# Rigorously Defining and Analyzing Medical Processes: An Experience Report

Stefan Christov, Bin Chen, George S. Avrunin, Lori A. Clarke, Leon J. Osterweil

Laboratory for Advanced Software Engineering Research (LASER)
University of Massachusetts at Amherst, Amherst, MA 01003
{christov, chenbin, avrunin, clarke, ljo} @ cs.umass.edu

David Brown, Lucinda Cassells, Wilson Mertens

D'Amour Center for Cancer Care, 3350 Main Street, Springfield, MA 01199
{david.brown, lucy.cassells, wilson.mertens} @ bhs.org

**Abstract.** This paper describes our experiences precisely defining the processes associated with preparing and administrating chemotherapy and then using those process definitions as the basis for analyses aimed at finding and correcting defects. The work is a collaboration between medical professionals from a major regional cancer center and computer science researchers. The work uses the Little-JIL language to create precise process definitions, the PROPEL system to specify precise process requirements, and the FLAVERS system to verify that the process definitions adhere to the requirement specifications. The paper describes how these technologies were applied to successfully identify defects in the chemotherapy process. Although this work is still ongoing, early experiences suggest that this approach can help reduce medical errors and improve patient safety. The work has also helped us to learn about the desiderata for process definition and analysis technologies that are expected to be more broadly applicable to other domains.

## 1 Introduction: The Problem and Our Proposed Approach

Medical errors cause approximately 98,000 patient deaths each year [1] in the United States. US Institute of Medicine (IOM) reports have suggested that the delivery of healthcare must fundamentally change to address medical errors (e.g., see [1] [2]). In particular, these studies suggest that many serious medical errors result from *system* rather than individual failures, leading the IOM to advocate the development of healthcare systems that directly address patient safety. In particular, the IOM report states, "what is most disturbing is the absence of real progress… in information technology to improve clinical *processes* [italics ours]" ([1], pg. 3). Thus, we have begun to investigate the application of software engineering process definition and analysis research to help reduce errors and improve safety in medical processes.

Our preliminary research (e.g., [3]) showed that many current medical processes are described only at a high-level of generality and are often not defined completely and precisely. Because of this, healthcare providers can find themselves in situations

that are not directly addressed by the processes they learned, and thus they may be unsure whether their actions conform to recommended care guidelines. In addition, aspects of current process descriptions are frequently vague, ambiguous, or inconsistent, allowing different providers to have different understandings of their specifics. Such descriptions may lead workers to believe they are following recommended guidelines when, in fact, their care has deviated, increasing the possibility of error.

In the work described here, software engineering researchers and medical experts developed precise, rigorous definitions of medical processes that capture both the standard cases and exceptional situations that can arise. The process definitions also capture the inherent concurrency and multi-tasking undertaken by busy healthcare providers, as well as details of the use of resources to perform the processes. Our investigations have indicated that there are somewhat different goals for defining and analyzing processes in different areas of medical practice, thus suggesting applying somewhat different approaches. For example, blood transfusion is primarily concerned with identification issues and emergency care is focused on improved patient flow.

In chemotherapy there seems to be an overriding concern for the identification and removal of process defects that create hazards to patient health and safety. These concerns suggest the value of at least two complementary engineering approaches, namely fault tree analysis and finite-state verification, each applied to a precise definition of safety-critical processes. Analysis of fault trees promises to indicate serious ramifications of incorrect performance of process steps [4, 5], while finite-state verification (e.g., [6, 7]) promises to identify sequences of tasks that, even if performed perfectly, could lead to safety violations [8, 9]. In this initial work in identifying and removing chemotherapy process defects, we focused on the latter, as the technologies to support it were more accessible to us. In particular, this paper describes efforts to evaluate the effectiveness of defining medical processes using a rigorously defined language, formally encoding the requirements for that process, carrying out finite-state verification of the processes to detect defects, and then improving the processes by defect removal. In the next section we present the Little-JIL process definition language and provide examples of how it was used to define a chemotherapy process. Section 3 describes and evaluates our experiences, and Section 4 overviews related work. Section 5 suggests some future research directions.

## 2 An Example: Chemotherapy Preparation and Administration

Chemotherapy is the use of chemical substances to treat disease. In its modern-day use, it refers primarily to the administration of cytotoxic drugs to treat cancer. Chemotherapy medications are typically highly toxic, and thus it is of overriding importance to be sure that the right patient receives the right medications in the right dosages at the right times. To assure this, elaborate processes are carried out that integrate the efforts of such diverse medical personnel as doctors, nurses, pharmacists, and clerical workers. Chemotherapy processes aim to speed the flow of treatment, while assuring that errors do not occur. Checks are in place to guard against committing such errors. Preliminary examination of these processes suggested that they are

large and complex, and their growing complexity makes it increasingly difficult to be sure they provide sufficient protection against the commission of errors.

