Abstract

Semantic case analysis provides a useful and powerful tool for an object-oriented analysis of software requirements. Two important areas of object-oriented requirements analysis are addressed: (1) identification of entities which should be modeled as objects in the software design; and (2) detection of ambiguity, inconsistency and incompleteness in the requirements documents. Available heuristics to identify these entities are based on coarse criteria, relegating much of the endeavor to intuition. We propose a set of refined and systematic heuristic rules to enhance the identification procedures. The proposed extensions further improve the requirements analysis process by focusing on basic verbs which are relevant to software design. The implemented system is capable of parsing the input requirements and extracting the objects, actions, and attributes for use in creating an object-oriented model of the requirements. A question-answering capability is provided to allow the user to ask specific questions about the model extracted from the requirements. Serving as a mediator, this system can lessen the chance for miscommunication by pointing out potential ambiguities, inconsistencies, imprecision, and other problems that can lead to misunderstanding between the client and the analyst.

Keywords and phrases: requirements engineering, methods and techniques, object-oriented requirement analysis, natural language processing, case grammar.

*This work was partially supported by NSF grant IRI-8804196 and by Tulane COR.
1 Overview

Requirements engineering (RE) has been recognized as a critical step in the production of software. Inadequate attention to this phase of the software life cycle has been a major cause of delay and expense [5, 8, 35, 38]. A representative view of the goals of RE has been given by Boehm [8]: “Software requirements engineering is the discipline for developing a complete, consistent, unambiguous specification—which can serve as a basis for common agreement among all parties concerned—describing what the software product will do (but not how it will do it; this is to be done in the design specification).” Agreement among parties implies understanding, and, indeed, it has been emphasized that one of the most important characteristics of a requirements document (RD) is that it be understandable [4, 38]. It is thus appropriate to view RE as the process by which the software user community imparts to designers and implementers an understanding of why the software is needed. However, no consensus exists about the RE process tasks. The following break-down of tasks is recurrent in the literature[16]:

1. Acquisition, Analysis, and Communication [13]
2. Problem Analysis, and Requirement Specification [15, 16, 10, 23]
3. Elicitation, and Modeling [28]
4. Elicitation, Formalization, and Validation [14, 21, 26, 29]

A synthesis of these views suggests the following major tasks in the RE process:

Requirements Acquisition: this task involves communicating with the client/user, understanding the problem, defining constraints, and establishing trade-offs;

Requirements Specification: this task involves the expression (formal or informal) of the behavioral and non-behavioral requirements; that is, a specification is a description of what the system does and what constraints are imposed by the user and the environment;

Requirements Analysis: this task involves investigating the properties of the specification (e.g., checking for inconsistencies, ambiguities, etc.), and developing an initial software model.

Problems of the requirements stage have been investigated since the early 1970’s. A first example is the SAFE project [3] whose goal was to convert informal specifications into formal specifications. The informal specification was formulated in a parsed natural language and conveyed the description of a well-formed process. Disambiguation was a major component in the analysis process. The modest success of this project on fairly small examples encouraged the project team into scaling up the goal. Two hard problems were identified: the conversion of informal specifications into formal ones, and the translation of high-level specifications into low-level ones. Consequently, the team researchers decided to concentrate on the second problem which led to the knowledge-based specification assistant (KBSA [25]). Two other early projects were PSL/PSA [34] and SADT [30], both supporting a dataflow-based methodology. PSL/PSA consists of a structured language (PSL) and an analyzer (PSA). SADT provides a graphical tool and a design methodology. Several enhancements to the dataflow-oriented methodology were subsequently proposed (e.g., SASS [17], DFD [39]). Software Requirements Engineering Methodology (SREM)[2] provides the necessary tools (RSL and REVS) to express and analyze software requirements. The Requirements Statement Language (RSL) is a formal language used to support a hierarchical state machine oriented modeling of a software system. The Requirements Engineering and Validation System (REVS) is then used to analyze and validate the model through simulation. A formal requirements modeling language (RML) is presented in [23]. RML is intended to capture the description of real-world entities in
an object-oriented framework. It is also intended to capture the knowledge about the application domain. Once a model is expressed in RML, formal analysis is possible. The TARA project [26, 27] provides an automated set of tools to animate a formal software specification. This process helps the analyst validate the specification. A different approach to requirements validation is the viewpoint resolution method [28]. This approach supports the analysis of different views of the same system with the intent of uncovering missing and contradictory information. A recent system that addresses the transition from informal requirements to formal specifications is the Requirements Apprentice (RA) [29]. RA assists the human analyst in the creation and modification of software requirements. It has three main tasks: knowledge representation and reasoning, interaction with the human analyst, and domain-specific knowledge maintenance.

An indirect approach to requirements analysis is the use of executable specification languages. OBJ [22] and PAISLey [40] are two well-established executable specification languages. We have also developed an executable specification language that can be used effectively for rapid prototyping [6, 7]. The FORREST project [14, 21] has synthesized a formal specification language based on modal logic. The intent of a such a language was to address some of the pitfalls of then-existing specification languages.

2 Object-oriented Requirements Analysis

Most of the surveyed approaches concentrate on providing formal languages for expressing software models and tools that perform analysis on these models. As such they either emphasize the analysis of a software model extracted from the informal requirements, or supports structuring mechanisms through a well-defined notation. This observation implies a distinction between the analysis of software models and the analysis of software requirements. Such a distinction warrants different tools and approaches. In addition, methods to effect the transition from the requirements to the specifications are needed; that is, methods to facilitate the model-extraction process are lacking.

