

User Guidance for Creating Precise and Accessible Property Specifications*

Rachel L. Cobleigh
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
rcobleig@cs.umass.edu

George S. Avrunin
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
avrunin@cs.umass.edu

Lori A. Clarke
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
clarke@cs.umass.edu

ABSTRACT

Property specifications concisely describe what a system is supposed to do. No matter what notation is used to describe them, however, it is difficult to represent these properties correctly, since there are often subtle, but important, details that need to be considered. PROPEL aims to guide users through the process of creating properties that are both accessible and mathematically precise, by providing templates for commonly-occurring property patterns. These templates explicitly represent these subtle details as options. In this paper, we present a new representation of these templates, a Question Tree notation that asks users a hierarchical sequence of questions about their intended properties. The Question Tree notation is particularly useful for helping to select the appropriate template, but it also complements the finite-state automaton and disciplined natural language representations provided by PROPEL. We also report on some case studies that are using PROPEL and on an experimental evaluation of the understandability of the disciplined natural language representation.

Categories and Subject Descriptors

D.2.1 [Software Engineering]: Requirements / Specifications—*elicitation methods, tools*

General Terms

Design, Verification

*This material is based upon work supported by the National Science Foundation under Award No. CCF-0427071, the U. S. Army Research Office under Award No. DAAD19-01-1-0564, and the U. S. Department of Defense/Army Research Office under Award No. DAAD19-03-1-0133. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, the U. S. Army Research Office or the U. S. Department of Defense/Army Research Office.

Keywords

Property specifications

1. INTRODUCTION

Property specifications are often used in requirements engineering to describe important aspects of a system's behavior. These specifications can then be used as the basis for software development and validation. Ideally, property specifications should be precise enough to support automated validation techniques and accessible enough to be readily understood by system developers. Automated validation tools typically accept property specifications represented in mathematical formalisms, such as temporal logic. Such formalisms have not been widely adopted by developers, in part because their use requires significant expertise [21]. In practice, developers tend to write requirements and design specification documents in natural language [16]. While natural language may offer accessibility, properties written with such informality are often ambiguous and thus are of limited value when doing rigorous analysis of the system. Additionally, accurately representing a property, even one that focuses on a very limited subset of the system's behavior, can be surprisingly difficult because of all the subtle details that should be considered. Overlooking these details often leads to inaccuracies that are not revealed until verification or testing, or perhaps even deployment. System developers may invest considerable effort trying to make sure that the system conforms to a property, only to later determine that the property has been specified incorrectly.

In previous work, we proposed a property-specification approach [20] that aims to guide users through the process of creating property specifications that are both accessible and mathematically precise. PROPEL, for "PROPErty ELucidator", is a tool that supports this approach by providing users with a set of property templates that explicitly indicate the variations that must be considered, thereby ensuring that important subtle details are not overlooked by users. Each template can be represented as a set of natural language phrases or as an extended finite-state automaton (FSA). Users can choose the appropriate variations using either or both representations, which are views of the same underlying representation. Users do not need to have expertise in a particular specification formalism to use the natural language representation. The resulting FSA specification, however, can be used as the basis for verifying system behaviors and other types of analyses.

Our previous work did not attempt to address the issue of how to guide users in selecting the appropriate template. A lack of guidance in this area was a weakness, since users often did not know how to choose among the templates and thus had difficulty getting started with the elaboration of their intended property. In this paper, we describe the Question Tree (QT) representation, which is designed to provide this additional user guidance. The QT representation is basically a decision tree, a representation that has often been used in requirements engineering. The content of the QT is based on natural language and its hierarchical format guides users through the elaboration of their intended property by asking questions and, for each question, providing a set of alternative answers for users to choose from. The QT breaks up the problem of deciding which template is most appropriate by asking users to consider only one differentiating attribute at a time. The hierarchical structure of the QT supports this isolation of concerns, only presenting questions to the users that are relevant in the context of their previous answers.

The next section of this paper reviews our property-specification approach and explains the concerns that motivated the introduction of the Question Tree representation. Section 3 discusses the Question Tree representation and gives an example of its use. Section 4 describes some preliminary evaluations of this approach. Section 5 describes related work, and Section 6 discusses the current status of PROPEL and future research directions.

2. THE PROPEL APPROACH

Our property specification approach is built upon the property patterns developed by Dwyer, Avrunin, and Corbett [7,8]. The property patterns work recognized that there are certain commonly-occurring types of properties that users of finite-state verification (FSV) tools, such as SPIN [13], SMV [15], INCA [4], and FLAVERS [9], want to check. Dwyer et al. observed that nearly all the properties found in the FSV literature could be classified into a small number of what they described as “high-level, formalism-independent, specification abstractions”. Each of the specification abstractions, or property patterns, is composed of two parts: a behavior (or “intent”, as it is called in the property patterns work) and a scope. A *behavior* describes the restrictions on occurrences of states or events, and a *scope* describes the parts of the state- or event-sequences within which those restrictions apply. For example, the Response behavior is a cause-and-effect relationship between a pair of states or events, in which the occurrence of the “cause” leads to an occurrence of the “effect”, and the Before scope requires the behavior to hold from the start of the state- or event-sequence until the first occurrence of a given state or event. The property pattern work identifies eight behaviors and five scopes that can be combined to create forty different properties. Dwyer et al. recognized, however, that other variations of the behaviors and scopes might be required, and their website [7] includes notes on how to modify the mappings to obtain some of these variations. This approach assumes that a developer with expertise in particular specification formalisms can identify the ways in which a desired property might differ from the forms given in the property patterns and, perhaps with some assistance from those notes, do any necessary tailoring.

Although the property patterns are extremely useful, it is still very difficult to accurately formulate a property specification based on them. In contrast, our approach extends these property patterns with templates that explicitly indicate the alternative *options* associated with each property pattern. These *property pattern templates* are each composed of a *scope template*, which contains options related to the selected scope, and a *behavior template*, which contains options related to the selected behavior. By making decisions about these options, users can create 32 possible scopes and 139 possible behaviors, which can be composed into over 4,400 different possible properties. As with the property patterns work, however, the space of properties available is not intended to cover all possibilities, just variations on the most commonly-occurring properties. Since the majority of the properties that Dwyer et al. found in their survey [8] are covered by only four of the eight behaviors that they identified, PROPEL currently supports variations on just those four¹. In addition, although the property patterns include both state- and event-based forms of the property patterns, our work currently concentrates only on the event-based forms of the property patterns.

