

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 experiences in using the precise definition of a process for chemotherapy administration and as the basis for analyses aimed at finding and correcting defects, leading to improvements in efficiency and patient safety. The work is a collaboration between Computer Science researchers and members of the professional staff of a major regional cancer center. The work entails the use of the Little-JIL process definition language for creating the precise definitions, the PROPEL system for creating precise specifications of process requirements, and the FLAVERS systems for analyzing process definitions. The paper describes the details of using these technologies, by demonstrating how they have been applied to successfully identify defects in the chemotherapy process. Although this work is still ongoing, early experiences suggest that our approach is viable and promising. 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 patients to die 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 error (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 in many cases current medical processes are often described only at a high-level of generality and are usually not defined completely and precisely. Because of this, healthcare providers can often find

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

In the work we describe here, software engineering researchers and medical experts developed precise, rigorous definitions of medical processes that capture not only the standard cases, but also the exceptional situations that can arise. The process definitions also captured the inherent concurrency and multi-tasking frequently undertaken by busy healthcare providers, as well as details of the complex use of resources in performing medical processes. Our preliminary investigations have indicated that there are somewhat different goals for defining and analyzing processes in different areas of medical practice, thus suggesting 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 suggested 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 possible effects 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 still lead to safety hazards [8, 9]. In this initial phase of our work in identifying and removing defects from chemotherapy processes, we have focused on the latter. In particular, this paper describes our efforts to evaluate the effectiveness of defining medical processes using a rigorously defined language, carrying out finite state verification analysis 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 some examples of how it was used to define a chemotherapy process. Section 3 describes and evaluates our experiences, and Section 4 summarizes some related work. Section 5 summarizes some of our other work on medical processes and suggests future directions for this research.

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 integrating 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 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, and e-government 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.

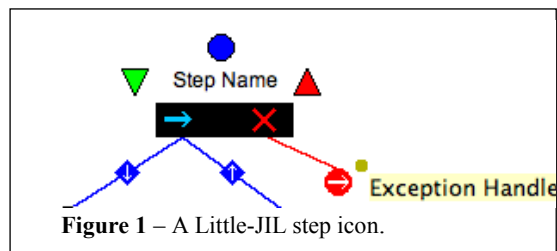
2.1 Principal Features of Little-JIL

Little-JIL [10, 11] was originally developed to define software development and maintenance processes. A Little-JIL process definition consists of three components, an *artifact collection*, a *resource repository*, and a *coordination specification*. The artifact collection contains the items that are the focus of the activities carried out by the process. The resource repository specifies the agents and capabilities that support performing the activities. The coordination specification ties these together by specifying which agents and supplementary capabilities perform which activities on which artifacts at which time(s).

A Little-JIL coordination specification has a visual representation, but is precisely defined using finite-state automata, which makes it amenable to definitive analyses. Among the features of Little-JIL that distinguish it from most process languages are its 1) use of abstraction to support scalability and clarity, 2) use of scoping to make step parameterization clear, 3) facilities for specifying parallelism, 4) capabilities for dealing with exceptional conditions, and 5) clarity in specifying iteration.

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

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. Assignment of an actual resource instance is done by a *resource manager*, which maintains a repository of available resources and their capabilities, and assigns a specific resource instance in response to the step's request. Each step requires one specially designated resource instance, called its *agent*, as the resource assigned responsibility for the performance of the step. Little-JIL agents may be 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 the step is to be executed and are connected to their parent by edges that may be annotated by specifications of the artifacts that flow between parent and substep and also by cardinality specifications. *Cardinality specifications* define the number of times the substep is instantiated, and may be a fixed number, a Kleene *, a Kleene +, or a Boolean expression (indicating whether the substep is to be instantiated). *Exception handlers* define how exceptions thrown by the step’s descendants are handled.

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. For example, a *sequential step* (right arrow) indicates that substeps execute from left to right. A *parallel step* (equal sign) indicates that substeps execute in any (possibly interleaved) order, although the order may be constrained by such factors as the lack of needed inputs. A *choice step* (circle slashed with a horizontal line) indicates step execution is by choosing among alternative substeps. A *try step* (right arrow with an X on its tail) mandates a sequence in which substeps are to be tried as alternatives.

