Complex Medical Processes as Context for Embedded Systems

George S. Avrunin* Lori A. Clarke*
Elizabeth A. Henneman† Leon J. Osterweil*

1 Introduction

Certification is aimed at ensuring that the requirements for a system are appropriate and complete and that the system satisfies those requirements. Even for cases where the requirements are relatively clear and unchanging and the main issue is making sure that the system satisfies them, certification presents a variety of difficult problems, and the presence of significant software components in the system exacerbates many of these difficulties. Certification must address the interaction of the software with the rest of the system (and with users), the correspondence between the software and the requirements, and the potential behavior of software that may have an extremely large number of possible inputs and executions.

Existing certification procedures, such as for avionics systems, address these problems by imposing strict standards for traceability of small units of the software back to the requirements and for the test coverage of the software. Meeting these standards is costly and they are necessarily imperfect, but they can be quite effective in ensuring the safety of systems for which the requirements are well-understood. A growing number of embedded systems, however, are intended for use in complex processes where the requirements for the system depend critically on the details of the process. If certification is to be useful for such systems, it must take the details of the process into account.

Consider, for example, the new generation of "smart" infusion pumps for intravenous administration of medications and fluids. Infusion pumps were introduced over 30 years ago as limited rate/volume devices. Current pumps, however, are used over a wide range of dosages and rates, from a milliliter or two per hour to many liters per hour, and may have several channels infusing

*Department of Computer Science, University of Massachusetts, Amherst, MA 01003, {avrunin, clarke, ljo}@cs.umass.edu
†School of Nursing, University of Massachusetts, Amherst, MA 01003, henneman@nursing.umass.edu

different medications. Errors in setting the pumps can lead to administration of 1000 times the intended dose of medication in a very short period. In response to this risk, manufacturers have introduced a new generation of "smart" pumps. These pumps are programmed (in the hospital) with libraries of drugs in use in divisions of that hospital, together with the usual concentrations, dosing units, and dosing limits for those drugs. More advanced versions of the pumps may also include information about drug interactions and provide various patient monitoring functions. A clinician using a pump typically designates an area of use (such as adult ICU or neonatal ICU) when the pump is powered on and the device is configured with appropriate libraries for that area. The clinician then completes the programming of the device, selecting the drug, concentration, etc. The pump may prevent the clinician from entering certain combinations (e.g., using the patient's weight to set dosage if the drug is not dosed by weight) and alerts the clinician if the dose exceeds pre-established limits, if the drug is already being administered on another channel, or some other hazardous condition is identified.

The way these pumps are used has enormous consequences for safety. Pumps may be used, even for the same patient, in a variety of contexts where the implications of particular dosages are quite different. For instance, some pumps provide an anesthesia mode for use in the operating room. In this mode, both the drug library and the nature of alarms and dose alerts are changed. The pump has to be changed to a different mode when the patient is moved to a post-operative or ICU setting. So the clinicians using the pump must be aware of its mode and the process must ensure that the mode is modified at appropriate times. In other settings, such questions as whether or not the clinician may override the stored settings and how responsibility for responding to alerts of improper dosages is assigned may be critical. The same device behavior may well be safe in certain uses and very dangerous in others. The safety of a smart infusion pump can only be certified in terms of the particular processes in which it is to be used.

But medical processes, especially in hospitals, are extremely complex. They are highly concurrent and exception-rich. They involve many human agents, each of whom may be participating in multiple processes at the same time. The execution of these processes often depends on whether various resources are available. Moreover, very often these processes are poorly specified, so even participants in the processes are not sure who is responsible for particular activities or what is to be done in unusual circumstances. Approaches to certification that assume that the requirements for a medical device are fixed and well understood are not adequate.

We are currently investigating a number of problems related to the description and analysis of complex medical processes. We are exploring methods for producing formal definitions of such processes, together with statements of the requirements they are intended to satisfy, that are precise and rigorous enough to support a variety of analysis methods, including finite-state verification, simulation, and others. We hope to identify safety problems, such as possible process executions on which an infusion pump is not correctly reset. In this

paper, we briefly describe this work and discuss ways in which it may provide a basis for more complete understanding of the behavior of devices in the context of the processes in which they are used, and thus for certification methods for sophisticated embedded systems.

