A Benchmark for Evaluating the Applicability of Software Engineering Techniques to the Improvement of Medical Processes

Stefan C. Christov
Dept. of Computer Science
University of Massachusetts
Amherst, MA 01003
christov@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

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

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

ABSTRACT

Problems in health care have gained prominence in recent years. To address such concerns, the software engineering and medical informatics communities have been developing a range of methodologies and tools for reasoning about medical processes. To facilitate the comparison of such methodologies and tools in terms of their applicability to health care, it would be desirable to have a set of medical examples, or *benchmarks*, that are easily available, described in considerable detail, and carefully characterized in terms of the real-world complexities they capture. This paper presents one such benchmark and discusses a list of desiderata that medical benchmarks can be evaluated against.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications

General Terms

Experimentation

Keywords

medical process, benchmark, software engineering methodology

1. INTRODUCTION

Problems in health care have gained prominence in recent years. A 2009 US National Research Council report [27] argues that "these persistent problems do not reflect incompetence on the part of health care professionals—rather, they are a consequence of the inherent intellectual complexity of health care taken as a whole and a medical care environment that has not been adequately structured to help clinicians avoid mistakes or to systematically improve their decision making and practice." The report also notes that current health care information technology is rarely used to support data-driven process improvement. To address such concerns, the software engineering and medical informatics communities have been developing a range of methodologies and tools for evaluating medical processes (e.g., [8, 11, 22]). The complex nature of medical processes—for example, the possible involvement of a diverse set of medical professionals treating a single patient, the abundance of exceptional situations, the concurrent execution of activities and real-time constraints—makes these processes hard to analyze and corresponding analysis methodologies hard to evaluate.

Because of the complexity of medical processes, it important to be able to automate, at least partially, the reasoning about them. Researchers have thus used modeling notations with formal semantics to make the process models amenable to automated analysis. In this paper, we refer to a process model created in such notations with rigorous, formal semantics as a *process definition*.

To empirically evaluate different process modeling notations and analysis techniques, researchers have applied them to different examples from the medical domain. For instance, the Little-JIL process definition language [6] and the FLAVERS finite-state verification analysis technique [15] have been applied to discover defects and suggest improvements in both blood transfusion and chemotherapy processes [8, 9]; the Asbru language [25] and the KIV theorem prover [5] have been used to reason about a jaundice protocol [28]; Message Sequence Charts [16] translated to guarded Labeled Transition Systems [19] have been combined with various kinds of static analyses to reason about a cancer therapy process [11].

Evaluating such process modeling notations and analysis techniques on medical examples is an important step towards a rigorous discipline of applying software engineering and medical informatics approaches to health care. An important question arises: "What are the relevant strengths

SEHC 2010 Cape Town, South Africa

and weaknesses of each modeling notation or analysis technique?" and consequently "For what kinds of medical processes should one use a certain process modeling notation or a certain analysis technique instead of another notation or analysis technique?" Questions like these are difficult to answer when each modeling notation or analysis technique has been evaluated in the context of a different medical example and when many of these medical examples are not easily accessible, are ambiguously described, or are not carefully characterized in terms of the real-world complexities they are meant to capture.

To facilitate the comparison of software engineering and medical informatics modeling notations and analysis techniques in terms of their applicability to health care, what seems to be needed is a set of medical examples, or *benchmarks*. Ideally, these medical benchmarks would be both easily accessible by the research community and rigorously described to reduce ambiguity and help ensure that different researchers are working on the same example. Preferably, these benchmarks should also be captured in notations that are relatively easy to understand and would be characterized in terms of the complexities of the real-world medical processes that they capture. Such characterizations would guide potential users in their choice of benchmarks. A set of medical benchmarks would not only facilitate the comparison of different process modeling notations and analysis techniques in terms of their applicability to health care, but it could also provide an infrastructure for controlled experimentation and reproducibility.

Benchmarks (e.g., [1, 12, 17]) have been successfully used in various other domains to facilitate the comparison of different systems and approaches. We believe that the use of benchmarks can also benefit the research community studying the applicability of software engineering and medical informatics process modeling notations and analysis techniques to health care. In addition to providing a basis for the comparison of different approaches and for controlled experimentation and replicability, a set of medical benchmarks could help lessen the burden on software engineering and informatics researchers of eliciting these processes from medical domain experts. Our experience indicates that eliciting an adequate description of a medical process can be very time-consuming and labor-intensive for both the domain experts and the computer scientists. Therefore, having a set of widely available medical benchmarks seems to be particularly valuable as it could significantly reduce costs associated with eliciting medical examples from scratch.

This paper presents one such medical benchmark, which is now being made available to the community. This benchmark is based on a blood transfusion process. The components of the benchmark we are making available now are: a blood transfusion process definition created in the Little-JIL language; a set of properties, or requirements, that a blood transfusion process must satisfy; and a set of bindings that establish the relationship between the blood transfusion properties and process definition.

The contributions of this paper are the blood transfusion medical benchmark, and a list of desiderata against which medical benchmarks can be evaluated. The next section of this paper discusses related work. The desiderata for medical benchmarks are presented in section 3. Section 4 describes the blood transfusion benchmark in detail, and section 5 concludes with an evaluation of the benchmark, its current status, and issues to be addressed in future work.

2. RELATED WORK

The medical community has created many descriptions of standard medical procedures and processes (e.g., [31,32]) for purposes such as promoting evidence-based practice or training novice health care professionals. These medical process descriptions are usually written using natural language as a narrative, as a checklist, or as a high-level flow chart. They aim to provide important procedural details but they have several drawbacks that make them difficult to use directly as benchmarks. These descriptions often contain poorly defined and inconsistently used terms, which could lead to different interpretations of the medical procedure, that in turn could complicate the comparison of the different technologies. Such medical process descriptions also tend to focus almost exclusively on the normative workflow and tend to omit important details about exceptional scenarios.

The medical informatics community has created several languages that can be used to represent medical processes (e.g., EON [20], GLIF [22], Asbru [25]). These languages tend to have constructs with well-defined semantics, and thus medical process descriptions specified in these languages tend to be more precise and less ambiguous than the natural language descriptions mentioned above. Some of these languages also have support for integrating domain ontologies into the process description, which can further reduce ambiguity. In this respect, medical process descriptions created in such languages are a step closer to being usable as benchmarks. Different languages, however, have different strengths and weaknesses in terms of the semantic constructs they provide to support the precise description of important aspects of medical processes, such as concurrency, exceptional flow, and real time constraints. The degree to which process descriptions capture such aspects is thus affected by the semantic richness of the language used to create these descriptions. To facilitate their use as benchmarks, such process descriptions should be accompanied by a clear statement of the important medical aspects they cover, as this would seem to be a useful guide to researchers in selecting the process descriptions that are most suitable for evaluating their technology.

