Analyzing Medical Processes*

Bin Chen
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
chenbin@cs.umass.edu

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

Elizabeth A. Henneman
School of Nursing
University of Massaschusetts
Amherst, MA 01003
henneman@nursing.umass.edu

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

Leon J. Osterweil
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
ljo@cs.umass.edu

Philip L. Henneman
Tufts-Baystate Medical Center
Springfield, MA 01199
philip.henneman@bhs.org

ABSTRACT

This paper shows how software engineering technologies used to define and analyze complex software systems can also be effective in detecting defects in human-intensive processes used to administer healthcare. The work described here builds upon earlier work demonstrating that healthcare processes can be defined precisely. This paper describes how finite-state verification can be used to help find defects in such processes as well as find errors in the process definitions and property specifications. The paper includes a detailed example, based upon a real-world process for transfusing blood, where the process defects that were found led to improvements in the process.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification; J.3 [**Life and Medical Sciences**]: Health

General Terms

Verification, Management, Reliability, Human Factors

Keywords

Finite-state verification, Model checking, Medical processes, Property specifications

1. INTRODUCTION

This paper describes how software engineering techniques that have been successfully applied in analyzing software systems can be effectively employed to detect problems in medical processes. This paper builds upon earlier work demonstrating that healthcare processes can be defined precisely using the Little-JIL process definition language [28]. Here, we describe our experiences in applying finite-state verification to precisely defined medical processes to identify process defects and then to confirm the effectiveness of corrections to those processes. Although described with respect to human-intensive, safety-critical medical processes, this work also suggests the applicability of these and related technologies to processes in other problem domains.

Medical errors are a major cause of death in our society. A 1999 report from the Institute of Medicine (IOM) [31] estimated that approximately 100,000 people die each year in US hospitals from preventable medical errors. There is ample anecdotal evidence that the complexity of the processes used to administer healthcare is a significant source of the problem. The healthcare literature is replete with documented evidence of such errors as administration of blood of the wrong type, misidentification of patients, and incorrect dosages of potentially lethal medications.

The medical community is aware of these problems and has approached them in a number of ways. One principal approach has been to devise mandated procedures for carrying out many healthcare activities, especially those identified as being particularly high-risk. Mandated procedures are generally described in considerable detail, sometimes in documents that are dozens of pages long. These documents consist largely of natural language text, often supplemented by diagrams. These documents are the basis for both the training of medical professionals and the actual processes performed in hospitals. Despite the care that went into the creation of such documents, as well as other safety measures, a subsequent IOM study [39] revealed that error rates in hospitals had not declined significantly in the five-year period following the initial IOM report.

Examination of documents used to describe medical processes suggest several reasons why such documents have proven to be inadequate. Documents describing such processes as blood transfusion (e.g., [50], [51]) provide good examples of the problem. Despite attempts to be complete, they contain terms that are poorly defined and inconsistently used, and important details are often missing, especially details for handling special cases that might arise. Recognizing such limitations, the medical community has

*Research partially supported by the National Science Foundation under awards CCF-0427071, CCR-0205575, and CCF-0541035, and by the U.S. Department of Defense/Army Research Office under awards DAAD19-01-1-0564 and DAAD19-03-1-0133.

ICSE'08, May 10–18, 2008, Leipzig, Germany.

tried to employ a number of modeling representations, but these are usually based upon such formalisms as data flow graphs that make it relatively cumbersome to represent the handling of exceptional cases or complicated concurrency and synchronization. As a result, these representations generally fail to represent the full complexity of these processes.

Indeed, the many diverse circumstances under which activities like blood transfusions must take place require processes of considerable complexity. Moreover, blood transfusion, like many other medical activities, requires coordinating the efforts of many different parties, often performing their activities in parallel. The complexity of a process, such as this one, increases the risk of defects. Software engineers will readily note that the software development community already deals with the creation of complex procedures (e.g., complex software systems) that present a range of difficulties analogous to those found in medical processes. This suggests that the approaches used in software engineering to build and analyze complex, distributed systems might be effective in defining and analyzing medical processes.

This paper describes our work on using finite-state verification to identify defects in actual medical processes. The example described in this paper is based on a blood transfusion process being studied by the nursing community and commonly used in hospitals. In our project, software engineers collaborated with healthcare professionals to define key parts of the processes and their desired properties in rigorous formalisms, and then applied finite-state verification to determine whether the process definition satisfied the properties. In doing so, process defects were detected and subsequently repaired. The verification also uncovered inaccuracies in our process definition and our property specifications. Since we use, or intend to use, these artifacts in a wide range of evaluation activities, detecting and correcting these problems is also vitally important.

In a broad sense, this work demonstrates the feasibility of medical process improvement, carried out in ways that are strongly analogous to software improvement approaches. It suggests the applicability of this approach to other domains as well as consideration of other software engineering tools.

In the next section of this paper, we provide a high-level description of the technologies that we employed, and Section 3 presents a detailed example. Section 4 discusses observations about this approach, and Section 5 outlines related work. The conclusion summarizes the contributions of this work and describes some areas of future research.

2. AN OVERVIEW OF THE TECHNIQUES AND METHODOLOGY USED

To evaluate the applicability of software engineering technologies to medical process definitions and analysis, we have used the Little-JIL process definition language [10], the PROPEL property elucidation system [17], and two finite-state verification systems, FLAVERS [22] and SPIN [29].

Modeling Processes Using Little-JIL: To analyze medical processes, it is important to develop precise, rigorous definitions of those medical processes first. The process definitions need to capture not only the standard cases, but also the exceptional situations. They also need to precisely specify the communication and coordination between medical professionals. In our approach, we use the Little-JIL language to define processes.

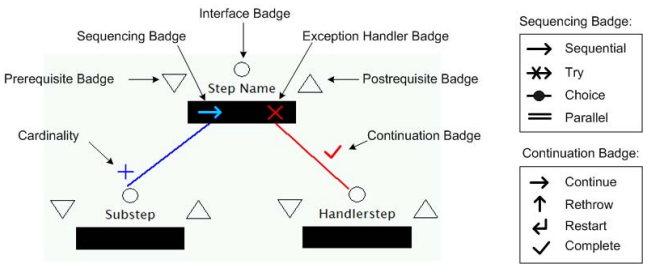

Figure 1: Little-JIL steps

Little-JIL was originally developed to define software development and maintenance processes. A Little-JIL process definition consists of three components, an *artifact specification*, a *resource specification,* and a *coordination specification*. The artifact specification contains the items that are the focus of the activities carried out by the process. The resource specification specifies the agents and capabilities that support performing the activities. The coordination specification ties these together by specifying which agents and supplementary capabilities perform which activities on which artifacts at which time(s). A Little-JIL coordination specification has a visual representation, but is precisely defined using finite-state automata, which makes it amenable to definitive analyses. Among the features of Little-JIL that distinguish it from most process languages are its 1) use of abstraction to support scalability and clarity, 2) use of scoping to make step parameterization clear, 3) facilities for specifying parallelism, 4) capabilities for dealing with exceptional conditions, and 5) clarity in specifying iteration.