2 Describing Medical Processes

Many medical processes, especially those carried out in hospitals, are highly complex. They may involve many practitioners, each of whom may be participating in several different processes at the same time. Exceptional conditions, which necessitate behavior other than that nominally specified, arise frequently—for example, resources may be unavailable when needed. These processes are usually specified in natural language documents, sometimes supplemented by flow charts. Such specifications lack the semantics needed to define such critical process characteristics as concurrency and exception management with the precision needed for careful analysis of the safety of the processes and with the clarity needed to make the definitions accessible.

We are using the Little-JIL process definition language [1,11] to formally define a number of complex medical processes, including the operation of a large Emergency Department (ED), the administration of outpatient chemotherapy, and in-patient blood transfusion [4]. Little-JIL is a visual language for defining the coordination of tasks that are to be executed by either computational or human agents. A process is defined in Little-JIL using hierarchically decomposed steps, where a step represents some specified task to be done by an assigned agent. Steps may also indicate prerequisites, postrequisites, and exception handling behavior that should be associated with the step. The language has a precise semantics, defined using finite state automata, and Little-JIL programs can be executed or can serve as the subject of careful static analysis.

Steps in a Little-JIL program have names and badges to represent control flow, exception handling, and pre- and postrequisites. The interface to a step specifies the resources to be used by the step, including the type of agent, such as a nurse, physician, or computer system, who is to execute the step and the flow of artifacts in and out of the step. Steps need only be defined once and can be referenced multiple times, where the semantics of each reference is quite similar to a procedure invocation.

Every non-leaf step in a Little-JIL process has a sequencing badge, which defines the order in which its substeps execute. The substeps of a *sequential* step are executed sequentially, from left to right, with successful completion of the parent step only after the successful completion of the last substep. For a *try* step, the substeps are attempted sequentially until one of them completes successfully; at that point execution of the parent step is considered to have completed successfully. The substeps of a *parallel* step are executed asynchronously (possibly concurrently) and the parent step completes successfully only when all substeps have completed successfully. For a *choice* step, the agent dynamically selects a substep to execute.

Any step in Little-JIL can throw exceptions when some aspect of the execution of that step fails, for example if a prerequisite or postrequisite is not satisfied or a necessary resource cannot be acquired. A thrown exception is handled by a matching exception handler (itself a Little-JIL step) associated with the parent of the step that throws the exception. If the parent step does not provide a handler for the particular exception, the exception is rethrown by the parent step. Exception handlers have badges that indicate how the step catching the exception executes after the handler finishes. The catching step may continue as if the substep that threw the exception had completed successfully, end its execution, rethrow the exception to its parent, or restart its execution from the beginning. A step may have a deadline specifying the maximum time allowed for completion. If a step continues to execute past its deadline, an exception is thrown. The language also includes constructs for describing the flow of artifacts between steps.

Figure 1 shows part of a Little-JIL definition of the blood transfusion process step "Single-Unit Transfusion Process." This step, shown at the top of the diagram, is a sequential step (as indicated by the arrow sequencing badge in the box below the step name), so its substeps, "Bedside Checks," "Gather Infusion Equipment," "Administer Unit of Blood Product," and "Post Transfusion Work," are executed in order unless an exception occurs. (All steps shown here with a solid black box under the step name are references to steps that are defined fully in other diagrams.) The annotations show the agents and resources associated with each step. Two exception handlers, "Unit of Blood Product Expiration" and "Suspected Transfusion Reaction" are shown, although the steps that might throw those exceptions are substeps of "Product Verification" and "Administer Unit of Blood Product" that are defined in other diagrams. The "Post Transfusion Work" step is a parallel step, so its substeps, "Discard Transfusion Materials," and "Record Infusion Information," may be performed concurrently.

