Formally Defining Medical Processes

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

Department of Computer Science
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, Springfield, MA 01199
{david.brown, lucy.cassells, wilson.mertens} @ bhs.org

Abstract. *Objectives:* To demonstrate a technology-based approach to continuously improving the safety of medical processes.

Methods: The paper describes the Little-JIL process definition language, originally developed to support software engineering, and shows how it can be used to model medical processes. A Little-JIL model of a chemotherapy process demonstrates how this model, and some process analysis technologies that are also briefly described, can identify process defects that pose safety risks.

Results: Rigorously modeling medical processes with Little-JIL and applying automated analysis techniques to those models helped identify process defects and vulnerabilities and led to improved processes that were reanalyzed to show that the original defects were no longer present.

Conclusions: Creating detailed and precisely defined models of medical processes that are then used as the basis for rigorous analyses can lead to improvements in the safety of these processes.

Keywords: medical safety, continuous process improvement, process modeling

1 Introduction: Technologies for Medical Process Improvement

Continuous process improvement has been employed to improve quality in such diverse areas as manufacturing, software development, and business administration. The work described in this paper outlines how this approach might be applied to address quality improvement in the medical domain. The need for quality improvement in medical care has become increasingly clear in recent years. Much of the attention to this problem was catalyzed by the IOM report, "To Err Is Human" (1), which estimated that in the United States approximately 98,000 deaths per year were attributable to avoidable errors. That report and a subsequent related IOM report (2) suggest that computer technologies should be employed to address the underlying problems and reduce the incidence of errors that cause needless cost and suffering. There are

This article is not an exact copy of the original published article in Methods of Information in Medicine. The definitive publisher-authenticated version of *Christov, S., Chen, B., Avrunin, G.S., Clarke, L.A., Osterweil, L.J., Brown, D., Cassells, L. and Mertens, W. 2008. Formally Defining Medical Processes. Methods of Information in Medicine. Special Topic on Model-Based Design of Trustworthy Health Information Systems, 47 (5). 392-398* is available online at:
http://www.schattauer.de/index.php?id=1948&L=1&artikel=23710&cHash=6a4f3ba1ef

many candidate technologies for doing this. We suggest that technologies originally developed to support continuous process improvement for software systems seem to be particularly applicable.

Continuous process improvement was espoused by W. Edwards Deming (3), whose work was applied successfully by the Japanese auto manufacturing industry. That early success led to the adoption of the principles of continuous process improvement in wider domains and more countries. The core of the idea is the so-called PDCA (Plan-Do-Check-Act) paradigm, often referred to as the "Deming Cycle" (although Deming himself refers to it as the "Shewhart Cycle", in honor of Walter Shewhart, who had articulated the ideas previously (4)). Fundamentally, this cycle posits that there is a process that is the central focus of improvement. In the Plan phase of the PDCA cycle, improvements to the process are formulated and considered. In the Do phase, the improvements are installed. In the Check Phase, the proposed improvements are analyzed and evaluated to see that they are indeed likely to effect improvements. In the Act phase, the modified process is actually deployed. The net effect of this full cycle should be demonstrable improvement. The resulting outcome of the cycle is then the subject of the Plan phase of the next cycle.

Our efforts to apply the PDCA cycle to improving quality of medical care have focused on defining medical processes and then applying analyses to them. In particular, we used a process definition language, Little-JIL, to define medical processes clearly, precisely, and in detail. In this paper, we concentrate mainly on this aspect of our approach – the creation of a precise process definition. We have also used a property specification system, PROPEL (5), to define event sequence properties and then have demonstrated that analyzers, such as FLAVERS (6) and SPIN (7), can be used to determine whether the Little-JIL-defined processes conform to the PROPEL-specified properties. We have undertaken several case studies to evaluate this approach and have found that the technologies mentioned above are indeed useful in supporting medical process improvement of the sort advocated by Shewhart and Deming.

In the next section, we present the Little-JIL process definition language and provide examples of how it is used to define a chemotherapy process. Section 3 describes our experiences, and Section 4 overviews related work. Section 5 suggests some future research directions.