As in the case of RE generally, OORA has not developed to the point of providing automated techniques for use with informal documents, but an emerging consensus indicates that the early stages of such analysis should focus on the identification of objects in the problem domain [1, 11, 33, 32]. Several approaches to OORA have been recently proposed [9, 11, 19, 33]. They all describe object-oriented methodologies with variations in the notation and strategies. It is clear that the notation can be unified and formalized. However, the strategies are highly informal and heuristic-driven. One ultimate goal of our research is to formalize the strategy.

In our view, the relative importance of the major tasks of RE depends on whether the requirements are expressed formally. Requirements expression is basically a problem formulation endeavor that attempts to capture the more or less vague concerns of the client or end user. Because of the uncertain nature of these concerns, requirements are often described in informal documents written in natural language. To make use of these documents, the requirements analysis activity must address natural language issues such as ambiguity and knowledge representation. Moreover, to facilitate the eventual software design, requirements analysis must also embrace a particular design method (e.g., dataflow, functional decomposition, or object-oriented). We have developed a prototype tool for direct analysis of informal software requirements that supports both natural language processing and object-oriented design.

2.1 Requirements Expression

Even though requirements expression is crucial to the analysis step, we have opted initially to emphasize analysis over expression to control the complexity of the problem and to concentrate on object-oriented analysis issues. To achieve this goal, we defined a constrained natural language
for requirements expression. This choice is motivated by studies that show that technical writings in scientific disciplines tend to be highly constrained and less varied in format than narrative text. Our expectations were that the grammatical structures used in software requirements would be constrained, regular, and capturable in a limited grammar. To address this conjecture, we studied several requirements documents written in English and noticed that the technical prose used in these documents does not resort to complex constructions. In fact, the structure of the sentences were usually either simple, or compound in a limited and regular way. Based on this study, we developed a grammar capable of expressing these constructions. (The resulting language is called Analyst’s Constrained Language, or ACL.) It should be noted that reading ACL documents is fairly natural, whereas writing them requires knowledge of the specific grammar rules. We tested the usefulness of this language by assigning two projects and several examples to our software engineering classes. The purpose of these assignments was to express the requirements of a software system in this constrained form of English. The results were surprisingly encouraging in two aspects: (1) in general, the students followed the grammar very closely, considering the short amount of training they received; and (2) the students were able to organize the requirements in a structured manner. We consider this preliminary evaluation successful in reinforcing our choice of constructs.

2.2 OORA Strategy

The main issue is the provision of a systematic and problem-independent method for the analysis of requirements. The purpose of the analysis is to extract, organize, and classify the information contained in the requirements documents. The proposed analysis method is based on the object-oriented approach to software development. This approach has gained acceptance as an effective methodology for developing software. Informal procedures and strategies have been proposed by several researchers to support the software development process defined by this approach. Abbot [1] provides the following object-oriented analysis procedure and associated heuristics:

1. Identification of the data types: A common noun in the informal strategy suggests a data type (class).

2. Identification of the objects: A proper noun or direct reference suggests an object.

3. Identification of the operators: A verb, attribute, predicate, or descriptive expression suggests an operator.

4. Formulation of the control to define the operators: The control structures are implied in a straightforward way by the English.

Steps 1 and 2 (identification of types and objects) are based on superficial identification criteria, thus rendering the identification process ad-hoc and non-automatable. Conceptually, the fundamental distinction between a class and an object derives from the intentional (generic) or extensional (existential) use of the noun phrase (NP). In general, distinguishing between the two uses may require a prohibitively large knowledge base. However, several observations concerning the technical writing style which typifies software requirements documents, as well as the success of our prototype system, show that useful design information can be automatically extracted from such documents by means of a relatively simple system.

First, at the design stage, it is more productive to generalize this notion of object/class by categorizing such entities as classes, and to think of objects as mere instantiations of these classes. This observation supports the hypothesis that classes—corresponding to intensional references in software requirements documents—relate to the “static semantics” of object oriented programs, while objects are dynamic elements which exist only during program execution. A second observation
relating to technical writing style is that multiple word senses and metaphor are not frequently encountered. The selection constraint mechanism which is employed in our case frame analysis system, however, has proved sufficient to distinguish most word senses found in requirements documents. Moreover, metaphor in technical documents, when it is used at all, seems to be used in a consistent manner (e.g., reading and writing of files). Experiments we have carried out show that a relatively simple grammar for a fragment of English (1) accounts for most of the constructions found in software requirements documents; (2) is easy to learn and apply in generating original documents; and (3) can be automatically processed using a relatively small knowledge base.

Step 3 (identification of operators) is syntactically straightforward, because verb phrases generally denote operators. However, there is a need for further discrimination among the operators. Verbs can be grouped into four major classes: state, action, achievement, and accomplishment [18]. This classification enhances and focuses the semantic analysis. It also provides more data pertaining to causality, entailment and temporality. The class of a verb can be determined statically, and thus, used as an attribute. In the few cases where a verb may belong to more than one class, a simple dynamic classification is needed.

Shlaer and Mellor [33] suggest a more comprehensive set of five categories of common nouns into which most object classes fall. There is a close correspondence between these categories and the noun domains we use to implement selection constraints for the various semantic cases allowed by each verb:

<table>
<thead>
<tr>
<th>Shlaer and Mellor categories</th>
<th>corresponding noun domain(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>roles</td>
<td>human</td>
</tr>
<tr>
<td>tangible things</td>
<td>animate, inanimate, machine, location</td>
</tr>
<tr>
<td>incidents</td>
<td>event</td>
</tr>
<tr>
<td>interactions</td>
<td>process</td>
</tr>
<tr>
<td>specifications</td>
<td>data, data structure, language</td>
</tr>
</tbody>
</table>

In addition to those above, we define three other noun domains which do not usually correspond to objects: measurement, state, and condition. A given noun lexeme, we have noted, will typically have senses in more than one of these noun domains. Thus, some form of word sense disambiguation, such as that provided by our selection constraint mechanism, is essential for successful application of the simple classificational heuristics proposed by Abbott or by Shlaer and Mellor.