Using PROPEL, a user instantiates a property by making decisions about, or *resolving*, each of the available options in a selected property pattern template. To resolve an option, the user selects one of the option’s possible *settings*. A property is said to be *partially instantiated* if not all of the options in its associated property pattern template have been resolved. For a property to be *fully instantiated*, not only must all the options be resolved, but the user must also define the set of events, or the *alphabet*, that is considered relevant to the meaning of the property. Like the property patterns, the property pattern templates have up to four pre-defined placeholders, or *parameters*, that can be associated with user-defined event names: at most two for the property’s behavior and at most two to represent the start and end delimiters for the scope. A parameter’s default name is displayed in the property pattern templates until the user associates a user-defined event with the parameter. We refer to events that are associated with the behavior parameters as *primary events*, or “events of primary interest”. Although the available property pattern templates have at most four parameters, users can define additional events, referred to as *secondary events*, or “other events in the alphabet of the property”. Secondary events can be used when a property must constrain more than just the occurrences of primary events. For example, it may be important that certain other events do not occur in between an occurrence of a “cause” and a subsequent occurrence of its “effect”, where the “cause” and the “effect” are the property’s primary events. In such a situation, users can include those other events in the alphabet of the property and define the property so that it explicitly prohibits all occurrences of those secondary events between the “cause” and its “effect”.

With the addition of the QT representation, the property pattern templates are represented using three different notations that together are meant to bridge the gap between

¹We do not yet support the chain property patterns, which involve ordered sequences of states/events, nor the bounded existence pattern, which limits the number of occurrences of a state/event.

precision and accessibility in property specification. Users can work with any one of these notations, or they may choose to use any combination of them. One notation is a graphical, extended FSA template representation that resolves to an FSA representation. The FSA template representation helps developers see the options that need to be considered by representing those choices with optional versions of FSA states, transitions, and transition labels. In addition, since the fully-instantiated form of this notation is an FSA, which is mathematically well-defined, this notation offers the precision necessary for many types of analysis, including FSV. In our experience, the FSA graphical notation seems to be more accessible than most other formal notations. A second notation is a disciplined natural language (DNL) template representation that is intended to appeal to those users who prefer a natural language description of their property. The DNL template representation provides a short list of alternative phrases that highlight the available options, as well as a few synonyms to support customization. The QT representation is a third representation. As mentioned, this representation was originally designed to help users select which property pattern template to use when elaborating their property. We soon discovered, however, that this notation was useful not only for helping users to select the appropriate property pattern template, but also for helping them to resolve the options associated with that selected template. After users select a template in the QT representation, the changes they make using any one of the representations are reflected in the other two representations wherever possible.

3. QUESTION TREES

The QT is an interactive format where the user is presented with a question and a choice of possible answers. A user can select only one answer for each question, and, based on which answer is selected, new questions and their associated answers may then be presented for further consideration. In keeping with the hierarchical nature of the QT, we refer to each new question that can be revealed after answering a given question as a *child question* of that previous answer, and we refer to the previous question as the *parent question* of that child question. When an answer to a question is selected and this selection reveals a new set of child questions, then, for as long as that answer remains selected, all those child questions and their associated answers remain visible and can be revisited if necessary. By selecting a different answer to that question, the user will explore a different set of child questions that are relevant to the new answer and will arrive at a different property altogether. The QT hides all questions that are not relevant to the currently selected answers, allowing the user to focus on one set of concerns at a time. The root question of the QT and its associated answers are always visible. In content, the QT text is similar to the corresponding DNL template text.

In PROPEL, we actually break the QT representation into two separate parts, one for the scope templates and another for the behavior templates. The QT representation is structured such that the questions that lead to the choice of scope or behavior template must be answered first, and in a pre-defined order, before questions that resolve the options associated with those templates. With one exception, which is discussed in Section 3.2, the questions about the options in

each of the scope and behavior templates are conceptually orthogonal to each other and are thus represented as sibling questions that can be answered in any order. Note that the answers to a given question are always orthogonal to each other and can be shown to the user in any order.

It is only after a user has selected scope and behavior templates by answering the initial questions in the QT that the user can switch to working with the FSA or DNL representations to resolve the options in the selected templates. Thereafter, the user can work with any or all of the three representations of the selected property pattern template to resolve the options. The user can use the QT to select a different property pattern template at any point in the elucidation of the intended property and the FSA and DNL representations will automatically update to match the new selection.

3.1 Behavior Question Tree

The Behavior QT (BQT) organizes the set of available behaviors into four behavior templates. The BQT’s guidance for choosing among the four behavior templates is structured along two major questions. First, how many events of primary interest are there, one or two? Second, if there are two events of primary interest, what is the relationship between them? Selecting answers to these questions selects a behavior template and the user can then continue to make decision about the selected behavior template’s options, if any exist, using any of the three notations.

3.1.1 Instantiating a Property’s Behavior

Figure 1 gives a PROPEL screen capture of the BQT representation of an instantiated behavior. In this figure, each tree node is either a question or an answer. Answers to a question are indented under the question and child questions are indented under the answers to their parent questions. The highlighted nodes in the BQT are answers that the user has selected in response to the questions. Unselected answers are not highlighted. Child questions are only visible when the user selects the appropriate answer to their parent question.

To see how the BQT in Figure 1 was derived, consider the following property specification, which is part of a case study we are performing:

If the nurse discovers that the patient’s type and screen (T&S) are not available in the lab, the nurse must obtain a blood specimen from the patient.

Let us assume that the PROPEL user identifies two events that are in the intended property’s alphabet, *no-T&S* and *obtain-specimen*, and that the user makes *no-T&S* the first primary event and *obtain-specimen* the second primary event². The first question the BQT asks the user about the intended property is “How many events of primary interest are there?” For this example, the user would choose the answer with two events. The BQT then reveals the second question, which is about the nature of the relationship

²This assignment of event names to the parameters would be done via a dialog box that is not shown here. After this assignment, the QT substitutes the assigned names for the parameters, as shown in Figure 1.

