \text{name}))
\]

\[
((\text{add-to} \ \text{wm} (\text{clean-the-deck} = \text{reg-number})))
\]

\[
(15 \ ((\text{agent-sailor} = \text{name}) \ (\text{object-boat} = \text{reg-number}))
\]

\[
\rightarrow
\]

\[
((\text{delete-from} \ \text{wm} (\text{agent-sailor} = \text{name}))
\]

\[
((\text{add-to} \ \text{wm} (\text{brings-up-the-sail} = \text{sail-type})))
\]
DATA Perspective Rules

(20 ((agent-officer =name) (object-boat =reg-number)
   (object-brush =type))
   -->
   (($delete-from wm (object-brush =type))
    ($add-to wm (clean-deck-with-brush =type =reg-number))))
(25 ((object-sail =sail-type) (object-vessel =reg-number)
   (agent-sailor =name))
   -->
   (($delete-from wm (object-sail =sail-type))
    ($add-to wm (pull-up-the-sail =sail-type)))))

On the above example, the fact (agent-officer name) of actor perspective has only to be compared with (agent-officer name), instead of being compared to all of the facts in the data perspective.

Justification

Although not changing the order of complexity of the comparison (O(n^2)), the heuristic substantially decreases the constant factor. If we assume that the number of facts in each rule is relatively small with respect to the number of rules, then the fact comparison in each rule comparison could be assumed to be a constant. Since we limit the comparison to only the facts of the same type, and there is at least one fact of each type (there are three types), then the constant factor has at least a threefold reduction.
5.3.2 The Recognition Strategy, Part II

The recognition strategy, part II addresses the following problem:

how to establish the analogy between facts of the different rule sets. That is, given two rules of each rule set and facts to be compared by type, 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 that represent each of the fact keywords. For example the pattern matcher would try to match "clean-the-deck" with "clean-deck-with-brush." The result of the match is a score, the score interval is [1,0]. The matcher used is described below.²

```
pattern-match (fact-rule-a fact-rule-b)
    score <- 0
    divide each fact-keyword into separate words
    for each word in set a
        if word member of verb-phrase b
            then score <- score + 1
    score <- score / # words in longer verb-phrase
    return (score)
```

²This pattern match is implemented by the Scheme function subkxyd in c-actr-d.s.
Example

Given the keywords "clean-the-deck" and "clean-deck-with-brush", the heuristic will produce the score 0.5.

Justification

This scoring, based on a simple pattern matcher, does not use any previous knowledge and relies on the premise that there is a reasonable probability that keywords will be the same or will be of a similar composition. By similar composition we mean verb phrases that are precisely not the same but do share words that are the same. As explained on the previous chapter, this relies on the concept of application vocabulary.

This heuristic mainly has problems with verbs used in different tenses, and with spelling errors or pluralization. Since each perspective is done separately, there is a chance, for instance, of cases such as the following. Suppose we have in one perspective the keyword "warning-signal-to-be-sent" and in the other perspective the keyword "signals-sent." The score result will be 0.2, although the keywords really share the same meaning.

The positive point of this heuristic is that facts usually have distinct verb-phrases such that false scores are less likely to occur, although this is not at all impossible. Let us examine the case of "warning-signal-to-be-sent" and "warning-signal-to-be-received." Although the resultant score is 0.8, the meaning is just the opposite. On the other hand, it is reasonable to argue that if "warning-signal-to-be-sent" is present in the actor perspective and "warning-signal-to-be-received" is
present in the data perspective, then it is likely that "warning-signal-to-be-sent" also will be present in the data perspective such that a score of 1 will be produced.

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.\(^3\)

Heuristic

\[
\text{fine-pattern-match (fact-rule-a fact-rule-b)}
\]
\[
\text{score } \leftarrow 0
\]
\[
\text{divide each fact-keyword into separate words}
\]
\[
\text{for each word in set a}
\]
\[
\text{if word member of verb-phrase b}
\]
\[
\text{or word is root of member of verb-phrase b}
\]
\[
\text{then score } \leftarrow \text{score + 1}
\]
\[
\text{if all words of shorter verb-phrase are members of longer verb-phrase}
\]
\[
\text{then score } \leftarrow 1
\]
\[
\text{else score } \leftarrow \text{score / \# words in longer verb-phrase}
\]
\[
\text{return (score)}
\]

Example

For exemplifying the heuristics of the viewpoint analysis, the same ship scene will be used. The difference here is that it is not two perspectives, but two viewpoints. As such, the VWPI description uses a different syntax/semantics (process perspective).

\(^3\)This fine pattern matcher is implemented by Scheme functions in fmtch.v.s.
VIEWPOINT A

(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)))
)

VIEWPOINT B

(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))}
)
Given the keywords "clean-the-deck" and "clean-deck-with-brush", the heuristic will produce the score 1.

Justification

The objective of using a pattern matcher that is less restrictive than the one used in comparing perspectives the objective is to find similar concepts with a different syntactic form. Relying on the application vocabulary concept, this pattern matcher looks for the roots of words and considers a match to exist if one fact-keyword is a subset of the other fact-keyword.

The comparison of roots is designed to avoid problems of plurality, verb tenses, adverbs, and adjectives. That is, it should be able to consider "clean" and "cleaned" or "word" and "words" as a match. The actual implementation, however, considers a match to exist when a fact-keyword in verb phrase a is a pattern in a fact-keyword in b and vice and versa. As pointed out in the next chapter, the improvement of this matcher is listed as future work.

As such, the same input to the pattern matcher is given to fine-pattern matcher, but the result is completely different. 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.
5.3.3 The Recognition Strategy, 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 likable rule pairs. That is, given a series of scores for keyword matching between facts of two rules, how to establish a scoring scheme of each rule combination to produce the most likable rule pairs.

Heuristic

This heuristic uses a different weigh 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 (maximum is chosen from the number of facts of the same type in each perspective). The scores for 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 scores combinations. The list in descending order exists for each rule of the actor perspective. Below we sketch the main logic of the heuristic.⁴

score (rule-a rule-b)

(let (total-score <-- 0)
  case:
    action: weight-type <-- 5
    object: weight-type <-- 2
    agent: weight-type <-- 3
  for each type
    (let (score-type <-- 0, fact-type <-- 0)

⁴The functions that implement these heuristics are present in c-actr-d.s and lst-mt-a.s.
for each fact

(let (score <- 0)
    score <- score + [weight-type * pattern-match
                      (fact-rule-a
                       fact-rule-b)]
    fact-type <- fact-type + 1)
    score-type <- score / fact-type
    total-score <- total-score + score-type)
append total-score to a list by actor rule
return (list)

sort (list)
    sort list in ascending order
    return (sorted list)

Example

Using the small example outlined in the part I of the recognition strategy, the
heuristic above produces the following list:

(((10 ((20 3.75) (25 2.375))) (15 ((25 4.375) (20 2.25)))).

This list tells us that the actor rule number 10 has a similarity score of 3.75 with
the data perspective rule 20 and a score of 2.375 with the data rule 25. The same
kind of information is available for the actor rule 15.
Justification

The first part of the heuristic is the assignment of weights to each fact type. The weights come from the metaphor made with the sentence structure subject, verb, and object. As such, there is no question which type has the higher weight. Textbooks in Natural Language Understanding [30, 93] point out that the verb carries more meaning than the other parts of the sentence. The subject (the instigator), because of its direct influence on the meaning provided by the verb, is then considered to be the second higher weight. The fact that agents and actions are in less number than objects in the rule horizontal structure confirms the weight hierarchy between types.