Two further observations can be made about the use of such simple heuristics for identifying classes of objects: (1) The proposed methods are informal and non-systematic; and (2) The strategy fails to account for the context in which noun phrases are used. Our system addresses these problems by automatically identifying noun phrases in the input document and indicating the relationship of each such noun phrase to every other one in the same sentential clause. In particular, these relationships are shown as semantic cases of the main verb of each clause. The analyst is thus presented not only with a semantic classification of each noun phrase, but with a characterization of its role in the operations denoted by verbs in the text. We contend that this kind of information leads to a more consistent identification of classes of objects than any method in current use.

2.3 Case Grammar

Because the initial task in OORA is the identification of objects, and informal specifications are typically expressed in natural language (NL), we believe that case grammar concepts can signifi-
cantly benefit the RE process. In case grammar, the verb is the focus of a sentence, and the noun phrases in a sentence each stand in a specific case relationship with the verb. Knowing the case role of each noun phrase, the requirements analyst can better judge how it should be represented in an object-oriented design.

Case grammar classifies each verb according to its case frame–the particular subset of semantic cases that must be filled by noun phrases whenever that verb is used in a sentence. No consensus exists as to a specific set of cases to be used in defining case frames. However, case grammarians, beginning with Fillmore [12, 20] have generally agreed that a small number of semantic cases is sufficient to model all the verbs in the language. Moreover, it is almost always stipulated that the same case may not occur more than once in a single sentential clause. We have implemented a prototype system based on six cases, informally defined as follows:

**Agent** – the initiator of the action

**Instrumental** – used by the Agent to perform the action

**Neutral** – the entity affected by the action

**Source** – starting point of the action

**Goal** – end point of the action

**Locative** – where the action takes place

The action referred to in the case definitions is that denoted by the verb. To better describe how verbs are modeled in terms of a particular set of cases, some general concepts are discussed below.

Many case schemes, particularly localistic schemes, represent several distinguishable roles with a single “generic” case by exploiting observed patterns of complementary distribution of such roles among different verbs. Source and Goal are examples of such generic cases; the verbs “change”, “give”, and “move” illustrate complementary distribution within these cases. Consider the following sentences:

1. The interrupt routine was changed from a sequential process to a parallel one.
2. The interrupt routine was given to the most experienced programmer.
3. The interrupt routine was moved from the kernel to microcode.

In each sentence, the Neutral case is filled by the noun phrase, “The interrupt routine”. In the first sentence, the Source and Goal, “a sequential process” and “a parallel one”, indicate the identity of the Neutral at the start and completion of the action, “change”. The verb “give”, in the second sentence, does not allow a Source, but the Goal, “the most experienced programmer”, manifests the possessor or assignee of the Neutral at the end of the action. Finally, in the third sentence, Source “the kernel” and Goal “microcode” specify the starting and ending location of the Neutral. (In fact, physical location is probably less significant than form in this example. Different Source and Goal relations are often viewed as abstractions of physical location, hence the term “localistic case system.”) It is seen from these examples that the Goal case is used to represent at least three different relations–identity, assignee and location. Because these relations are distributed in complementary (i.e., mutually exclusive) fashion among the different verbs, however, they may all be represented by the Goal case without allowing sentences that violate the single case per clause convention.
As noted above, each verb is characterized by the semantic cases involved in its use. Some cases are optional for some verbs, a fact that may be explained by distinguishing between the propositional and the modal content of a sentence. Every verb, when used properly in a sentence, expresses some proposition about one or more particular entities. The cases that are conceptually required for the proposition to make sense are called propositional cases. By contrast, modal information may be added to any complete proposition. Such information is generally considered to modify the proposition as a whole, making it more specific as to time, manner, place, etc. Case grammar is more concerned with the propositional content than the modality of sentences. Indeed, modal information is not always expressed in terms of noun phrases that can be readily classified as cases. For example, sentence adverbs and auxiliary verbs (e.g., modals "can", "should", etc.) contain modal information. Nevertheless, our system includes two cases, Instrumental and Locative, which are usually optional and may, as such, be considered modal cases. (It may be noted that sentences in passive form frequently omit reference to an Agent, as in the examples above. As such, the Agent might be considered an optional case, but a distinction is made between roles that are conceptually required but unexpressed and those that simply aren’t required. Thus, if we know that a particular action takes place, we usually imagine that some entity initiates it even if the identity of that entity is unknown.)