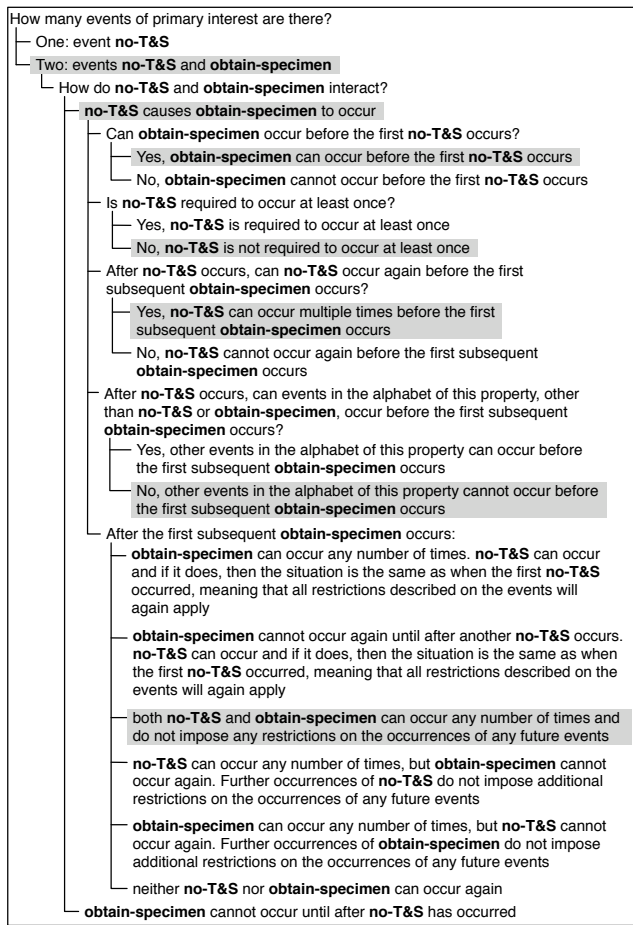


Figure 1: An Example Response Behavior in the Behavior Question Tree

between the two events. The BQT offers two choices: “no-T&S causes obtain-specimen to occur” or “obtain-specimen cannot occur until after no-T&S has occurred”. If the lab does not have a T&S for the patient, the nurse must obtain a blood specimen from the patient so that the lab can get a T&S. Thus, the user selects the first choice about how the two events are related, which is the Response behavior template. The BQT then reveals the five child questions that concern the Response behavior template’s various options. In this section, we continue to elucidate this behavior by using the BQT, but since a behavior template has been selected, the user could now use any or all of the three alternative representations to continue elucidating the details of the property’s behavior.

Since the Response behavior template’s child questions are orthogonal to each other, they can be answered in any order. In this discussion, we answer them in the order that they are shown in the BQT. The first child question asks whether *obtain-specimen* is allowed to occur before the first occurrence of *no-T&S*. Since there are many reasons to obtain a blood specimen besides just discovering that no T&S is available, the user selects the “Yes” answer. To give a brief illustration of how the QT, the FSA template, and the DNL template representations are related, the same option

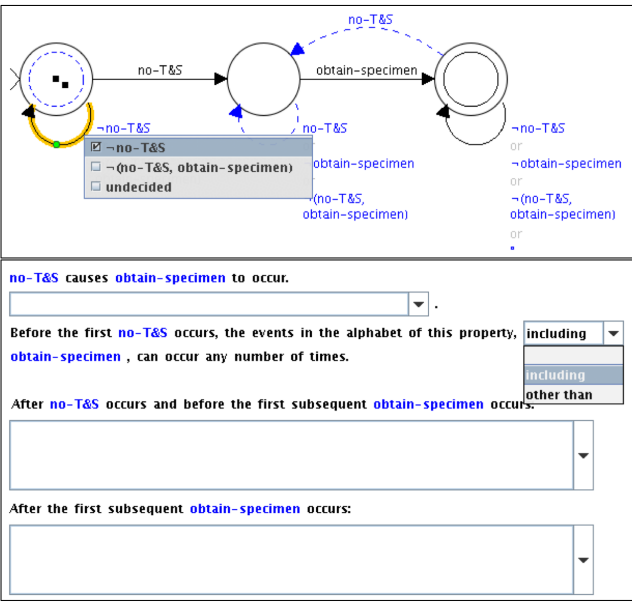


Figure 2: Example Partially-Instantiated FSA and DNL Templates

setting that is selected to answer that question in the BQT is selected in the FSA and DNL template representations in Figure 2, which gives PROPEL screen captures of the FSA and DNL template representations of the Response behavior template at this point. In the FSA template, the start state’s self-loop is on all the events in the alphabet of the property except for *no-T&S*, where the “~” shorthand notation is a set-complement operator for all (i.e., both primary and secondary) events in the alphabet. In the DNL template, the second combo box contains the selected phrase “including”. Throughout the elucidation of this property, changes can be made to any of the three representations and those changes will be reflected in all of the representations simultaneously.

The next child question in the BQT representation of the Response behavior template asks whether *no-T&S* is required to occur at all. Since the lab may have a T&S for the patient available when the nurse inquires, the user selects the “No” answer. The next child question asks whether, after the first *no-T&S* has occurred, can *no-T&S* occur again before the first subsequent *obtain-specimen* occurs. Since the nurse can repeatedly check with the lab to find out if the patient’s T&S is available before obtaining a blood specimen, the user selects the “Yes” answer.

The next child question asks whether, after the first *no-T&S* occurs, events other than either *no-T&S* or *obtain-specimen* can occur, before the first subsequent *obtain-specimen* occurs. Here is an example where the user could put secondary events into the alphabet of the property and constrain those secondary events by deciding whether they are allowed to occur or are prohibited from occurring between a *no-T&S* and the first subsequent *obtain-specimen*. There is an event that should never occur after *no-T&S* and before the first subsequent *obtain-specimen*: the nurse can never label the specimen vial (*label-vial*) before putting a blood specimen in the vial. Thus, the user could indicate that *label-vial* is

a secondary event and then select the answer, “No, other events in the alphabet of the property cannot occur before the first subsequent *obtain-specimen* occurs”.³

The final child question in the BQT representation of the Response behavior template asks what can happen after the first *obtain-specimen* occurs; whether further *obtain-specimen* events can occur, whether new *no-T&S* events can occur, and if new *no-T&S* events can occur, whether they again require subsequent *obtain-specimen* events to follow them. The BQT presents the user with six answers to this question, each of which represents one possible combination of decisions about the above issues. After the nurse obtains a blood specimen and sends it to the lab, the nurse can check repeatedly with the lab to find out if the patient’s T&S is available yet. The nurse would normally check with the lab until the T&S is available, and seeing that it is not available in the interim would not mean that the nurse has to go obtain more blood specimens from the patient. The nurse may also be asked to obtain blood specimens for purposes other than a T&S, so after obtaining the initial blood specimen for the T&S, the nurse could obtain other blood specimens. Thus, the user selects the third of the six possible answers, as indicated in Figure 1.

3.2 Scope Question Tree

The Scope QT (SQT) provides guidance for expressing an intended property’s scope. The SQT organizes the set of available scopes by using four scope templates, based on the scopes proposed by Dwyer et al. in [8]. Their property patterns include five scopes and, as was true with the behaviors, they expect users to have considerable expertise with the specification formalisms in order to tailor those scopes as needed. Our scope templates extend the property pattern scopes in a way similar to how the behavior templates extend the Dwyer et al. behaviors. Specifically, we explicitly provide a number of possible variations in the scopes and present these alternative options to the user via scope templates. These four scope templates, with their options, can be used to create the five Dwyer et al. scopes and many more.