In an effort to compare several languages for specifying medical guidelines, Peleg et al. [23] specified portions of two medical guidelines—for managing chronic cough and hypertension—in these languages. These guidelines and their corresponding specifications seem to focus on providing physicians with recommendations or heuristics about what tests to order and what treatments to prescribe based on symptoms and test results. The benchmark presented in this paper focuses mostly on the processes through which healthcare providers actually deliver these treatments by interacting with each other, the patient, and the resources required for care. We believe that both of these perspectives on medical processes are important and that benchmarks from each perspective are needed in order to evaluate the relative strengths and weaknesses of software engineering technologies for improving medical processes.

Researchers have already used process descriptions created in some of the languages mentioned above to evaluate the applicability of several software engineering techniques to health care (e.g., [11, 28]). To the best of our knowledge, however, the medical process descriptions that were used

for these evaluations are not publicly available to be used as benchmarks by other groups.

As noted, benchmarks have been successfully used in other domains to evaluate the strengths and weaknesses of different approaches and tools. The software testing community has developed an infrastructure (that contains a collection of programs, versions, test cases, and scripts) to support controlled experimentation with software testing techniques [12] and has utilized this infrastructure to evaluate different testing techniques (e.g., [13]). The software process modeling community developed the ISPW-6 Software Process Example [17] to "facilitate understanding, comparing, and assessing the various approaches that are being pursued for software process modeling". Since then, many researchers have used the ISPW-6 benchmark to evaluate approaches to software process modeling ([2] mentions some examples). The SPEC benchmarks [1] have been extensively used to empirically evaluate a wide range of compiler approaches (e.g., compiler optimization techniques [3, 33])

3. DESIDERATA FOR MEDICAL PROCESS DESCRIPTIONS

This section discusses desiderata that should be taken into account when considering medical processes and when designing benchmarks based on such processes. We would hope that over time, the set of available medical benchmarks would provide representative examples of all these aspects. This list is not exhaustive but contains aspects that we have identified in our case studies from the medical domain and in our interactions with medical professionals. As the community develops a set of medical benchmarks and gains more experience with their design and use, we would expect this list to be augmented.

Detail and Precision.

It is important that medical process descriptions be *detailed* enough to support meaningful reasoning about the process. For example, if one wishes to reason about the risks that can arise if a paper copy of a treatment plan becomes inconsistent with an electronic version, the process description needs to provide enough detail about the use of the paper and the electronic versions of the treatment plan to detect such inconsistencies. If a medical process description is to be utilized as a benchmark to compare the capabilities of different, and probably automated, analysis approaches, then it needs to be *precise* enough to avoid ambiguity and to ensure that the results from applying these analyses are based on process descriptions that are sufficiently complete and consistent. To further help reduce ambiguity, it is desirable to use a notation with well-defined semantics for representing the medical process description.

Artifacts.

An *artifact* is an instance of a type of object (e.g., a blood tube, a treatment plan, or a patient medication label) that is consumed, produced, or used in a health care process. As mentioned in the previous paragraph, reasoning about certain features of a medical process may necessitate information about artifacts such as the versions of a treatment plan. Thus, it is often desirable that a medical process description incorporate an explicit mechanism for specifying the artifact types and instances that are involved in the process.

Agents.

The involvement of a diverse group of medical professionals is typical for many situations in health care. For example, in a chemotherapy preparation and administration process, physicians, nurses, pharmacists, and support staff may all participate in the provision of care for a single patient. In addition, each health care professional is usually responsible for certain kinds of tasks, but under some circumstances, may perform tasks outside their traditional role. For example, a physician does not typically administer an injection but, in an emergency situation, may choose to do so. For the purposes of this discussion, we define an *agent* to be an instance of a type of human, machine, or software system that is capable of performing some set of activities (e.g., triage nurse, physician, electronic health record, robot that performs a surgery). To capture the complexity of real-world medical processes, process descriptions often need to provide information about the different agent types, and sometimes about particular agent instances (e.g., physician Phil or robot R2D2), involved in a process.

Medical professionals are continuously making decisions as they perform their tasks. These decisions are influenced by both the patient's medical condition and by the medical professional's personal working style. As a result, such decisions cannot be specified a priori and it is often desirable that medical process descriptions provide support for letting agents make decisions. For human agents, this might involve a sense of "free choice", while for automated agents this might be represented by nondeterminism.

Resources.

Resources are instances of artifacts or agents for which there typically is contention. Health care processes often involve a variety of such resources (e.g., surgeons, beds, and X-ray machines). The availability of these resources can play a critical role in the quality, efficiency, and cost of health care provision. Thus, if a medical process description is to be used to reason about, or to evaluate tools that can be used to reason about, issues such as quality, efficiency, and cost, then that process description needs to contain adequate representation of the resources used in the process. Since it is generally the case that resources are entities for which there is contention, representing them in a process description often requires additional information, such as access control policies, priorities, and resource capacities.

Aspects of Process Flow.

Medical process descriptions often need to specify complex process flow in order to provide an adequate representation of the real-world processes.

Exceptional behavior: Deviations from normal workflow occur frequently in most medical processes. For example, in a situation where a patient needs a blood transfusion, the nurse responsible for the transfusion can face a variety of non-normative events: the patient may be missing an ID band, in which case the nurse does not know if this is the right patient to receive the ordered unit of blood; the patient may be unconscious or unable to speak the language that the nurse speaks, in which case the nurse will not be able to perform the standard procedure of obtaining two patient identifiers (e.g., name and date of birth) by asking the patient for them; or the patient may have an adverse reaction to the transfused unit of blood, in which case the nurse

has to discontinue the transfusion immediately, stabilize the patient, and consult with a physician before attempting to restart the transfusion. In all these situations, the agent (in this case a particular nurse) faces exceptional circumstances and is forced to deviate from the normal execution of the process to handle those circumstances appropriately.

Sometimes, the exact location in the process flow and the exact type of exceptional event that can arise can be predicted and planned for. For example, if during normative flow there is an activity requiring a nurse to check that a piece of information on an artifact is correct, one can expect that in certain exceptional situations, this check may fail. Many medical processes, especially processes performed in highly dynamic environments such as an emergency room, however, exhibit exceptional events, such as interruptions, whose exact type and location in the process flow are harder to predict. Such exceptional events can be caused by events that are external to the process under consideration (for example, a nurse may be paged to go to a different hospital unit where help with a patient in critical condition is needed, thus interrupting the nurse's current activity) and can occur at almost arbitrary locations in the process flow. Our experience with modeling medical processes and interactions with domain experts indicates that exceptional situations or deviations from the "happy path" are common in the medical domain, suggesting that special attention needs to be paid to them when creating medical process descriptions.