According to these principles the action has weight 5, the agent has weight 3, and the object has a weight of 2. The question then, is how effectively are these weights, i.e., why not have scores of 3, 2, and 1. The main reason for the 5, 3, 2 weights is the idea of maintaining a close proportion between the fact types. As such, action : agent has a proportion of $\frac{5}{3}$, action : object's proportion is $\frac{5}{2}$, and agent : object has a proportion of $\frac{3}{2}$. Therefore the action is 1.7 times more important than the agent and 2.5 times more important than the object, while the agent is 1.5 times more important than the object. With the experiments to date, there is reason to believe that this heuristic is fair.

The second part of the heuristic deals with the number of facts of each type that are present on the rule. The division of the score (weight-type * patter-match) by the number of facts in the compared type is necessary to avoid unfairness. A simple example demonstrates this.

actor rules data rules
rule 1: (object-a =j)      rule 2: (object-a =j)
      (object-b =j)      (object-b =j)
      (object-c =j)      (object-c =j)

If no division is made then the pair (1, 2) has score of 6 and the pair (1, 5) score of 3. With the division, (1, 2) has a score of 2 and (1, 5) score 3.

The third general heuristic is the addition of the score types to produce the rule combination score as well as the sorting of the combined scores. A descendent sort is necessary to promote those data rules with higher score to the top of the actor rule/data rule list.

Heuristic

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 as well as add important missing ones.\(^5\)

\[
\text{score (rule-a rule-b)}
\]

\[
\text{(let (total-score <-- 0)}
\]

\[
\text{case:}
\]

\[
\text{action: weight-type <-- 5}
\]

\[
\text{object: weight-type <-- 2}
\]

\(^5\)The Scheme functions that implement these heuristics are present in find-syn.s, cp-1-2.s and lst-nt-v.s.
agent: weight-type <-- 3
for each type
  (let (score-type <-- 0, fact-type <-- 0)
    for each fact
      (let (score <-- 0, weight-ty <-- weight-type, resul <-- 0)
        if weight-ty = 5
          then weight-ty <-- weight-ty * 1.25
        resul <-- fine-pattern-match (fact-rule-a fact-rule-b)
        if resul = 1 or member-dic (keyword-a keyword-b)
          then weight-typ <-- weight-ty * 1.75
        score <-- score + [weight-ty * resul]
        fact-type <-- fact-type + 1)
    score-type <-- score / fact-type
    total-score <-- total-score + score-type)
  append total-score to a list by actor rule
return (list)

sort (list)
  sort list in ascending order
return (sorted list)

member-dic (keyword-a keyword-b)
  if the pair (keyword-a keyword-b)
    is member of the dictionary of similar actions
    then #T
    else #F

Example
Using the ship example, the above heuristic produces the following list.
Justification

The introduction of weight multipliers \((1.75)\) and \((1.25 \times 1.75)\) has the objective of reducing misleading scores produced by a less restrictive pattern matcher. This restriction appears to be more fair, not only for giving more importance to action matches, but also for the object and agent matches as well.

5.4 Heuristics Used for the Elaboration Problem

Picking the first pair on 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, and
- rules that may condense matches, thus generating false similarities.

5.4.1 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.
For each actor perspective list (rules in the data perspective sorted by higher scores), 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.\(^6\)

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.

elaboration (list-of-actor-data-rules-combination-score)

for each actor rule

elaboration-1 (actor-rule list-of-data-rules)

elaboration-2 (actor-rule list-of-data-rules

list-of-most-probable-pairs)

if there exists a data rule not used in the elaboration-1

then <message #19> ; actor perspective is missing

rule with respect the actor

perspective

elaboration-1 (actor-rule list-of-data-rules)

(let (tagged, and saved scores are lists)

get the first two rules.

case:

1) if there are no data rules

then <message #19> ; data perspective is missing

\(^6\)The Scheme functions that implemented this strategy are present in listmg.\(\$\) and in fn-mtec-a.\(\$\) (functions fnatch-ac).
rule with respect to the actor perspective

2) if both pairs have score < minimal score (1.4)
   then <message #18>

3) if both are tagged and (# of actor rules * 1.5) <
   # data rules and there is not a
   next untagged and score first tagged * 1.3 <
   saved tagged score
   then <message #18>

4) if both tagged and score of first tagged >
   saved tagged score and score of second tagged >
   saved tagged score
   then update saved scores
   get the two tagged.

5) if both tagged and score of first tagged >
   saved tagged score and (score of second tagged * 1.15) <
   saved tagged score and there is a next untagged
   then update saved score
   get the first tagged and the next untagged.

6) if both tagged and (score of first tagged * 1.15) <
   saved tagged score and score of second tagged >
   saved tagged score and there is a next untagged
   then update saved score
   get the second tagged and the next untagged.

7) if both are tagged and (# of actor rules * 1.5) <
   # data rules * 1.2 and there is a next untagged
   and score of first tagged (not >) 2 * score of next
   untagged
   then get the two next untagged.

8) if both tagged and there is a next untagged and
   2nd tagged (not >) 2 * score the next untagged and
first tagged (not >) 2 * 2nd tagged
then get the 2nd tagged and the next untagged.

9) if first tagged and 2nd untagged and
(# of actor rules * 1.5) < # data rules and
first tagged (not >) 2 * 2nd untagged and score of
first tagged < (saved tagged score * 1.15)
then get 2nd untagged.

10) if both tagged and (score of first tagged * 1.15) <
saved tagged score and (score of second tagged * 1.15) <
saved tagged score and there is a next untagged and
first tagged (not >) 2 * next untagged
then get the next untagged.

11) if second data rule score < 1.4 (minimal score)
then get only the first data rule.
otherwise)
update the score of the selected data rules
get the first and second rules
update the list of tagged(data rules already used)
return (list-of-most-probable-pairs messages) )

elaboration-2 (actor-rule list-of-data-rules list-of-most-probable-pairs)
(let ((misses <-- 0))
if one pair is in perfect match on the action type
and the other is not
and the pair in perfect match has not more misses
than the other
then select the pair in perfect match
else select both pairs
return (list-of-most-probable-pairs) )
5.4.2 Example

The example below is the comparison of author’s actor and data perspective of the book inventory example.

The recognition strategy produced the following list.

(1 ((1 4.66666666666667) (6 4.3333333333333) (2 3.83333333333333)
(7 3.6) (4 3.26190476190476) (5 3.0952380952381) (8 1.6) (3 1.6)
(9 0.866666666666667) (11 0.63333333333333) (10 0.53333333333333)
(12 0.444444444444444))

(2 ((2 4.66666666666667) (7 4.22222222222222) (1 3.83333333333333)
(6 3.6) (4 3.26190476190476) (5 3.0952380952381) (8 1.5) (3 1.5)
(10 0.63333333333333) (11 0.63333333333333) (9 0.53333333333333)
(12 0.444444444444444))

(3 ((4 4.66666666666667) (5 4.16666666666667) (1 3.26190476190476)
(2 3.26190476190476) (6 2.92857142857143) (7 2.92857142857143) (8 1.5)
(3 1.5) (11 0.63333333333333) (9 0.53333333333333) (10 0.53333333333333)
(12 0.444444444444444))

(4 ((3 4.66666666666667) (8 4.66666666666667) (5 1.66666666666667) (6 1.5)
(7 1.5) (1 1.6) (2 1.5) (4 1.5) (12 0.86111111111111111)
(11 0.63333333333333) (9 0.53333333333333) (10 0.533333333333333))

(5 ((7 2.33333333333333) (1 2.) (2 2.) (6 2.) (4 1.85714285714286)
(5 1.85714285714286) (8 1.5) (3 1.6) (10 0.791666666666667)
(11 0.66666666666667) (9 0.66666666666667) (12 0.633333333333333)))

The application of the elaboration strategy produced the following list of most probable pairs.