3.2.1 The Scope Templates

A property’s behavior must hold in every *scope interval* in any event sequence, where a scope interval is a portion of the event sequence that is defined in terms of a *starting delimiter*, an *ending delimiter*, or both. For example, a scope interval might begin with the first occurrence of the starting delimiter and end with the first subsequent occurrence of the ending delimiter. The scope templates use parameters named **start** and **end** for the starting and ending delimiters, respectively, that can be replaced with user-defined event names. In scope templates that do not have the **start** parameter, the beginning of the event sequence is the default starting delimiter. Likewise, in scope templates that do not have the **end** parameter, the end of the event sequence is the default ending delimiter. Our approach puts two restrictions on the starting and ending delimiters. The

³A separate property would be needed to prevent *label-vial* from occurring before *obtain-specimen* occurs in other situations, since our approach limits each property to specifying a very focused subset of system behavior.

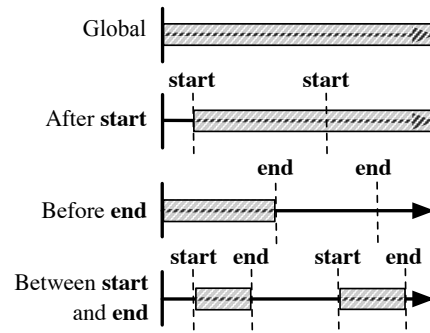


Figure 3: Examples of Scopes Based on the Four Scope Templates

first is that the starting and ending delimiters cannot be in the alphabet of the behavior. We make this restriction because some undesirable interactions could occur; for example, a user could create a property with a Response behavior whose stimulus event is the same as the scope ending delimiter, and thus no event sequence could ever satisfy this property. The second restriction is that the starting and ending delimiters must be different events. This restriction also avoids some complications, but it does not allow some forms of two-event alternation. Relaxing these restrictions requires careful consideration and is something we intend to explore in the future.

The four scope templates available in PROPEL are: Global, After **start**, Before **end**, and Between **start** and **end**. The Global scope template has no alternative options for the user to consider and does not use either delimiter; the behavior is required to hold from the start of the event sequence until the end of the event sequence. The Global scope template is the default for a new property. The After **start** scope template requires that in any event sequence, the behavior must hold after an occurrence of the **start** event until the end of that event sequence. The Before **end** scope template requires that the behavior must hold from the start of any event sequence until the first occurrence of the **end** event in that sequence. The Between **start** and **end** scope template requires that the behavior must hold after an occurrence of the **start** event until the first subsequent occurrence of the **end** event. Only this scope template can define multiple scope intervals in a single event sequence. Figure 3 shows examples of four scopes that could be instantiated based on the four scope templates. In this figure, the four horizontal lines labeled by the scope template names are the timelines on which the events occur in sequence. The vertical dashed lines that are labeled with either “**start**” or “**end**” denote points in the event sequences at which those events occur. The shaded regions on a given line show where the scope intervals exist in the event sequence.

There are a number of options associated with each scope template. The Between **start** and **end** scope template contains all the options that are associated with the After **start** and Before **end** scope templates, plus one more option that neither of those templates contains, so we only describe the Between **start** and **end** scope template in detail here.

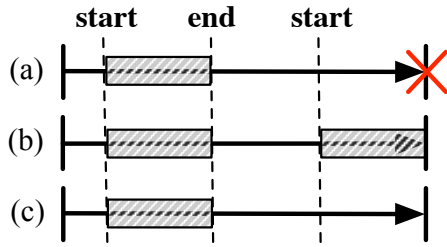


Figure 4: Three Ways To Interpret a Missing End Delimiter

The Between **start** and **end** scope template has five options. One option determines whether **start** must occur at least once in a given event sequence. Another option determines what happens if **start** occurs more than once before the end of the given scope interval. There are two alternative settings associated with this option. One setting is that the first occurrence of **start** begins this scope interval and later occurrences of **start** are ignored. This is the setting used in the example After **start** scope in Figure 3. The other possible setting is that only the last occurrence of **start** (before a subsequent occurrence of **end** that ends this scope interval) begins this scope interval; that is, each occurrence of **start** resets the beginning of this scope interval.

Another option in the Between **start** and **end** scope template determines whether multiple scope intervals can exist in a single event sequence. There are two possibilities of what can happen after **end** occurs to end a scope interval. One possibility is that multiple scope intervals can exist (consecutively) in the event sequence, because a subsequent occurrence of **start** potentially begins a new scope interval. The second possibility is that there can be only one scope interval in the event sequence, because a subsequent occurrence of **start** does not begin a new scope interval.

Figure 4 illustrates the two remaining options in the Between **start** and **end** scope template, using the same notation as is used in Figure 3. All the other options can be resolved in any order, but for these two options, the first must be resolved before the second. The first option determines whether **end** is required to subsequently occur after an occurrence of **start**. Figure 4(a) corresponds to the setting where **end** is required to subsequently occur after an occurrence of **start**. If **end** does not occur, the event sequence cannot satisfy the property, which is denoted by this timeline being marked with an “X”. The other possible setting is that **end** is not required to occur after an occurrence of **start**. In this case, the second option must be resolved. This option determines what should happen if, after an occurrence of **start** begins a scope interval, **end** never occurs to end this scope interval. Figure 4(b) shows the setting where this scope interval exists until the end of the event sequence, even though the second occurrence of **start** has no subsequent occurrence of **end** to end the scope interval. Figure 4(c) shows the other possible setting where this potential scope interval does not exist without a subsequent occurrence of **end** to terminate it.⁴

⁴The two possible settings for this option distinguish two of

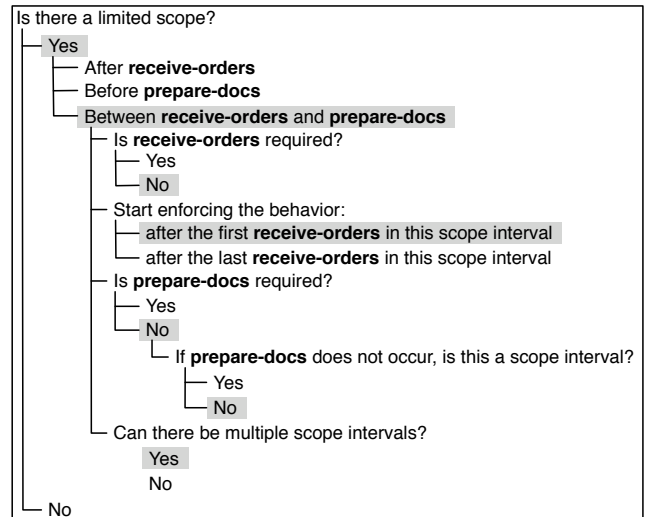


Figure 5: An Example Between Scope in the Scope Question Tree

3.2.2 Instantiating a Property’s Scope

In Figure 5, we use the SQT to instantiate a scope for the example property given in Section 3.1.1. Suppose that the full property specification is the following:

After the nurse receives a physician’s order to transfuse blood into a patient, if the nurse discovers that the patient’s type and screen (T&S) are not available in the lab, the nurse must obtain a blood specimen from the patient before the nurse can prepare documentation for picking up the necessary unit(s) of blood product from the blood bank.