Concurrency and Synchronization: In many medical processes, different activities may happen simultaneously. For example, the activities of processing and analyzing a patient's test results (e.g., evaluation of laboratory and X-ray results) could happen in parallel. In addition, at different points in the process, information may need to be exchanged or coordinated before other activities can proceed. For example, surgery may not proceed until the lab and X-ray results have both been received.

Real time constraints: Time-critical activities are prevalent in certain types of medical processes. During a surgery, for example, it may be necessary for tasks to be performed within a certain time frame, either absolute (e.g., each blood unit must be transfused within thirty minutes or arriving on the patient care unit) or relative to each other (the patient's blood pressure must be taken before administering an antihypertensive medication).

4. THE BLOOD TRANSFUSION BENCHMARK

This section first provides a high-level description of the in-patient blood transfusion process, the real-world medical process on which the benchmark components are based. Then, the individual components of the benchmark as well as the tools used to produce and analyze them are described. This paper presents some simple examples of the benchmark components; the full benchmark is available at `https://collab.cs.umass.edu/groups/laser_library/wiki/daf17/BloodTransfusionBenchmark.html`.

The benchmark components were created in the course of several years of collaboration between medical professionals and computer scientists. The main goals of the project were to define and analyze complex, high-risk medical processes to improve their safety and efficiency. At the same time, this project afforded opportunities to develop and improve certain software engineering techniques for defining and reasoning about medical processes. The benchmark components are based on a standard blood transfusion process. Standard process descriptions from the medical literature, however, tend to lack precise specification of exceptional scenarios. Thus, the knowledge and experience of the domain experts we worked with was used to describe common exceptional scenarios in the process definition. Some lower-level details in the process definition cover tasks related to computerized physician order entry (CPOE), which are specific to institutions that have CPOE systems.

4.1 Overview of the Process

In the blood transfusion process, a nurse receives a physician's order to transfuse one or more units of blood into a patient. To carry out the order, the nurse needs to perform several subprocesses: 1) checking that the patient's blood type and screen have been performed and if they have not, obtaining the blood specimen which allows the type and screen to be performed; 2) preparing documentation, and picking up the unit of blood from the blood bank; 3) performing the transfusion; and 4) performing follow-up documentation. Our benchmark for blood transfusion focuses mainly on the first and third of these subprocesses, since they are the most safety-critical.

If the type and screen have not been performed, the nurse needs to obtain a blood specimen from the patient so that the type and screen can be performed. Before the nurse can obtain the specimen, a physician's order to do so is needed. There is a possibility, however, that the computer system is temporarily down, in which case the physician needs to use a special downtime requisition form. Once the nurse has the physician order for obtaining a blood specimen, the nurse needs to obtain the appropriate specimen labels and equipment for specimen collection, verify the patient's identity, confirm that the information on the patient's identification band matches the information of the specimen label, label the specimen, perform the blood draw, and send the blood specimen to the lab. This subprocess must be conducted in the order described with one exception—the blood label may be applied either prior to immediately after obtaining the specimen for type and cross. The important safety property is that no other process may occur between obtaining and labeling the specimen. Complications can arise at several points in this subprocess (e.g., the information on the patients ID band does not match the information on the specimen label). All of these details are important for a high-risk process such as blood transfusion and together with all of the above exceptional scenarios that may arise, make the blood transfusion process particularly challenging yet interesting to study.

The subprocess for performing the actual transfusion is also quite complex. If a patient is losing a lot of blood, the physician may order the administration of several units of blood. In that case, the nurse may need to manage several transfusions simultaneously. Before administering a unit of blood, the nurse needs to clinically assess the patient, obtain the equipment necessary for the transfusion, and perform bedside checks. Again, each of these tasks has its own complexities—the clinical assessment may reveal problems in patient history that need to be addressed before the transfusion can be started, necessary equipment may be unavailable, or there could be problems with verifying the patient's

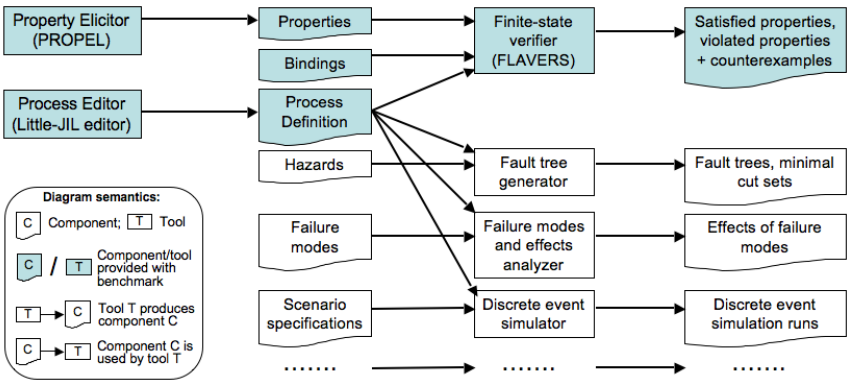

Figure 1: Benchmark architecture

identity. In addition, a subset of the bedside checks—the verification of the blood product information—needs to be performed in cooperation by two different nurses as a safety measure. Only after having completed these tasks can the nurse start the actual infusion of blood into the patient. During the infusion, patients may develop what is known as a "transfusion reaction". If this happens, the nurse needs to react quickly and perform an elaborate process of handling the transfusion reaction, which includes immediately stopping the infusion, obtaining a blood bank evaluation, notifying the physician and then acting according to the physician's orders. Finally, after the infusion of each blood product has been completed, the nurse needs to perform post-infusion tasks, such as performing another clinical evaluation of the patient, recording infusion information and discarding appropriately the infusion materials

Most of the activities in the blood transfusion process are performed by a nurse, who is responsible for the blood transfusion. Tasks performed by other agents, however, are also an important part of the process. Some examples are blood bank staff tasks for preparing the correct unit of blood and physician tasks for ordering the blood transfusion and deciding if the patient develops a transfusion reaction. The blood transfusion process requires a rich assortment of resources and artifacts (e.g., physician order, patient ID band, unit of blood product, etc.). The blood transfusion process is also interesting in terms of process flow. The normative flow is complex and has many important details itself, but elaborate deviations from the normative flow can also arise as a result of exceptional events, such as realization that the patient's type and screen are unavailable, determining that the information on the patient's ID band does not match the information on the blood product tag, or observing that the patient develops an adverse reaction after the infusion of blood has been started. The process exhibits some concurrency as a nurse may need to perform multiple blood transfusions for the same patient simultaneously. Real time constraints are also inherent to the process because. For example, the nurse needs to act within a certain time if a patient develops a transfusion reaction and a unit of blood from the blood bank must be used or returned to the blood bank within thirty minutes.