(((1 1 6) (2 2 7) (3 4 5) (4 3 8) (5 7 1)))
The case of actor rule 5 is interesting. Since both 7 and 1 data rules were already used and the cardinality difference was greater than 20%, the heuristic did not succeed in getting the next untagged data rule, mainly because in this case the next untagged rule would be (10 0.7916666666666667), which is too low a score to be considered.

It also produced the following messages.

(*table* (19 (12) (11) (10) (9)))

(*table* (12 (19)) (11 (19)) (10 (19)) (9 (19)))

The first list points out that message 19 - "actor perspective is missing a rule with respect to the actor perspective" - was issued for the data rules 9, 10, 11, and 12. The second list is just an inverted version of the first list.

5.4.3 Justification

The heuristics for the elaboration strategy were designed to minimize the problems related to misleading scores in such a way that the combinations or pairs chosen have more fidelity. Their performance depends on the set of conditions covered by the case structure shown above, and is strongly based on the following assumptions:

- A rule in one rule set should have one rule that is similar to it in the other rule set.
- The actor perspective has fewer rules than the data perspective.
• In general, the number of rules in the data perspective rules is at most twice
  the number of rules in the actor perspective

• By selecting, when applicable, the two most probable analogies in the data
  perspective, for each actor perspective rule the chance of missing a possible
  pair is reduced.

• Scores for a chosen pair have to be at least equal to the minimum score, thus
  the existence of pairs which do not conform to this criteria indicates that one
  perspective is missing information which is present in the other perspective.

Although the first assumption states that in general there is a one to one
relationship between rules, it is the case that sometimes more than one rule in the
first rule base and more than two rules in the second rule base should be evaluated
together. In this case the heuristics provided here will not evaluated all the rules
 together. A consequence of this is that we may not be detecting potential conflicts
and missing information or we may issue conflicts and missing information messages
that could have been avoided.

For each of the case conditions used in the heuristics, there is a logical expla-
nation which is listed per case condition number.

• Cases 1, 2, 3, and 11: identification of missing information.

• Cases 4, 5, and 6: identification of a misleading tagged score, i.e., a data rule
  already used in combination with an actor rule, but which has a score strong
  enough to be repeated.
- Cases 7, 8 and 9: identification of misleading high scores for already tagged data rules, as well as avoidance of false missing information due to the cardinality of the data perspective rule set to be superior to the cardinality of the actor set. These heuristics try to guarantee that in the case where there is a possibility of more than one data perspective rule for each actor perspective rule, other matching combinations besides the ones with higher scores are considered. This is important, since there are situations where rules that are too general concentrate matches and do not allow other rules, with a possibility for a better analogy, to be considered.

- Case 10: avoids the selection of tagged rules that have a score inferior to the combination already used. It avoids false similarities as well as rule scores which are significantly different from each other.

The minimal score is defined to be 1.4, because there should be at least a 40% match on the object type, a 40% match on the agent type, and 40% on the action match. Since an action, in the case of "error" is really an object, the weight used in this case is the object weight. Consequently, the minimum score is:

\[ ms = (3 \times 0.4 + 2 \times 0.4 + 2 \times 0.4). \]

In the case of heuristics that issue a critique (message), i.e., rules not found in the other perspective, the message's objective is to point out that it is possible for a whole concept (rule) to be missing from the other set of rules.

The elaboration-2 heuristics when given two probable pairs, further narrow the search space by selecting the pair in which there is a perfect match of the fact keyword of the actor type, and which has fewer mismatches.
5.5 Heuristics Used for the Evaluation Problem

The evaluation problem is further divided into three subproblems. They are named evaluation problem part I, part II and part III. Each one is addressed at a different sub-subsection where the problem is stated and the heuristic, the example, and the justification are presented.

5.5.1 The Evaluation Strategy, 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. That is, given the most probable pairs, 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 (case grammar). 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 (12), the agent-input (11), the object-invariant (12), the object-input (11), and the action-output (30).
The comparison is made using the pattern matcher already described, and a list of perfect matches, as well as lists of non-matched facts are produced. The heuristics also take a closer look at the attributes of facts that have a perfect match and compare them.\textsuperscript{7}

\texttt{id\,identification-of-perfect-matches (actor-rule data-rule)}

\texttt{(let (perfct-mtchs, not-mtched-actr, and not-mtched-data be lists)
}

\texttt{case:
}

\texttt{1) is actr-agt-i member of data-agt-in
}

\texttt{\#T - update-append perfct-mtchs
}

\texttt{\#F - insert facts in not-mtched-(actr, data)
}

\texttt{2) is actr-agt-in member of not-mtched-data:rl.12
}

\texttt{\#T - update-append perfct-mtchs
}

\texttt{delete not-mtched-data:rl.12
}

\texttt{\#F - insert actr-agt-in in not-mtched-actr
}

\texttt{3) is actr-obj-in member of data-obj-i
}

\texttt{\#T - update-append perfct-mtchs
}

\texttt{\#F - insert facts in not-mtched-(actr, data)
}

\texttt{4) is not-mtched-actr:rl.22 member of data-obj-in
}

\texttt{\#T - update-append perfct-mtchs
}

\texttt{delete not-mtched-actr:rl.22
}

\texttt{\#F - insert data-obj-in to not-mtched-data
}

\texttt{5) is actr-act-o member of data-act-o
}

\texttt{\#T - update-append perfct-mtchs
}

\texttt{update-append data-act-o in perfct-mtchs
}

\texttt{\#F - insert facts in not-mtched{(actr, data)}
}

\texttt{for facts in perfect match compare the attributes
}

\texttt{if actor-attribute different from data-attribute
}

\textsuperscript{7}The Scheme functions that implement the strategy are the \texttt{fmatch-ab series}, present in \texttt{fn-mtc-a.s.}
then <message #1> ; the attributes of a perfect
; match do not correspond

member (actr-type-class data-type-class)
  score <- partial-match (kywd-actr kywd-data)
  if score = 1
    then return (#T)

Example

In the example seen in the recognition strategy section, we selected the pair
actor rule 15 and data rule 25 and submitted them to the heuristics. The result is:

- perfect-matches (*table* (1 (3) (2) (1 (((agent-sailor) (=name)) ((agent-sailor) (=name)))))
- not-mtched-actr (*table* (1 (30 ((brings-up-the-sail) (=sail-type))) (22 ((object-boat) (=reg-
  number))) (12) (11))))
- not-mtched-data (*table* (1 (30 ((pull-up-the-sail) (=sail-type))) (22 ((object-vessel) (=reg-
  number))) (21 ((object-sail) (=sail-type))) (12))).

In other words, the only perfect match found was the agent, "agent-sailor." The other
facts were classified as either those not found in the data perspective
(not-mtched-actor), or those not found in the actor perspective (not-mtched-data).

---

\(^8\)The result from the elaboration strategy for the ship example was: "((10 20 . 25) (15 25 .20))"
Justification

The major guideline used in this set of heuristics is the horizontal structure of the rule base as established in VWPl (viewpoint language). That is, a rule is composed of facts, facts are composed of keywords and attributes, and facts have types and classes. The facts are objects, agents, and actions. Objects are compared with objects, agents are compared with agents and actions are compared with actions. Since there may be two classes of agents and objects (cases 2 and 4), the comparison is made for each.

Again the measure of similarity is based on the pattern matcher defined in the recognition strategy. For those facts considered to be in perfect match, there is a critique with respect to the matching of their attributes. If the attributes are different, message 1 is issued.