A final significant feature of case grammar is that each verb is further characterized by selection constraints on the nouns that may fill its cases. This implies that all nouns in the language are classified in terms of a predetermined set of semantic domains. The case frame for each verb, then, includes a designation of the allowable noun domains for each case to be filled. In our system, the domains of nouns for each case in the case frame are a subset of the following:

Animate –person, cat, horse, ⋅⋅⋅
Inanimate –e.g., desk, chair, book, ⋅⋅⋅
Human –e.g., programmer, analyst, operator, ⋅⋅⋅
Machine –computer, disk drive, printer, ⋅⋅⋅
Process –program, process, algorithm, ⋅⋅⋅
Data –information, character, document, ⋅⋅⋅
Data Structure –file, record, address, ⋅⋅⋅
Event –interrupt, change, overrun, ⋅⋅⋅
Condition –completion, ambiguity, progress, ⋅⋅⋅
Language –algebra, alphabet, grammar, ⋅⋅⋅
Location –e.g., center, place, side, ⋅⋅⋅
State –e.g., maximum, minimum, parity, ⋅⋅⋅
Measurement –e.g., amount, group, precision, ⋅⋅⋅

Many nouns are classified into more than one domain. Multiple classification of a noun indicates that it may refer to different concepts (i.e., have different senses or meanings). “Address”, for example, listed above as an example of a Data Structure, might also be used to refer to the symbol or number (i.e., the data itself) that makes up an address, or in an abstract sense as a location of some other data. Thus, any verb having a case allowing a noun from the Data Structure, Data, or Location domains could be used with “address” filling that case. Selection constraints can serve to
disambiguate nouns, when the intersection of the possible domains for a noun and the allowable domains for the case it fills includes only one domain. Though not all instances of sense ambiguity in nouns can be resolved in this manner, selection constraints provide an effective heuristic. This is particularly true when the knowledge base containing the constraints and noun domains is designed for a technical application such as software requirements analysis.

2.4 Knowledge base extensions

This basic design of case grammar may readily be represented in first order logic (FOL) in a manner consistent with conventional notions about verbs and noun phrases. In particular, each verb in the language would correspond to a k-place predicate, where k is the number of semantic cases in the design. The domain of the logical model for the design would comprise the set of all the nouns in the language, augmented by a null marker to indicate illegal cases in the case frames. Noun domains would be represented by one place predicates. The knowledge base would then comprise one set of axioms for nouns and another set for verbs. Each noun axiom would simply be a disjunction of all the noun domain predicates possible for that noun. For each verb there would be an axiom of the form:

\[(1) \text{ < verb > (} x_1, x_2, \cdots, x_k \text{)} \rightarrow (F_1 \& F_2 \& \cdots \& F_k)\]

where \text{ < verb > } is the symbol for the verb (i.e., the verb lexeme itself); \(x_1, x_2, \cdots, x_k\) are variables; and \(F_1, F_2, \cdots, F_k\) are formulas of either form (2) or (3):

\[(2) < nd_m, 1 > (x_m) | < nd_m, 2 > (x_m) | \cdots | < nd_m, l_m > (x_m)\]

\[(3) x_m = < \text{null } >\]

where \(m\) is in \{1, ..., k\}; \text{ < nd_m, 1 >, < nd_m, 2 >, ..., < nd_m, l_m >} predicate names corresponding to the allowed noun domains for the \(m^{th}\) case of the verb; and \text{ < null >} is the null marker. Form (2) would be used for allowed cases of the verb; form (3) for illegal ones.

It is assumed that any NL processing system based on case grammar will incorporate essentially syntactic mechanisms that map NL sentences to some form equivalent to FOL predicates. These mechanisms must include means for identifying the head noun in each noun phrase available to fill a verb case, so that its noun domain can be looked up. Modal information not expressed in the form of noun phrases must also be dealt with or removed from consideration. The actual process of case frame analysis—determining which noun phrases fill the various required and optional cases—may involve consideration of selection constraints together with relevant syntactic information. In addition to case selection constraints, then, case frame analysis may use information about the position of each noun phrase (subject or object); its introductory preposition; and whether the sentence is in active or passive form.

Several observations may be made about case grammar based on the foregoing FOL model. First, verbs are not related in any way by the model to one another. Moreover, the only way nouns in the model are related to one another is by classification in the same noun domains. Finally, aside from the noun classifications and syntactic information which are actually used to determine its semantic case, nothing in the basic case grammar model allows inferences to be made about a particular noun instance (or the noun phrase it heads) based on knowledge of its case. We propose to extend the semantic treatment of the basic case grammar model by a disciplined treatment of noun, verb and case relationships.

To be useful, the knowledge base must include case frames (axioms) for a significant number of verbs (i.e., several hundred). Extending the semantics for verbs on an ad hoc basis would thus be
very cumbersome. The result, we believe, would be highly redundant as well. We therefore propose
to first map the case frames for general verbs into those of a small number of “basic” verbs. Since
the basic verbs themselves would also be in the lexicon, the representational domain would, in
effect, be a subset of the input domain. That is, each input sentence would be mapped to an initial
case frame format with its own verb as a predicate, and then to a second format with one or more
basic verbs as predicates. It is assumed that some sentence(s) in the input language formed from
the appropriate basic verbs and noun phrases, would be mapped directly to the second format.

It may be objected that some elements of meaning may be lost in mapping to the representa-
tional domain, since different input sentences may be mapped to identical representational forms.
For present purposes, it is assumed that the “meaning” of a sentence or predicate is the set of
inferences which can be drawn from it (given, of course, some propositional context and possibly
some set of constraints on the inferencing mechanism.) Two observations are offered in response
to this objection:

1. Different input sentences may, in fact, have identical meanings. The verbs, for example, may
   be synonymous.

2. We allow a single case frame from the input domain to be mapped into multiple basic verb
case frames (e.g., in sequence, as a conjunction, etc.) In this sense, meaning that resides in
lexical diversity is changed to equivalent meaning based on representational complexity.

While it is thus possible in theory to preserve meaning within the proposed format, it may
prove very difficult in practice to design basic verb representations which do so. Indeed, it may
be desirable, in the context of the software design process, to ignore possible meanings which are
not relevant to the application at hand. Thus, in generating extended axioms or case frames for
non-basic verbs (i.e., formulas for translating non-basic verb predicates to basic verb predicates),
we endeavor to capture just those elements of meaning pertinent to requirements analysis in general
and object-oriented design in particular.