Artifacts and artifact flows—An *artifact* is an entity (e.g., a physical entity or data item) used or produced by a step. Parameter declarations in the step interface (circle atop the step bar) list the artifacts the step uses (IN parameters) and the artifacts the step produces (OUT parameters). Artifacts flow in two different ways, hierarchically between parent and child, and through data channels. Parent-child artifact flow is indicated by edge annotations showing the artifacts and their direction of flow.

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.

Requisites—A Little-JIL step optionally can be preceded or succeeded by a step executed before or after execution of the main body of the step. A *pre-requisite* is represented by a down arrowhead to the left of the step bar, and a *post-requisite* is represented by an up arrowhead to the right of the step bar. Requisites facilitate checking that step execution is appropriate, and that it has been acceptable. The failure of a requisite triggers an exception.

Exception Handling—A step in Little-JIL can signal the occurrence of exceptional conditions when some aspect of the step’s execution fails (e.g., the violation of one of the step’s requisites). This triggers a matching *exception handler* at an ancestor of the step that throws the exception. Little-JIL can also be used to specify how execution continues after the exception is handled.

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 illustrated in Figure 2. This is the top-level coordination diagram of the process and thus represents it at a high level of abstraction. The actual Little-JIL process definition consists of more than 250 steps and cannot be addressed in its entirety due to space

limitations on this paper. Here, we present part of the process that is concise but representative of many of the interesting issues that arise in defining and analyzing the full chemotherapy process.

Figure 2 indicates that before the chemotherapy process can begin, there is a prerequisite that the patient has been referred to the clinic for cancer management (note the arrowhead on the left of the root step *chemotherapy process*). The information about the prerequisite is encoded in the step declaration, and in order to reduce visual clutter, the diagrammatic representation of the process uses a colored triangle to indicate the presence of a prerequisite. The *chemotherapy process* is decomposed into three substeps that can be executed in parallel (note the equal sign in the step bar). The first substep, *prepare for and administer first cycle of chemotherapy*, is further 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*, done by a Medical Doctor (MD); *perform initial review of patient records*, done by a Practice Registered Nurse (RN) and a Triage Medical Assistant; *perform pharmacy tasks*, which are the pharmacy verifications done by a Pharmacist; *perform patient teaching*, done by a Nurse Practitioner; *perform final tasks (day before chemo)* done by a Pharmacist and a Clinic RN; and *the first day of chemo* tasks, done again by a Pharmacist and a Clinic RN. Each one of the above six steps is, in turn, further decomposed into substeps and the decomposition continues until each step can map to an atomic activity or a satisfactory level of abstraction is achieved. Substep decomposition is used by Little-JIL to incorporate details and thus proceeds to any level desired.