In developing Little-JIL definitions of medical processes, our expectation was that the medical professionals would provide descriptions of the processes in the form of documents and oral presentations of the roles of the individual participants. The computer scientists would then construct initial Little-JIL definitions and present these to the medical professionals, who would clarify points that had been misunderstood by the computer scientists. After a number of iterations of this procedure, we expected to be able to construct reasonable Little-JIL definitions of the processes.

In reality, things are somewhat more complicated. First, of course, each side needs to learn to understand the terminology of the other. The computer scientists need to learn enough of the medical context to understand the factors that are important for the process and the medical professionals need to learn both the Little-JIL notation and what the computer scientists mean by such things as exceptions. This takes some time and there are inevitable misunderstandings while it is going on. We have been pleased to observe that, with some training, many of the medical professionals become quite fluent in Little-JIL and are able to understand the details of Little-JIL diagrams.

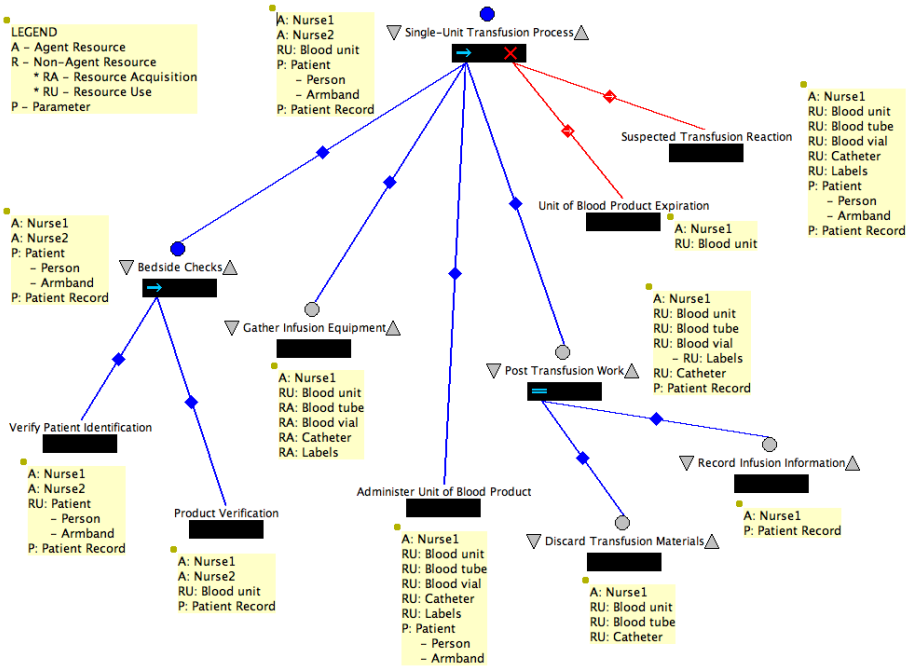

Figure 1: Little-JIL Diagram for Single-Unit Transfusion Process

Second, and more important, is the fact that many of these complex processes are poorly understood even by the medical professionals. As noted earlier, the existing descriptions of the processes are often quite incomplete. For example, even when documents describing the process exist, they often describe only the nominal execution and do not consider the exceptional conditions that arise if some step in the nominal execution fails. Furthermore, we have found repeatedly that the different participants in the process have very different views of the process, leading to serious misconceptions about what other participants will do. For instance, we discovered a mismatch between nurses and pharmacists about how data on body size were used to determine chemotherapy dosages. Incorrect assumptions about how other agents (including "smart" devices) use data or perform validity and safety checks are an important safety issue in themselves. Furthermore, of course, accurate information about the activities of the agents in the process is necessary to determine the precise requirements against which the functioning of a "smart" device must be certified.

3 Analysis

We are interested in constructing formal definitions of medical processes because we want to apply various analysis techniques to those definitions in order to

identify potential sources of medical errors. (We are also interested in using these formal definitions as a basis for automation support, identifying inefficiencies, and clarifying the roles of agents, but these aspects of our work are less relevant to certification.) In this section, we discuss the kinds of analysis we are interested in and sketch some of the progress we have made.