The first step in designing a representational system as proposed is to designate a set of basic
verbs. Several criteria should be considered in this task. First, of course, the verbs chosen should be
commonly used, and their meanings should be widely and consistently understood. This will insure
that expressions translated into the representational domain will not give rise to misunderstanding
between the software designer and the end user. At the same time, the basic verbs should denote
concepts relevant to software design, since the requirements will be used for this purpose. The
goal of reducing lexical diversity, discussed above, is advanced by keeping the total number of basic
verbs small. We propose two further criteria that are related to the concept of representational
complexity: (1) senses in many abstract “verb domains”; and (2) both Source and Goal cases are
allowed.

In the preceeding section, it was suggested that the specific roles represented by the various
cases, particularly Source and Goal, may vary from one verb to another. In fact, different case
relationships may occur in different senses of the same verb. Thus, in one instance a particular
verb might have a meaning in which Source and Goal stand for one set of relationships, but in
another instance the same verb could be used in a different sense, with different Source and Goal
relationships. The notion of complementary distribution is not violated, however, because only
one Source or Goal case is involved in a single verb instance. Just as classification of a particular
noun into several noun domains suggests a corresponding number of different senses, different
interpretations of a verb’s cases correspond to its distinct senses. These different interpretations
(location, possession, identification, etc.) have been described as the concrete location domain and
its abstract extensions [12]; we refer to them all simply as verb domains.
In accordance with the criteria stated above, we wish to select verbs with senses in more than one verb domain as basic verbs. Obviously, simply mapping from one verb to another does not increase surface complexity. Our strategy is to add expressions, as part of the process of mapping to the representational domain, that will serve to identify the verb domain of the basic verb. Such added expressions may involve the generation of Source and Goal cases for the basic verbs, as suggested by the final criterion above. We think it more likely, however, that the verb domain of each basic verb instance may be more readily determined by examining its Neutral case. The process of mapping from input to representational domain is illustrated by the following examples:

Input domain:

(4) The system assigns the CPU to an interrupt routine.
(5) The system assigns a name to an interrupt routine.

Representational domain:

(4a) The system changes control of the CPU [from X] to an interrupt routine.
(5a) The system changes the identity of an interrupt routine [from X] to a name.
(5b) The system changes control of a name [from X] to an interrupt routine.

Both input sentences (4) and (5) have the same Agent, “The system”, and Goal, “an interrupt routine.” Mapping from the input verb “assign” to the basic verb “change”, however, requires elaboration of the Neutral case so there is no confusion between the identificational and possessional verb domains. In sentence (4), “assign” is used in its more common sense, to denote a change of control of its Neutral case. Determination of the verb sense in sentence (4) depends on the noun domain–Machine–of the Neutral, “the CPU”. When the Neutral of “assign” is in the Data domain, as in sentence (5), the expected verb domain is identificational, as in interpretation (5a). Sentence (5) is actually ambiguous with respect to verb domain, however, since possessional interpretation (5b) cannot be definitely ruled out. Such ambiguity can be automatically detected. Moreover, the addition of an appropriate term in the representational form as the head noun of the Neutral (“control”, “identity”, etc.) facilitates immediate recognition of the applicable verb domain.

The examples above also show that the Goal may, as in (4a) and (5b), or may not, as in (5a), be mapped directly from the input to the representational domain. In (5a), in fact, the Neutral, “the identity of an interrupt routine”, is formed by adding a term to the Goal of the input sentence. Still one further feature of the proposed system is illustrated by the basic verb “change”, which allows (but does not require) a Source. Thus, the phrase “from X” could be included in (4a), (5a) and (5b) to bring to attention the absence of information about the Source in the corresponding input sentence.

3 Implementation

We have implemented a prototype system which automatically performs semantic case analysis on input sentences. The sentences must conform grammatically to a fragment of English called Analyst Constrained Language (ACL). The current lexicon for the system includes approximately 500 each of nouns, verbs and other words, and the knowledge base comprises a designation of case frames and selection constraints for verbs, possible noun domains for nouns, and syntactic roles of other words. The structure of the implementation of the analysis system is depicted in Figure 1. It consists of a natural language processing subsystem (NLPS) and an object-oriented analysis subsystem (OOAS). Informal requirements written in ACL are input to this system. NLPS performs morphological,
lexical, syntactic and semantic analysis on them. A decorated tree altogether with case frames are passed from NLPS to OOAS. From these two structures, roles, relationships, and entailments are derived and represented as relational tables. These tables are subsequently used to answer questions to elicit the requirements and to display the information contained in the tables.

Output from the implemented system for each sentence analyzed shows the main verb and the noun phrases that fill its required and optional cases. The noun domains of the case fillers are shown as well. We have also developed a set of simple heuristics to be applied by the analyst in using the system output for object oriented requirements analysis. These heuristics serve as a filter to indicate which noun phrases in the input sentence are the most likely choices to be represented as objects (or classes) in an object-oriented software design described by the sentence. To illustrate the use of output from the prototype system, example sentence (4) from the preceding section is again considered. The system output for this sentence includes the following information:

**Main verb:** assign

**Agent:** “The system” (Machine)

**Neutral:** “the CPU” (Machine)

**Goal:** “an interrupt routine” (Process)

To guide the analyst in using this information, the following heuristic rules are applied:

1. The noun domains of Human, Animate, Inanimate, Machine, Event, Process, Data Structure, and Language are most strongly suggestive of objects.