The new phrases in the example property imply a beginning and an end to when it is required that the nurse check the lab for the T&S and subsequently obtain a blood specimen if necessary. Let us assume that the user identifies two new events in the property that correspond to the nurse receiving a physician’s order (*receive-order*) and the nurse preparing documentation for blood pick-up (*prepare-docs*), and let those two events be associated with the **start** and **end** parameters, respectively. (Again, we do not show the dialog box where event names are assigned to the scope parameters, although the results of this assignment are shown in Figure 5.) Given these events, the user decides that the intended property’s behavior is only required to hold within certain limits and selects the “Yes” answer to the first question in the SQT. The SQT then reveals the three delimited scope template choices. Since “Between *receive-order* and *prepare-docs*” is the only choice that refers to both the starting and ending delimiters, the user selects this answer and the SQT reveals four child questions that the user must answer about the alternative options in the selected scope template. In this section, we continue to elucidate this scope by using the SQT, but since a scope template has been selected, the user can now use any or all of the three alter-

the scopes in the Dwyer et al. work: their “After **start** Until **end**” scope corresponds to the first setting (Figure 4(b)) and their “Between **start** and **end**” scope corresponds to the second setting (Figure 4(c)).

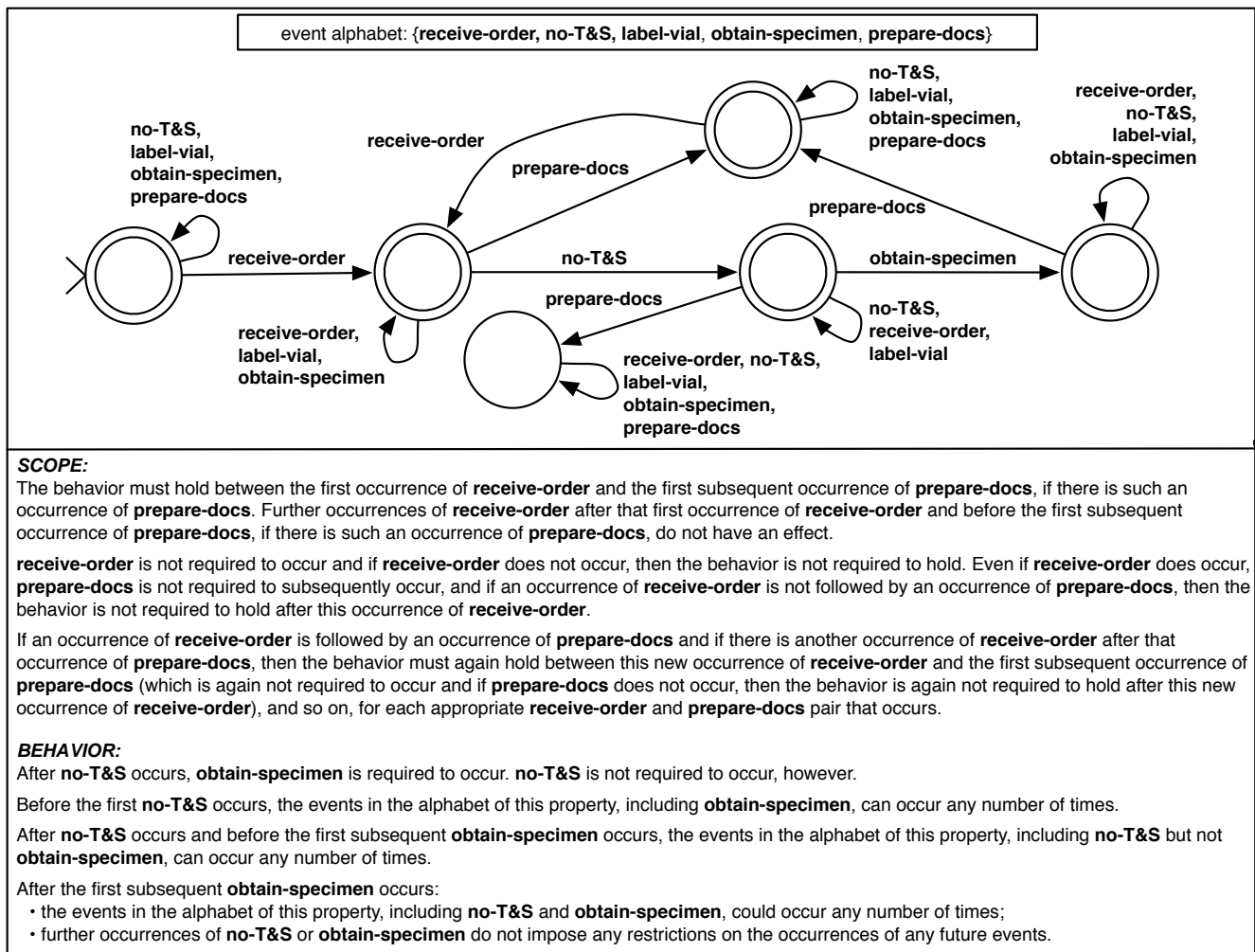


Figure 6: Example FSA and DNL Property Representations

native representations to continue elucidating the details of the property’s scope.

Since the Between **start** and **end** scope template’s child questions are orthogonal to each other, they can be answered in any order. In this discussion, we answer them in the order that they are shown in the SQT. The first child question asks whether *receive-order* is required to occur at least once. Since it is possible for a patient to be admitted to a hospital and to never require a blood transfusion, the user selects the “No” answer to this question. The next child question is concerned with what should happen if more than one *receive-order* occurs before the first subsequent *prepare-docs* occurs: should the given scope interval be started after the first occurrence of *receive-order* or the last occurrence of *receive-order*? For simplicity, let us assume that the intended property is meant to be in the context of a single patient. In this situation, the most common reason that the nurse would receive multiple physician orders for the patient’s blood transfusion is that one or more physicians, not realizing that a transfusion had already been ordered, would place additional, unnecessary, orders. In this case, the nurse

would contact the physicians who have sent the redundant orders and would ask them to retract those orders. Given this knowledge, the user decides to select the answer “Start enforcing the behavior after the first *receive-order* in this scope interval”.