Our work began by defining some example chemotherapy processes. Earlier work in defining processes in such other domains as software development, scientific data processing [10], and e-government [11] suggested that a powerful process definition language would be needed. We chose to use the Little-JIL process definition language because our previous experience suggested that semantic features of this language were likely to be effective in defining processes in the chemotherapy domain.

### 2.1 Principal Features of Little-JIL

Little-JIL [12, 13] was originally developed to define software development processes. A Little-JIL process definition has three components, an *artifact collection*, a *resource repository,* and a *coordination specification*. The artifact collection contains the items that are the products of the process. The resource repository specifies the agents and capabilities that support performing the activities. The coordination specification ties these together, 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, where a step represents a task to be done by an assigned agent. Figure 1 shows the iconic representation of a single step. Each step has a name and a set of badges to represent control flow among its sub-steps, its *interface* (specifying its input/output artifacts and the resources it requires), the exceptions it handles, etc. A step with no sub-steps is a *leaf step*. *It* represents an activity performed by an agent, without any process guidance. A full description of Little-JIL is provided in [13]. Below we present some Little-JIL features used in the example presented in this paper.

**Resources and Agents**—Each Little-JIL step interface specifies the types of resources required to support execution of the step. Some examples of resources are physicians, infusion suites, and accesses to medical records. Each step has one specially designated resource, called its *agent*, which is assigned responsibility for the performance of the step. Little-JIL agents may be humans, groups of humans, or automated devices.

**Substep Decomposition**—Little-JIL steps may be decomposed into two kinds of substeps, *ordinary substeps* and *exception handlers*. Ordinary substeps define how each step is executed and connected to its parent through edges annotated

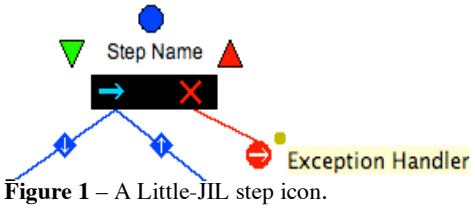


**Figure 1** – A Little-JIL step icon.

by specifications of the artifacts that flow between parent and substep. *Exception handlers* define how exceptions thrown by the step's descendants are handled.

**Step sequencing**—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. Little-JIL has four step kinds. Our example uses two, namely, the *sequential step* (right arrow) indicating that substeps execute from left to right, and the *parallel step* (equal sign) indicating that substeps execute in any (possibly interleaved) order, although the order may be constrained by such factors as the lack of needed inputs.

**Data Channels**—*Data Channels* are named entities that act like buffers, directly connecting specifically identified source step(s) with specifically identified destination step(s). This construct helps define how streaming data is handled and can also be used to synchronize concurrently executing steps.

**Exception Handling**—A Little-JIL step can throw an exception when some aspect of step execution fails. This triggers execution of a matching *exception handler* defined at an ancestor of the step that throws the exception.

### 2.2 An Example Using Little-JIL to Define a Chemotherapy Process

An example Little-JIL definition of a portion of a chemotherapy process is shown in Figure 2. This is the top-level coordination diagram of the process and thus represents it at a high level of abstraction. The entire Little-JIL process definition has more than 250 steps and thus cannot be shown in its entirety here. Elicitation of the process required two semesters of weekly meetings between process developers and medical professionals. The part of the process presented here is concise but representative of many interesting issues that arise in defining and analyzing the full process.

Figure 2 indicates that the process is decomposed into two substeps executing in parallel (note the equal sign in the step bar). Each substep is further decomposed down to the level of leaf steps for which the process definer is unable to, or uninterested in, providing process detail and guidance.

The first substep, *prepare for and administer first cycle of chemotherapy*, of the root step *chemotherapy process* is decomposed into six substeps to be executed in sequence (note the arrow pointing to the right in the step bar). The six substeps of *prepare for and administer first cycle of chemotherapy* are the major stages of the chemotherapy process: *perform consultation and assessment* is done by a Medical Doctor (MD); *perform initial review of patient records* by a Practice Registered Nurse (RN) and a Triage Medical Assistant; *perform pharmacy task* by a Pharmacist; *perform patient teaching* by a Nurse Practitioner; *perform final tasks (day before chemo)* by a Pharmacist and a Clinic RN; and *the first day of chemo*, is done again by a Pharmacist and a Clinic RN.