3.1 Properties

For safety analyses of processes, we need to know what constitutes unsafe behavior. In particular, to apply rigorous analysis techniques, we need precise statements of various properties that the process is expected to enforce. It is, however, very difficult to get such statements. Although a variety of formal notations for specifying properties or requirements have been proposed, including various kinds of automata and temporal logics, it is hard even for experts in these notations to capture the desired behavior precisely. Moreover, these formal representations are unlikely to be understandable to the domain experts—in our case, the medical professionals—who know what the system is supposed to do. On the other hand, natural language descriptions that are understandable by the domain experts are typically ambiguous and incomplete. Consider the following simple property for blood transfusion:

> *Before starting to transfuse each unit of blood product into a patient, the nurse must verify the patient's identifying information.*

This looks straightforward, but there are a number of details that must be specified before a precise formal specification can be created. For example,

- Is the nurse required to verify the patient's identifying information at least once (whether or not the blood transfusion subsequently occurs)?
- Can the nurse verify the patient's identifying information more than once before the blood transfusion begins?
- Can anything happen between the nurse verifying the patient's identifying information and the start of the blood transfusion? (E.g. can this verification take place hours, or days before the transfusion? Can the nurse leave the room between verification and the start of the transfusion?)

We have been developing a property specification approach and a corresponding tool, called Propel, for "PROPerty ELucidator" [9, 10], that aims to guide specifiers through the process of creating property specifications that are both precise enough to be used by FSV tools and accessible enough to be understood by domain experts. To provide this guidance, the approach focuses on helping specifiers to elucidate subtle, but important, property details that need to be considered but are often overlooked. The properties are represented using templates that explicitly indicate the possible variations associated with these details. Our approach currently provides three alternative, but coordinated, representations of these templates and, depending on the developer's preferences, these representations can be used in isolation or in combination. These

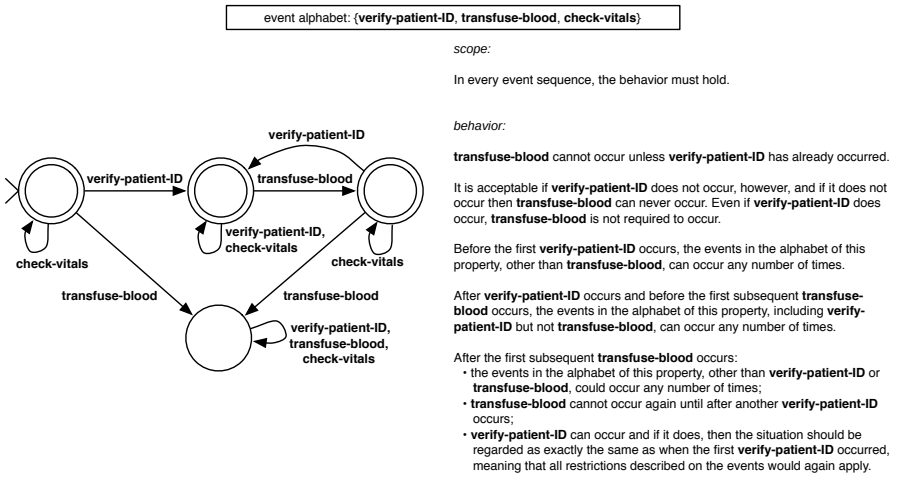

Figure 2: FSA and DNL versions of a blood transfusion property

representations are: a graphical depiction of a finite-state automaton (FSA), which offers precision and can be used as the basis for verification and other types of automated analyses; a "disciplined" natural language (DNL) description, which offers accessibility; and an interactive question tree format, which offers accessibility and additional user guidance. Figure 2 shows the FSA and DNL representations of an elaborated version of the property mentioned above. Given a suitable model of a blood transfusion process, analysis techniques could be used to determine if any possible execution of the process, including, for instance, ones in which a nurse is called to assist with another patient after having started the transfusion process, can violate the property.