2 An Example: Chemotherapy Preparation and Administration

Chemotherapy medications are typically highly toxic, and thus it is of overriding importance 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. 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.

We began by defining some example chemotherapy processes. Earlier work in defining processes in such other domains as software development, scientific data proc-

essing (8), and e-government (9) 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 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 restrict data and control flow, 3) facilities for specifying concurrency, 4) capabilities for dealing with exceptional conditions, 5) capabilities for specifying the utilization of resources, and 6) 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. Each black bar in Figure 1 is an iconic representation of a step with some of its features. Each step has a name and a set of badges to represent control flow among its substeps, its *interface* (specifying its input/output artifacts and the resources it requires), the exceptions it handles, etc. A step with no substeps is a *leaf step*. It represents an activity performed by an agent, without any process guidance. Below we present some Little-JIL features.

Resources and Agents—A Little-JIL step interface (represented by a filled circle above the step name) specifies the types of resources required to support execution of the step. Some examples of resources are infusion suites and medical records. Each step has one special resource, called its *agent*, which has responsibility for performing 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 are connected to their parent by edges annotated 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 in the step bar; e.g., the equal sign on the step *chemotherapy process* in Figure 1) that defines the order of substep execution. Little-JIL has four step kinds. The example depicted in Figure 1 uses two, 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 resources.

Channels—*Channels* are named entities that act like buffers, directly connecting specifically identified source step(s) with specifically identified destination step(s). This construct supports non-parameterized data flow and helps synchronize concurrently executing steps.

Exception Handling—A Little-JIL step can throw an exception when some aspect of its execution fails. This triggers execution of a matching *exception handler* defined at an ancestor step of the step throwing the exception. Figure 1 shows an exception handler *consider alternative treatment* (connected to the X in the root step bar), which is triggered when one of the children of the root step throws a matching exception.

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

Figures 1, 2, and 3 depict part of a Little-JIL definition of a chemotherapy process. Figure 1 is the top-level 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. The part of the process definition that is depicted here is concise but representative of many interesting issues that arise in defining the full process.

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 1, 2 and 3 do not display full details of the resources and artifacts declarations in each step but represent them iconically by the step's interface circle.

Figure 1 indicates that the process definition is decomposed into two substeps that can be executed in parallel (note the equal sign in the step bar). In the full process definition, each substep is further decomposed down to the level of leaf steps for which the process definer is unable to provide, or uninterested in providing, process detail. As noted above, Figure 1 also shows that the root step *chemotherapy process* has a substep *consider alternative treatment* that is as an exception handler (note the

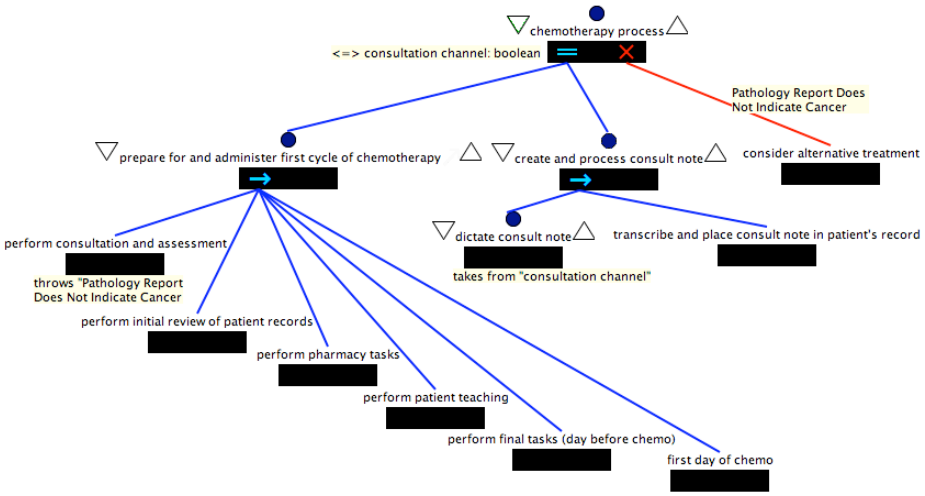

Figure 1: A coordination diagram of Little-JIL chemotherapy process

"X" sign on the *chemotherapy process* step bar to which the step *consider alternative treatment* is connected).