In the case of viewpoint analysis, the heuristics are very similar to the ones used in the perspective analysis. The differences include the use of the process perspective structure, the use of the finer pattern matcher, and the use of the action dictionary.

An example will illustrate the use of the action dictionary. In the example seen in the recognition strategy, we selected the pair viewpoint a, rule 15 and viewpoint b, rule 25 and submitted it to the heuristics. The result is:

- perfect-matches (*table* (1 (2 (((sail-is-up) (=sail-type)) ((sail) (=sail-type))) (3 (((brings-up-the-sail) (=sail-type)) ((pull-up-the-sail) (=sail-type))) (1 (((sailor (=name)) ((sailor) (=name)))))

- not-matched-frstvw (*table* (1 (20) (21) (22 ((boat) (=reg-number))) (31) (12)))

The result from the elaboration strategy for the ship example is: "(((10 20) (15 25)))".
The actions are found to be in perfect match as well as the fact-keyword "sail-is-up" and "sail". The actions are found to be in perfect match because they belong to the action dictionary.

5.5.2 The Evaluation Strategy, 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. That is, given the set of nonmatched rule set-a facts and the non matched rule set-b facts, how to choose the most probable matches between the facts.

Heuristic

The set of heuristics for this strategy is similar to the one used for the list of rules at 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 result, between candidate-pairs (which have some similarity) and missing-information (facts that did not find a pair).10

10The Scheme functions that implement the strategy are the \texttt{match-d} series, present in \texttt{fn-mtc-a.s}. 
identification-fact-pairs (not-matched-actr not-matched-data)

(let (already-used, list-pair, candidate-pairs, 

and missing-info be lists) 

while type in both lists are equal 

compare facts from not-matched-actr list with 

not-matched-data list 

score <-- partial-match (kywd-actr attr-actr 

kywd-data attr-data) 

append to list-pair the facts identified with 

type-class and the pair score.

end while 

sort list-pair 

for each pair in the list-pair 

check to see if it is a possible pair 

#T append each fact of the pair 

to a list of already-used 

append the pair to the candidate-pairs list 

append the non used facts to missing-info list 

return (candidates-fact-pairs, missing-info-from-rule) )

partial-match (keyword-a attributes-a keyword-b attributes-b) 

(let (score <-- 0) 

divide each fact into separate words 

for each word in set a 

if word member of verb-phrase b 

then score <-- score + 1 

for each attribute in set a 

if attribute member of attributes-b 

then score <-- score + 1 

score <-- score / (# words in longer verb-phrase + 

# attributes in longer attributes list)
return (score) )

Example

Using the same example as in part II, i.e., actor rule 15 and data rule 25, the further processing of the nonmatched facts will produce both the facts that are candidates to be a pair and the facts that are found to be missing from each rule.

The result using the lists shown in part II is as follows.

candidate-pairs:

(*table* (3 (((30 brings-up-the-sail =sail-type)

30 pull-up-the-sail) =sail-type) 0.8))

(2 ((((22 object-boat =reg-number)

22 object-vessel =reg-number) 0.6666666667)))

missing-info:

(((object-sail) .21)))

The actions "brings-up-the-sail" and "pull-up-the-sail" as well the objects "object-boat" and "object-vessel" will be analyzed further.

The object "object-sail" was found to be missing from the actor rule 15.

Justification

The heuristic used above is similar to the scoring heuristic used in the recognition problem (part III); scores are higher for facts that have a better syntactic matching. In this case, however, since the search space was considerably smaller, the partial match score not only considers keyword matches, but the matching of
attributes as well. The use of the attributes is supposed to help in the identification of similarities.

Based on the scores, candidate pairs are assigned and facts that are not selected are appended to a list of prospective missing information. Both the candidate pairs and the missing information lists, will be analyzed to determine discrepancies.

The weak point of this set of heuristics is that it does not take into consideration that a candidate pair may well be a perfect match between facts. As such, different verb tenses, plurals, and spelling can lead to false discrepancies.

In the case of viewpoint analysis the only differences are the use of the finer pattern matcher and the use of the action dictionary.\(^{11}\)

### 5.5.3 The Evaluation Strategy, Part III

The evaluation strategy, part III addresses the following problem:

how to differentiate among inconsistencies, wrong information, and missing information. That is, given the list of candidates pairs, the list of missing information, and hierarchies how to differentiate between inconsistencies, wrong information and missing information among facts.

\(^{11}\)The Scheme functions that implement the strategy are the `match-d` series in `fn-mtc-v.s`. 
Heuristic

The heuristics for this strategy are the ones that ultimately define if the lack of similarity between facts is determined by a problem of missing information or wrong information. The heuristics also detect inconsistencies.

A series of heuristics for using hierarchies in helping to determine the discrepancies are listed, as well as other heuristics based on score and on type-class.\(^{12}\)

\[\text{analyze-facts-from-candidates-pairs (actor-fact data-fact)}\]

\[\text{case:}\]

1) if actor:kywd is a root in hierarchy and
   data:kywd is a leave in hierarchy
   then is-a: <message #2> ; specialization
   ; inconsistencies
   parts-of: <message #7> ; decomposition
   ; inconsistencies

2) if actor:kywd is a leave in hierarchy and
   data:kywd is a root in hierarchy
   then is-a: <message #3> ; specialization
   ; inconsistencies
   parts-of: <message #8> ; decomposition
   ; inconsistencies

3) if actor:kywd is a leave in hierarchy and
   data:kywd a leave in hierarchy and
   both roots are equal
   then is-a: <message #4> ; facts in contradiction
   parts-of: <message #9> ; facts in contradiction

4) if actor:kywd is a leave in hierarchy and

\(^{12}\)The Scheme functions that implement the strategy are the fnmatch-e and fnmatch-h series, present in fn-mtc-a.s.
data:kywd is a leave in hierarchy and
and roots are not equal

then is-a: <message #5> ; missing facts
parts-of: <message #10> ; missing facts

5) if actor:kywd is member of hierarchy and
data:kywd is not a member of hierarchy

then is-a: <message #6> ; is-a is incomplete
parts-of: <message #11> ; parts-of is incomplete

6) if data:kywd is member of hierarchy and
actor:kywd is not a member of hierarchy

then is-a: <message #12> ; is-a is incomplete
parts-of: <message #13> ; parts-of is incomplete
otherwise)
probability (actor-fact data-fact)

probability (actor-fact data-fact)

if there is only one combination in the type and
the class and type are equal for both facts
then <message # 20> ; facts in contradiction
else if score >= .5
then <message #14> ; score-percentage of
; contradiction
else <message #15> ; score-percentage of
missing information

analyze-facts-from-missing-info (actor-fact data-fact)

if there is a possible combination of types
then case:

1), 2) 3), 4), 5) and 6) as in

analyze-facts-from-candidate-pairs
otherwise)

if there is only one combination in the
  type and the class and type are equal
  for both facts
  then <message # 7> ; facts in contradiction

if actor-fact

then <message #17> ; missing fact from
  ; data perspective

if data-fact

then <message #16> ; missing fact from
  ; actor perspective

Example

Using the ship example and the hierarchies below,

(*table* (2 (ship deck sail))) ; parts-of-hrrchy

(*table* (1 (agent agent-officer agent-sailor))

(2 (object-vessel object-ship object-boat))) ; is-a-hrrchy

we got the following table of messages:

(*TABLE*)

(3 (1525 (((22 OBJECT-BOAT =REG-NUMBER) 22 OBJECT-VESSEL =REG-NUMBER) .
  0.666666666666667)) (1025 (((22 OBJECT-SHIP =REG-NUMBER) 22 OBJECT-VESSEL
  =REG-NUMBER) . 0.666666666666667)))