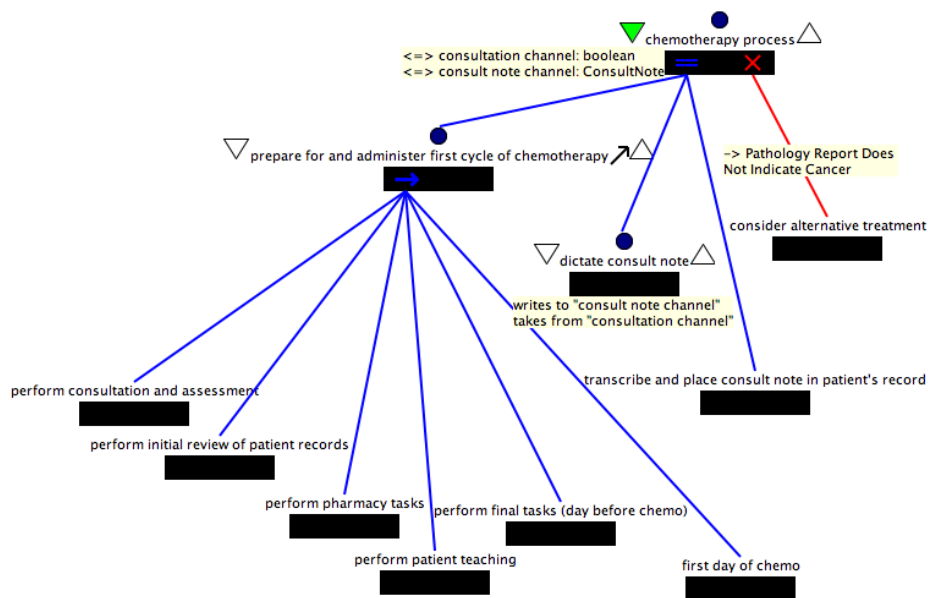


Figure 2: A coordination diagram of Little-JIL chemotherapy process

The second substep, *dictate consult note*, of the root step, *chemotherapy process*, can be executed in parallel (again, note the equal sign in the step bar of the *chemotherapy process* step) with the steps *prepare for and administer first cycle of chemotherapy* and *transcribe and place consult note in patient's record*. However, the “consultation channel,” declared at the parent step, and shown in Figure 2 as a comment elaborating the step's interface (shown iconically by the circle above the step) is also used to synchronize substep execution. Thus, execution of *dictate consult note* requires an input parameter from the “consultation channel,” and the step that writes to this channel is *perform patient consultation* (not shown for lack of space), which is part of *perform consultation and assessment* in Figure 2. Thus, *dictate consult note* cannot start until *perform patient consultation* 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 more flexibility in how long after consultation the MD may actually dictate the consult note. Specifically, the consult note itself is primarily 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 started. This step sequencing flexibility is captured precisely by the coordination diagram in Figure 2, which shows the *dictate consult note* step in parallel with the *perform patient consultation and assessment* step as long as the *perform patient consultation* step has been already completed – with synchronization by the “consultation channel.”

The third substep, *transcribe and place consult note in patient's record*, of the root step, *chemotherapy process*, can also be executed in parallel with the other substeps of the root step, although its use of the consult note as an input constrains its execution to follow the execution of *dictate consult note*, the step that generates the consult note. The further decomposition of this step is shown in Figure 3.

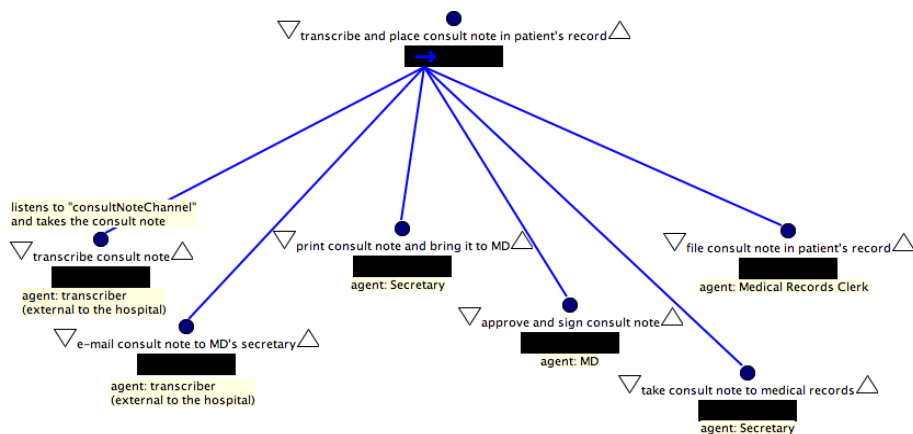


Figure 3: The task decomposition of *transcribe and place consult note in patient's record*

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, it is possible that the doctor determines that the patient's pathology report does not indicate cancer. In this case, the *Patient's 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 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.

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 hides some interesting complexities. We note that the step *transcribe consult note* uses a blocking channel, called “ConsultNoteChannel.” The step waits to read from the blocking channel, and thus until an artifact is written to the channel, the *transcribe consult note* step cannot start and none of the other substeps in the diagram can start either, since they all need to be executed in sequence (note the arrow pointing right in the step bar of the parent step *transcribe and place consult note in patient's record*). The step that writes to the channel “ConsultNoteChannel” is *dictate consult note* in Figure 2. Thus, the step *transcribe consult note* in Figure 3 cannot start until the step *dictate consult note* in Figure 2 writes an artifact to the channel. This synchronization 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 (see Figure 3).