The "consultation channel" (shown in Figure 2 as a comment elaborating the root step's interface, which is represented iconically by the circle above the step) is declared at the root step, *chemotherapy process*, and is used to synchronize execution of the root's two substeps, which execute in parallel with each other. *Create and process consult note* is a sequential step (note the arrow in the step bar), meaning that its substeps are executed in order. The step *dictate consult note* cannot start until *perform patient consultation* (not shown for lack of space), which is a substep of *perform con-*

*sultation and assessment,* completes and writes a parameter to the "consultation channel," i.e. an MD cannot dictate the consult note before evaluating the patient's condition. On the other hand, there is flexibility in how long after consultation the MD may actually dictate the consult note. Specifically, the consult note is primarily used for billing and legal purposes, and it does not directly affect the creation of a treatment plan or the administration of chemotherapy based on that treatment plan. Thus a doctor may choose to dictate the consult note right after evaluating the patient or later, while the tasks in *prepare for and administer first cycle of chemotherapy* are already underway. This step sequencing flexibility is captured precisely by the coordination diagram in Figure 2, which shows the *dictate consult note* step potentially executing in parallel with the step *perform consultation and assessment*. These two steps synchronize their efforts when the substep of *perform patient consultation and assessment* step sends a parameter to the step *dictate consult note* via the "consultation channel."

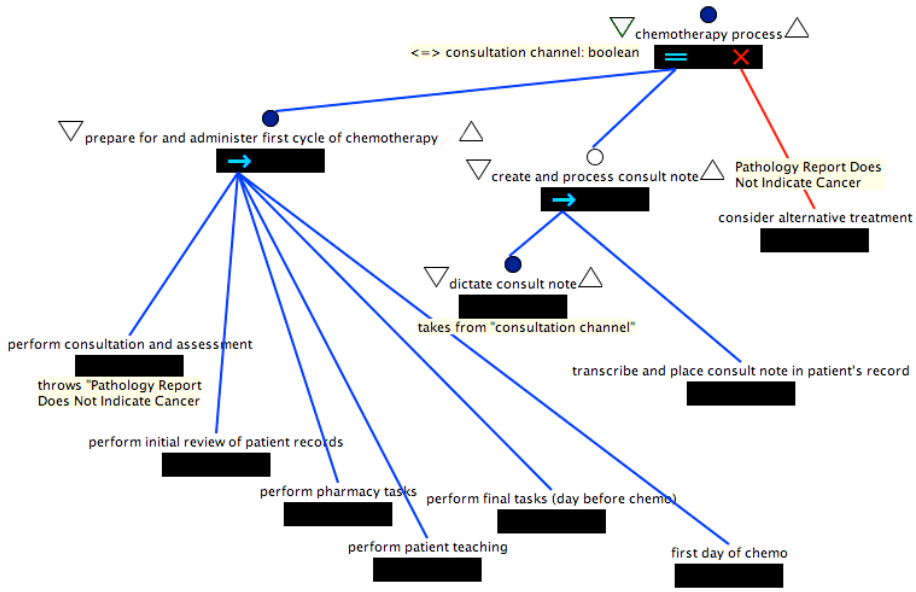


**Figure 2:** A coordination diagram of Little-JIL chemotherapy process

Figure 2 also shows that the root step *chemotherapy process* has a substep *consider alternative treatment* that acts as an exception handler (note the "X" sign on the *chemotherapy process* step bar to which the step *consider alternative treatment* is connected). In the step *perform consultation and assessment* in Figure 2, the doctor may determine that the patient's pathology report does not indicate cancer. In this case, the *Pathology Report Does Not Indicate Cancer* exception is thrown (the decomposition of the *perform consultation and assessment* step is not shown due to space limitations). The exception propagates up the step decomposition tree until it reaches a matching handler. Thus, control is transferred to the exception handler step *consider alternative treatment* and appropriate action is taken.

Note that the diagrams in Figures 2 and 3 do not include all the information needed for completeness of a Little-JIL process definition. A diagram is created using the Little-JIL visual editor, which allows the developer to suppress visualization of process details for the sake of clarity. Thus, Figures 2 and 3 do not display the resources and artifacts declarations in each step; just representing them by the circle above the step.