(16 (1520 ((OBJECT-BRUSE) . 21)) (1525 ((
  OBJECT-SAIL) . 21)) (1025 ((OBJECT-SAIL) . 21)) (1020 ((OBJECT-BRUSH) . 21)))

(20 (1520 (((30 BRINGS-UP-THE-SAIL =SAIL-TYPE) 30

CLEAN-DECK-WITH-BRUSH =TYPE

=REG-NUMBER) . 0)) (1525 (((30 BRINGS-UP-THE-SAIL =SAIL-TYPE) 30

...
Message 3 is reporting on specialization inconsistencies in rules 15 and 25 and rules 10 and 25 between "object-boat" (rule 15) / "object-ship" (rule 10), and "object-vessel", as confirmed by the is-a hierarchy. Message 16 tells us that "object-sail" is missing from the actor rules 15 and 10 and "object-brush" is missing from rule 10 and rule 15. Message 20 points out that there is a contradiction between the actions of each combination pair. Message 4 reports on the contradictions between agents for the rules combination 15 and 20 and 10 and 25, and a contradiction in the combination 10 and 20 regarding "object-ship" and "object-boat". Message 1 reports that "agent-officer" has conflicting attributes for the pair actor rule 10 and actor rule 20.

The ship example is interesting, since the elaboration strategy is not able to distinguish between the ideal most probable pairs, so each rule is checked against the other. This case happened mainly for two reasons:

- the same cardinality for the rule sets, and
- the different action type keywords used.
As already stated, two cases occurring together are not expected to be common. However, even in the case that the elaboration strategy does not produce the ideal output, the distinction between contradictions and missing information (evaluation strategy) is effective.

**Justification**

The analysis to distinguish between wrong information, inconsistencies, and missing information is accomplished by the two sets of heuristics.

- analyze-facts-from-candidate-pairs, and
- analyze-facts-from-missing-info.

Both sets of heuristics use the hierarchies to help in the identification and classification of discrepancies. That is, given two facts from the different perspectives, they can either be in contradiction or each perspective is missing the other perspective fact. The hierarchies provide semantic information complementary to the perspective's rules, and are the target of consistencies checks.

The reasoning for the heuristics described above with relation to the hierarchies are as follows.

- For cases 1 and 2: if both facts are present in the hierarchy and one is the root and the other a leave, then there is evidence to argue that there is a specialization or decomposition inconsistency.

- For case 3: if both facts are leaves of the same root, i.e., under the same root the actor keyword and the data keyword are different, then there is a
indication that a contradiction exists. In other words, if both facts are linked to the same more general concept then there is a better chance that the facts are in contradiction rather than that each one be missing from the others perspective rule.

- For case 4: if both facts are leaves of the type hierarchy and their roots are different, there is a strong indication that both perspectives are missing each other's facts, since they belong to different concepts in the hierarchies.

- For case 5: this heuristic only points out if the hierarchy is complete with respect to the facts being considered.

It is important to note that the critique using hierarchies is done for each one of the hierarchies (e.g., is-a, parts-of) and does not combine the two for performing the critiques. If a message is issued from the is-a critique, the parts-of critique is skipped.

The hierarchies heuristics are used only if at least one of the fact-keywords belongs to the hierarchies. Thus, hierarchies are used as a complement and, depending on their quality, may improve the job of differentiating between types of problems. If hierarchies are not available, the distinction between problem types is based on the following.

- If both facts have the same class type and are the only combination on the class type, then there is a contradiction between facts. The reason for this is that because both facts have the same type and the same class, there is a high indication that they are in conflict.
• If the facts belong to the candidate-pairs list and no message has been issued yet, with the exception of messages 6 and 11, then the determination of contradiction and missing information is based solely on the score produced by Part II of this strategy (evaluation). The score is properly transformed into percentages such that the message indicates the chances of the message being close to the real situation. The problem with this heuristic is that it is too dependent on the syntactic forms, and as such there is a reasonable chance that it sometimes provides a message that is just the opposite of the real case.

• If the facts belong to the missing-information list and no message has been issued by the hierarchies' heuristics, with the exception of messages 6 and 11, then the facts are considered to be missing information from the other perspective, i.e., is the messages 16 or 17 are issued.

The overall justification of these heuristics described above is the difficulty of differentiating between wrong information and missing information based solely on the verb phrase present in the keywords. Although keyword comparison reflects some semantic information (semantic grammars and case grammars), it has its limitations. The use of hierarchies (semantic information), makes it possible for distinctions with better performance, because the hierarchies may present useful clues not expressed in the rules descriptions. The word may is emphasized since there is no complete guarantee that this will always be the case. A counter-example is a hierarchy that presents inconsistencies. For example, suppose that an element "car" in the is-a hierarchy with root "transportation" and "truck" is in the is-a hierarchy with root "heavy-vehicles." A keyword match will indicate missing information (which is not the case), and a hierarchy-driven critique (case 4) also
will give the same incorrect diagnosis. If the hierarchies instead had “truck” as a leave of “transportation”, the correct message would have been produced (case 3).

There is a considerable difference between the perspective strategy and the one used for viewpoints. When comparing viewpoints, there are four hierarchies that need to be considered: the is-a hierarchy for both the first and second viewpoint, and the parts-of hierarchy for both the first and second viewpoint.

Heuristic

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.\(^{13}\)

analyze-facts-from-candidates-pairs (fact-a fact-b)

if both in hierarchy

then check-both-hierarchies (fact-a fact-b)

else check-one-hierarchy (fact-a fact-b)

check-both-hierarchies (fact-a fact-b)

get root of each hierarchy (is-a or parts-of)

try to establish relationship

test each leave of hierarchy against each other

sore-hrrcy ← summation of score using very-fine-mtch / average number of entries

if score-hrrcy > .79

then case:

\(^{13}\)The Scheme functions that implement the strategy are the fmtch-e and fmtch-h series, present in fn-mtc-v.s.
1) if both views are root
    then semi-perfect-matches
2) if both are leaves
    then semi-perfect-matches
3) if kwid-1 is a leave and kwid-2 a root
    then #3 #8 ; contradictory facts
4) if kwid-1 is a root and kwid-2 a leave
    then #25 #26 ; contradictory facts
otherwise)
    probability (actor-fact data-fact)
else    probability (actor-fact data-fact)

check-one-hierarchy (fact-a fact-b)

(let (score-hrrchy <-- 0)
  if one is in hrrchy (is-a or parts-of)
    then try to establish relationship
      test the fact not in the hierarchy against each
      element in the other's fact hierarchy
    score-hrrchy <-- summation of score using very-fine-mtch /
      average number of entries

  if score-hrrchy > .79
    then case:

      5) kwid1 is a root
         then #21 #22 #6 #11 ; missing information and
         ; warning of incomplete hrrchy

      6) kwid1 is a leave
         then #21 #22 #6 #11

      7) kwid2 is a root
         then #23 #24 #13 #12

      8) kwid2 is a leave
         then #23 #24 #13 #12
otherwise)

probability (fact-a fact-b)

else probability (fact-a fact-b)

probability (actor-fact data-fact)

if there is only one combination in the type and
the class and type are equal for both facts
then <message # 20> ; facts in contradiction
else if score >= .6
then <message #14> ; score-percentage of
contradiction
else <message #15> ; score-percentage of
missing information

analyze-facts-from-missing-info (actor-fact data-fact)

if there is a possible combination of types
then case:
1), 2), 3), 4), 5), 6), 7) and 8) as in
analyze-facts-from-candidate-pairs
otherwise)

if there is only one combination in the
type and the class and type are equal
for both facts
then <message # 20> ; facts in contradiction
if actor-fact
then <message #17> ; missing fact from
data perspective
if data-fact
then <message #18> ; missing fact from
actor perspective
Example