Working with the medical professionals involved in the processes we are studying, we are using Propel to specify properties that the processes are intended to satisfy. We have found that, as with formalizing the process definitions, the effort to specify properties precisely is useful for the medical professionals even before checking those properties. Careful specification of the properties has exposed a number of issues involving such things as the unclear use of language in policy documents and the identification of exceptional conditions in which a property need not hold.

3.2 Applying Analysis Techniques to Medical Processes

Once the requirements for a system have been specified, hopefully completely, certification involves showing that the system satisfies those requirements. In our work we have used Propel to capture key requirements as properties defined by means of FSAs. We have then used these FSAs in applying finite-state verification (FSV) technology to check adherence of the processes to the properties.

FSV techniques, such as model checking [3], were developed to check whether a concurrent system satisfies certain properties, such as freedom from deadlock or the occurrence of one event always being followed by the occurrence of another. Testing, which is certainly the most widely used method for evaluating whether a system meets its requirements, is particularly problematic for concurrent systems. Even for sequential systems, it is usually infeasible to test more than a tiny fraction of the possible inputs. For concurrent systems, differences in the order of events in different parts of the system may lead to different behavior on subsequent executions of the same test case. Thus, correct behavior of the system on one execution with a given test case does not imply that the system will behave correctly on a later execution with the same inputs.

In order to apply an FSV technique to check properties of a system, it is necessary to construct a model that represents the executions of that system in a form that can be utilized by that technique. We have built automated model construction tools that take a Little-JIL process definition and construct models for three tools that implement different FSV techniques, FLAVERS [6], SPIN [7], and LTSA [8]. Our model constructors first translate the Little-JIL definition into the Bandera Intermediate Representation (BIR) [5] and then translate from the BIR into the representation required by the FSV tool. In order to reduce the size of the models and improve the efficiency of the verification, we apply several abstractions and optimizations during the translation process. These transformations are *conservative*, in the sense that the verification process will never falsely report that a property holds for all process executions. (It is sometimes the case that a much smaller model can be constructed if it is allowed to include extra "executions" that do not correspond to executions of the real system, so we may occasionally get incorrect reports that a property does not hold on some "execution.")

We are still refining the Little-JIL definitions of the ED and outpatient chemotherapy processes, but we have applied FSV methods to a number of properties of the blood transfusion process [4]. Most properties we checked were successfully verified, but we have detected some interesting errors involving exceptions and concurrent behavior that lead to unexpected event orderings. We are currently investigating ways to model the failure of a process step that is not recognized by the agents involved, e.g., a nurse comparing the label on a unit of blood with the patient's identification bracelet and erroneously saying that they match. With better ways of modeling such failures, we will also be able to use FSV techniques to measure redundancy in processes by determining, for example, that all possible executions of the process involve three successive checks of patient identity. If we can use actual data to conclude that two checks are enough to ensure an appropriate level of safety, the process can be modified to eliminate the third check.

We have also recently begun the investigation of other methods for analyzing formal process definitions. We mention two of these here, though space limitations prevent a discussion of details. First, we are developing a simulation engine that runs directly off the Little-JIL process definition. This will allow us to investigate the impact of changes in such things as staffing and resource

availability, or even changes in the basic process itself, on such factors as waiting times in the ED.

We are also exploring the automatic generation of fault trees from Little-JIL definitions [2]. Fault Tree Analysis is widely used in safety engineering to identify the possible causes of hazardous conditions. Manual construction of fault trees for complex processes is extremely time-consuming and error-prone, and the automatic construction of fault trees may be very useful in detecting safety problems in medical processes.

4 Discussion

The correct behavior of devices used in complex, multi-agent medical processes must be understood in the context of those processes. Behavior that is safe in one process context may well be dangerous in a slightly different one. The certification of medical devices used in such processes must take the processes into account.