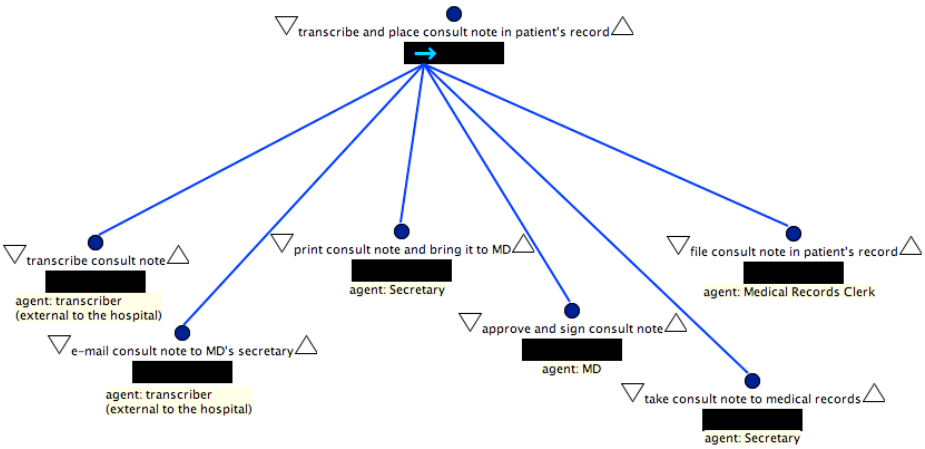


**Figure 3:** The task decomposition of *transcribe and place consult note in patient's record*

To illustrate the use of process decomposition, Figure 3 decomposes the substep *transcribe and place consult note in patient's record* of the root step *chemo process*. Although trivial at first glance, the diagram in Figure 3 supports some interesting complexities. First, note that *transcribe and place consult note in patient's record* is the second substep of the sequential step *create and process consult note* (Figure 2). This, means that *transcribe and place consult note in patient's record* cannot start until the step *dictate consult note* has completed. This sequencing mechanism is a faithful representation of the real world situation. In this process, the doctor dictates the consult note on the phone. The doctor's message is recorded and triggers the tasks of the transcriber, who is external to the clinic. The transcriber listens to the message, transcribes the consult note, emails it to the doctor's secretary and so on.

Another interesting aspect of the diagram in Figure 3 is the diverse set of agents that execute the steps – Transcriber, Secretary, Medical Doctor, and Medical Records Clerk. Thus, the timely manner in which the step *transcribe and place consult note in patient's record* is performed depends on the availability of all those agents. In a later section, we will see that the time of completion of the *transcribe and place consult note in patient's record* step relative to the time of completion of other steps in the process is important for satisfying some of the properties of the process.

### 2.3 Using PROPEL and FLAVERS Analysis to Look for Process Defects

In this section, we present a short, simplified example of the application of finite-state verification to the chemotherapy process definition. Finite-state verification techniques algorithmically check all possible paths through a model of a system to determine whether any execution of the system can violate a specified system property. In the work described here, we have used the FLAVERS [7] finite-state verifier, although other tools (e.g., [14]) could have been used. Our model of the system is an annotated control flow graph derived from the Little-JIL process definition. For our purposes, a property is a specification of the requirements for some aspect of the behavior of the system. Thus, the property is a specification against which a system is to be verified. For example, a property might state that a certain event cannot occur until after some other event occurs. Our work focuses on developing such properties with the help of domain experts (chemotherapy medical professionals in this example), eliciting a process definition from domain experts, and finally comparing the process definition against the properties. If a property is violated, we change the process (assuming the property is correctly specified) and verify the modified process against the property. We iterate the above procedure until the process satisfies the property and thus the process is improved.

In our analysis, properties are encoded as finite-state automata (FSA) and represent constraints on the sequences of events that could occur during executions of the process.

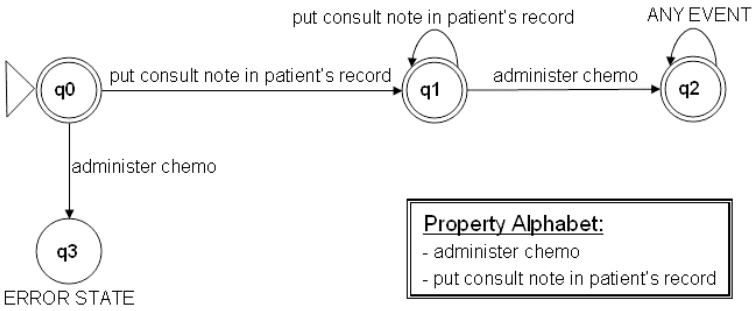


**Figure 4:** An FSA corresponding to the chemotherapy property "Before Chemotherapy Can Be Administered to a Patient, that Patient's Consult Note Needs to Be Put in that Patient's Record." A transition labeled with ANY EVENT means that the transition is taken if any event from the alphabet of the FSA occurs. The ERROR STATE is a trap state, i.e. it is a non-accepting state, such that once the automaton enters that state, it remains in it regardless of what other events occur.

The FSA in Figure 4 corresponds to the property "Before Chemotherapy Can Be Administered to a Patient, that Patient's Consult Note Needs to Be Put in that Patient's Record." The events in this property are *put consult note in patient's record* and *administer chemo*. The event *put consult note in patient's record* is bound to the step *file consult note in patient's record* in Figure 3. The event *administer chemo* is bound to the step *administer chemo drug* which is a part of the subprocess decomposition of the step *first day of chemo* in Figure 2.

At the start of execution of the process, the automaton in Figure 4 is assumed to be in its start state *q0* (indicated by the triangle to the left of state *q0*). Execution of *put*

*consult note in patient's record* causes the FSA to transition from state *q0* to state *q1*. Then if *administer chemo* is encountered during execution of the process, the FSA transitions from state *q1* to state *q2*. The state *q2* is an accepting state (indicated by doubled circle). Thus, *put consult note in patient's record* followed by *administer chemo* is a valid sequence of events in the chemotherapy process. On the other hand, if *administer chemo* occurs before *put consult note in patient's record* (the transition from state *q0* to state *q3* in the FSA shown in Figure 4), the automaton ends up in an *ERROR* state (*q3*) indicating that this causes the property to be violated. Also note that if *consult note is put in patient's record* does not occur at all, then the automaton will remain in its start state *q0*, which is also an accepting state thus indicating that the property is satisfied.