Using the ship example and the hierarchies below

(*table* (2 (ship deck sail))) ; parts-of-hrrchy of viewpoint-a
(*table* (2 (ship deck sail))) ; parts-of-hrrchy of viewpoint-b

(*table* (1 (crew sailor officer)))
   (2 (vessel ship boat))) ; is-a-hrrchy-viewpoint-a
(*table* (1 (crew sailor officer)))
   (2 (ship vessel boat))) ; is-a-hrrchy-viewpoint-a

we got the following tables of messages:

(*TABLE*

   (16 (1525 ((SAIL-OK) 20)) (1020 ((BRUSH) 21)))
   (1 (1020 1 ((OFFicer
   ) OFFicer) (=RANK =NAME) =NAME 2 ((DECK-CLEAN) DECK-CLEANED) (=REG-NUMBER)
   =REG-NUMBER)))

(*TABLE* (1525 (16)) (1020 (16) (1)))

Message 16 was issued for reporting that the fact "sail-ok" was missing from viewpoint-a rule 15 and that "brush" was missing from rule 10. Message 1 reports a contradiction (wrong information) on attributes, and was issued for the following facts: "deck-clean" and "deck-cleaned", "officer" and "officer", and "clean-the-deck" and "clean-the-deck-with-brush."

As the table of messages reflects, the facts "boat" in rule 15 and "vessel" in rule 25 were considered to be semi-perfect matches. That is, although they conflict according to the pattern matcher, an analysis of the hierarchy concludes that they
should be representing the same concept. The same event occurred between rules 10 and 20 for "boat" and "ship".

Justification

Although sharing some resemblance with the heuristics used in the perspective analysis, the heuristics for differentiating between: wrong information and missing information are different. The difference is greater with respect to the hierarchies heuristics.

Below we provide a rationalization of each of the cases presented.

- For case 1: if both facts of both viewpoints are roots in their respective hierarchies and the hierarchies are related (score > .79), then there is reason to believe that they are semi-perfect matches.

- For case 2: the same argument holds as above, with the difference that they are leaves.

- For case 3: if the hierarchies of both facts are related but one is a root and the other is a leaf, the conflict of decomposition or specialization is an indication that there is a contradiction between them, perhaps one is wrong and the other is correct.

- For case 4: the same rationalization holds as for case 3.

- For cases 5, 6, 7, and 8: if one fact belongs to a hierarchy and the other does not, and if the fact not belonging to a hierarchy has a score greater than .79 with any leave or root of the hierarchy to which the other fact belongs, then
based on a conflict of decomposition or specialization, there is an indication that both facts are missing in each other's viewpoint.

The other heuristics are the same as those applied in the perspective analysis.

5.6 Case Studies, Results and Observations

Assuming that the engine has implemented the set of heuristics of the static analyzer, the next step is to have some demonstration of the effectiveness of those heuristics in determining discrepancies between views and perspectives.

The test to verify that the engine implemented the heuristics was conducted with small examples that were then compared to the application of the heuristics manually. This test, although not exhaustive since we are dealing with a prototype, has shown that the engine carries out the designed heuristics.

An empirical demonstration that the static analysis heuristics are effective in finding missing information and wrong information was carried out by using two different sorts of problems and four different people. The case studies confirm the effectiveness of the static analyzer in finding discrepancies and classifying them. Although we did not study the effectiveness of the missing information and wrong information as an agenda, we observed that in two of the cases studied there was a confirmation of our "agenda" hypothesis (see Sections 5.6.3 and 5.6.4). The case studies also point out two interesting problems related to viewpoint analysis in general and to the implemented heuristics.
In the next Subsections we will overview each of the case studies, providing a description of the problem addressed, the actors involved, the rules produced, and the conclusions drawn from each case. A complete report [55] gives all the rule sets used in the case studies, as well as the tables of messages produced by the static analyzer.

5.6.1 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 [35].

Using a manual for using WVP1 [58], subject D first found out the keywords of the problem and then coded the three perspectives. In addition, subject D had some interaction with this author in order to clarify the procedures in the manual as well as on the formalism. The perspectives where analyzed and a series of discrepancies was found.

Subject D analyzed each one of the reported discrepancies and commented on them. Most of them were basically due to naming conflicts, but some pointed out missing information and conflicts. The best example of the kinds of the discrepancies found in Subject D’s case is:

- conflict between objects used for the action “check-stock”, and
- missing information about “order-information” in the data perspective.
The final version of Subject D's view is considerably different from the previous perspectives, and he acknowledges the usefulness of the "critiques". A problem observed by D is 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 to distinguish useful messages from not useful ones. In this specific case a generalization of messages is not hard. We refer to these classes of problems in Chapter 6.

5.6.2 Subject E's Perspective Analysis

Subject E, a senior computer science student at UCI with minor experience in data processing programming, used the IWSDD [47] Lift Problem, an English description of a controller for lifts. As with subject D, the author provided explanation of the contents of the manual [58] to subject E.

Subject E produced three perspectives and received a set of "critiques." These critiques were produced manually and then were 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. The most significant information provided by the analysis was:

- "there is a conflict between agents responsible for the action 'cancel-emergency,' the data perspective has "user" where the actor perspective has "supervisor".
Subject E did not provide any feedback with respect to any problems with the method.

5.6.3 Subject D’s and Subject J’s Viewpoint Analysis

Subject D’s viewpoint, using the view determined by the perspective analysis referred to above, and the author’s viewpoint towards the Gane’s description were analyzed. The first set of critiques were 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 a a missing information. The most useful messages were the following:

- Subject D did not have an action for the supervisor responsibility;
- Subject J action, “scan-to-see-the-tile,” did not have the fact, “book-title”.

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. In Chapter 6 we mention future work on this subject.
5.6.4 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 in computer science and extensive industrial experience. Subject T followed the same manual as did subject D and subject E, but with less tutorial explanation from the author.

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 out 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, where 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. This observation is connected with the problem of building an application vocabulary, and we refer to it in Chapter 6.
Chapter 6

Conclusions

In this chapter we compare viewpoint analysis with the other validation strategies and detail the contributions made. As a conclusion we highlight related topics that need further research.

6.1 Viewpoint Resolution and Very Early Validation

In Chapter 3 we studied the validation problem. As pointed out there, it is very hard or impossible to solve the validation problem of a model against a universe of discourse. Research has been trying instead to increase the approximation between the resulting model and the universe of discourse.

Researchers in software engineering have been working on methods and tools that improve this approximation. We have analyzed four different strategies. Qualitative analysis of them, based on their input to the process called Delta Computation, were presented. Analyzing the proposed viewpoint strategy using the same schema of analysis, we have determined the following:
Figure 6.1: Using Viewpoint Resolution
The Viewpoint Resolution strategy is able to distinguish between inconsistencies, wrong facts, and missing facts.

\[
\Delta = \begin{cases} 
\text{inconsistencies}(facts_a,facts_b) \\
\text{wrong facts}(facts_a,facts_b) \\
\text{missing facts}(facts_a,facts_b)
\end{cases}
\]

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

(-) it 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.