4.2 Benchmark components

The components included in the benchmark are a *definition* of the blood transfusion process, a set of *properties*, or requirements, that the blood transfusion process must satisfy, and a set of *bindings* that relate the events in the properties to steps in the process definition. In addition to the benchmark components, we also provide a set of results obtained from applying finite-state verification techniques to these components and the tools that we used to create those finite-state-verification results.

Figure 1 shows an architectural view of how the benchmark components relate to each other and indicates some of the tools and artifacts that could be used to populate these components. The second column in Figure 1 (i.e., "Properties", "Bindings", "Process Definition", etc.) corresponds to benchmark components; the first column shows the tools that we actually used to create some of these components; the third column shows other tools that could utilize the benchmark components for various kinds of analyses; and the fourth column shows potential results that might be generated from these tools. The shaded benchmark components and tools are those for which specific instances are being made available to the community for use. We have utilized the Little-JIL language editor to create a definition of the blood transfusion process in Little-JIL [6] and the Propel property elicitation tool [10] to create a set of blood transfusion properties. To establish the correspondence between the properties and the process definition, we used the Little-JIL environment (not shown) to create a set of bindings between the events in the properties and the step names in the Little-JIL definition, since these components are usually created independently of each other. The Little-JIL definition, the properties, and the bindings are input to the FLAVERS finite-state verifier [15], which then tries to determine if all possible traces through the process conform to the stated properties. The verification results consist of a confirmation that the blood transfusion process definition satisfies a given property or of a counterexample showing a process definition trace that violates the property. The shaded components and tools are described in more detail in the ensuing subsections.

The shapes that are not shaded in Figure 1 are components that can potentially be added to the benchmark and tools that can potentially operate on the existing or the newly added benchmark components. These components serve as examples of how the benchmark can continue to grow and the tools are ones that we currently have under development. In Figure 1, the definition of the blood

transfusion process can be accompanied by a set of potential hazards, which can then be used together with a fault tree generator to produce fault trees for the blood transfusion process. A fault tree and the accompanying minimal cut sets can help identify the minimal sets of events that need to occur in a process for a hazard to occur [7, 30]. To complement the information obtained from fault tree analysis, failure mode and effect analysis (FMEA) [26] can be applied. For failure mode and effect analysis, the definition of the blood transfusion process is used together with a set of failure modes to generate a table showing the effects resulting from those failure modes. One can also supply a set of scenario specifications and use them together with the blood transfusion process definition to drive discrete event simulation [24]. The resulting simulation runs can be used, for example, to reason about the effect of different resource mixes (e.g., number of physicians, nurses, beds) on quantities of interest (e.g., patient waiting time in an emergency room) so that a resource mix that optimizes these quantities of interest can be found.

We do not expect the architecture shown in Figure 1 to be final. We hope that as the community continues to study the applicability of software engineering approaches to tackling health care problems, other benchmark components and tools that support different types of analysis will be added. We also hope that the benchmark components we provide can be used with different tools and approaches and can serve as a basis for comparing the relative strengths and weaknesses of such tools and approaches. Finally, we hope that other medical case studies will be shared with the community. These case studies could have attributes similar to the blood transfusion process or they could focus on other aspects of medical processes that are not evident or emphasized by the blood transfusion process (e.g., interruptions).

Process definition.

The definition of the blood transfusion process included in the benchmark was created in the Little-JIL process definition language [6] using the Little-JIL editor. Little-JIL's support for concurrency and synchronization, exception handling, resources and artifacts makes it suitable for modeling the complex aspects of medical processes, as discussed above, and its formally defined semantics make processes defined in it amenable to rigorous analysis. A Little-JIL process definition consists of three main specifications—a resource specification, an artifact specification, and a coordination specification. The resource specification defines the agents and resources (human and non-human) needed to perform process activities. The artifact specification defines the products of the process activities. The coordination specification brings these two together by defining which agents, using which resources, perform which activities on which artifacts at which times. The main building blocks of the Little-JIL coordination specification are steps. A step corresponds to an activity performed by a human or non-human agent and, in the graphical representation, is shown iconically by a black bar. A Little-JIL process definition is a hierarchical decomposition of steps where each step can be decomposed into substeps to an arbitrary level of detail.

Figures 2 and 3 show two subprocesses of the Little-JIL definition of the blood transfusion process and illustrate some of the complexities and the detail it captures. Figure 2 shows the decomposition of the step *perform pre-infusion work*, part of the larger blood transfusion process definition. *Perform pre-infusion work* is a sequential step (indicated by the right arrow in the step bar), which means that the agent(s)[1] need to perform the substeps in order from left to right.

Three of the substeps of *perform pre-infusion work* in Figure 2 can throw exceptions (indicated by the notes[2] under the step bars). Exceptions correspond to events that require a deviation from the desired process flow. For example, if the nurse finds a problem in the patient's history while assessing a patient, the nurse needs to first handle that problem before continuing with any other tasks. When a Little-JIL step throws an exception, the exception propagates up the step hierarchy until a matching exception handler is found, and then executed. Thus, when the step *assess patient* throws *ProblemFoundInPatientHistory* exception, the matching exception handler *handle ProblemFoundInPatientHistory* is executed (exception handlers are connected to the "X" in the step bar of *perform pre-infusion work*). Similarly, when the other two substeps of *perform pre-infusion work* throw exceptions, their corresponding handlers are executed.

Exception handlers are themselves steps and can therefore be hierarchically decomposed to an arbitrary level of detail and can throw exceptions themselves. The handler *handle ProblemFoundInPatientHistory*, for example, can throw *ReceivedOrderToDiscontinueTransfusion* exception, which corresponds to the situation when the physician has decided that blood transfusion cannot be completed, (perhaps, for instance, due to a problem in the patient's history), and has ordered that the nurse discontinue the work involved in performing that transfusion. When the handling of an exception is completed, execution continues based on the resumption semantics indicated in the handler. Little-JIL supports four different resumption semantics. In the blood transfusion process definition shown in Figure 2, for example, after handling the exception *ProblemFoundInPatientHistory* is completed, the parent step needs to be restarted, which means that the nurse needs to attempt to carry out *perform pre-infusion work* again. If after completing *handleProblemFoundInPatientHistory* the nurse restarts the parent step, this will happen because the physician decided that the problem is not significant enough and the transfusion can go ahead (this communication with the physician is part of the exception handler). At this point, the nurse will never throw the *ProblemFoundInPatientHistory* exception because of this problem again since he/she already has resolved this problem with the physician.