In our project, automata such as the one in Figure 4, were generated by the PROPEL (PROPerty ELucidator) system [15, 16]. PROPEL facilitates the elucidation of properties by providing three different representations of a property—a question tree view, a disciplined English view, and a finite-state automaton view—and assuring that the three views automatically remain synchronized with each other. The different views aim to bridge the gap between the natural language in which the properties are elicited from domain experts and the rigorous, but usually not trivial to specify correctly, mathematical formalism of the finite-state automaton used by the verification tool FLAVERS. Each view also explicitly indicates subtle choices that need to be made and questions that need to be answered in order to specify a property, such as whether certain events must always occur or whether other events can occur multiple times. For the example chemotherapy process, there are dozens of important safety and legal properties to be verified. Our experiments indicate that PROPEL is adept at supporting the definition of such properties.

Having defined the process in Little-JIL and created the property automaton using PROPEL, we then used the finite-state verifier FLAVERS to check whether the process satisfies the property on all possible paths of execution. If it does not, i.e. if a process execution can drive the property automaton to a non-accepting state, then FLAVERS reports the violation and produces a trace of the process execution that leads to the property violation. The verification example in this paper may appear relatively straightforward, given the simple property, but we note that it entails considerable challenges. The fact that the root step *chemotherapy process* is parallel requires that FLAVERS explore all possible execution interleavings of the substeps, creating a very large space of alternatives to be explored. The use of channels further complicates the verification. The fact that the chemotherapy process is of a significant size (more than 250 steps) makes the verification state space very large. FLAVERS employs optimization techniques and thus can usually cope with the verification of properties of processes whose size is similar to that of this chemotherapy process.

In fact, FLAVERS reported that this chemotherapy process example can violate the property presented in Figure 4, and it produced a trace of a valid execution of the process where *administer chemo drug* occurs before *file consult note in patient's record* completes. Although a channel imposes some synchronization between the parallel activities in the process, the verifier detected that concurrent execution can allow at least one execution sequence that leads to a property violation. Thus, this result identified a process defect, but it also raises an interesting question about whether

legal and privacy issues may have received much less attention than medical safety issues and thus may not be fully addressed by standard medical processes.

## 3. Experience and Evaluation

Working with the chemotherapy process suggests that our approach can lead to improvements in the processes. We were able to identify process defects and raise issues resulting in defect elimination. The medical professionals involved in the project have found benefit in this work. They are even considering using the formal process definition as the basis for training documents and guidelines for medical staff.

The very task of eliciting details from the medical professionals about the chemotherapy process and capturing those details formally in Little-JIL lead to the discovery of many of the problematic aspects in the process. One of the first observations after interviewing several different medical professionals was that the terminology used for the chemotherapy process guidelines contained some inconsistencies. For example, words like “verify”, “confirm”, “check”, “match”, and “consistent” were used loosely. The same word used at different times or in different contexts often had different meanings, even when used by the same individual. Since many of the critical errors that may occur in a process like chemotherapy may arise from neglecting small details (e.g. not checking to see if the patient height or weight measurements on which the chemotherapy dose is based are sufficiently up-to-date), we had to develop a precise naming template that disambiguated the use of different terms. Thus, our experience suggests that the effort of defining and analyzing complex medical processes can benefit if some kind of ontological structure of the domain knowledge is present.

We also found that process guidelines usually contain adequate details when describing common, standard scenarios. However, process guidelines did not provide enough details, or often any details, for handling many non-typical cases. For example, there were places in the process where an agent confirms the correctness of some information and, if the confirmation succeeds, the agent continues on with the rest of the defined tasks. However, if the confirmation fails, then in many cases the process lacked specific instructions detailing how the agent should proceed. In some cases, we noted that different agents were handling the exception differently depending on personal style, level of experience, and the individual approach of other medical professionals involved in the recovery from the failure. While modeling the process with Little-JIL, the rich exception handling semantics of the language forced us to think about exceptional scenarios and ask specific questions about the exact process to be executed following the throwing of an exception, the agents involved in resolving that exception, and the place in the process to which control gets transferred once the exception has been handled. Questions like “What do you do when the check you make fails?” and “Which task do you proceed with and which tasks do you need to redo when you have resolved the problem?” typically triggered discussions among the medical professionals that resulted in more complete and rigorous specification of how to deal with these exceptional cases, thus improving the process overall.