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 right arrow in the step bar). The six substeps of *prepare for and administer first cycle of chemotherapy* are the major stages of the chemotherapy process. Although the agent assignments are not shown in this diagram, *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.

In this example, a channel is used to specify that an MD cannot dictate the consult note before evaluating the patient's condition. But, since the consult note is primarily used for billing and does not directly affect the patient's treatment, the 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 underway. This step sequencing flexibility is captured by the diagram in Figure 1, which shows that the *dictate consult note* step can potentially execute in parallel with the step *prepare for and administer first cycle of chemotherapy*. At the same time, the "consultation channel" imposes the additional restriction that the MD cannot dictate the consult note before evaluating the patient's condition – the step *dictate consult note* takes a parameter from the "consultation channel" (declared at the root step so that it is visible, hence usable, by all of its descendants) and thus cannot start until *perform patient consultation* (shown in Figure 2), which is a substep of *perform consultation and assessment,* completes and writes a parameter to the "consultation channel.

Figure 2 shows the decomposition of the step *perform consultation and assessment* from Figure 1. Since *perform consultation and assessment* is a sequential step (right arrow in the step bar), its substeps need to execute in the order specified in the diagram. Thus, first the patient has to fill out medical history forms, then a medical assistant (MA) has to measure height and weight, record them, and check the vital signs of the patient. After that, the medical doctor (MD) has to examine the patient, perform

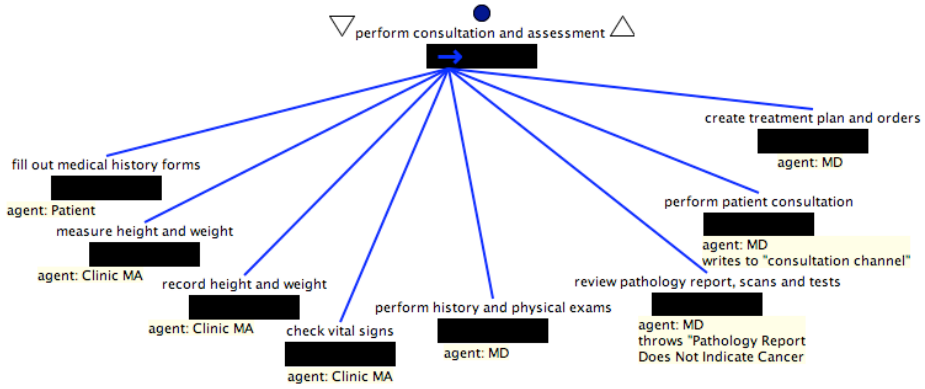

Figure 2: The task decomposition of *perform consultation and assessment*

reviews, consult the patient, create a treatment plan, and enter orders in the system. Figure 2 illustrates the ability of Little-JIL to capture information about the agents (represented as annotations in this figure) who execute the tasks in a process. Figure 2 also demonstrates the use of exceptions to model non-standard scenarios in a medical processes – if the MD discovers that the pathology report does not indicate cancer, the step *review pathology report, scans and results* throws an exception and control is transferred to the matching exception handler *consider alternative treatment*, which was discussed in the context of Figure 1. Finally, Figure 2 shows the use of channels to provide synchronization among steps in a Little-JIL process definition. The step *perform patient consultation* writes a parameter to the "consultation channel" and thus it needs to execute before the step *dictate consult note* (in Figure 1), which reads from the "consultation channel", can start execution.

Figure 3 decomposes the substep *transcribe and place consult note in patient's record* of the root step *chemo process*. Note that the process shown in this diagram provides further details of the handling of the consult note. Figure 1 specifies that *transcribe and place consult note in patient's record* is the second substep of the sequential step *create and process consult note*. 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. Except for needing to wait for the availability of the consult note, this can happen in parallel with the tasks in *prepare for and administer first cycle of chemotherapy*.