In this paper, we have outlined some of our current research on describing and analyzing medical processes. We are currently developing formal process definitions for a large emergency department, for outpatient chemotherapy, and for blood transfusion. Our work is based on the Little-JIL process definition language and makes use of the Propel tool for specifying properties of these processes. We are applying several finite-state verification tools to check properties of the medical processes, as well as investigating other ways of analyzing the processes to evaluate their safety and efficiency.

We see several ways in which this work could support certification of medical devices. The device could be treated as an agent in the Little-JIL definition of a process. The behavior of the device could be represented either in Little-JIL or in constraints used in finite-state verification, and safety properties of the process including the device could be verified. Similarly, a finite-state model of the device could be constructed and that model (or an abstracted version of it constructed after some initial analysis) could be combined with the finite-state model of the process used for verification. In a related way, an abstracted version of the finite-state model of the process definition could be used with the device to show that the device behaves safely as long as the process satisfies certain constraints. In this way, the device could be certified within a "process envelope," and the certification would only hold when the device was used in a process satisfying these constraints.

Certification of devices that do not have rich interactions with a complex process is certainly not a simple problem. Certification of medical devices that do have such interactions presents many additional challenges. But many medical devices do interact in complicated ways with complex processes and those challenges must be addressed if the safety and effectiveness of those devices are to be ensured.

Acknowledgments

This research was partially supported by the National Science Foundation under grants CCR-0205575 and CCF-0427071, by the U.S. Army Research Laboratory and the U.S. Army Research Office under agreement DAAD190110564, and by the U.S. Department of Defense/Army Research Office under agreement DAAD190310133. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation, the U.S. Army Research Office or the U.S. Department of Defense/Army Research Office.

References

[1] A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, J. Stanley M. Sutton, and A. Wise. Little-JIL/Juliette: A process definition language and interpreter. In *Proceedings of the 22nd International Conference on Software Engineering*, pages 754–757, Limerick, Ireland, June 2000.

[2] B. Chen, G. S. Avrunin, L. A. Clarke, and L. J. Osterweil. Automatic fault tree derivation from Little-JIL process definitions. In *Proceedings of the Software Process Workshop/Workshop on Software Process Simulation (SPW/ProSim 2006)*, Shanghai, May 2006. to appear.

[3] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, 2000.

[4] L. A. Clarke, Y. Chen, G. S. Avrunin, B. Chen, R. Cobleigh, K. Frederick, E. A. Henneman, and L. J. Osterweil. Process programming to support medical safety. In M. Li, B. Boehm, and L. J. Osterweil, editors, *Unifying the Software Process Spectrum: International Software Process Workshop, SPW 2005*, number 3840 in LNCS, pages 347–359, Beijing, May 2005.

[5] J. C. Corbett, M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Păsăreanu, Robby, and H. Zheng. Bandera : Extracting finite-state models from Java source code. In *Proceedings of the 22nd International Conference on Software Engineering*, pages 439–448, June 2000.

[6] M. B. Dwyer, L. A. Clarke, J. M. Cobleigh, and G. Naumovich. Flow analysis for verifying properties of concurrent software systems. *ACM Trans. Softw. Eng. Meth.*, 14(3):359–430, 2004.

[7] G. J. Holzmann. *The* Spin *Model Checker*. Addison-Wesley, Boston, 2004.

[8] J. Magee and J. Kramer. *Concurrency: State Models & Java Programs*. Wiley, 1999.

[9] R. L. Smith, G. S. Avrunin, and L. A. Clarke. From natural language requirements to rigorous property specifications. In *Workshop on Software*

Engineering for Embedded Systems (SEES 2003): From Requirements to Implementation, pages 40–46, Chicago, IL, Sept. 2003.

[10] R. L. Smith, G. S. Avrunin, L. A. Clarke, and L. J. Osterweil. PROPEL: An approach supporting property elucidation. In *Proceedings of the Twenty-Fourth International Conference on Software Engineering*, pages 11–21, Orlando, FL, May 2002.

[11] A. Wise. Little-JIL 1.0 language report. Department of Computer Science Technical Report UM-CS-1998-024, University of Massachusetts, Amherst, MA, 1998.