The resource and artifact modeling capabilities of Little-JIL also led to interesting questions during the interviewing stage that exposed some deficiencies in the process.

For example, the chemotherapy process relies heavily on a paper copy of a treatment plan, which is an artifact created at the earlier stages of the process and then verified independently and signed by medical professionals. However, doctors enter changes to a treatment plan electronically, which sometimes leads to inconsistencies between the current electronic version and the paper copy that circulates among the medical professionals. The artifact model of Little-JIL and the need to precisely describe and distinguish between paper and electronic records led to the discovery of such issues.

The expressive power of Little-JIL proved to be useful for the definition of the process in the chemotherapy case study. The powerful exception handling mechanisms in the language enabled the process definition to reflect the real world process more accurately. The capabilities the language provides for modeling resources (both agent and non-agent) and artifacts were an important part of the specification of the process. The synchronization mechanism and channel support for specifying direct communication between steps was also useful in this process definition. Hierarchy and abstraction were beneficial in helping to keep down the size of the chemotherapy process and the many different levels of abstraction at which it was defined.

The graphical notations in Little-JIL facilitated the communication of computer science concepts to the medical professionals. We usually tried to present the process to the medical professionals in textual, natural language form, but we were often asked to show the Little-JIL diagrams as they provide clearer understandings. While medical professionals are unlikely to ever write their own process definitions in Little-JIL, our experiences suggest that it is not unreasonable to expect they will be both able and willing to read Little-JIL process definitions.

The task of interviewing domain experts and specifying precisely the high level goals and requirements that the medical process needs to meet, proved to be beneficial. We worked on identifying properties at a higher level of abstraction, a level at which the property's events are not tightly coupled to concrete steps in the process definition, but rather are used to capture universal safety and legal goals that need to be satisfied no matter how the process is implemented. This approach introduced a different perspective and helped medical professionals view the process in a new light. Instead of focusing only on "what is being done", the process was approached by asking questions like "Why is this done?" and "What goal is met by this sequence of steps?" Such types of questions also helped expose deficiencies in the process and triggered discussions about how to address them. While considering the motivation behind parts of the process and the objectives that certain sequences of steps are trying to achieve, the medical professionals often identified steps that were either misplaced or missing from the process guidelines. Thus, property elicitation itself played an important role in enhancing the process.

PROPEL was of great value in facilitating the correct specification of properties. Previous experience indicated that specifying a property in a mathematical formalism, like a finite-state automaton or a temporal logic, is often not trivial and subtleties are often not captured easily or correctly. For example, consider the requirement that if patient height and weight data (used to determine correct dosage) are "stale" (i.e. the measurements are not recent enough), then height and weight must be remeasured before administration of chemotherapy. A correct formal specification of this must address such issues as whether the data can become stale several times and, if so, whether a single remeasurement is sufficient, whether the data always becomes stale,

whether remeasurement is necessary if chemotherapy is not administered for some reason, etc. In addition to the finite-state automaton view of a property, PROPEL provides natural language template, where users select phrases, and a question tree view that explicitly asks questions, like the ones above. All three of these views are equivalent and assist the user in capturing the subtleties of the property.

So far our efforts have focused on capturing the chemotherapy process in Little-JIL and specifying properties using PROPEL. Our initial use of FLAVERS focused on verifying relatively simple properties, and most of them were satisfied. In most of the cases when the verifier detected a violation, it was due to an omission or error in the process definition or property specification. However, the example in the previous section shows that our verification approach could identify real violations and pinpoint weaknesses in the process. We expect that when we begin to analyze more complicated properties over larger processes that hide potential concurrency, our approach will lead to the discovery of more defects in the process.

We note that as the size of the process under verification increases, so does the state space that needs to be explored. Large processes, like the chemotherapy one, with inherent parallelism and complex exception handling specifications, stress the importance of utilizing verifiers that scale well. At this point, our work indicates that the performance of the FLAVERS system seems to be capable of acceptable scaling. For example, the verification of the property presented in Fig. 4 took less than ten seconds of computing time running on a standard desktop computer.

## 4. Related Work

There has been some recent work using process definition and analysis to improve medical processes. For example, the Protocure II project [17] has goals that are quite similar to ours in that medical protocols are formally specified and verified. As part of that project, a protocol for jaundice and its properties were modeled in the Asbru language[18]. The protocol that was analyzed consists of 40 plans (where the plans seem to be similar to Little-JIL steps), whereas the chemotherapy process that we analyzed consists of over 250 steps. The Little-JIL process definition supports more detailed representation of the process, including support for exceptions and complex agent interactions. The Protocure researchers also encountered ambiguous use of medical terms, incomplete information, and inconsistencies that may support different conclusions. In another study that was also part of the Protocure II project[19], the Asbru model of the jaundice protocol and its properties were verified using the SMV model checker.