The next child question is concerned with whether or not *prepare-docs* is required to occur at least once and end the given scope interval. It is possible that a physician may order a blood transfusion for a patient but that for some reason (e.g., the physician retracts the order, the patient’s status changes), the blood transfusion never actually occurs. In such a situation, the nurse may never have to pick up the units of blood product from the blood bank, and thus the nurse may never have to prepare the documentation for doing so. Knowing this, the user selects the choice where *prepare-docs* is not required to occur to end the given scope interval. The SQT then reveals the child question under this answer, which asks whether the behavior still must hold if *prepare-docs* never occurs to end this scope interval. Since events might occur that would negate the need to check for a T&S or obtain a blood specimen (e.g., the physician could

Case Study	Total # of Properties	PROPEL Expresses	Percent Covered
Blood Transfusion	23	16	70%
Chemotherapy	33	31	94%
Emergency Dept.	19	18	95%
Blood Bank	9	5	56%
TOTAL	84	70	83%

Table 1: Case Study Summary

cancel the order before the nurse gets to prepare documentation), the user selects the “No” answer to the question: “If *prepare-docs* does not occur, is this a scope interval?”

The final child question is about what happens after an occurrence of *prepare-docs* ends a scope interval. Does a subsequent occurrence of *receive-order* begin a new scope interval in which the nurse discovers that there is no T&S available, thereby requiring that a new blood specimen be obtained? It is certainly possible that after the nurse has prepared documentation for blood pick-up (and probably after the nurse also completes the administration of the blood transfusion), the nurse may receive another physician order for a blood transfusion for the same patient. Given this knowledge, the user selects the “Yes” answer to the question: “Can there be multiple scope intervals?” With this final answer, the user has instantiated a scope. Given the example sequence of starting and ending delimiters in Figure 4, this scope would match Figure 4(c).

Now that the user has answered all the questions in both the BQT and the SQT and has associated events with the selected behavior and scope templates’ parameters, the example blood transfusion property is fully instantiated. As described earlier, there is now also an FSA and a DNL paragraph that describe this property (see Figure 6).

4. EVALUATION

We are currently evaluating this approach by using PROPEL to specify properties in four case studies. We have also completed an experimental evaluation of the accessibility of the DNL property representation.

4.1 Case Studies

We have used PROPEL to represent the properties encountered in 4 case studies we are undertaking in the medical domain. For each case study, Table 1 shows the name of the medical process, the number of properties, and how many of those properties can be expressed in PROPEL. We have encountered a total of 84 properties thus far and were able to use PROPEL to express 83% (70 of 84) of them. In the first three case studies, we are eliciting the properties from several medical professionals by interviewing them, capturing the properties first in natural language, and then together using PROPEL to elaborate the precise details of those properties. Not surprisingly, the nurse involved with the first case study has only been comfortable with the QT and DNL representations of the properties and does not refer to the FSA representation at all. She has observed that misinterpreting the subtle details that the options expose is a common source of errors in medical processes and that guidelines that precisely specified those details would contribute greatly to-

wards improving education and practice in medicine. The fourth case study is the only study that did not involve medical professionals directly; instead these properties were initially obtained from a blood bank laboratory manual and then an industrial software engineer from a medical software company worked with us to specify 3 of the properties using PROPEL. In contrast to the medical professional described above, this user preferred to work solely with the QT and FSA notations. She specified the properties by using the QT notation to resolve all the options in the selected scope and behavior templates, and then used the FSA notation to check the properties’ correctness.

Most of the case study properties were concerned with checks that must be done before administering a particular treatment or with how to handle negative reactions that a patient may have to treatment. We found that 63% of the behaviors are some form of Precedence, where one event cannot occur until after another event occurs (e.g., the checks that must be done before a treatment), and 21% of the behaviors are some form of Response (e.g., the necessary response to a patient’s negative reaction to a treatment). It is interesting to note that the relative prevalence of these two behaviors differs from what was reported in [8]; there, 5% were some form of Precedence and 44% were some form of Response, the most prevalent of the behaviors surveyed in that work. It is likely that this difference is due to the fact that these two collections of properties are drawn from different domains; the earlier work was primarily a survey of the FSV literature. With respect to the properties’ scopes in our case studies, 86% are Global and 8% are some form of Between. This distribution of the scopes is close to the distribution reported in [8]; there, 79% were Global and 7% were some form of Between.

Since PROPEL uses an event-based model without real-time support, we model the 12 of the properties that refer to state information or timing constraints with events that are checks of deadlines or data values, respectively. Other specification approaches that address these concepts more directly might be preferable in the long term. Of the 14 properties that cannot be handled in PROPEL, we expect that all but 2 can be handled by fairly straightforward extensions that we plan to make: chain property patterns, a pattern where one event blocks another event from occurring, and disjunction⁵ will be supported soon, and we expect that conjunction can be supported by using an extension to the property patterns, such as one of the composite propositions in [17]. With these extensions, we believe that PROPEL would be able to express 98% of the properties we have seen so far in these case studies. As these case studies continue to evolve, it will be interesting to see if PROPEL continues to be as effective.

4.2 DNL Experimental Evaluation

Since first describing PROPEL in [20], we have repeatedly tried to improve the natural language phrases used in the DNL. For the current version of the system, we conducted an experiment in which we tried to evaluate how well users understand the DNL representations of properties. For this study, we asked 14 human subjects with a background in

⁵While we can create a more abstract event that is a disjunction of the two events, the domain experts wanted to model them as two distinct events.

computer science (graduate students and technical staff from our department) to translate properties represented in DNL into equivalent FSA representations. We then measured how well the subjects understood the DNL by comparing their FSAs to the PROPEL FSAs that the DNL was intended to describe. Given the limited number of subjects who participated, we selected a representative sample of the PROPEL properties. Each of the 14 subjects was given four properties represented in DNL. One of those four was a very simple property with a Global scope and a 1-event behavior that was included for training purposes only and, thus, are not counted in the results described here. The remaining three properties varied in the complexity of their scopes. In addition, each property was given to only one subject and the overall set of properties covered all of the option settings, although obviously not all possible combinations of those settings.

The experimental results indicated that 40% (17 of 42) of the subjects’ FSAs were an exact match for the PROPEL FSAs and 64% (27 of 42) were “close” to the PROPEL FSAs. We say that a subject’s FSA is “close” to the PROPEL FSA if the subject’s FSA can be transformed into the PROPEL FSA by changing no more than one⁶ transition (i.e., adding it to the FSA if it is missing, or changing its label or destination state) or the accepting status of one state. Although we could determine the language accepted by one of the FSAs but not the other, it is not clear how we would use that language to measure the size of the difference between the two FSAs; instead we use this cruder transformation measure.