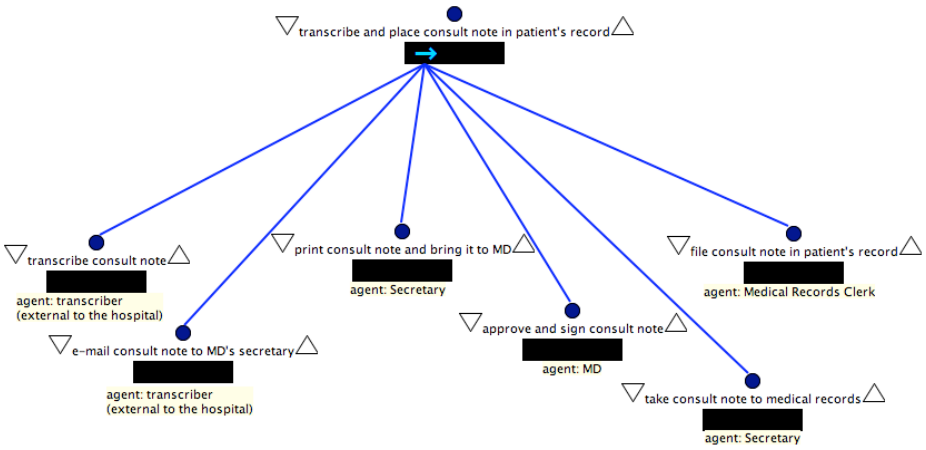

Figure 3: The task decomposition of *transcribe and place consult note in patient's record*

3. Experience

The very task of eliciting details from the medical professionals about the chemotherapy process and capturing those details formally in Little-JIL led to the discovery of several defects in the process. Applying analysis techniques also helped us detect defects. Finite-state verification, for example, was used to determine if specified goals, or properties, are always satisfied on all possible execution paths through the process definition. One of the properties required to hold in the chemotherapy process stated "Before Chemotherapy Can Be Administered to a Patient, that Patient's Consult Note Needs to Be Put in that Patient's Record." This means that the step *administer chemo drug* (which is part of the step *first day of chemo* in Figure 1) cannot be performed until the step *file consult note in patient's record* (shown in Figure 3) has been completed. We modeled this property as a finite-state automaton using the PROPEL system (5) and then used the FLAVERS finite-state verifier (6) to check whether the process satisfies this property. Although a channel imposes some synchronization between the parallel activities in the chemotherapy process, FLAVERS detected that concurrent execution can allow at least one execution sequence that leads to a property violation, i.e. *administer chemo drug* occurs before *file consult note in patient's record* completes. Detailed discussion about analyzing Little-JIL definitions of medical processes is beyond the scope of this paper, but a more comprehensive treatment of the subject is presented in (12).

The discovery of defects led to changes in the chemotherapy process definition to eliminate those defects. For example, we found that some traces through the process definition could bypass a check to see if the patient's height or weight, on which the chemotherapy dose is based, are sufficiently up-to-date. After careful scrutiny it was determined that this defect was not merely a process definition error, but an actual error in the process. The medical professionals then proposed changes in the process definition. The modified process definition was then reanalyzed with respect to all the properties, not just the one that caused this defect. The process and its process definition were subsequently improved so this check always occurred on all possible traces before chemotherapy could be administered. The medical professionals involved in the project found benefit in this process improvement cycle.

One of the observations that became apparent during the early interviews with the medical professionals was that the terminology used to describe the chemotherapy process was sometimes inconsistent. For example, words like "verify", "confirm", and "check" 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 medical process may arise from neglecting small details, we developed a glossary that disambiguated the use of different terms. Thus, our experience suggests that the effort of defining and analyzing complex medical processes can benefit if such a glossary is employed.

We also found that process guidelines usually contain adequate details when describing common, standard scenarios, but do not provide enough details, or often any details, for handling many exceptional cases. For example, there were places in the guidelines where an agent is to confirm the correctness of some information and, if the confirmation succeeds, the agent is to continue with the rest of the defined tasks.