Noumeir has also pursued similar goals, but using a notation like UML to define processes [9]. Others (e.g., [20]), view medical processes as workflows and use a workflow-like language to define processes and drive their execution. But, we note that these projects seem to place less emphasis on analysis.

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 [21], error reporting systems [22], and process automation in laboratory settings [23], such as those where blood products are prepared for administration. In other work, Baye-

sian belief networks have been used as the basis for discrete event simulations of medical scenarios and to guide treatment planning (e.g., [24]).

Many languages and diagrammatic notations have been used to define processes. Some incorporated use of a procedural language [25]. Others used rules [26] and modified Petri Nets [27] to define processes. More recently, the workflow [28] and electronic commerce [29] communities are pursuing similar research. None of these approaches, however, seem able to support process definitions that are both clear and precise enough. Main failings of these approaches include inadequate specification of exception handling, weak facilities for controlling concurrency, lack of resource management, and inadequate specification of artifact flows.

There has also been considerable work on the analysis of code and models of systems. Finite-state verification, or model checking (e.g., [6, 7, 14]), approaches construct a finite model that represents all possible executions of the system and then analyze that model algorithmically to detect executions that violate a particular property specified by the analyst. A major concern of these techniques is controlling the size of the state-space model, while maintaining analytic precision. Our team has analyzed and evaluated various finite-state verification approaches [30], and developed verifiers such as FLAVERS [7] and INCA [31]. Our work seems to be among the first that has applied FSV approaches to process definitions [16].

## 5. Conclusion

The finite-state verification approach presented in this paper supports checking whether or not a process satisfies certain properties, but it assumes that all agents involved in the process perform their tasks without errors. However, human errors do occur in medical processes and thus complementary forms of analysis are also useful. Thus, for example, we have used a blood transfusion process definition as the basis for the automatic generation of a fault tree representation of this process and have used the fault tree to identify single points of failure in the process, thereby reducing its vulnerability to failure [5]. Similarly, our studies of delays in a hospital Emergency Department (ED) have underscored the potential for resource management to improve efficiency in the ED's processes [32]. In response, we are developing technologies to create discrete event simulations from process definitions in order to support reasoning about how to improve efficiency through better resource management.

In conclusion, we observe that this work has shown considerable promise and has suggested extensions in several directions. We propose to pursue further research in this domain. We expect that this research will provide further insights into how process definition and analysis technology can help improve the safety and efficiency of the processes in this critical domain.

## Acknowledgments

This research was funded by the US National Science Foundation under Award No. CCF-0427071 and by the U. S. Department of Defense/Army Research Office under Awards No. DAAD19-03-1-0133 and DAAD19-01-1-0564. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. National Science Foundation, U. S. Department of Defense/Army Research Office, or the U.S. Government.

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 Rachel Cobleigh and Irene Ros, who helped elicit the chemotherapy process and properties, and to many members of the staff of the D'Amour Center for Cancer Care, who graciously donated their time and expertise.

## References

1. Kohn, L.T., Corrigan, J.M., Donaldson, M.S. (eds.): To Err is Human: Building a Safer Health System. National Academy Press, Washington, DC (1999)
2. Reid, P.P., Compton, W.D., Grossman, J.H., Fanjiang, G. (eds.): Building a Better Delivery System: A new Engineering/Healthcare Partnership. National Academies Press, Washington. DC (2005)
3. Henneman, E.H., Cobleigh, R.L., Frederick, K., Katz-Bassett, E., Avrunin, G.A., Clarke, L.A., Osterweil, L.J., Andrzejewski, C., Merrigan, K., Henneman, P.L.: Increasing patient Safety and Efficiency in Transfusion Therapy using Formal Process Definitions. Transfusion Medicine Reviews **21** (2007) 49-57
4. Burgmeier, J.: Failure Mode and Effect Analysis: An Application in Reducing Risk in Blood Transfusion. Quality Improvement **28** (2002) 331-339
5. Chen, B., Avrunin, G.S., Clarke, L.A., Osterweil, L.J.: Automatic Fault Tree Derivation from Little-JIL Process Definitions. SPW/PROSIM 2006, Vol. 3966. Springer-Verlag LNCS, Shanghai, China (2006) 150-158
6. E.M. Clarke, J., Grumberg, O., Peled, D.: Model Checking. MIT Press (2000)
7. Dwyer, M.B., Clarke, L.A., Cobleigh, J.M., Naumovich, G.: Flow Analysis for Verifying Properties of Concurrent Software Systems. ACM Trans. on Software Engineering and Methodology **13(4)** (2004) 359-430
8. Cobleigh, J.M., Clarke, L.A., Osterweil, L.J.: Verifying Properties of Process Definitions. ACM SIGSOFT Intl. Symp. on Software Testing & Analysis. ACM Press, Portland, OR (2000) 96-101
9. Noumeir, R.: Radiology Interpretation Process Modeling. Journal of Biomedical Informatics **39(2)** (2006) 103-114
10. Boose, E.R., Ellison, A.M., Osterweil, L.J., Clarke, L., Podorozhny, R., Hadley, J.L., Wise, A., Foster, D.R.: Ensuring reliable datasets for environmental models and forecasts. Ecological Informatics (in press) (2007)
11. Schweik, C.M., Osterweil, L.J., Sondheimer, N., Thomas, C.: Analyzing Processes for E-Government Development: The Emergence of Process Modeling Languages. Journal of E-Government **1(4)** (2004) 63-89