Although these translation percentages are low, the inherent difficulties of accurately translating from natural language to FSAs, even for simple properties, would likely bias the results. The more complicated properties, specifically those with a Between scope, had particularly low percentages. The completed property in Figure 6 has a Between scope and illustrates why it can be so difficult for subjects to manually translate from the DNL to the FSA. Although the DNL representation of the Between scopes could probably be improved, it is likely that the subtleties of this complex scope make translating any description of it very difficult. For example, 5 of the subjects’ 10 Between scope FSAs that were deemed not “close” omitted a state that even we initially missed when we developed the PROPEL FSAs. Given all the potential pitfalls, we believe the percentage of “close” FSAs indicates that the DNL representation is a promising approach for supporting precise and accessible property specifications.

5. RELATED WORK

Many property specification approaches aim to provide either accessibility or precision; few approaches try to provide both. Of those that do try to provide both, there are two main variations in how these qualities are addressed. One variation is to use graphical or tabular approaches and the other variation is to combine a natural language (NL) description of a property with a formal model. In the approaches that use the latter variation, there are three directions that have been pursued: one is to use NL-processing

⁶Multiple transitions from one state to another are treated as one transition with multiple labels that goes from the former state to the latter state.

Property Complexity Category	Total # of Properties	Exact Match	1 Error	% Close
After/Before scope, 1-event behavior	6	5	1	100%
Between scope, 1-event behavior	8	1	2	38%
Global scope, 2-event behavior	14	9	3	86%
After/Before scope, 2-event behavior	8	2	3	63%
Between scope, 2-event behavior	6	0	1	17%
TOTAL	42	17	10	64%

Table 2: DNL Experimental Results

(NLP) techniques to extract formal models from property specifications that are written in a restricted subset of NL, another direction is to simply annotate formal models with NL descriptions, and a third direction is to enable users to develop both the formal model and the NL description in parallel. PROPEL takes this latter approach.

For the purposes of brevity, we will only mention in passing some of the property specification approaches that are focused on addressing issues other than how to combine an NL description of a property with a formal model. Structured natural language recommendations (e.g., the NL templates in [5]) provide accessibility but not precision, and formal specification languages (e.g., the temporal logics in [3]) provide the opposite. Graphical approaches such as [19] and tabular approaches such as [12] provide some level of accessibility while still supporting a formal model, but these approaches do not incorporate NL outside of the names of the events or states used in the model, so we do not discuss them further here.

NLP-based approaches that aim to support both precision and accessibility in property specifications extract various formal models from properties expressed in several different restricted subsets of NL. There have been a number of approaches that translate these restricted subsets of NL into different logics, such as the work in [1]. The Attempto Controlled English (ACE) project [10] uses NLP to translate NL property specifications into first-order logic and also provides annotated NL templates for non-expert users. Similar to the work done for ACE but with a much less restricted subset of NL, Ambriola and Gervasi [2] have developed the CARL and CICO/CIRCE tools to translate NL property specifications into propositional logic and back again. Based on the restricted subsets of NL that these two tools use, they can also provide suggested NL phrases for common relationships between propositions. Gervasi and Zowghi [11] noted a limitation in this approach, however: in their experience, “propositional logic was found to be adequate to express high-level requirements, but not the details of how the system should behave”. In addition, one limitation of all these NLP-based approaches is that they must make assumptions about how to interpret the NL property specifications that they are given, because those specifications are often ambiguous. It is possible that presenting users of NLP tools with interpretation alternatives like the options given in PROPEL might help in improving the accuracy of the resulting formal property representations. In any case, the use of natural language in PROPEL is much less ambitious than that of these NLP-based approaches. PROPEL provides two property representations that are based on natural language (the QT and the DNL) and a formal property representation (FSA), but our work does not attempt to understand NL,

even in restricted domains.

Gervasi and Zowghi [11] recently conducted an experiment that is similar to our DNL experiment, to evaluate their NLP tool, CARL. To discover whether humans would interpret a set of NL property specifications the same way that CARL did, the researchers asked 15 subjects to translate the NL property specifications into propositional logic and then answer a multiple-choice questionnaire about valid inferences that could be made from the set of logical expressions. CARL’s translations matched the subjects’ translations in 72% of the cases, and the authors say that this percentage was probably due to human misinterpretation of the NL statements. It is interesting to note that this was the translation percentage, even for properties that could be expressed as simple propositional invariants. This translation percentage was only slightly higher than the percentage in our DNL experiment, and supports our assertion that translation from natural language into a formal model can be difficult, even for simple properties.

Another approach to combining an NL description of a property with a formal model is to simply annotate the formal model with NL comments that are not in themselves intended to function as property specifications, but instead are just a means of conveying the basic gist of what the property is meant to express. Several approaches take this route, including the Dwyer et al. property patterns work [7,8], which PROPEL extends with templates that capture variations in the subtle details of those patterns. Unlike the property patterns work, however, PROPEL provides a DNL representation that is intended to function as an accessible property specification. A number of approaches, such as Drusinsky’s (N)TLCharts [6] and the work done by Mondragon, et al. [17,18], extend the property patterns and, like Dwyer et al., also include NL annotations that are not intended to be used as property specifications. The (N)TLCharts provide some very brief NL descriptions of simple temporal logic properties (based on the Dwyer et al. NL descriptions) to annotate Statecharts. In their Prospec tool, Mondragon et al. extend the property patterns via Composite Propositions (CPs), which are logical formulae that compose multiple proposition primitives in a predefined way (e.g., conjunction). Prospec offers NL and formal-logic descriptions of these CPs and binary decision trees to guide users to a choice of one of the five property pattern scopes and one of the CPs. These decision trees impose a pre-defined order on the decisions, but some of the decisions could actually be made in a different order. The PROPEL QT, by contrast, is not required to be binary (thus affording slightly more flexibility in the order that questions can be answered) and it focuses on clearly expressing a number of variations in the subtle details in the scope and behavior templates. While the NL descriptions in Prospec map to the underlying formal logics that the tool supports, the NL descriptions are not intended to be used as property specifications; they are mainly just expansions on the Dwyer et al. NL descriptions. In contrast, the fully-instantiated forms of the PROPEL QTs and the FSA and DNL templates are all designed to be used as property specifications. Finally, Konrad and Cheng [14] have also built on the property patterns. They provide an NL-based representation of their real-time property patterns that is modeled after our DNL, as well as a formal represen-

tation. Their NL-based property representation is designed to capture only the basic concepts of their real-time behaviors, however, since they do not yet support the possible variations in those behaviors nor any kind of decision-tree guidance for selecting among their real-time property patterns.

6. CONCLUSIONS

The results from our case studies showed that PROPEL could handle most of the properties that were encountered. It is interesting to note that of the four case studies, only one was specifically selected to evaluate PROPEL. The three other case studies are part of a larger investigation on ways to reduce medical errors. Although the property patterns covered 92% of the properties examined in [8], these properties were mostly selected from the finite-state verification literature, which was the domain used to help develop the property patterns. In our case studies, the medical professionals had no knowledge of finite-state verification or requirements engineering. Thus, it is somewhat surprising that we achieved similarly promising results; we are able to handle 83% of the properties in these case studies, although we currently only support four of the property pattern behaviors with our templates. In our case studies, computer scientists formulated the properties based on discussions with the medical experts. The medical professionals were taken aback by the amount of detail required to precisely capture their guidelines. In fact, one nursing faculty member remarked that working with us on this project significantly changed the way she views and teaches medical procedures.