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. If the Resource Manager has received requests from other steps for those agent resources, and especially if those requests also carry a higher priority, the execution of the substeps of *transcribe and place consult note in patient's record* can be stalled. 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 of those agent resources. 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., [12, 13]) could have been used. Our model of the system is the automaton representing 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 capturing it precisely using Little-JIL, 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 sequences of event executions in the process. Before starting the finite-state verification analysis, the events used in the property must be bound to steps in the process. An example chemotherapy process property is shown in Figure 4.

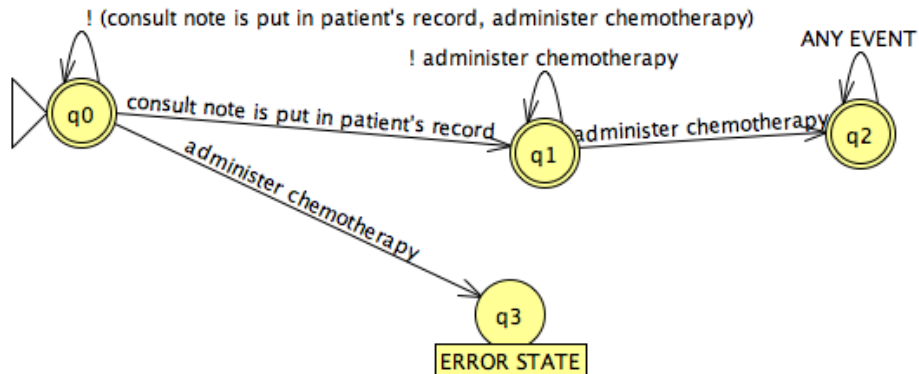


Figure 4: An FSA corresponding to the chemotherapy property “Before Chemotherapy Can Be Administered to a Patient, that Patient’s Consult Note Needs to Be Present in that Patient’s Record.” A transition labeled with ANY EVENT means that the transition is taken regardless of what other events from the alphabet of the FSA occur. 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 exclamation point (!) stands for set complement, i.e. the transition is taken on any event from the alphabet except the ones following the exclamation point.

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 *consult note is put in patient’s record* and *administer chemotherapy*. The event *consult note is put in patient’s record* is bound to

the step *file consult note in patient's record* in Figure 3. The event *administer chemotherapy* 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. The automaton in Figure 4 is originally at its start state $q0$ (indicated by the triangle to the left of state $q0$). Execution of *consult note is put in patient's record* causes the FSA to transition from state $q0$ to state $q1$. Then if *administer chemotherapy* is executed the FSA transitions from state $q1$ to state $q2$ and the automaton ends up in an accepting state $q2$ (indicated by doubled circle). Thus, *consult note is put in patient's record* followed by *administer chemotherapy* is a valid sequence of events in the chemotherapy process. On the other hand, if *administer chemotherapy* occurs before *consult note is put in patient's record* (the transition from state $q0$ to state $q3$ in the FSE 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 [14, 15]. 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. They also explicitly indicate 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. Some examples are “Before chemotherapy can be administered to a patient, the clinic's pathologist must review that Patient's pathology” and “Before a patient's treatment plan can be approved and before all the verifications of that patient's chemotherapy orders can be completed, that patient's chemotherapy orders must be consistent with that patient's treatment plan.” Our experimentation is indicating 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. In addition, 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 veri-

fication of properties of processes whose size is similar to that of this chemotherapy process.

In fact, the FLAVERS verifier reported that the chemotherapy process in this example can violate the property presented in Figure 4, and it produced a trace of a valid execution of the process such that the step *administer chemo drug* occurs before the step *file consult note in patient's record* completes. Although there exists some synchronization between the parallel activities in the process (represented by the channels), the verifier was able to detect that concurrent execution can allow for at least one sequence of steps whose execution leads to a property violation. This analytic result detects a deficiency and raises issues about improving that particular aspect of the process. 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 points 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 (like 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 enough detail and elaboration when referring to common, standard scenarios. However, process guidelines did not provide enough details, or indeed any details, for 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. When 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 the process when dealing 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 for 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. As much as possible, we 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 by different implementations of the process. 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 lead to exposing some deficiencies in the process and triggering 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 stale patient height and weight data (used to determine correct dosage) must be re-measured 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. Besides the ability to directly edit the finite state automaton of a property, PROPEL provides a question tree view that contains questions, like the ones above, and thus further assists 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 and finite state verification have 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.