If the confirmation fails, then the guidelines often lack specific instructions detailing how the agent should proceed. In some cases, we observed that different agents were handling the exceptional cases differently. While modeling the process with Little-JIL, the rich exception handling semantics of the language encouraged us to think about exceptional scenarios and to ask specific questions about the process to be executed following the occurrence of an exception, the agents involved in resolving that exception, and the place in the process where control is transferred once the exception has been handled. Questions like "What do you do when the check 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.

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. Doctors, however, 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 distinguish between paper and electronic records led to the discovery of such issues.

Overall, we found that the rich semantics of Little-JIL proved useful for defining the chemotherapy process. The exception handling mechanisms enabled the process definition to reflect the real world process more accurately. Modeling resources (both agent and non-agent) and artifacts were an important part of the specification of the process. The channel synchronization mechanism for specifying direct communication and synchronization among steps was also useful. Hierarchy and abstraction were beneficial in helping to reduce the size of the process definition and in allowing the process to be defined at different levels of abstraction.

Elicitation of the process required almost two semesters of weekly meetings between process developers and medical professionals. In these meetings usually there were two graduate students and at least one faculty member along with two or three medical professionals. The medical professionals comprised different combinations of physicians, pharmacists, nurses, and medical assistants. The graphical notations, as well as the language's constructs supporting abstraction and exception handling, facilitated the communication of computer science concepts to the medical professionals. We usually presented the process to the medical professionals in textual, natural language form, but we were often asked to show the Little-JIL diagrams. Although we believe that it is most likely that the Little-JIL definitions will be written by computer scientists or medical informatics specialists, our experiences suggest that medical professionals, with a little training, can become comfortable reading Little-JIL process definitions.

4. Related Work

The medical informatics community has developed several languages for specifying medical processes (e.g. Asbru (13), EON (14), Glare (15), GLIF (16), PRO*forma* (17)). Similarly to Little-JIL, these languages model medical processes as a collection of tasks and provide support for hierarchical decomposition, decisions, goals, concurrency, and exception handling. Some languages, however, support certain features better than others. For example, Little-JIL separates normal flow from exceptional flow, provides a means to pass information about the exception and its context to exception handlers, and provides various continuation options after an exception has been handled. We found that these language features, intended for specific and articulate support of exception handling, are extremely important when modeling medical processes, since exceptions frequently arise, and their representation in a process definition should be made particularly clear and accessible to medical professionals who are counted upon to validate definitions of their processes. Similarly, Little-JIL also supports abstraction well by supporting parameterized procedure invocation. This language feature likewise adds to the clarity of Little-JIL process definitions, facilitating their comprehensibility. The other languages mentioned above do not seem to provide equivalent semantic richness to facilitate process definition comprehensibility.

Little-JIL and PRO*forma* are general-purpose process modeling languages, whereas, EON and GLIF are designed specifically to model processes from the medical domain. These domain specific languages also provide support for drawing upon domain ontologies. This would be an interesting feature to consider adding to Little-JIL to encourage the consistent use of terminology. In addition, Little-JIL's support for timing is not as strong as that provided by the above languages.

Some of these languages have also been used as the basis for formal analysis. For example, as part of the Protocure II project (18), Asbru (19, 20) has been used with the KIV theorem prover (21) and with the SMV model checker (22) . Glare has been used with SPIN (7). The rigorous semantics of Little-JIL allow for fully automated translation of Little-JIL process definitions to input languages of formal verifiers. We have built tools that automatically translate Little-JIL process definitions to the input representations of FLAVERS (6) and SPIN. These tools have helped us avoid manual translation, which is time-consuming and error-prone. There also is automated support for translating Asbru into the internal representation used by SMV and KIV. We have also developed and used PROPEL (5), which provides natural language support for specifying mathematical properties. Using PROPEL, FLAVERS, and SPIN, we have verified Little-JIL process definitions and discovered errors in real medical processes (12).

Much of the related work in the medical informatics domain has focused on modeling medical guidelines that describe the treatment of a single patient with a particular diagnosis. Risks to patient safety, however, arise not only from errors in such guidelines, but from problems in the processes through which health-care providers actually deliver these treatments by interacting with each other, the patient, and the resources required for care (1). Our work has largely been concerned with modeling and analyzing these organizational, or system, processes, and Little-JIL's support for

abstraction and facilities for specifying agent types, resources, and exceptional behavior have been correspondingly important.