2. The cases of Agent and Instrumental are most strongly suggestive of objects.

3. The Neutral case also indicates the existence of an entity that should be modeled as an object, but the case filler may not refer directly to the object. This is particularly true if the noun domain of the case filler is not one of those listed in rule (1). The head noun of the Neutral case may be an attribute of the object, while the object itself may be found elsewhere in the case filler phrase or in a related case (Source, Goal or Locative).

Applying these rules to the information from the input sentence, the analyst would conclude that “The System” should be modeled as an object, and that “the CPU” should probably be
modeled as an object as well. It would also be concluded that if “the CPU” were not established as an object, “an interrupt routine” might be so designated instead.

The current prototype, as a byproduct of semantic case analysis, also performs automatic detection of some forms of ambiguity, inconsistency, and incompleteness. In particular, ambiguity is suggested when more than one of the possible noun domains for a case filler are allowed by the applicable case restraints. By similar analysis, inconsistency is indicated when none of the noun domains for a particular noun phrase match the case restraints for the case it is otherwise forced to fill. Incompleteness is suggested by the absence of sufficient noun phrases to fill all the required cases for the main verb of a sentence.

In the proposed extension to the basic case grammar system, the case frames for each verb in the knowledge base are supplemented by information allowing mapping to a representational domain. Case information for representations in terms of basic verbs offer the analyst several advantages over the system output described above:

1. Basic verbs focus on those elements of meaning most relevant to software design.

2. The case frames for basic verbs associate objects with the Agent, Instrumental and Neutral cases, in accordance with the heuristic rules above, more consistently than other verbs in the input lexicon.

3. More extensive information about incompleteness is available, since the case frames for basic verbs allow more cases than other verbs in the input lexicon.

4. The Neutral case filler is modified to show precisely the attribute or relation that is affected by the action denoted by the verb.

5. The analysis is focused on the basic verbs kernel; thus, addition of other verbs will have no effect on the analysis process.

Besides offering the analyst enhanced information for manual analysis, the output from the extended case grammar system may be used more readily for further automated semantic analysis. Features of the extended system that make such further analysis more manageable include (1) the relatively small number of verbs for which semantic routines must be designed; and (2) the regular form of the Neutral case expression.

4 Semantic Case Frame Analysis

The semantic analysis phase associates meaning with the input text by constructing a knowledge representation of the sentences. Case frame was chosen as the preliminary semantic representation. This choice was motivated largely by the requirements of object-oriented modelling. The components of the object-oriented model, objects, operations, and attributes, can be readily identified from the tables generated by the case frame analysis.

Case frame analysis takes the parsed requirements, consults the lexicon for necessary information on the verbs and nouns in the various phrases of the sentence, and attempts to fill in the case slots in the case frames of the action verbs using the available information. Four phases of analysis—(1) locating and identifying the verb, (2) identifying the prepositional and noun phrases which are candidates for cases, (3) classifying the required cases, and (4) classifying the optional cases—are illustrated below for the following text, adapted from an example by Shlaer and Mellor[33]:

1. Every TSAR [Tape Storage and Retrieval] housing is equipped with a tape robot.
2. The TSAR performs its intended function with the tape robot.

3. Every tape robot on site is installed in a TSAR.

4.1 Locating and Identifying the Verb

The first step in case frame analysis is to locate the main verb in the sentence. The analysis consults the ACL grammar rules to find the location of the main verb and interacts with the lexicon to obtain information on each word in the sentence. In the first sentence above, the main verb, “equip”, has the following entry in the lexicon:

Word: EQUIP  
Category: ACTION  
Agent: HUMAN, MACHINE, PROCESS  
Agent Flag: OPTIONAL  
Instrumental:  
Instrumental Flag: ILLEGAL  
Neutral: MACHINE, PROCESS  
Neutral Flag: REQUIRED  
Source:  
Source Flag: ILLEGAL  
Goal: HUMAN, MACHINE, PROCESS, INANIMATE  
Goal Flag: REQUIRED  
Locative:  
Locative Flag: ILLEGAL  
Particle: WITH  
Particle Flag: REQUIRED

The main verbs for the remaining sentences are identified in the same manner, and the corresponding lexical entries located:

Word: PERFORM  
Category: ACTION  
Agent: ANIMATE, HUMAN, MACHINE, PROCESS  
Agent Flag: OPTIONAL  
Instrumental: MACHINE, PROCESS  
Instrumental Flag: OPTIONAL  
Neutral: PROCESS
4.2 Identifying Prepositional and Noun Phrases

Prepositional phrases that are candidates for cases in the verb’s case frame must come after the main verb. Those that come before modify the noun phrases which they follow. In the third sentence, for example, the prepositional phrase “on site” cannot serve as a case since it is an attribute of the noun “robot”, indicating the location of the robot. For each prepositional phrase that follows the main verb, the preposition in the phrase is stored, and the domain(s) of the main noun in the noun phrase immediately following the preposition is (are) retrieved from the lexicon. One candidate prepositional phrase is identified in the first sentence:

prepositional phrase: with a tape robot
Preposition: WITH
Domains of ROBOT: MACHINE, INANIMATE

Similarly, one candidate phrase is identified in each of the other sentences. A problem arises, however, because our current dictionary does not have an entry for “TSAR”. To allow processing to continue, the lexical analyzer invokes a routine to query the user for necessary information about
any such new words it encounters. For nouns, the user need only supply the appropriate domain
names in response to prompts from the query routine. These prompts may be in menu form, and
may include definitions of the available domains. Generating entries for new verbs is more involved,
but we have observed that the overwhelming majority of new words encountered in requirements
documents are nouns.