A Little-JIL coordination specification consists of hierarchically decomposed steps (see Figure 1), where a step represents a task to be done by an assigned agent. Each step has a name and a set of badges to represent control flow among its sub-steps, its *interface* (a specification of its input/output artifacts and the resources it requires), the exceptions it handles, etc. A step with no sub-steps is called a *leaf step* and represents an activity to be performed by an agent, without any guidance from the process.

Little-JIL steps may be decomposed into two kinds of substeps, *ordinary substeps* and *exception handlers*. Ordinary substeps define how the step is to be executed and are connected to their parent by edges that may be annotated by specifications of the artifacts that flow between parent and substep and also by cardinality specifications. *Cardinality specifications* define the number of times the substep is instantiated, and may be a fixed number, a Kleene *, a Kleene +, or a Boolean expression (indicating whether the substep is to be instantiated). *Exception handlers* define how exceptions thrown by the step's descendants are handled.

A non-leaf step has a *sequencing badge* (an icon on the left of the step bar; e.g., the right arrow in Figure 1) that defines the order of substep execution. For example, a *sequential step* (right arrow) indicates that substeps execute from left to right. A *parallel step* (equal sign) indicates that substeps execute in any (possibly interleaved) order, although the order may be constrained by such factors as the lack of needed inputs. A *choice step* (circle slashed with a horizontal line) indicates a choice among alternative substeps. A *try step* (right arrow with an X on its tail) indicates the sequence in which substeps are to be tried as alternatives.

A Little-JIL step can be optionally preceded or succeeded by a *pre-requisite,* represented by a down arrowhead to the left of the step bar, or a *post-requisite,* represented by an up arrowhead to the right of the step bar. Pre-requisites check if the step execution context is appropriate before starting execution of the step, and post-requisites check if the completed step execution satisfied its goals. The failure of a requisite triggers an exception.

Channels are message passing buffers, directly connecting specified source step(s) with specified destination step(s). Channels are used to synchronize and pass artifacts among concurrently executing steps.

Specifying Properties Using PROPEL**:** A property is a specification of the requirements for some aspect of the behavior of a process. In the medical domain, properties are often specified as policies in natural language so that they can be easily understood by the medical professionals. Such informal descriptions, however, are often vague and ambiguous and need to be translated into rigorous mathematical formalisms such as automata or temporal logics that can be used as the basis for verification. This is a surprisingly difficult task. Even experienced developers may overlook subtle, but important, details. In our approach, we use PROPEL to support specifying formal properties.

PROPEL guides users through the process of creating properties that are both accessible and mathematically precise. PROPEL provides users with a set of property templates, each of which can be viewed as an extended Finite-State Automaton representation, a Disciplined Natural Language representation, or a Question Tree. Each representation contains options (or questions) that explicitly indicate the variations that must be considered, thereby ensuring that users do not overlook important subtle details. In addition, the Question Tree can be used to guide the user in selecting the appropriate template. All three representations are views of a single underlying representation so that a change in any representation is reflected automatically in the others.

Verifying Processes Using FLAVERS and SPIN: Finite-state verification (FSV) techniques, such as model checking [15], involve the construction of a finite model that represents all possible relevant executions of a system with respect to the property to be evaluated. Then algorithmic methods are employed to determine whether the particular property holds for the model. A number of FSV tools have been proposed; for this work, we have used FLAVERS and SPIN. We chose these tools because we were familiar with them (FLAVERS was developed in our laboratory), they represent distinct modeling and checking approaches, and we could build on existing technology to construct models for them.

To construct models of Little-JIL processes, we first translate the Little-JIL into the Bandera Intermediate Representation (BIR) [30]. BIR is a guarded-command language for describing state-transition systems and was intended to support translation into the input languages of a variety of model checkers. A translator from BIR to the Promela language used by SPIN was constructed by the Bandera team, and we have built a translator for FLAVERS. Medical processes entail substantial amounts of concurrency and exception handling. This leads to very large state spaces, making scaling an important issue. Therefore, we use several optimizations and abstractions to reduce the size of the generated model. Most of these are performed during the Little-JIL to BIR translation to take advantage of the scoping and hierarchy in Little-JIL. All the transformations are conservative for the property and process definition. This means that a process will not be reported to be consistent with a property unless that is indeed the case. Spurious violations, property violations in the model that do not correspond to any real trace through the process, could be introduced by these optimizations, but this problem arises rarely, and when it does can often be dealt with by using various model refinement techniques.

Methodology: To evaluate the effectiveness of this approach, we are engaged in three case studies, the blood transfusion case study described in more detail here, as well as a case study on emergency room patient flow and one on an outpatient chemotherapy process for treating breast cancer.

For each case study, a small team of computer scientists meets regularly with a group of medical professionals to elicit the process definition and the properties. The medical professionals are responsible for describing the processes and their requirements and for reviewing the material created by the computer scientists. The computer scientists are responsible for defining the process in Little-JIL, the properties in PROPEL, and for doing the analysis, as well as enhancing the tools as needed. The procedure that we follow is to first focus on the process definition. The benefits of capturing the blood transfusion process formally from the perspective of the nursing profession is described in [28].

While the process is being defined, it is not unusual for goals or high-level requirements to be mentioned, and these are recorded by the computer scientists. After the process definition has begun to stabilize, meetings are then held to elicit a more complete set of requirements. The requirements are first stated informally in natural language. The computer scientists then work with the medical professionals to agree on a glossary of terms that are used to more systematically describe the requirements, still in natural language. After agreement is reached on these statements, computer scientists work closely with the medical professionals to develop the detailed property specifications using PROPEL.

It is after this point that the computer scientists apply the FSV tools to evaluate whether the process definition is consistent with the stated formal properties. As with programs, it is often the case that the FSV tools find problems in the process definition and in the property specifications as well as in the process. Our long-range plans include using the process definitions to support fault tree analysis, simulations, and even process-guidance in the clinical setting. Thus, it is extremely important that the process definition accurately reflects the process and, of course, it is important that the process does not violate correctly formulated properties.