4. Related Work

There has been some prior work in using process definition and analysis to improve medical processes. For example, the Procure II project [16] has goals that are quite similar to ours, but uses a rather different, AI-based, linguistic paradigm for defining processes. Noumeir has also pursued similar goals, but using a notation like UML to define processes [9]. Others (e.g., [17]), 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 [18, 19], error reporting systems [20], and process automation in laboratory settings [21], such as those where blood products are prepared for administration. In other work, Bayesian belief networks have been used as the basis for discrete event simulations of medical scenarios and to guide treatment planning (e.g., [22]).

Many languages and diagrammatic notations have been used to define processes. Some incorporated use of a procedural language [23]. Others used rules [24] and modified Petri Nets [25] to define processes. More recently, the workflow [26] and electronic commerce [27] communities are pursuing similar research. This work has shown that some notations aid process understanding, while others provide the semantic rigor needed to support verifying processes to varying degrees of certainty. None, however, seems able to support process definitions that are 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, 12, 13]), works by constructing a finite model that represents all possible executions of the system and then analyzing 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 [28], and developed verifiers such as FLAVERS [7] and INCA [29]. Our work seems to be among the first that has applied FSV approaches to process definitions [8].

5. Extensions of the Work

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 as devices for providing further insights into process quality, efficiency, and vulnerability. 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. 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 management of resources.

We have applied our process improvement approach to a broad range of domains such as labor-management negotiation, elections, and scientific data processing. The work in each domain has shown the need for additional language facilities and a broader research focus, but has confirmed the general applicability of our approach, thus pointing to the need for interesting complementary work.

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 lead to notable improvements in the quality of medical processes, and we also expect it to lead to better understandings of

how process definition and analysis technology can become key components in the more effective engineering of methods in this critically important 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, who had major responsibility for the development of Little-JIL, as well as Barbara Lerner and Aaron Cass, who also made major contributions. Major contributions were made by Rachel Cobleigh who elicited the chemotherapy properties, specified them formally in PROPEL and took part in the elicitation of the process. Contributions were also made by Irene Ros and many members of the staff of the D'Amour Center for Cancer Care.

References

1. L.T. Kohn, J.M. Corrigan, M.S. Donaldson (Eds). *To Err is Human: Building a Safer Health System*. Washington, DC: National Academy Press, 1999.
2. P.P. Reid, W.D. Compton, J.H. Grossman, G. Fanjiang (Eds). *Building a Better Delivery System: A new Engineering/Healthcare Partnership*. Nat. Academies Press, Washington. DC, 2005.
3. E.H. Henneman, R.L. Cobleigh, K. Frederick, E. Katz-Bassett, G.A. Avrunin, L.A. Clarke, L.J. Osterweil, C. Andrzejewski, K. Merrigan, P.L. Henneman, Increasing patient Safety and Efficiency in Transfusion Therapy using Formal Process Definitions, *Transfusion Medicine Reviews*, **21**, 1, pp. 49-57, January 2007.
4. J. Burgmeier, Failure Mode and Effect Analysis: An Application in Reducing Risk in Blood Transfusion. *Quality Improvement* **28**, 331-339, 2002.
5. B. Chen, G.S. Avrunin, L.A. Clarke, L.J. Osterweil, Automatic Fault Tree Derivation from Little-JIL Process Definitions, SPW/PROSIM 2006, Shanghai, China, May 20-22, 2006, Springer-Verlag LNCS. **3966**, pp. 150-158.
6. E.M. Clarke, Jr., O. Grumberg, D. Peled, *Model Checking*, MIT Press, 2000.
7. M.B. Dwyer, L.A. Clarke, J.M. Cobleigh, G. Naumovich, Flow Analysis for Verifying Properties of Concurrent Software Systems. *ACM Trans. on Software Engineering and Methodology*, **13**(4) 359-430, 2004.
8. J.M. Cobleigh, L.A. Clarke, L.J. Osterweil, Verifying Properties of Process Definitions, *ACM SIGSOFT Intl. Symp. on Software Testing & Analysis*, Portland, OR, ACM Press, 2000:96-101
9. R. Noumeir, Radiology interpretation process modeling. *Journal of Biomedical Informatics* **39**(2) 103-114, 2006.