Something to consider is the fact that in a viewpoint resolution strategy, the user has a more indirect participation as compared with the prototype strategy, where the user, without being familiar with the "model", may express their opinion about its behavior. The viewpoint approach is more oriented to the ones who have built the model, and this task is usually performed by a non-user. Although this consideration is important, more important is the fact that VWPl is an executable language, and a description in VWPl is easily prototyped [53].
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 unfolding tacit knowledge are improved. There is less dependency on observers and readers.

6.2 Main Contribution

Considering the following challenge.

in the process of requirements analysis, provide an early validation scheme capable of distinguishing among inconsistencies, wrong information, missing information, without relying on a previous encoded domain

and acknowledging the results from Chapter 5, we have demonstrated the capability of the proposed strategy as a means of helping in the fact-validation process of requirements analysis. In order to support the conclusion that viewpoint resolution can be used as a means of very early validation, it is necessary to acknowledge:

1. the proof of the existence of an automated engine to perform validation through static analysis of two sets of production system rules,

2. the demonstration of the capability of the heuristics wired in the engine to properly point out for inconsistencies, wrong information, and missing information between different viewpoints.
The engine has to exist so that item 2 can be checked. It is also true that the non-existence of an engine would make it impossible to have an early validation based on the proposed scheme, since it would be impractical to have the method done by hand. By this argumentation the proof of the existence of the engine (1) is bound by its availability, which has been proved by its use.

Once the engine is available, it is necessary, however, to check the capability of its heuristics (2). This not only implies a test on the effectiveness of the heuristics, but also a test to ensure that the engine carries out those heuristics. This test, although not exhaustive since we are dealing with a prototype, has shown that the engine does carry down the designed heuristics. The effectiveness of the heuristics has been demonstrated by the examples given in Chapter 5, as well as the experiences analyzed in the Chapter 5 section “Case Studies.”

There are three main groups of heuristics: one to determine the analogy between different rules (resulting in the most probable pairs between the different rule bases), another to determine the analogy between different facts inside a pair of rules (a pair of rule is composed of one rule from each viewpoint being compared), and a third to determine which problems (consistency, correctness, and completeness) exist in a pair, as well as in the rule bases. We named these group of heuristics recognition, elaboration and evaluation, respectively. In Chapter 5, their objective was detailed and their effectiveness analyzed. In this analysis we identified not only the strong points, but also their weakness as well.

In conclusion, contrary to the usual approach of attacking the whole process of requirements analysis, which is so difficult, we tried to focus on a subproblem of requirements analysis, fact-validation. The results reported in this work are
expected to provide a contribution to the software engineering knowledge on the process of fact-validation, and in particular on using viewpoints in this process.

6.3 Highlights of the Contributions

- The presentation of a process-oriented analysis of the task of elicitation. The construction of a model, where the component parts of elicitation are factored out and explained (Chapter 2). The component fact-validation is explained in detail. A classification of problems in fact-validation is given (Figure 3.2), as well as the creation of a schema for comparing different strategies for validation (Chapter 3).

- The clear distinction between viewpoint analysis and viewpoint integration (Figure 1.3). Resolving conflicts between viewpoint holders and their posterior integration is pointed out as a problem more akin to social aspects of software engineering and of managerial expertise (Chapter 4).

- Usually systems analysts use an implicit and non-systematic way of handling different viewpoints. Our viewpoint resolution model makes this process explicit and systematic.

- The formalization of viewpoints by the VWPl language. VWPl has a horizontal rule structure oriented towards the representation of functionality and specific types of constraints (Chapter 4). The use of semantic descriptions for the language (semantic grammar and case grammar). The use of a language that is executable in the PRISM [52] system, thus providing the opportunity for a dynamic analysis of a viewpoint.
• The presentation of a method for the use of VWPl. That is, unlike the usual practice in software engineering research, a representation is proposed together with a method to use and apply it (Chapter 4).

• The presentation of a formal and automated strategy for analyzing viewpoints (Chapter 5).

• The demonstration by examples and by small "real" cases of the capability of the proposed strategy for analyzing viewpoints (Chapter 5).

• The unexplored possibility of this strategy being used in a cumulative manner to achieve a closer approximation to the universe of discourse, i.e., by successive applications of viewpoint resolution. Given two viewpoints, resolve and integrate them, comparing the result with that of another viewpoint.

6.4 Future Work

As a result of this work, we believe that six related topics deserve further investigation. Those topics are briefly described and their main research questions are highlighted.

Investigation of Partial Match Algorithms. As pointed out by this thesis, partial matches of fact-keywords within an existing application vocabulary are the keystone to the recognition and evaluation strategies. As such, better algorithms for partial matching in this situation could improve the conclusions of the viewpoint resolution method.
The elaboration of partial matching algorithms that could take into account words which are derived from other words, must address adverb formation, adjective formation, verb tenses, and the related problems.

Partial match algorithms could also have learning mechanisms to help and improve future use. The construction of dictionaries as a function of viewpoint integration could be a way of giving the algorithms learning capabilities.

Applicability Range of Viewpoint Resolution. Although primarily targeting the problem of fact-validation in requirements analysis, there are other areas to which the viewpoint resolution idea could be applied. Further investigation of those areas of applicability may indicate that this process is transferable to other scenarios.

One such scenario is in software maintenance. Before producing a maintenance request, different systems analysts could use their views in improving their "understanding" not only of the change to be made but also of the software to be modified. In this case, the number of rules will be far less than in the case of performing a complete requirements analysis.

Another scenario to examine is the incremental application of the method, i.e., instead of approaching the whole problem and trying to find all of facts and code all of the rules, the method could be used in different levels of abstraction, presupposing a previous organization of the problem. For instance, trying first to encode the functions presented in the universe of discourse and not the detailed processes would make the rule set more manageable and would provide quicker feedback.
Investigations on the applicability of the strategy, together with experimentation, should also evaluate and produce data to determine the tradeoffs of using more costly requirements analysis methods compared with methods that have a fast turnaround. This has resemblance to the comparison between viewpoint analysis and prototyping made in Section 6.1.

Another aspect to be taken in consideration is the notion of application vocabulary. Our method assumes its existence as natural if fidelity with the universe of discourse terminology is preserved, but further investigations should consider the construction of dictionaries through a negotiation process before the rules are encoded. In other words, a previous integration of the vocabulary is performed before viewpoints are analyzed by the static analyzer. As mentioned above, the dictionary could be constructed after viewpoint integration, where here the dictionary is constructed before the rules are compared. Investigating this tradeoff will provide valuable data for further developments.

Human-Interfaces. One of the aspects that our implementation does not emphasize is with respect to the human interface aspects of the static analyzer. We believe that applied research with the static analyzer should address this issue. There are three major areas where further work can be done: the presentation of messages, the gathering of facts, and the generalization of messages.

With respect to the presentation of messages, a prettyprinter should be available to translate the internal forms to English text messages.

The gathering of facts and their encoding in rules could be assisted by a system composed of a dialogue and a formatter for rules and hierarchies. This idea
has been explored in a previous paper [53] dealing with an automated assistant for
the elicitation process.

One of the major problems with the static analyzer is that it generates some
messages that are not pertinent or that are somehow duplicated. An example of this
is when in one viewpoint has the same missing object in several rules. In this case
the analyzer gives a message for each rule, thus producing “annoying” redundancies
for the user. The way to avoid these kinds of messages is to provide a filter that
analyzes all messages before prettyprinting them. One kind of filter could be one
which generalizes several messages and issues just a global one. Although this will
help to reduce the amount of output, it does not eliminate those messages that
compare rules that do not have much in common.¹

Dynamic Analysis of Viewpoints. As stated earlier the viewpoint representa-
tion used for static analysis of viewpoints is itself an executable language. In Leite
[53] it was pointed out that the examination of traces from the execution of a rule
base representing fact-finding could lead to interesting clues with respect to the
behavior of requirements.