12. Cass, A.G., Lerner, B.S., McCall, E.K., et al: Little-JIL/Juliette: A Process Definition Language and Interpreter. Intl Conf. on Software Engineering, Limerick, Ireland (2000) 754-758
13. Wise, A.: Little-JIL 1.5 Language Report. Lab. for Advanced SW Eng. Research (LASER), Dept. of Comp. Sci, UMass, Amherst (2006)
14. Holzmann, G.J.: The SPIN Model Checker. Addison-Wesley (2004)
15. Smith, R.L., Avrunin, G.S., Clarke, L.A., Osterweil, L.J.: PROPEL: An Approach To Support-ing Property Elucidation. 24th Intl. Conf. on Software Engineering, Orlando, FL (2002) 11-21
16. Cobleigh, R.L., Avrunin, G.S., Clarke, L.A.: User Guidance for Creating Precise and Acces-sible Property Specifications. 14th ACM Symposium on the Foundations of Software Engineering, Portland, OR (2006) 208-218
17. Protocure II, http://www.protocure.org. (2006)
18. Teije, A.t., Marcos, M., Balser, M., Croonenborg, J.v., Duelli, C., Harmelen, F.v., Lucas, P., Miksch, S., Reif, W., Rosenbrand, K., Seyfang, A.: Improving Medical Protocols by Formal Methods. Artificial Intell. in Medicine **36(3)** (2006) 193-209
19. Baumler, S., Balser, M., Dunets, A., Reif, W., Schmitt, J.: Verification of Medical Guidelines by Model Checking – A Case Study. 13th International SPIN Workshop (2006) 219-233
20. Ruffolo, M., Curio, R., Gallucci, L.: Process Management in Health Care: A System for Preventing Risks and Medical Errors. Business Process Mgmt (2005) 334-343
21. Voak, D., Chapman, J.F., Phillips, P.: Quality of transfusion practice beyond the blood transfusion laboratory is essential to prevent ABO-incompatible death. Transfusion Medicine **10** (2000) 95-96
22. Battles, J.B., Kaplan, H.S., Schaaf, T.W.v.d., Shea, C.E.: The Attributes of Medical Event Reporting Systems for Transfusion Medicine. Arch Pathology Laboratory Medicine **122** (1998) 231-238
23. Galel, S.A., Richards, C.A.: Practical Approaches to Improve Laboratory Performance and Transfusion Safety. Am. J. Clinical Pathology **107 (Suppl 1)** (1997) S43-S49
24. Gaag, L.C.v.d., Renooji, S., Witteman, C.L.M., Aleman, B.M.P., Taal, B.G.: Probabili-ties for a Probabilistic Network: A Case-Study in Oesophageal Cancer. Artificial Intelli-gence in Medicine **25(2)** (2002) 123-148
25. Sutton, S.M.J., Heimbigner, D.M., Osterweil, L.J.: APPL/A: A Language for Software-Process Programming. ACM Trans. on Software Engineering and Methodology **4(3)** (1995) 221-286
26. Ben-Shaul, I.Z., Kaiser, G.: A Paradigm for Decentralized Process Modeling and its Realization in the Oz Environment. 16th Intl. Conference on Software Engineering (1994) 179-188
27. Bandinelli, S., Fuggetta, A., Ghezzi, C.: Process Model Evolution in the SPADE Environment. IEEE Transactions on Software Engineering **19(12)** (1993)
28. Paul, S., Park, E., Chaar, J.: RainMan: A Workflow System for the Internet. Usenix Sympo-sium on Internet Technologies and Systems (1997)
29. Grosof, B., Labrou, Y., Chan, H.Y.: A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML. ACM Conf. on Electronic Commerce (EC 99), Denver, CO (1999) 68-77
30. Avrunin, G.S., Corbett, J.C., Dwyer, M.B.: Benchmarking Finite-State Verifiers. Software Tools for Technology Transfer **2** (2000) 317-320
31. Corbett, J.C., Avrunin, G.S.: Using Integer Programming to Verify General Safety and Liveness Properties. Formal Methods in System Design **6** (1995) 97-123
32. Raunak, M.S., Osterweil, L.J.: Effective Resource Allocation for Process Simulation: A Position Paper. 6th Intl. Workshop on Software Process Simulation and Modeling (ProSIM 2005), St. Louis, MO (2005)