Assuming the user designates the domains “Machine” and “Location” for the noun “TSAR”,
the remaining prepositional phrases for the sentences above are analyzed as follows:

<table>
<thead>
<tr>
<th>Sentence (2)</th>
<th>Preposition: WITH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domains of ROBOT: MACHINE, INANIMATE</td>
<td></td>
</tr>
<tr>
<td>prepositional phrase: with the tape robot</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sentence (3)</th>
<th>Preposition: IN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domains of TSAR: MACHINE, INANIMATE</td>
<td></td>
</tr>
<tr>
<td>prepositional phrase: in a TSAR</td>
<td></td>
</tr>
</tbody>
</table>

Noun phrases (i.e., those that are not preceded by a preposition) are classified as subject or
object. Every sentence must include a subject noun phrase, which precedes the verb, and may
optionally contain one or two object noun phrases. As with each prepositional phrase, the domains
of the main noun in each subject or object are retrieved from the lexicon. Again, the user query
routine is invoked to obtain appropriate classifications for “housing”. Assuming the user designates
the domains ”Inanimate” and ”Location” for ”housing”, the subject and object noun phrases for
the sentences above are:

<table>
<thead>
<tr>
<th>Sentence (1)</th>
<th>Subject: Every TSAR housing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domains of HOUSING: INANIMATE, LOCATION</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sentence (2)</th>
<th>Subject: The TSAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domains of TSAR: MACHINE, INANIMATE</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sentence (2)</th>
<th>Object: its intended function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain of FUNCTION: PROCESS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sentence (3)</th>
<th>Subject: Every tape robot on site</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domains of ROBOT: MACHINE, INANIMATE</td>
<td></td>
</tr>
</tbody>
</table>

4.3 Classifying Required Cases

The main steps of case frame analysis start here. With the information so far accumulated from the
earlier phases, the analysis starts classifying the prepositional and noun phrases by first classifying
the required cases. The intersection of the two sets of domains, one of the main noun in the phrase
being considered for a case and one of the case itself, determines whether that phrase qualifies to
be classified as the particular case, i.e., its main noun must be in one of the defined domains for
that case. For example, if the domains defined for the agent case are HUMAN and MACHINE,
the main noun of a phrase being considered for the agent case must belong to at least one of these
domains.

In the example sentence (1), there are two required cases, the neutral and the goal. The verb
“equip” also requires a particle, which introduces the neutral case. (Our treatment of “with”
as a particle here might not be accepted by all grammarians, but we have found it convenient
to deal with many similar verb-preposition combinations in uniform fashion. As pointed out,
many distinctions necessary in the analysis of unconstrained NL are superfluous when dealing with
technical documents.) Since “with a tape robot” is thus required to fill the neutral case in the first
sentence, the goal case must take the subject “Every TSAR housing”.

As can be seen, the requirements for the neutral case of “equip” and the domains for “robot” both include MACHINE, while the domain INANIMATE is associated with both the goal case and the noun “housing”. Though there was no need to examine the selection constraints in identifying the cases for the first sentence, this step is performed nevertheless as a consistency check. For example, had the user specified only LOCATION as a domain for housing, the resulting violation of case constraints would have been reported by the system.

In the second sentence, the neutral is the only required case. The sentence is active and so the domain of the main noun of the object noun phrase, “function”, is tested against the domains of the neutral case for a domain match. The domain of “function” is PROCESS, which is one of the domains of the neutral case for the verb “perform”.

Again, only the neutral case is required in the third sentence. The form of the sentence is passive, however, and so the subject rather than the object is tested for a domain match. The phrase, “Every tape robot on site” is thus identified as the neutral in sentence three.

4.4 Classifying Optional Cases

When the required case slots have been filled, the case frame analysis proceeds to fill the optional case slots. In sentence (1) above, all of the available phrases have been used to fill the required cases, so analysis of the optional cases is omitted.

There are two optional cases for the verb “perform” in sentence (2): AGENT and INSTRUMENTAL. Since the sentence is in active form, the requirements for the agent case slot are compared with the subject, “The TSAR”, and a satisfactory match is found. The remaining phrase in sentence (2), “with the tape robot”, meets the selectional constraints and includes an appropriate preposition, “with”, for introducing an instrumental, and so is deemed to fill the remaining optional case slot.

In sentence (3), the only available phrase remaining, “in a TSAR”, meets the selectional constraints for either of the optional cases. Since the preposition ”in” introduces locative rather than agent, however, it is still possible to complete the case frame automatically.

5 Object-oriented Analysis Using the Instantiated Case Frame

The purpose of case frame analysis of informal requirements is to organize and represent the information provided in the documents in a meaningful way to help the software analyst identify those entities which should be modeled as classes. The instantiated case frames from the example sentences are:

<table>
<thead>
<tr>
<th>VERB</th>
<th>NOUN PHRASE</th>
<th>CASE</th>
<th>NOUN DOMAIN(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>equip</td>
<td>Every TSAR housing a tape robot</td>
<td>GOAL</td>
<td>INANIMATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NEUTRAL</td>
<td>MACHINE</td>
</tr>
<tr>
<td>perform</td>
<td>The TSAR its intended function the tape robot</td>
<td>AGENT</td>
<td>MACHINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NEUTRAL</td>
<td>PROCESS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSTRUMENTAL</td>
<td>MACHINE</td>
</tr>
<tr>
<td>install</td>
<td>Every tape robot on site a TSAR</td>
<td>NEUTRAL</td>
<td>MACHINE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LOCATION</td>
<td>MACHINE</td>
</tr>
</tbody>
</table>

The domain for each of the noun phrases identified is the intersection of the possible domains from the dictionary entry for the head noun with the verb’s allowed domains for each case. In each case, selectional constraints have served to narrow the choice to a single domain for each noun phrase. Using the guidelines suggested by Shlaer and Mellor, (i.e., considering the noun domain alone), the analyst might conclude that each of the noun phrases above should be modeled as an
object or class. The additional information about case relationships, however, allows a more refined analysis.