3. AN EXAMPLE

Blood transfusion, although a common procedure, involves considerable risk to the patient, and experts believe that adverse events involving transfusion are significantly underreported. Indeed, transfusion medicine professionals were among the first to develop classification schemes for medical errors, but most of the work in this area has focused on the handling of blood products in the laboratory rather than at the point of care where the actual transfusion occurs [28].

One of us (E. Henneman) is involved in the development of guidelines regarding the safe administration of blood and

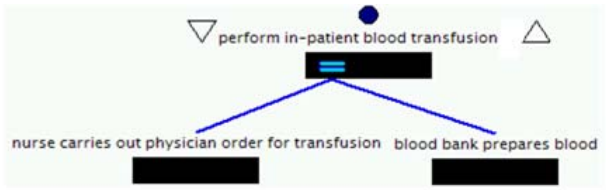

Figure 2: Root of the transfusion process

blood products. As part of this work, she identified a checklist [51] from a standard nursing reference [50] as a good example of a description of the standard transfusion process from the standpoint of the nurse administering the transfusion. This checklist has 40 items (some of which have a number of sub-items) ranging from "Administers pretransfusion medication as prescribed" and "Obtains IV fluid containing normal saline solution and a blood transfusion administration set" to "Compares the patient name and hospital identification number on the patient's identification bracelet with the patient name and hospital identification number on the blood bank form attached to the blood product" and "Using aseptic technique, attaches the distal end of the adminstration set to the IV catheter." To support her evaluation of such descriptions of the clinical transfusion process, we modeled the process described by the checklist in Little-JIL, and composed it with a model of the process that the hospital blood bank performs. The process model focuses particularly on representing the blood bank's interactions with the nurse, and abstracts away most of the details of the complex activity that the blood bank performs to prepare blood products for administration to patients.

In our Little-JIL model, shown in Figure 2, the root of the transfusion process is a parallel step (note the equal sign in the lefthand side of the step bar), "perform in-patient blood transfusion," whose children are the steps "nurse carries out physician order for transfusion" and "blood bank prepares blood." This root step begins after a physician orders a transfusion, and each of its substeps is further elaborated in separate Little-JIL diagrams, shown in Figures 3 and 4.

The substeps of the root step in Figure 3, which are numbered to show the correspondence with the items in the checklist, are carried out in order from left to right (note the right arrow in the lefthand side of the step bar). Although its interface specification is elided from the figure to reduce clutter, the step "verify informed consent has been obtained" may throw the exception "NoPatientConsent" if the nurse cannot verify that consent has been obtained.

The root step has an exception handler whose purpose is to obtain this consent. Similarly, "verify physician's order" may throw an exception if the order is incomplete. The step "notify blood bank to prepare blood" puts a message with the order into the "order blood" channel (here, too, the actual specification of this use of the channel has been elided; a yellow documentation annotation serves as a visual reminder), from which it is to be retrieved by a step performed by the blood bank. The steps "obtain infusion materials," "obtain blood product from blood bank," and "perform transfusion" are themselves elaborated in additional diagrams that are not included here. In the "obtain blood product from blood bank" diagram the nurse repeatedly checks the "blood bank status" channel for a "blood product ready" message, modeling the behavior of the nurse who calls the blood bank repeatedly to see if the blood product is ready yet, and then picks up the blood product once it is ready.

As shown in Figure 4, the blood bank's process begins with execution of the "receives notification from nurse" step, whose execution begins by taking an order from the "order blood" channel and then continues by putting a "blood product not ready" message into the "blood bank status channel." The blood bank process continues by executing the "obtain blood type and screen" step, which is performed by checking the lab for a current type and screen for this patient, normally obtained from a blood specimen drawn earlier in the hospitalization. If current type and screen are not available, the process indicates that an exception is thrown. In that case, the blood bank puts a "blood type and screen unknown" message into the "blood bank status" channel and waits for the specimen. Once the blood bank has prepared the blood (represented by the "prepares blood" step), the process specifies that the blood bank replace the "blood product not ready" message in the "blood bank status" channel with a "blood product ready" message. The process concludes with execution of the "gives blood to nurse" step.

Although we have elicited and formalized more than 60 properties that should be satisfied by a safe blood transfusion process, we focus here only on the property that, once the nurse notifies the blood bank to prepare the blood product, the nurse will eventually pick up the blood product. When we attempt to verify that our process satisfies this property, the verification tool reports a violation and produces as a counterexample a path through the process in which the patient's type and screen are not available and the blood bank is then unable to prepare the blood. Indeed, our model of the nursing process reflects the assumption in the checklist that the type and screen have been obtained prior to the transfusion order, an assumption that usually, but not always, holds. This analysis has thus identified a problem with the process specified by the checklist, namely that it has failed to deal with this exceptional situation. Our experience has indicated that such problems are often found in existing natural language process descriptions. It is particularly interesting that, as we noted earlier, this checklist is unusually detailed and complete, and yet it does not carefully specify the required behavior of the nurse in a number of exceptional cases, such as this one.

Upon discovering this process defect, medical professionals suggested improving the process by inserting a step mandating that the nurse respond to the "blood type and screen unknown" message by drawing a blood specimen for determining type and screen. Verification of the modified process then showed that blood would always be prepared, though the sample for type and screen might not be drawn until after all of the other preparations for the transfusion have taken place and the nurse has called the blood bank (represented in the process by reading from the "blood bank status" channel) to see if the blood is ready for pickup. This can involve significant delay and risk to the patient, and such errors are not uncommon in clinical practice. To address this process defect, we next modified the nursing process to have the nurse check for the availability of the type and screen *before* notifying the blood bank of the transfusion order and, if necessary, drawing the specimen for type and screen at that time. By making this process modification, the possibility of this delay is eliminated.