As mentioned earlier, all four substeps of *perform pre-infusion work* and the three exception handler steps can be hierarchically decomposed into lower levels of detail (in fact, all of the above steps are further decomposed in the process definition included in the benchmark, except for *confirm presence of IV catheter* and *gather infusion materials*). We show the decomposition of the step *perform bedside checks* in Figure 3. It is a sequential step, meaning that its substeps *verify patient ID to ID band* (whose elaboration is not

[1]We elide agent and artifact information from these diagrams to avoid clutter, but this information is part of the Little-JIL process definition. The types of agents involved in the blood transfusion process are nurse (2 different instances), physician and blood bank staff.

[2]The notes are not part of the Little-JIL visual syntax but they are included in the diagrams here for clarity.

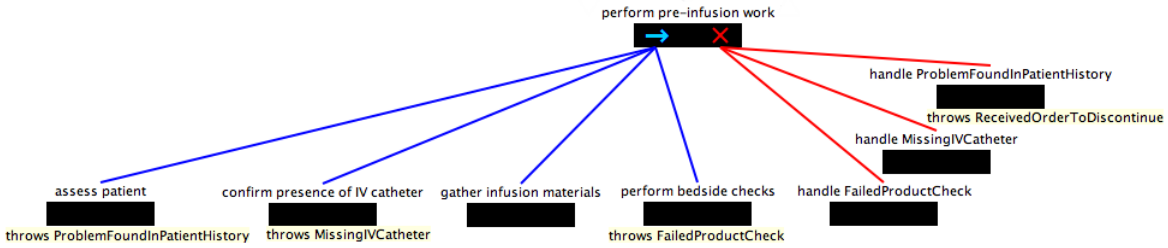

Figure 2: Elaboration of *perform pre-infusion work*

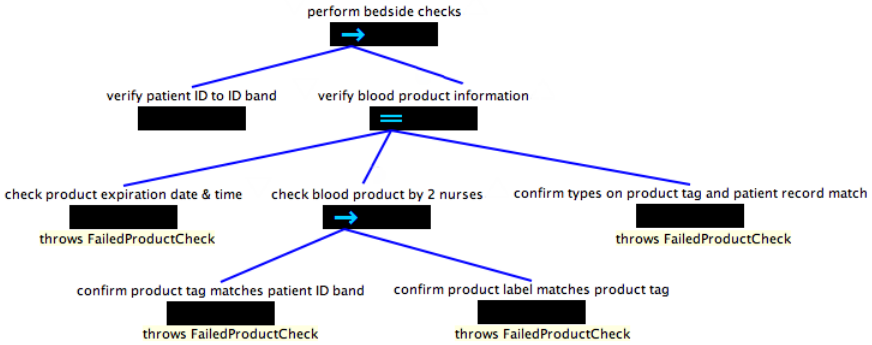

Figure 3: Elaboration of *perform bedside checks.*

shown here due to space limitations) and *verify blood product information* need to be executed in that order. *Verify blood product information* is a parallel step (indicated by the equal sign in the step bar), which means that its substeps can be executed in any order, including concurrently. Finally, the step *check blood product by 2 nurses* is decomposed into two substeps whose elaboration is again not shown due to space limitations, but is included in the benchmark. Four of the steps in Figure 3 throw a *FailedProductCheck* exception, which propagates up the step hierarchy and is handled by *handle FailedProductCheck* in Figure 2.

Properties.

Another component included in the benchmark is a set of properties related to the behavior described in the blood transfusion process definition. A *property* is a requirement or a goal that a system or process must satisfy. A property is usually independent of any particular implementation of a specific process or system and thus is required to hold across different processes, and indeed often across different organizations at which the processes are performed. The properties included in the benchmark are constraints on the event sequences that can occur during the execution of a blood transfusion process. For example, a property may state that one event cannot occur until after another one has occurred, or that some particular event must always occur after some other particular event.

Eliciting precise and accurate properties is known to be difficult and error prone [10, 29]. To facilitate property elicitation and specification, we created a set of blood transfusion properties using the Propel tool [10]. Propel provides templates for most of the common finite-state verification patterns identified in [14]. Each template provides options that the user must select in order to elaborate the full details of a property. Initially the specifier indicates the main events and answers a few questions until it becomes clear which specification pattern template applies. The specifier can then continue to answer questions using the Question Tree (QT) view, can select from sentences using a Disciplined Natural Language (DNL) view, or can select from optional labels and transitions in the Finite-State Automaton (FSA) view. No matter which of the views the specifier prefers, questions, such as "Does event B have to occur?", "Is event B allowed to occur more than once before event A occurs?", "Does the property need to hold only within a limited scope?", "Are there any exceptional scenarios where the property is not required to hold?", etc., need to be addressed when elaborating the property. The Propel specifications of the properties provide answers to these detailed questions.

Propel aims to bridge the gap between natural language, which is understandable by domain experts but is also imprecise, and a mathematical formalism, which is precise but can be hard for domain experts (and even computer scientists) to understand. Each property specified with Propel has two final representations—a finite state automaton and a disciplined natural language paragraph expressing the different aspects of the property. We believe that the natural language representation of the properties in the benchmark, although verbose, makes them easier for users of the benchmark to understand, while the formal representation is unambiguous and provides a reliable basis for comparison of software engineering tools and the methodologies that use them.

To give a sense of the kinds of properties included in the blood transfusion benchmark, we present one of the proper-

High-level property statement

Before infusing each unit of blood product into a patient, it must be checked that the medical record number on that patient's ID band matches the medical record number on the tag affixed to the unit of blood product.

Disciplined Natural Language (DNL) property statement

Scope:
1. From the start of any event sequence through the end of that sequence, the behavior must hold.

Behavior:
1. The events of primary interest in this behavior are ***infuse unit of blood product*** and ***check MRN on ID band and product tag match***.
2. There are no secondary events of interest.
3. ***infuse unit of blood product*** is not allowed to occur until after ***check MRN on ID band and product tag match*** occurs.
4. ***check MRN on ID band and product tag match*** is not required to occur.
5. Even if ***check MRN on ID band and product tag match*** does occur, ***infuse unit of blood product*** is not required to occur after ***check MRN on ID band and product tag match*** occurs.
6. After ***check MRN on ID band and product tag match*** occurs, but before the first subsequent ***infuse unit of blood*** occurs, ***check MRN on ID band and product tag match*** is allowed to occur again, zero or more times.
7. After ***check MRN on ID band and product tag match*** and the first subsequent ***infuse unit of blood product*** occur:
   - ***infuse unit of blood product*** is not allowed to occur again until after another ***check MRN on ID band and product tag match*** occurs;
   - ***check MRN on ID band and product tag match*** is allowed to occur again and, if it does, then the situation is the same as when the first ***check MRN on ID band and product tag match*** occurred, meaning that the restrictions described in parts 5, 6, and 7 would again apply.