10. A.G. Cass, B.S. Lerner, E.K. McCall, et al. Little-JIL/Juliette: A Process Definition Language and Interpreter, *Intl Conf. on Software Engineering*. Limerick, Ireland, 754-758, 2000.
11. A. Wise, Little-JIL 1.5 Language Report, Lab. for Advanced SW Eng. Research (LASER). Dept. of Comp. Sci., UMass, Amherst, Tech. Report, 2006.
12. G. J. Holzmann, *The SPIN Model Checker*, Addison-Wesley, 2004.
13. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV vers. 2: An Open-Source Tool for Symbolic Model Checking, *Computer Aided Verification Conf.*, Springer-Verlag, 2002, 359-365.
14. R.L. Smith, G.S. Avrunin, L.A. Clarke, L.J. Osterweil, PROPEL: An Approach To Supporting Property Elucidation, *24th Intl. Conf. on Software Engineering*, Orlando, FL, 11-21, 2002.
15. R.L. Cobleigh, G.S. Avrunin, L.A. Clarke, User Guidance for Creating Precise and Accessible Property Specifications, *14th ACM Symposium on the Foundations of Software Engineering*, Portland, OR, 208-218, 2006.
16. A. ten Teije, M. Marcos, M. Balsler, J. van Croonenborg, C. Duelli, F. van Harmelen, P. Lucas, S. Miksch, W. Reif, K. Rosenbrand, A. Seyfang, Improving Medical Protocols by Formal Methods. *Artificial Intell. in Medicine*, **36** (3), 193-209, 2006.
17. M. Ruffolo, R. Curio, L. Gallucci, Process Management in Health Care: A System for Preventing Risks and Medical Errors, *Business Process Mgmt.* 334-343 2005.
18. D. Voak, J.F. Chapman, P. Phillips, Quality of transfusion practice beyond the blood transfusion laboratory is essential to prevent ABO-incompatible death. *Transfusion Medicine* **10**: 95-96, 2000.
19. M.L. Foss, S.B. Moore, Evolution of Quality Management: Integration of Quality Assurance Functions Into Operations, or "Quality is Everyone's Responsibility". *Transfusion* **43** 1330-1336, 2003.
20. J.B. Battles, H.S. Kaplan, T.W. van der Schaaf, C.E. Shea, The Attributes of Medical Event Reporting Systems for Transfusion Medicine. *Arch Pathology Laboratory Medicine* **122**, 231-238, 1998.
21. S.A. Galel, C.A. Richards, Practical Approaches to Improve Laboratory Performance and Transfusion Safety, *Am. J. Clinical Pathology* **107** (Suppl 1):S43-S49, 1997.
22. L.C. van der Gaag, S. Renooji, C.L.M. Witteman, B.M.P. Aleman, B.G. Taal, Probabilities for a Probabilistic Network: A Case-Study in Oesophageal Cancer, *Artificial Intelligence in Medicine*, **25**(2), 123-148.
23. S.M. Sutton Jr., D.M. Heimbigner, L.J. Osterweil, APPL/A: A Language for Software-Process Programming, *ACM Trans. on Software Engineering and Methodology*, **4** (3), 221-286, 1995.
24. I.Z. Ben-Shaul, G. Kaiser, A Paradigm for Decentralized Process Modeling and its Realization in the Oz Environment, *16th Intl. Conference on Software Engineering*, 179-188, 1994.
25. S. Bandinelli, A. Fuggetta, C. Ghezzi, Process Model Evolution in the SPADE Environment. *IEEE Transactions on Software Engineering* **19**(12) 1993.
26. S. Paul, E. Park, J. Chaar, RainMan: A Workflow System for the Internet, *Usenix Symposium on Internet Technologies and Systems*, 1997.
27. B. Groszof, Y. Labrou, H.Y. Chan, A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML, *ACM Conf. on Electronic Commerce (EC 99)*, Denver, CO, 68-77, 1999.
28. G.S. Avrunin, J.C. Corbett, M.B. Dwyer, Benchmarking Finite-State Verifiers, *Software Tools for Technology Transfer* **2**, 317-320, 2000.
29. J. C. Corbett, G.S. Avrunin, Using Integer Programming to Verify General Safety and Liveness Properties, *Formal Methods in System Design* **6**, 97-123, 1995.