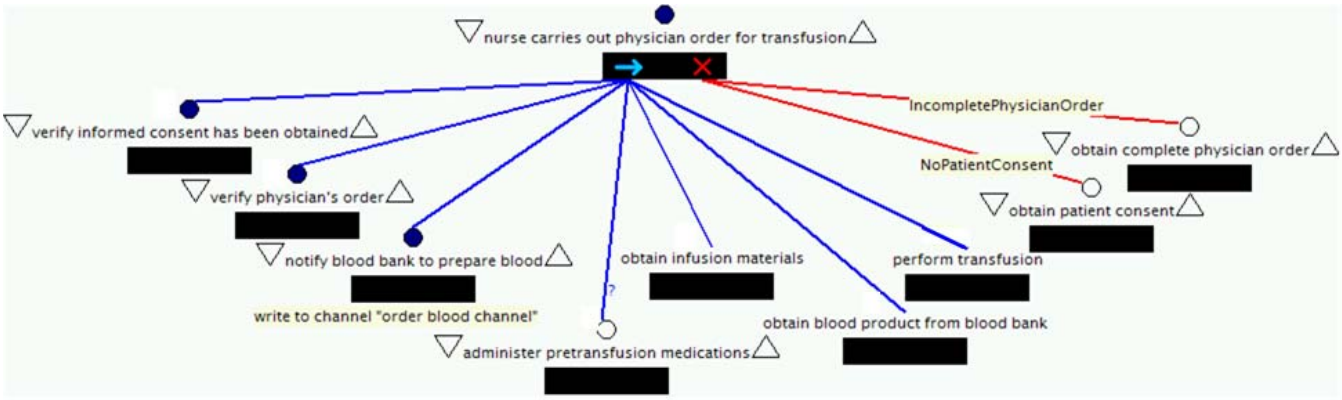

Figure 3: Elaboration of "nurse carries out physician order for transfusion"

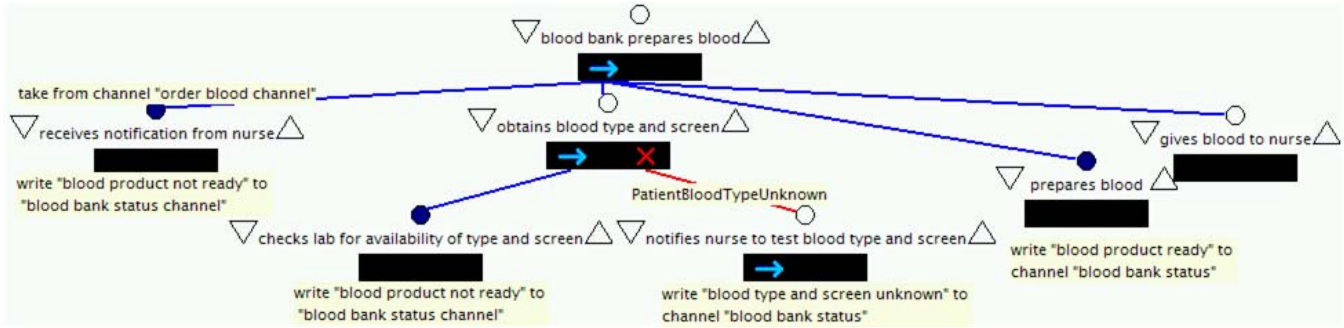

Figure 4: Elaboration of "blood bank prepares blood"

For those hospitals using electronic order entry, another concern is whether the nurse will see a transfusion order promptly. In about 10% of US hospitals, orders for transfusions are entered at a computer terminal by the physician and sent electronically to the blood bank and nurse. In such situations the nurse will see the transfusion order only when viewing a particular "task list" page in the patient's electronic record. To evaluate whether the nurse will respond to such an order promptly, we adapted the corrected version of the nursing and blood bank processes and introduced an abstract model of the physician's activities. In this model, the processes performed by the physician, the nurse, and the blood bank are defined to execute in parallel with each other and the nurse's process is defined to repeatedly make a choice between performing other nursing tasks or checking for a transfusion order. The resulting process specifies that if the nurse checks for a transfusion order and finds that one has been issued by the physician, the nurse then follows a process that is similar to the one previously described, but modified to indicate that the blood bank has received the order electronically from the physician rather than from the nurse. Figure 5 shows the top-level diagram of this process.

We then tried to verify the property that, if a physician orders a transfusion and certain exceptions (such as the patient refusing consent) do not occur, then the nurse will see the order. The verifier reported that the property does not hold in the case where the nurse never checks for the transfusion order. We understand that, in actual practice, the press of other urgent tasks may indeed cause a nurse to consult the task list only at the start and end of a shift, when the nurse going off duty reviews cases with the nurse coming on. Unlike other pages in the electronic patient record, such as those to order and record the administration of medication, the task list page is not frequently referenced during typical nursing procedures. A few large research hospitals or groups of hospitals with the resources to write their own systems have modified those systems so that the nurse sees an alert indicating that the transfusion has been ordered whenever the patient's record is accessed, and the alert remains visible until it is acknowledged. Hospitals that use commercial electronic order systems are often unable to get such changes made. (Some hospitals have resorted to entering transfusion orders on the medication page of the record since that page is consulted frequently by the nurse, even though this causes other difficulties with the electronic order system.)

We modified the process to reflect such an alert by explicitly distinguishing nursing activities that use the computer from those that do not and adding a pre-requisite to the steps representing activities involving the computer. That pre-requisite is a check for the existence of a transfusion order, representing an alert that informs the nurse of the existence of the order. We then tried to verify the property under the assumption that at least one activity involving use of the computer occurs after the physician orders the transfusion. With this assumption, which, as noted above, reflects the fact that many nursing activities in hospitals with electronic order systems do involve interaction with the computer system, the property holds. The important point is that the original problem with the process could be detected using FSV and the effectiveness of the modification

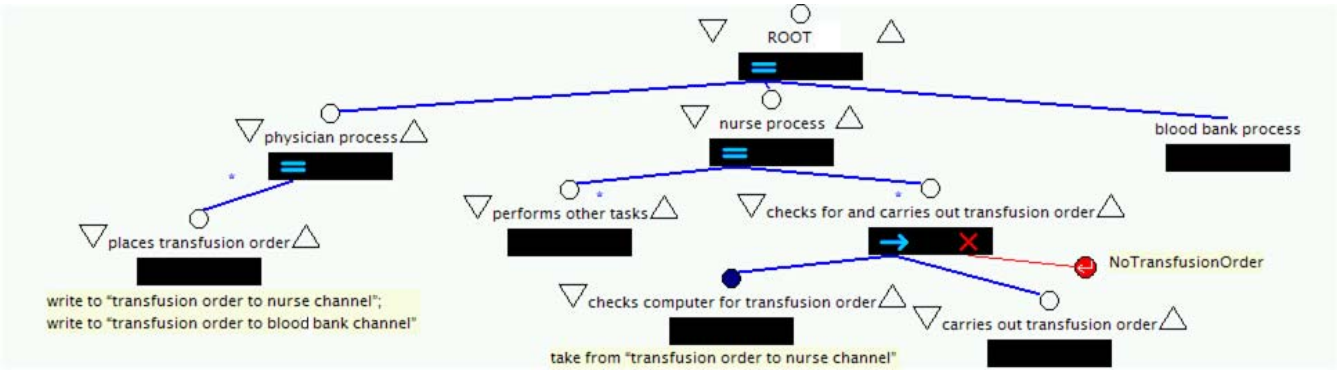

Figure 5: Process model for hospitals with electronic transfusion orders

could be subsequently evaluated, including the assumptions that were needed to make it valid.