The majority of the properties that could not be supported by PROPEL appear to be ones that will be supported in the next planned version of the system. All involved property patterns or simple extensions that we do not yet support. A few involved real-time constraints or combinations of state and events that perhaps could be better supported by representations designed to address those types of properties [12,14].

Results from the DNL experiment were also promising. For the majority of the properties in the experiment, the computer scientists could exactly or almost exactly formulate an FSA representation of the intended property based on the DNL descriptions used in PROPEL. Mistakes tended to be made when the Between scope was involved. Although we will reexamine the phrasing for this scope, there is no doubt that it is a difficult concept that results in a complex FSA. We may just have to acknowledge that the Between scope can not be represented succinctly or quickly comprehended in its entirety.

Although anecdotal, we have a few observations about the elicitation support provided by PROPEL. The various domain experts seemed comfortable with the QT representation for selecting the scope and behavior templates and for resolving the options in those templates. The industrial computer scientist used the FSA representation to evaluate whether the answers she gave in the QT representation provided what she wanted. Non-computer scientists, on the other hand, liked the QT representation for creating a property, preferred the DNL for reading about a property, and avoided the FSA representation altogether. We also found

that the QT and DNL representations are relatively easy to understand, even for non-computer scientists.

The PROPEL tool supports all the capabilities described in this paper. While creating properties for the case studies, we soon discovered that we needed support for collections of properties. PROPEL provides two capabilities that support property collections. One is a project directory hierarchy, where users can define projects and subprojects and then associate each property with a project in the hierarchy. Another capability is the summary view, which provides a tabular representation of the project organization and attributes of the properties associated with a project (and possibly its subprojects). For example, a summary could be created that lists all the primary events, secondary events, scope events, and comments associated with each property in the project. We have found that a summary provides a useful high-level view of a set of properties and their focus.

There are several issues that we intend to address in the future. In addition to supporting chain patterns, there are several other patterns that we plan to support, since these patterns have arisen several times in our case studies. Some of these patterns, such as chains, are included in the property patterns and some are not, such as the concept of one event blocking the subsequent occurrence of another event and the concept of alternation. Properties that involve a conjunction or disjunction of events also merit further scrutiny. Finally, we intend to investigate the possibility of loosening the restrictions on the scopes' alphabets.

7. ACKNOWLEDGMENTS

We thank Elizabeth Henneman and Jamieson M. Cobleigh for their helpful comments, and we also thank Valerie A. Gartland, Vitaliy Lvin, David Miller, Matthew O'Connell, and Andrew Roberts for their contributions.

8. REFERENCES

- [1] S. S. Ali. A logical language for natural language processing. In *Proc. of the Tenth Biennial Canadian Artificial Intelligence Conf.*, Banff, Alberta, Canada, May 1994.
- [2] V. Ambriola and V. Gervasi. On the systematic analysis of natural language requirements with CIRCE. *Automated Software Eng.*, 13(1):107–167, Jan. 2006.
- [3] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.
- [4] J. C. Corbett and G. S. Avrunin. Using integer programming to verify general safety and liveness properties. *Formal Methods in System Design*, 6:97–123, 1995.
- [5] C. Denger, D. M. Berry, and E. Kamsties. Higher quality requirements specifications through natural language patterns. In *Proc. of the IEEE Int. Conf. on Software – Sci. Tech. and Eng.*, pages 80–91, 2003.
- [6] D. Drusinsky. Visual formal specification using (N)TLCharts: Statechart automata with temporal logic and natural language conditioned transitions. In *Int. Workshop on Parallel and Distributed Systems: Testing and Debugging*, Santa Fe, NM, Apr. 2004.
- [7] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Property specification patterns web site: <http://www.cis.ksu.edu/santos/spec-patterns/>.
- [8] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *Proc. of the 21st Int. Conf. on Software Eng.*, pages 411–420, Los Angeles, CA, May 1999.
- [9] M. B. Dwyer, L. A. Clarke, J. M. Cobleigh, and G. Naumovich. Flow analysis for verifying properties of concurrent software systems. *ACM Trans. on Software Eng. and Methodology*, 13(4):359–430, Oct. 2004.
- [10] N. E. Fuchs, U. Schwertel, and R. Schwitter. Attempto Controlled English - not just another logic specification language. In *Proc. of the Eighth Int. Workshop on Logic-based Program Synthesis and Transformation*, pages 1–20, 1998.
- [11] V. Gervasi and D. Zowghi. Reasoning about inconsistencies in natural language requirements. *ACM Trans. on Software Eng. and Methodology*, 14(3):277–330, July 2005.
- [12] C. L. Heitmeyer. Software Cost Reduction. In J. J. Marciniak, editor, *Encyc. of Software Eng.* Wiley-Interscience, Jan. 2002.
- [13] G. J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Eng.*, 23(5):279–294, 1997.
- [14] S. Konrad and B. H.C.Cheng. Real-time specification patterns. In *Proc. of the 27th Int. Conf. on Software Eng.*, pages 372–381, May 2005.
- [15] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic Publishers, 1993.
- [16] L. Mich, M. Franch, and P. Novi Inverardi. Market research for requirements analysis using linguistic tools. *Req. Eng. J.*, 9(1):40–56, 2004.
- [17] O. Mondragon and A. Gates. Supporting elicitation and specification of software properties through patterns and composite propositions. *Int. J. of Software Eng. and Knowledge Eng.*, 14(1):21–41, Feb. 2004.
- [18] O. Mondragon, A. Q. Gates, and O. Sokolsky. Generating properties for runtime monitoring from software specification patterns. Technical Report UTEP-CS-04-21, U. of Texas at El Paso, 2004.
- [19] M. H. Smith, G. J. Holzmann, and K. Etessami. Events and constraints: A graphical editor for capturing logic requirements of programs. In *Proc. of the Fifth IEEE Int. Symp. on Req. Eng.*, Aug. 2001.
- [20] R. L. Smith, L. A. Clarke, G. S. Avrunin, and L. J. Osterweil. Propel: An approach supporting property elucidation. In *Proc. of the 24th Int. Conf. on Software Eng.*, pages 11–21, Orlando, FL, May 2002.
- [21] A. van Lamsweerde. Formal specification: A roadmap. In A. Finkelstein, editor, *The Future of Software Engineering*, pages 147–159. ACM Press, 2000.