Finite-state automaton (FSA) property representation

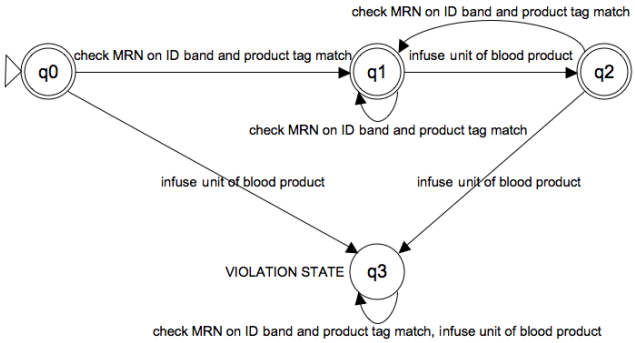

Figure 4: An example property.

ties here[3]: *"Before infusing each unit of blood product into a patient, it must be checked that the medical record number (MRN) on that patient's ID band matches the medical record number on the tag affixed to the unit of blood product"*. This property has two main events, namely *infuse unit of blood product* and *check MRN on ID band and product tag match*, and it is an instance of the *Precedence* specification pattern [14]. The informal English description, the fully elaborated English description, and the corresponding FSA for this property are shown in Figure 4. These three items are provided for all the properties in the benchmark.

Each property has three elements: an alphabet, a scope, and a behavior. The alphabet is the set of events of interest for a given property. The *scope* specifies over what parts of an execution of the process/system the property is required to hold. The property presented in Figure 4, for example, is required to hold throughout the entire execution of the blood transfusion process but, in general, a scope can specify that a property is required to hold before or after the occurrence of a given event or between the occurrences of two events. The scope can also indicate whether there are certain exceptional events after whose occurrence the property may no longer be required to hold. The *behavior* expresses the constraints between the property events that are required to hold within the specified scope. The scope and behavior are determined by selecting the options associated with the selected pattern, no matter which representations are used. In addition to the events used to delimit the scope, the primary events participating in the property pattern, and any exceptional events, properties can also have *secondary events*. Secondary events are events that might be restricted or allowed to happen between primary events. Thus, secondary events can be used to express constraints, such as "secondary event S is not allowed to occur between the occurrence of primary event A and the subsequent occurrence of primary event B."

The DNL property statement in Figure 4 elaborates the high-level property statement given informally above. The finite-state machine in Figure 4 is an equivalent representation of the DNL representation of the property, but it is formal and is thus amenable to automated analysis. The alphabet of the FSA in Figure 4 contains the primary events *infuse unit of blood product* and *check MRN on ID band and product tag match*. The FSA is originally in its initial state $q0$. For this property, $q0$ is an accepting state (indicated by the doubled circle), which means that even if none of the events in the alphabet occurs on an execution of the blood transfusion process, the property still holds. If an event from the property alphabet does occur, then a corresponding transition (labeled with that event) in the FSA is taken. For example, if the event *infuse unit of blood product* occurs on a process execution while the property is in its initial state $q0$, the transition that drives the FSA to state $q3$ is taken. In the FSA for this particular property, $q3$ is a *violation state*, meaning that if the FSA ends up in this state, the property has been violated. State $q3$ is a non-accepting (indicated by the single circle) trap state, meaning that once the FSA enters this state, it can never leave it, no matter what events occur afterwards. Thus, if *infuse unit of blood product* drives the FSA from $q0$ to $q3$, the FSA will remain in the violation state and never leave it, which in turn corresponds to a property violation since *infuse unit of blood product* occurred before the event *check MRN on ID band and product tag match* has occurred. On the other hand, if *check MRN on ID band and product tag match* occurs followed by *infuse unit of blood product*, the property will be driven from $q0$ to $q1$ and then to $q2$. State $q2$ is accepting and the property holds since *check MRN on ID band and product tag match* followed by *infuse unit of blood product* is an event sequence allowed by the property. The transitions and the states in the FSA in Figure 4 encode precisely the information in the DNL property statement.

Bindings.

To check whether a process definition satisfies a given property, one needs to provide a mapping from the abstract events used to define the property to the actual events that take place in the course of execution of the process definition. We call such mapping a *binding*. For each of the properties specified in PROPEL that are included in the benchmark, we

[3]In the interest of simplifying the discussion, the presented property is a slightly simplified version of the property included in the benchmark.

also provide the binding that establishes how the events in the property relate to the events arising from execution of the blood transfusion process definition. Frequently the executions of steps in a process definition are the events that are bound to events in a property. Thus, to be able to verify the property mentioned in the previous paragraph, for example, the event *check medical record number on ID band and product tag match* is bound to the execution of the step *confirm product tag matches patient ID band*, which is part of the process shown in Figure 3. The property event *infuse a unit of blood product* is bound to the execution of the step *infuse unit of blood product*, which is in a different part of the blood transfusion process definition (not shown in this paper due to space limitations) that is executed after the nurse has completed the step *perform pre-infusion work*. Property events do not have to be bound only to the executions of steps in a process definition. They can also be bound, for example, to the use/definition of parameters or to the throwing of exceptions. The set of bindings included in the benchmark has several such examples.

Creating the bindings between the property events and the constructs from the process definition is a human intensive process that can be difficult and error-prone. Given a set of events from a property, one needs to choose what kind of process construct to bind the events to (e.g., steps, parameters, exceptions) and also find the appropriate instance of these process constructs (i.e., a particular step, parameter, exception) in a potentially large process definition. The toolset that we make available with the benchmark components provides capabilities that facilitate the creation of bindings, such as automated search for process constructs to which events could be bound.

The need to bind abstract events of high-level properties to concrete entities in a process definition is not specific to the use of PROPEL to verify properties of Little-JIL process definitions. This need also arises when verifying properties specified in other notations against process definitions (or other models) written in other formalisms. Analysis results and performance can differ significantly with minor variations in the bindings that do not initially appear significant [21]. Thus, including a set of bindings in the benchmark seems essential if the properties and the process definition included in the benchmark are to be used as the basis for evaluating different analysis techniques.