Verification of each of the properties we have examined for the blood transfusion process takes about 10 seconds or less on a 1.86 GHz Pentium M processor and uses a few megabytes of memory.

4. DISCUSSION

This project has helped the medical professionals understand and improve their processes in a variety of ways. In this section we discuss the impact of FSV on process understanding and improvement and some of the issues that have arisen in applying FSV techniques to medical processes.

Impact of Verification on Process Definition: Specifying properties for verification and attempting to verify them identified a number of problems in our process definitions. In some cases, simply trying to formulate the properties precisely pointed out gaps in the formal process definitions, for instance because there simply were no steps that could be bound to the events used to specify those properties. In some cases, this was because the part of the process intended to address the issue being captured by a property was simply missing and had to be added. In other cases, the problem was that certain steps needed to be further decomposed in order to identify the substeps that should be bound to the property events.

Verification efforts also helped us to find several subtle errors in the formal process definitions that had remained undiscovered despite careful inspection by both software engineers and domain experts. For example, the blood transfusion process definition specified that if discrepancies occur during the "verify blood product" step, then "verify blood product" is to be terminated and a "Failed Blood Product Check" exception is to be thrown. The handler "handle failed blood product check", which refers to a step defined in other diagrams (not shown here due to lack of space), requires the nurse to send the blood product back to the blood bank and obtain a replacement blood product. The exception continuation badge of this handler was specified to be "continue", which implied that, after this exception has been handled in this way, the process continues with the nurse signing the blood bank form after obtaining the replacement product. This process thus violated the property "The nurse must verify the blood product before it is transfused to the patient", where the event "verify the blood product" is bound to completion of the step "verify blood product". It is clear that this would introduce the potential for serious medical error, and in the actual process the nurse verifies the patient identification and the product identification again after obtaining the replacement product. Thus, attempting to verify this property indicated that the exception's continuation badge was wrong and needed to be changed to "restart".

The accuracy of the formal process definitions is, of course, critical to the utility of the analysis—verifying properties of a model that does not reflect the real process provides no information about the real process and may in fact lead to dangerous overconfidence in the safety of the process. Furthermore, our process models are also intended to be used for other types of analysis, simulation, and possibly even guidance in the clinical setting, so detecting and correcting errors in the models is important for other reasons as well.

Impact of Verification on Processes: The analysis of the processes described in this paper identified problematic defects in the processes, helped determine the causes of the problems, and subsequently provided some assurance that the modified processes were indeed improvements. This was the case with other processes from our case studies as well.

Perhaps the greatest direct benefit of verification, however, is the assurance it provides that the revised process satisfies the previously violated property as well as other previously verified properties. In practice, modifications to the processes are usually made incrementally, with changes introduced to address some perceived problem. Sometimes this problem is a "sentinel event," an occurrence in the execution of the existing process where a patient has been put at risk, and thus changes are introduced to prevent the recurrence of that event. In other cases, changes are introduced to increase efficiency or make the process more convenient for the medical professionals or more comfortable for patients. But the medical professionals have very few methods for assessing the impact of such changes, and it is hard for them to determine whether the changes really do address the intended problem and do not introduce new problems. For example, our analysis of a chemotherapy process detected a deadlock that was introduced by a change that was intended to prevent ill patients from having to make an extra trip to the chemotherapy site. When this deadlock arose in practice, the medical professionals involved in the process broke the deadlock by ad hoc means. It took some additional process modifications and further analysis to be sure that this would not lead to any reductions in the number of safety checks in the process. For life-critical processes, the ability

to evaluate proposed changes in the process without having to first put them in place is very significant.

Issues in Applying FSV to Medical Processes: There are a number of obstacles to applying FSV techniques to complex, human-intensive processes. First, it is hard to get the formal definitions of the processes right. The amount of effort invested by both the medical professionals and the computer scientists involved in our project is considerable. The computer scientists have to learn enough of the medical terminology and context to understand what the medical professionals are saying and the medical professionals have to think very hard about the details of their processes and the possible exceptions. This is itself a complex, human-intensive activity. As we have described above, aspects of the modeling and verification process help detect errors in the formal definitions, but getting the right formal definitions at a suitable level of detail for verification of the properties related to medical safety is not easy.

A second obstacle is that specifying properties for verification is also hard. Domain experts often had problems being precise about the high-level properties that they wanted their processes to obey. In addition, these high-level statements were often incorrect. This frequently occurred because the domain experts did not consider possible exceptional situations that could impact the property. For example, the property "The nurse must verify patient's identification bracelet matches patient's stated name and birth date before infusing blood product" requires that the event "verify patient's identification bracelet matches patient's stated name and birth date" must occur on every execution of the process. In attempting to verify this property, however, the verifier reported a violation, identifying the possibility that if the patient refuses to sign the consent form, the process cannot proceed, and hence the event "verify patient's identification bracelet matches patient's stated name and birth date" will not occur. The actual property should have taken this constraint into consideration and instead have stated that "After the patient signs the consent form, verify patient's identification bracelet matches patient's stated name and birth date."

Once the abstract statements of the desired properties have been chosen, the events in those statements must be bound to specific events in the formal process definitions for verification. It can be quite difficult to determine which step or steps should be bound to which events. We encountered problems with this because the sources for the property specifications (i.e., medical guidelines) were sometimes different from the sources for the process definition or because the two descriptions were at very different levels of abstraction; sometimes both problems arose. The property specifications frequently had to be broken down from a high, abstract level (e.g., "the right drug to the right patient at the right time") to lower-level specifications that could be mapped to the process step names (e.g., the patient's name and date of birth as given by the patient match the name and date of birth in the chart). Sometimes high-level abstract specifications were mapped to several low-level properties, stated in the terminology of the step names, and sometimes property events needed to be represented by more than one process step name. Similar problems occur for software systems when high-level system requirements need to be mapped to lower level properties stated with respect to the details in the system design or implementation.