The exploration of different traces produced by different viewpoints could
provide a dynamic analysis of viewpoints. The research on this topic would, to a
certain extent, provide an analysis of how to integrate a combination of prototype
strategy and viewpoint strategies.

¹This perception may vary between the viewpoint holders, i.e., one may think that there is a
similarity between two rules of a pair while the other may not.
Experimentation. Usage and experimentation of the proposed method is necessary to further evaluate its range of applicability as well as to improve the heuristics. Validation of requirements is a problem which concerns a mapping from an informal description to a formal description. Because of this, empirical results from the application of a method, which is capable of distinguishing between types of problems, could lead to important observations about the problem. In order to enhance the heuristics of the static analyzer, in addition to providing better partial matchers, it is necessary to have a series of empirical experimentations to better evaluate the scoring algorithms, as well as to test other scoring strategies.

Controlled experiments, as well as broader experimentation, should be designed in order to measure the effectiveness of applying a method that classifies and identifies validation problems. What kind of problem is more frequent? Which is the correlation, if any, between completeness problems and correctness problems? How do consistency problems help in the detection of other problems? These are the kinds of questions that experimentation with the method could help to answer.

A valuable experiment would be to evaluate the impact of making viewpoint resolution explicit and systematic in the behavior of systems analysts. A possible drawback of this method could be if it is perceived as a way of measuring competence between peers.

Viewpoint Integration. Since the complete viewpoint resolution strategy has to deal with the Delta computation, i.e., the resolution itself and the integration of viewpoints, it is necessary research this topic further.
Although it is a problem more akin to the social aspects of computing and to the management of software, it may be possible that methods and tools in the area of software engineering may help beyond the results reported in this work. Artificial intelligence approaches could help in providing probable integration candidates, based on some other description of the reality besides VWPl, although the integration problem is more difficult than viewpoint analysis.

Viewpoint Resolution in a Requirement Analysis Environment. As observed, the task of requirements analysis is in itself a very complex and difficult process. The availability of methods and tools for the systems analyst to use has been pointed out by several authors [32] [97] [83] as being of major importance in order to improve productivity in requirements analysis.

Our opinion is that an approach similar to program environments be used as a form of integrating tools and methods to help the systems analyst. We view viewpoint resolution as one of the methods and tools that should be present in such an environment.

A further development of the rule base representation would cause it to migrate to a more powerful representation. Since VWPl is not intended to be the language for the resulting model of requirements analysis, its migration to a semantic net based representation is one of the tasks that should be performed in such an environment. VWPl already has some semantic net characteristics, i.e., hierarchies and entities-attributes relationships are well mapped. The difficult part would be the transformation of rules to a semantic net formalism. Since the rules
are operational by definition it would be better to migrate to a hybrid formalism like KRIPTON [15].
Appendix A

VWPl Grammar

Here we describe the VWPl grammar, using standard BNF [72] and notations derived from semantic grammar [78]. 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 VWPl are reflected on the abstract forms produced by the VWPl parser.

The semantics of the language VWPl, is a subset of PRISM [52] semantics, since in our case the control strategy, nondeterministic firing, is fixed.

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 ")" "(" IsaHierarchies ")"
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 | DataTypes

Rhs := ActorsRhs | ProcessRhs | DataTypes

ActorsLhs := "(" AgentInvariantSeries ")"

"(" 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
Appendix B

The Static Analyzer

The objective of this appendix is to give an overview of the design of the static analyzer. We present the major modules used and give a brief description on them. The major data structures are presented as well as some of the experience gained working with Scheme. The static analyzer has approximately 3000 lines of code.

B.1 Overall Design

The static analyzer is composed of six major modules (see Figure B.1). The interaction between these modules is carried out using files. There were two reasons for using files as the interface between modules. First, because the modules are independent, having separate modules loaded at different times helps the development process. Secondly, is the easy of use of files in Scheme provides a nice extension to memory.

The first module is the parser. It reads the VWPl description and produces an internal form organized by type and class of facts. The production of the internal form is done by analyzing the left-hand side and the right-hand side of each rule. The parse checks for set membership in order to differentiate between types, and
Figure B.1: Overall Design of the Static Analyzer
uses the perspective name and rule position of the facts to determine classes. Each perspective (e.g., data, actor, process) has a different set of Scheme functions to implement its parser.

The second module is the first part of the recognizer. It identifies, depending on the perspectives being compared, the common facts between types and classes that are comparable. It attaches a score to each compared pair of facts.

The third module is the second part of the recognizer. It adds up the scores by type and classes and summarize them by rules. Thus, it produces the score for each rule after adding the series of facts. The scores are sorted, such that for each rule of the first viewpoint, the pairs of the second viewpoint are arranged in descending order by scores.

The fourth module is responsible for choosing those pairs that are most probable from the sorted list of pairs. It tries to eliminate misleading scores. It reads the sorted list and produces a list of pairs such that the first viewpoint rule has at most two second viewpoint rules. It also identifies missing rules in both rule sets.

The fifth module examines each pair again, this time in more detail. It recomputes the score of each pair of facts, chooses the most similar pairs for facts, analyzes their scores together with the hierarchies, and produces a table of messages.

The sixth module contains the partial matchers functions. Depending on where it is used, a different sort of matcher is used. It produces a score, given a set of patterns.
Besides these six basic modules, there are two general utilities modules used throughout the analyzer: the set functions and the tables functions. They are mostly borrowed from Abelson and Sussman [1]

B.2 Data Structures

All of the data structures used are list based. We used three kinds of data structures: tables, trees, and sets. Tables and sets use special operators that recognize their level of abstraction. Trees use operators that are low-level Scheme constructs.

The internal form as well as the score list of rule pairs are tree-like structures. In the case of the internal form, displayed in Figure B.2, the tree has three levels: type and class, rule numbers, and the facts. The facts are the leaves. In the score list, shown in Figure B.3, there are three leaves as well: the first viewpoint rule number, the second viewpoint rule number, and the scores.

The tables are used mostly in the evaluation part of the static analyzer. One of them keeps candidate pairs (see Figure B-4) between facts of a rule pair. It has one key, the class of the candidate pairs. The other table is the message table, see Figure B-5. It has two keys, the message number and the rule pair.

Sets are used, for example, in determining missing rules. Consequently, with the set of rules used for the first viewpoint in addition to the set of rules for the second viewpoint, we can determine the rules missing in the first viewpoint by finding the set difference between the second viewpoint and first viewpoint sets.
Figure B.2: Internal Form Tree

Figure B.3: Score Tree

Figure B.4: Candidate Pairs Table
B.3 Observations from Using Scheme

Scheme is a lexical scoped Lisp developed at MIT by Sussman and Steele [91]. The implementation used was developed by Texas Instruments for the PC class of machines.

At first on our project, we used global variables for the majority of shared data. As we became more knowledgeable about Scheme, mainly from reading the Abelson and Sussman book “Structure and Interpretation of Computer Programs” [1], our functions started to use more local variables. Although local variables are preferred in good software engineering practice, sometimes we used global variables just because they enable the results to be checked faster.

A problem we discovered with the Texas Instrument implementation was the impossibility of using “define” inside “define,” and as such the obligation of using “let” or “letrec.” Of the most interesting functions we developed were those in the third module, which were inspired by a merge function presented in [1].

Overall, our experience with Scheme was very good. The language completely fulfilled our expectations, and we managed to implement the static analyzer as we had designed it.
Bibliography


85. SIGSOFT. Special Issue on Rapid Prototyping. Dec. 1982. Several Authors.