4.3 Accompanying tools and components

In addition to the main benchmark components—the blood transfusion process definition, properties, and bindings—we also make available a set of tools. The toolset contains the Little-JIL editor, Visual-JIL, which was used to create the blood transfusion process definition and which can be used to view the graphical representation of the definition and to modify that definition. The PROPEL property elucidator, which was used to specify the blood transfusion properties and which can be used to view and modify these properties, is also included in the toolset. PROPEL can be used to export the FSA representation of the properties in standard external formats, such as XML. Finally, the toolset contains FLAVERS/Little-JIL, which was used to create the bindings and to run the finite-state verification analyses given the process definition, properties, and bindings.

We also provide the finite-state verification (FSV) results obtained from applying FLAVERS/Little-JIL. The toolset discussed in the previous paragraph automatically translates a Little-JIL process definition into a model that represents all sequences of relevant events that could occur on process executions, where the relevant events include the primary, secondary, exceptional, and scope events of the property being verified. Then FLAVERS [15] uses this model to algorithmically check whether the model satisfies a given property. If the model does satisfy the property, FLAVERS reports that. Otherwise, it produces a counterexample, which is a trace through the process on which the property is violated. We provide the verification results based on the blood transfusion process and properties as an additional component to the benchmark. For the blood transfusion example there are currently 23 properties in the benchmark. We were able to specify 21 of them in PROPEL and then use FLAVERS/Little-JIL to check whether the process definition satisfies them. The results are included in the benchmark website. The other 2 properties involved real time constraints and we were unable to specify them in PROPEL. We expect to add more properties to the benchmark as we elicit and formalize them.

5. DISCUSSION AND FUTURE WORK

This section discusses some of the strengths and limitations of the current version of the blood transfusion benchmark. We believe that this information will be useful to researchers who may decide to use the blood transfusion benchmark to evaluate different technologies on a medical process. We conclude by discussing some open issues related to the collection and management of medical benchmarks.

5.1 Current status of the blood transfusion benchmark

Currently, the definition of the blood transfusion process included in the benchmark covers most of the aspects of medical processes discussed in section 3. The process definition provides a significant amount of detail for important parts of the blood transfusion process (i.e., checking for patient blood type and screen and performing the infusion of a unit of blood product) and it is rigorous and precise since it is defined in Little-JIL, a language with formally defined semantics. Exceptional scenarios, such as what happens when the patient's blood type and screen are unavailable or what happens when the patient develops a transfusion reaction, are also well represented in the process definition. The agents responsible for the execution of the steps in the process are specified, as well as the most important artifacts and resources used in the process. This process definition contains only a few examples of concurrent execution. The process definition does not represent well the real time constraints, such as that the nurse must clinically assess the patient 15 minutes after the blood transfusion has been started. This is mainly due to the fact that the Little-JIL language currently has weak support for real time constraints.

Another limitation of the benchmark is that the process definition is currently specified only in Little-JIL. Thus, users of the process definition will need to familiarize themselves with Little-JIL's syntax and semantics to be able to fully understand the definition of the blood transfusion process. We are currently working on a tool that can automatically generate a natural language description from a Little-JIL process definition. We plan to include the generated natural language description of the blood transfusion process as a com-

ponent of the benchmark as soon as it becomes available. We believe that this natural language process description will facilitate the understanding of the blood transfusion process by people unfamiliar with Little-JIL's semantics. The natural language description by itself, however, does not fully satisfy the need for a precise and unambiguous description. In addition, it is long and verbose because of all the process details that it needs to express. One could argue that a natural language description that is totally independent of any process definition notation would be preferable to one generated from a specific process definition. But we believe that for this description to be accurate and to capture all necessary information, the description would have to become very artificial, long, and detailed—similar to the one that is automatically generated—and we are not optimistic about a description that is independent of a formal notation being sufficiently unambiguous and precise.

The set of blood transfusion properties included in the benchmark covers a wide range of requirements for blood transfusion processes, ranging from policy and regulatory requirements, such as the need for a consent form to be signed before performing blood transfusion, to purely clinical requirements, such as the specific response required for dealing with a transfusion reaction. As mentioned earlier, properties are in general high-level and are independent of a specific system or process and thus we expect that the set of properties included in the benchmark can be used with a variety of blood transfusion processes, as performed in different hospitals. A potential limitation of the set of properties comes from the fact that PROPEL, the tool used to create the FSA and DNL representations of the properties, supports only event-based properties. Event-based properties can encode constraints on the order of occurrence of certain process events but are weak in encoding constraints that involve state information. The specification patterns that PROPEL is based on also do not support real time constraints. There has been some work on extending these specification patterns with support for real-time constraints [18] and such extensions to PROPEL will be considered in the future. Most of the properties that we elicited from the domain experts, however, turned out to be event-based properties with no real time constraints and thus we were able to create formal FSA representations and a detailed DNL descriptions for these properties. There were a few properties that involved real time constraints. For these, we did not create an FSA or a DNL description, but we provide the high-level natural language statements as part of the benchmark.

We are currently working on some of the other tools and benchmark components (represented by unshaded shapes in Figure 1). Our group has elicited a description of an emergency room process from domain experts and has applied discrete event simulation techniques to reason about optimal resource allocation strategies [24]. We are also investigating how automated fault-tree analysis and failure mode and effect analysis techniques can be used to analyze medical processes.

5.2 Benchmark Issues

The prospect of accumulating a set of benchmarks to be used for the evaluation of the applicability of software engineering technologies to solving health care problems raises several important issues. As medical benchmarks get modified, having an adequate version control mechanism in place will be necessary. This will allow researchers to know which versions they are comparing and to track modifications. Modifications will need to be clearly documented, specifying such key aspects as content and authorship.

As the community accumulates a larger number of benchmarks, organizing them could also become an issue. How will benchmarks be categorized? How do different benchmarks relate to each other? Answers to such questions will not only help researchers find the most appropriate benchmark(s) for evaluating their technologies, but may also influence the creation of benchmarks for specific needs. The list of benchmark desiderata discussed in this paper can serve as a starting point to provide one possible categorization.

There has been evidence in the research literature that even when easily accessible, unambiguously described, and carefully characterized benchmarks are available, comparisons of different methodologies are still difficult to make [4], and thus we acknowledge that a set of medical benchmarks, like the one described in this paper, will not solve all problems in comparing different methodologies. We believe, however, that creating such benchmarks is an important step and will constitute an improvement of the community's ability to evaluate the relative strengths and weaknesses of different software engineering and medical informatics methodologies when applied to health care.

6. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under awards CCF-0427071, CCF-0820198, and IIS-0705772#2. 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 authors gratefully acknowledge the work of Rachel Cobleigh and Huong Phan, who made major contributions to the development of the Little-JIL definition and properties of the blood transfusion process, of Dr. Philip L. Henneman, who provided medical expertise during the project, of Bin Chen and Heather Conboy, who worked on the FLAVERS/Little-JIL toolset and helped with the creation of the bindings and with the verification, and of Alexander Wise who contributed to the development of the Little-JIL language.

7. REFERENCES

[1] Standard Performance Evaluation Corporation (SPEC) http://www.spec.org.

[2] V. Ambriola, R. Conradi, and A. Fuggetta. Assessing process-centered software engineering environments. *ACM Trans. Softw. Eng. Methodol.*, 6(3):283–328, 1997.

[3] M. Arnold, M. Hind, and B. G. Ryder. Online feedback-directed optimization of Java. In *OOPSLA '02*, pages 111–129, 2002.

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

[5] M. Balser, W. Reif, G. Schellhorn, K. Stenzel, and A. Thums. Formal system development with KIV. In *FASE '00: Proc. Third Int. Conf. on Fundamental Approaches to Softw. Eng.*, pages 363–366, London, 2000. Springer-Verlag.

[6] A. G. Cass, B. S. Lerner, J. Stanley M. Sutton, E. K. McCall, A. Wise, and L. J. Osterweil. Little-JIL/Juliette: a process definition language and interpreter. In *ICSE '00: Proc. 22nd Intl. Conf. Softw. Eng.*, pages 754–757, 2000.

[7] 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, 2006.

[8] B. Chen, G. S. Avrunin, E. A. Henneman, L. A. Clarke, L. J. Osterweil, and P. L. Henneman. Analyzing medical processes. In *ICSE '08: Proc. 30th Intl. Conf. Softw. Eng.*, pages 623–632, 2008.

[9] S. Christov, B. Chen, G. S. Avrunin, L. A. Clarke, L. J. Osterweil, D. Brown, L. Cassells, and W. Mertens. Rigorously defining and analyzing medical processes: An experience report. *Models in Softw. Eng.: Workshops and Symposia at MoDELS 2007, Reports and Revised Selected Papers*, pages 118–131, 2008.

[10] R. L. Cobleigh, G. S. Avrunin, and L. A. Clarke. User guidance for creating precise and accessible property specifications. In *SIGSOFT '06/FSE-14: Proc. 14th ACM SIGSOFT Intl. Symp. Found. Softw. Eng.*, pages 208–218, 2006.

[11] C. Damas, B. Lambeau, F. Roucoux, and A. van Lamsweerde. Analyzing critical process models through behavior model synthesis. In *ICSE '09: Proc. 2009 31st Intl. Conf. Softw. Eng.*, pages 441–451, 2009.

[12] H. Do, S. Elbaum, and G. Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Softw. Eng.*, 10(4):405–435, 2005.

[13] H. Do and G. Rothermel. An empirical study of regression testing techniques incorporating context and lifetime factors and improved cost-benefit models. In *SIGSOFT '06/FSE-14: Proc. Intl. Symp. Found. Softw. Eng.*, pages 141–151, 2006.

[14] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE '99: Proc. 21st Intl. Conf. Softw. Eng.*, pages 411–420, 1999.

[15] 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, 2004.

[16] Internation Telecommunication Union, Standardization Sector. Message sequence charts, recommendation Z.120.

[17] M. Kellner, P. Feiler, A. Finkelstein, T. Katayama, L. Osterweil, M. Penedo, and H. Rombach. Software process modeling example problem. In *Proc. 6th Intl. Softw. Process Workshop*, pages 19–29, 1990.

[18] S. Konrad and B. H. C. Cheng. Real-time specification patterns. In *ICSE '05: Proc. 27th Intl. Conf. Softw. Eng.*, pages 372–381, 2005.

[19] J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. Wiley, 2006.

[20] S. T. Mark and M. A. Musen. A flexible approach to guideline modeling. In *Proc. of AMIA Symposium*, pages 420–424, 1999.

[21] G. Naumovic, G. S. Avrunin, L. A. Clarke, and L. J. Osterweil. Applying static analysis to software architectures. In M. Jazayeri and H. Schauer, editors, *Software Engineering—ESEC/FSE '97*, volume 1301 of *LNCS*, pages 77–93, Zurich, 1997. Springer Verlag.

[22] M. Peleg, A. Boxwala, O. Ogunyemi, Q. Zeng, S. Tu, R. Lacson, E. Bernstam, N. Ash, P. Mork, L. Ohno-Machado, E. Shortliffe, and R. Greenes. GLIF3: The evolution of a guideline representation format. In *Proc. AMIA Symposium*, pages 645–649, 2000.

[23] M. Peleg, S. W. Tu, J. Bury, P. Ciccarese, J. Fox, R. A. Greenes, R. Hall, P. D. Johnson, N. Jones, A. Kumar, S. Miksch, S. Quaglini, A. Seyfang, E. H. Shortliffe, and M. Stefanelli. Comparing computer-interpretable guideline models: A case-study approach. *JAMIA*, 10:2003, 2002.

[24] M. Raunak, L. Osterweil, A. Wise, L. Clarke, and P. Henneman. Simulating patient flow through an emergency department using process-driven discrete event simulation. In *SEHC '09: Proc. 2009 ICSE Workshop on Softw. Eng. in Health Care*, pages 73–83,2009.

[25] Y. Shahar, S. Miksch, and P. Johnson. The Asgaard project: A task-specific framework for the application and critiquing of time-oriented clinical guidelines. In *Artificial Intelligence in Medicine*, pages 29–51, 1998.

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

[27] W. W. Stead and H. S. Lin, editors. *Computational Technology for Effective Health Care: Immediate Steps and Strategic Directions*. National Academies Press, 2009.

[28] 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 Intelligence in Medicine*, 36(3):193–209, 2006.

[29] A. van Lamsweerde. Formal specification: A roadmap. In A. Finkelstein, editor, *The Future of Software Engineering*, pages 147–159. 2000.

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

[31] J. M. Wilkinson and K. V. Leuven. Procedure checklist for administering a blood transfusion. http://davisplus.fadavis.com/wilkinson/Procedure_Checklists/PC_Ch36-01.doc.

[32] J. M. Wilkinson and K. Van Leuven. *Fundamentals of Nursing*. F. A. Davis Company, 2007.

[33] W. Zhang and B. Ryder. Constructing accurate application call graphs for Java to model library callbacks. In *SCAM '06: Proc. Sixth IEEE Intl. Workshop on Source Code Analysis and Manipulation*, pages 63–74, 2006.