Because errors in the formal process definitions and property specifications were often not detected until the first rounds of verification, the verification process could be very lengthy. This, of course, is also the case with verification of software systems—much of the early effort of verification is devoted to finding problems in the model and the properties. For human-intensive processes, like many medical processes, however, this may be even more significant since the initial artifacts from which process definitions and properties are derived are less concrete and precise than, say, source code.

Finally, one expects problems of scale in FSV. For the sorts of processes that we have analyzed so far, these have not been serious. In particular, some of our optimizations are able to take advantage of aspects of the structure of Little-JIL process definitions to reduce the size of the models. But our examples have largely been restricted to very small configurations, such as one nurse performing one transfusion on one patient. As we extend this work to consider processes involving many medical professionals carrying out many activities to treat many patients, we expect that the time and memory resources required for verification may begin to limit the applicability of our methods. As for verification of software systems, we will look for new abstractions to reduce the size of the models and new domain-specific verification techniques that take advantage of special features of these processes.

5. RELATED WORK

Process Definition: Many languages and diagrammatic notations have been evaluated as vehicles for defining processes. For example, APPL/A [44] used a procedural language, MARVEL/Oz [8] used a rule-based language and SLANG [5] used modified Petri Nets to define processes. More recently, the workflow [36] and electronic commerce [27] communities have pursued similar research. In the medical domain, several languages, such as Asbru [41], GLARE [33], and PROforma [43], have been especially designed for representing clinical protocols and guidelines using an AI-based linguistic paradigm. Noumeir has also pursued similar goals, but using a notation like UML to define processes [34]. Others (e.g., [40]), view medical processes as workflows and use a workflow-like language to define processes and drive their execution. None of these process definition approaches, however, seems able to support process definitions that are both sufficiently clear and sufficiently broad and precise to support analysis of the sort described in this paper. The main problems with these approaches include inadequate specification of exception handling, weak facilities for controlling concurrency, lack of resource management, and inadequate specification of artifact flows. We believe that the Little-JIL language addresses these problems relatively more successfully, although it still has inadequacies, such as the lack of good support for specifying timing constraints and transactions.

Property Specification: There are many property specification approaches that aim to provide both accessibility and precision. The Attempto Controlled English (ACE) project [24] uses a natural language processing technique to translate natural language (NL) property specifications into first-order logic. It also provides annotated NL templates for non-expert users. Ambriola and Gervasi [1] have developed the CARL and CICO/CIRCE tools to translate NL property specifications into propositional logic and back again. One

limitation of these approaches is that the translator and the user may have different interpretations of NL specifications due to the ambiguity in those NL specifications. Interpretation alternatives like the options in PROPEL might help in improving the accuracy of the resulting formal property representations. Unlike these approaches, PROPEL does not attempt to understand NL, even in restricted domains. Some other approaches, including the Dwyer et al. property patterns work [21], and Drusinsky's (N)TLCharts [20], simply annotate the formal model with NL comments. Unlike the DNL representation in PROPEL, the NL comments 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.

FSV: There has been a great deal of work on the analysis of software artifacts. Most of this work has been focused on analysis of code or models of systems. Finite-state verification, or model checking, techniques (e.g., [15], [13], [29]), work by constructing a finite model that represents all possible relevant executions of the system and then analyze that model algorithmically to detect executions that violate the particular property. Our team has been involved in the analysis and evaluation of various finite-state verification approaches [3], and the development of verifiers such as FLAVERS [22] and INCA [18]. Our work seems to be among the first that has applied FSV approaches to process definitions [16], [12], [32]. As noted above, one of the major concerns of these techniques is controlling the size of the state space, while maintaining precision in the analysis result. Many abstraction and reduction techniques (e.g., [19], [14], [26]) have been used to tackle this problem. The optimization and abstraction approaches we have taken in this work are not new. They are however, effective, since they can take advantage of Little-JIL's scoping and hierarchy to achieve important reductions.

Improving Medical Processes: To our knowledge, there has been very little work on using formal methods to improve medical procedures. In [45], a medical protocol is modeled in the Asbru [41] language. To analyze the protocol, the model is automatically translated to a formal representation for the interactive theorem prover KIV [4]. This approach was applied to two real-life medical protocols, a jaundice protocol and a diabetes mellitus protocol. In [7], the Asbru model is translated into the input language of the SMV model checker and a simple abstraction is used to reduce the model. This work also used the jaundice protocol as an example and found errors in it. The blood transfusion process that we analyzed seems to be notably more complex than the protocols analyzed in these two papers. The Little-JIL model of the blood transfusion process consists of about 120 steps (some of which are essentially invocations of previously defined steps) while the Asbru model of the jaundice protocol has only about 40 plans (a plan is the counterpart of a step in Little-JIL). And unlike the Asbru model, which only defines the normal procedure, our blood transfusion process models specify real-world exception handling that could greatly change the control-flow of the normal process. Our blood transfusion processes also define the interactions among various medical professionals, making the models even more complex. In [46], the clinical guideline defined in GLARE [33] is translated to a Promela [29] representation and verified by SPIN. However, no details of the evaluation are presented in the paper.

There have been other approaches to improving medical safety, as well, but much of the emphasis of this work has been targeted towards quality control measures [49], [23], error reporting systems [6], and process automation in laboratory settings [25], such as those where blood products are prepared. In other work, Bayesian belief networks have been used as the basis for discrete event simulations of medical scenarios and to guide treatment planning (e.g., [47]).

6. CONCLUSIONS AND FUTURE WORK

This project has demonstrated some of the benefits of applying selected software technologies to improve healthcare processes. We found, for instance, that the very act of using technologies that support process and property elicitation was effective in identifying defects in medical processes. For example, Little-JIL's facilities for exception management invite process eliciters to inquire about exceptional behavior as a routine part of their interviewing of domain experts. In doing so exception management issues are brought forward. Similarly PROPEL's interactive questioning about the details of properties has also proven to be effective in causing domain experts to confront property details that otherwise are typically overlooked.

It is true that addressing all of these details carefully, as invited by these technologies, causes the process and property elicitation processes to be lengthy. Typically the process and property elicitations and reviews took place through weekly meetings over a period of several months. While this cost is admittedly high, we believe that it is more than repaid by the quality of the processes and properties obtained and by the improvements achieved.

Indeed, we believe that these costs will be further amortized as we use these process definitions and property specifications as the basis for further types of analysis. In preliminary work we outlined how our process definitions can be used to automatically generate Fault Trees [11] that can then be used as the basis for Fault Tree Analysis (FTA) and Failure Mode and Effects Analysis (FMEA) [42, 48]. Such analyses seem likely to be effective in supporting such diagnoses as the presence of single points of failure and how faulty performance of a process step may impact subsequent process executions. We have also begun working on the automatic generation of simulations from Little-JIL process definitions. As these process analysis efforts proceed, we hope to discover that they are mutually supportive, and that the combined value of such analyses richly repays the costs of elicitation of processes and properties.