New heuristic guidelines are needed to use the information in the instantiated case frames. Accordingly, we contend that the strongest candidates for objects are related to noun phrases filling agent, instrumental and neutral case slots. In many instances, the head noun itself will refer directly to the entity (or process, proposition, etc.) to be modeled as an object (or class). Particularly when the neutral case is involved, however, the head noun may denote an attribute, operation or entity related to what is ultimately made an object.

Applying these heuristic rules to the case frames for the example sentences, it is seen that “tape robot” appears as the neutral case in sentences (1) and (3), and as the instrumental case in sentence (2). The analyst would thus deem it a strong candidate for an object. Another probable object is the “TSAR”, since it appears as the agent in the second sentence. Finally, the “intended function” (of the TSAR) appears as the neutral in sentence (2). Since it refers to an operation of the TSAR, however, this phrase is less likely to be an object than the other phrases just discussed.

The results of the analysis compare very favorably with the actual design from which the example sentences are taken. Of the four items (TSAR housing, tape robot, TSAR, and intended function) referred to in the sentences, those indicated as the strongest object candidates—the tape robot and the TSAR—are the only ones actually modelled as objects. Moreover, several specific functions—subclasses of the general “intended function” of the TSAR—are also modelled as objects in the actual design, thus supporting the validity of the proposed heuristics relating to the neutral case.

6 Additional Analysis

Besides identifying objects, classes and operations, an important goal of requirements analysis is the detection of incompleteness, inconsistency, and ambiguity. Though formal definitions are not given, we have suggested that some forms of these properties are detected during case analysis. When the required cases for a verb cannot be filled either because there are insufficient noun phrases or because the available noun phrases violate the verb’s selection constraints, incompleteness or inconsistency are implied. It should be kept in mind, however, that not all such situations necessarily result from deficiencies in the requirements documents, since the system’s dictionary entries cover only the usage most typical of technical writing. When word senses outside the scope of the dictionary are used, whether correctly or incorrectly, they are brought to the attention of the analyst. Conversely, word sense ambiguity is indicated when the intersection of the possible domains for a noun and the allowable domains for the case slot it fills includes more than one member. It might be possible to add mechanisms for resolving such ambiguity, but to do so could unduly complicate the system.

One common form of structural ambiguity which may be readily detected involves candidate prepositional phrases for the locative case that immediately follow another noun phrase. For example, an agent phrase, “by a technician”, may be added to example sentence (3), viz: “Every tape robot on site is installed by a technician in a TSAR.” Without some further ambiguity resolution mechanism, it is impossible to determine whether the phrase “in a TSAR” should be associated with the verb (i.e., as a locative case) or as a modifier with the preceding noun phrase. Indeed, many instances of such ambiguity cannot be resolved by any means. Once alerted to structural ambiguities in the requirements, the analyst can break the offending sentences into simpler ones that are more exact in meaning.

As a by-product of the analysis, the system records information useful in assessing the quality of requirements documents as they are processed. In particular, backtracking path lengths and frequency counters on ambiguities in the text are maintained. These two quantities are taken to
be indirectly proportional to the quality of the document. That is, if the system detects a large number of ambiguities and performs substantial backtracking to process a document, the analyst is likely have difficulty interpreting the document as well. Such an assessment, then, indicates that the substantive results of analysis are less reliable than for documents with fewer detected ambiguities and shorter backtracking path lengths.

7 Conclusion

Semantic case analysis provides a useful and powerful tool for the object-oriented requirements analyst. Two important areas of OORA are addressed: (1) identification of entities which should be modeled as objects in the software design; and (2) detection of ambiguity, inconsistency and incompleteness in the requirements documents.

The proposed extension to case grammar further enhances the requirements analysis process by focusing on basic verbs which are relevant to software design. Moreover, the representational format of the extended system guides the semantic treatment of verbs in the input lexicon in a uniform manner, and makes further automated semantic analysis more straightforward.

The implemented system is capable of parsing the input requirements and extracting the objects, actions, and attributes for use in creating an object-oriented model of the requirements. A question-answering capability exists to allow the user, either the client or the analyst, to ask specific questions about the model extracted from the requirements. From this inspection of the model, the user would be able to detect differences between his/hers and the system’s interpretations and proceed to make the necessary modifications to clarify his intentions.

The ability of the analysis system to process directly the requirements can be of great benefit to both the software engineer and the end user. Serving as a mediator, this system can lessen the chance for miscommunication by pointing out potential ambiguities, inconsistencies, imprecision, and other problems that can lead to misunderstanding between the client and the analyst.

The quality of the requirements documents can be assessed as a by-product. Indeed, the system maintains backtracking path lengths and frequency counters on ambiguities in the text. These two quantities are taken to be indirectly proportional to quality of the document. That is, the larger the counters and the lengths are, the poorer the quality is. In other words, if the system has to resolve a large number of ambiguities and perform substantial backtracking to process a document, then the analyst will have serious difficulties in interpreting such a document.

Acknowledgements We would like to thank Dzung Hoang, Van Nguyen, Judy Maxwell, and Johnette Hassell for their contributions to this research.

References


Yeh, R., P. Zave, A. Conn and G. Cole, Jr., “*Software Requirements: New Directions and Perspectives*”, in [36], pp 519-543.