The processes studied in the project have all been human intensive. As noted, the underlying technologies were first developed for software systems and have been considered for combined hardware/software systems. We envision expanding the scope of this project to include medical devices and the human processes involved in employing those devices. We believe that it is important to not only verify the device but to evaluate it in the context in which it will be employed. We have shown in preliminary work [2] that the properties can be quite different in different contexts.

Ultimately we envisage the development of a process environment in which process definition tools are smoothly integrated with a spectrum of process analysis capabilities. Such a support environment would hopefully lead to a systematic and well-reasoned approach to process improvement. Our primary focus is on processes in the healthcare com-

munity, but in other work we are also exploring processes in use in such other diverse domains as labor-management dispute resolution [35], ecological data processing [9], and elections [37].

Finally we would like to emphasize that the benefits of this work are not restricted to effecting improvements only in the application domains. Our work has also resulted in improvements to our process definition language and in our requirements engineering and analysis capabilities. Little-JIL's semantic capabilities, for example, have been broadened and sharpened in response to needs that became manifest as we defined processes in the healthcare domain. Our understanding of the difficulty of defining resources and the ways in which processes specify needs for them was also sharpened considerably by our work on healthcare processes. This is leading to challenging new directions in resource specification and management [38]. Other needs are continually being recognized, leading to a range of research challenges, most of which have direct relevance to software engineering.

We regard the project described in this paper as only an early indication of the many possible ways in which software engineering technologies can be applied to new domains. Doing so offers the strong prospect of benefit to those domains and also to the further development of the software technologies themselves.

7. ACKNOWLEDGEMENTS

The authors gratefully acknowledge the work of Sandy Wise, Barbara Lerner, and Aaron Cass, who made major contributions to the development of Little-JIL, to Heather Conboy and Jamieson Cobleigh, who made major contributions to the development of FLAVERS, to Rachel Cobleigh, who developed PROPEL, and to Houng Phan, who helped to elicit the blood transfusion process and properties.

8. REFERENCES

[1] V. Ambriola and V. Gervasi. On the systematic analysis of natural language requirements with circe. *Automated Software Eng.*, 13(1):107–167, 2006.

[2] G. S. Avrunin, L. A. Clarke, E. A. Henneman, and L. J. Osterweil. Complex medical processes as context for embedded systems. *ACM SIGBED Rev.*, 3(4):9–14, 2006.

[3] G. S. Avrunin, J. C. Corbett, and M. B. Dwyer. Benchmarking finite-state verifiers. *Software Tools for Technology Transfer*, 2(4):317–320, 2000.

[4] M. Balser, W. Reif, G. Schellhorn, K. Stenzel, and A. Thums. Formal system development with kiv. In T. Maibaum, editor, *Fundamental Approaches to Software Engineering*, volume 1783 of *LNCS*, pages 363–366, 2000.

[5] S. C. Bandinelli, A. Fugetta, and C. Ghezzi. Software process model evolution in the SPADE environment. *IEEE Trans. on Softw. Eng.*, 19(12), December 1993.

[6] J. Battles, H. Kaplan, T. van der Schaaf, and C. Shea. The attributes of medical event-reporting systems: experience with a prototype medical event-reporting system for transfusion medicine. *Arch. Pathology Laboratory Medicine*, 122:231–238, 1998.

[7] S. Bäumler, M. Balser, A. Dunets, W. Reif, and J. Schmitt. Verification of medical guidelines by model checking - a case study. In A. Valmari, editor, *SPIN*, volume 3925 of *LNCS*, pages 219–233, 2006.

[8] I. Z. Ben-Shaul and G. E. Kaiser. A paradigm for decentralized process modeling and its realization in the oz environment. In *16th international conference on Software Engineering*, pages 179–188, 1994.

[9] E. R. Boose, A. M. Ellison, L. J. Osterweil, L. Clarke, R. Podorozhny, J. L. Hadley, A. Wise, and D. R. Foster. Ensuring reliable datasets for environmental models and forecasts. In *Ecological Informatics*, volume 2, pages 237–247, 2007.

[10] A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, J. Stanley M. Sutton, and A. Wise. Little-jil/juliette: A process definition language and interpreter. In *22nd Int. Conf. on Softw. Eng.*, pages 754–757, Limerick, Ireland, 2000.

[11] B. Chen, G. S. Avrunin, L. A. Clarke, and L. J. Osterweil. Automatic fault tree derivation from little-jil process definitions. In *SPW/ProSim*, volume 3966 of *LNCS*, pages 150–158, Shanghai, May 2006.

[12] S. Christov, B. Chen, G. S. Avrunin, L. A. Clarke, and L. J. Osterweil. Rigorously defining and analyzing medical processes: An experience report. In *1st International Workshop on Model-Based Trustworthy Health Information Systems*, September 2007.

[13] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV version 2: An opensource tool for symbolic model checking. In *Proc. Int. Conf. on Computer-Aided Verification*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.

[14] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, September 1994.

[15] E. M. Clarke, O. G. Jr., and D. A. Peled. *Model Checking*. MIT Press, 2000.

[16] J. M. Cobleigh, L. A. Clarke, and L. J. Osterweil. Verifying properties of process definitions. In *ACM SIGSOFT Int. Symp. on Software Testing and Analysis*, pages 96–101, Portland, OR, August 2000.

[17] R. L. Cobleigh, G. S. Avrunin, and L. A. Clarke. User guidance for creating precise and accessible property specifications. In *14th ACM SIGSOFT Int. Symp. on Foundations of Software Eng.*, pages 208–218, Portland, OR, November 2006.

[18] J. C. Corbett and G. S. Avrunin. Using integer programming to verify general safety and liveness properties. *Formal Methods System Design*, 6(1):97–123, 1995.

[19] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, 1977.

[20] D. Drusinsky. Visual formal specification using (n)tlcharts: Statechart automata with temporal logic and natural language conditioned transitions. In *International Workshop on Parallel and Distributed Systems: Testing and Debugging*, April 2004.

[21] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state

verification. In *21st Int. Conf. on Softw. Eng.*, pages 411–420, Los Angeles, May 1999.

[22] M. B. Dwyer, L. A. Clarke, J. M. Cobleigh, and G. Naumovich. Flow analysis for verifying properties of concurrent software systems. *ACM Trans. Softw. Eng. Methodol.*, 13(4):359–430, October 2004.

[23] M. Foss and S. Moore. Evolution of quality management: integration of quality assurance functions into operations, or "quality is everyone's responsibility". *Transfusion*, 43(9):1330–1336, September 2003.

[24] N. E. Fuchs, U. Schwertel, and R. Schwitter. Attempto controlled english - not just another logic specification language. In P. Flener, editor, *8th International Workshop on Logic Programming Synthesis and Transformation*, number 1559 in LNCS, pages 1–20, 1998.

[25] S. Galel and C. Richards. Practical approaches to improve laboratory performance and transfusion safety. *American Journal of Clinical Pathology*, 107(4):S43–S49, 1997.

[26] S. Graf and H. Saïdi. Construction of abstract state graphs with pvs. In *9th Int. Conf. on Computer Aided Verification*, number 1254 in LNCS, pages 72–83, 1997.

[27] B. N. Grosof, Y. Labrou, and H. Y. Chan. A declarative approach to business rules in contracts: courteous logic programs in XML. In *ACM Conf. on Electronic Commerce*, pages 68–77, Denver, CO, 1999.

[28] E. A. Henneman, G. S. Avrunin, L. A. Clarke, L. J. Osterweil, C. Andrzejewski, Jr., K. Merrigan, R. Cobleigh, K. Frederick, E. Katz-Bassett, and P. L. Henneman. Increasing patient safety and efficiency in transfusion therapy using formal process definitions. In *Transfusion Medicine Review*, volume 21, pages 49–57, January 2007.

[29] G. J. Holzmann. *The SPIN Model Checker*. Addison-Wesley, 2004.

[30] R. Iosif, M. B. Dwyer, and J. Hatcliff. Translating java for multiple model checkers: The bandera back-end. *Formal Methods in System Design*, 26(2):137–180, March 2005.

[31] L. T. Kohn, J. M. Corrigan, and M. S. Donaldson, editors. *To Err Is Human: Building a Safer Health System*. National Academy Press, Washington, D.C., 1999.

[32] B. S. Lerner. Verifying process models built using parameterized state machines. In *ACM SIGSOFT Int. Symp. on Software Testing and Analysis*, pages 274–284, New York, USA, 2004.

[33] G. Molino, P. Terenziani, S. Montani, A.Bottrighi, and M. Torchio. Glare: a domain-independent system for acquiring, representing and executing clinical guidelines. In *J. of the Amer. Medical Informatics Association (JAMIA) Symposium supplement*, 2006.

[34] R. Noumeir. Radiology interpretation process modeling. *J. of Biomedical Informatics*, 39(2):103–114, 2006.

[35] L. J. Osterweil, N. K. Sondheimer, L. A. Clarke, E. Katsh, and D. Rainey. Using process definitions to facilitate the specification of requirements. Technical report, Department of Computer Science, University of Massachusetts Amherst, 2006.

[36] S. Paul, E. Park, and J. Chaar. Rainman: a workflow system for the internet. In *USENIX Symp. on Internet Technologies and Systems*, Berkeley, CA, 1997.

[37] M. S. Raunak, B. Chen, A. Elssamadisy, L. A. Clarke, and L. J. Osterweil. Definition and analysis of election processes. In *SPW/ProSim 2006*, volume 3966 of *LNCS*, pages 178–185, Shanghai, May 2006.

[38] M. S. Raunak and L. J. Osterweil. Effective resource allocation for process simulation: A position paper. In *6th International Workshop on Software Process Simulation and Modeling*, St. Louis, MO, May 2005.

[39] P. P. Reid, W. D. Compton, J. H. Grossman, and G. Fanjiang, editors. *Building a Better Delivery System: A New Engineering/Health Care Partnership*. National Academy Press, Washington, D.C., 2005.

[40] M. Ruffolo, C. Information, and R. Curia. Process management in health care: A system for preventing risks and medical errors. In *Business Process Management*, pages 334–343, 2005.

[41] Y. Shahar, S. Miksch, and P. Johnson. The asgaard project: a task-specific framework for the application and critiquing of time-oriented clinical guidelines. *Artificial Intelligence in Medicine*, 14(1-2):29–51, 1998.

[42] D. H. Stamatis. *Failure Mode and Effect Analysis: FMEA from Theory to Execution*. Amer Society for Quality, March 1995.

[43] D. R. Sutton and J. Fox. The syntax and semantics of the proforma guideline modeling language. In *Journal of the American Medical Informatics Association*, volume 10, pages 433–443, Sep-Oct 2003.

[44] J. S. M. Sutton, D. Heimbigner, and L. J. Osterweil. Appl/a: a language for software process programming. *ACM Trans. on Software Engineering and Methodology*, 4(3):221–286, 1995.

[45] A. ten Teije, M. Marcos, M. Balser, J. van Croonenborg, C. Duelli, F. van Harmelen, P. Lucas, S. Miksch, W. Reif, K. Rosenbrand, and A. Seyfang. Improving medical protocols by formal methods. *Artificial Intell. in Medicine*, 36(3):193–209, 2006.

[46] P. Terenziani, L. Giordano, A. Bottrighi, S. Montani, and L. Donzella. Spin model checking for the verification of clinical guidelines. In *ECAI 2006 Workshop on AI techniques in healthcare: evidence-based guidelines and protocols*, August 2006.

[47] L. van der Gaag, S. Renooji, C. Witteman, B. Aleman, and B. Taal. Probabilities for a probabilistic network: a case study in oesophageal cancer. *Artificial Intelligence in Medicine*, 25(2):123–148, June 2002.

[48] W. Vesely, F. Goldberg, N. Roberts, and D. Haasl. *Fault Tree Handbook (NUREG-0492)*. U.S. Nuclear Regulatory Commission, Washington, D.C., Jan. 1981.

[49] D. Voak, J. Chapman, and P. Phillips. Quality of transfusion practice beyond the blood transfusion laboratory is essential to prevent abo-incompatible death. *Transfusion Medicine*, 10(2):95–96, June 2000.

[50] J. M. Wilkinson and K. V. Leuven. *Fundamentals of Nursing*. F. A. Davis Company, June 2007.

[51] J. M. Wilkinson and K. V. Leuven. Procedure checklist for administering a blood transfusion. `http://davisplus.fadavis.com/wilkinson/PDFs/Procedure_Checklists/PC_Ch36-01.pdf`, 2007.