Noumeir has also pursued similar goals using a UML-like notation to define processes (23). Others (e.g., (24)), have viewed medical processes as workflows and have used workflow-like languages to define processes and drive their execution. The models created by these projects seem to be less amenable to formal analysis.

Other approaches to improving medical safety have targeted quality control measures (25), error reporting systems (26), and process automation in laboratory settings (27). In other work, Bayesian belief networks have been used as the basis for discrete event simulations and to guide treatment planning (e.g., (28)).

5. Conclusion

This paper presents some of the benefits that arise from the use of a process definition language to describe medical processes. The Little-JIL process definition language provides a rich set of semantic features. We overviewed some of those features and demonstrated how they could be used in an example chemotherapy administration process. While developing the process definition, a number of serious potential defects in the actual process were detected. This resulted in the process definition being modified and, after careful scrutiny, the corresponding process updated to remove those defects.

Since the process definitions can become quite large and complex, manually reviewing these definitions is not sufficient. Instead, we advocate the use of automated analysis techniques that can help detect defects. We briefly indicated how finite-state verification helped detect process defects in the process definition and in the actual processes.

Finite-state verification supports checking whether a process satisfies certain properties, but it assumes that all agents involved in the process perform their tasks without errors. Other types of analysis, such as fault tree analysis (29), consider what happens if tasks are not done correctly. We have explored automatically generating a fault tree from a Little-JIL process definition and then using the fault tree to identify single points of failure and other vulnerabilities (30). Our studies of delays in a hospital Emergency Department have underscored the potential for resource management and discrete event simulation to improve efficiency in medical processes (31).

This work has shown considerable promise and has suggested extensions in several directions. Further research should provide insights into how process definition and analysis technology can be used to improve medical processes.

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 notwith-

standing 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. NSF, U. S. DOD/Army Research Office, or the U.S. Government.

We gratefully acknowledge the contributions of Sandy Wise, Barbara Lerner, and Aaron Cass, who worked on the development of Little-JIL, of Rachel Cobleigh and Irene Ros, who helped elicit the chemotherapy process and properties, and of Ann Garbecki, Gina Parisi, Sally Irelan, Gary Bessette and other members of the staff of the D'Amour Center for Cancer Care, who graciously donated their time and expertise.

Correspondence to:
Stefan Christov
316 Department of Computer Science
University of Massachusetts at Amherst
Amherst, MA 01003, USA
Tel: +1 413 545 2146
Fax: +1 413 545 1249
E-mail: christov@cs.umass.edu

References

1. Kohn LT, Corrigan JM, Donaldson MS, editors. To Err is Human: Building a Safer Health System. Washington, DC: National Academies Press; 1999.
2. Institute of Medicine. Crossing the Quality Chasm: A New Health System for the 21st Century. Washington D.C.: National Academies Press; 2001.
3. Deming WE. Out of the Crisis. Cambridge: MIT Press; 1982.
4. Shewhart WA. Economic Control of Quality of Manufactured Product: D. Van Nostrand Co.; 1931.
5. Cobleigh RL, Avrunin GS, Clarke LA. User Guidance for Creating Precise and Accessible Property Specifications. ACM Symp on the Foundations of Software Engineering; 2006; Portland, OR 2006. p. 208-18.
6. Dwyer MB, Clarke LA, Cobleigh JM, Naumovich G. Flow Analysis for Verifying Properties of Concurrent Software Systems. ACM Trans on Software Engineering and Methodology. 2004;13(4):359-430.
7. Holzmann GJ. The SPIN Model Checker: Addison-Wesley; 2004.
8. Boose ER, Ellison AM, Osterweil LJ, Clarke L, Podorozhny R, Hadley JL, et al. Ensuring Reliable Datasets for Environmental Models and Forecasts. Ecological Informatics. 2007;2:237-47.
9. Schweik CM, Osterweil LJ, Sondheimer N, Thomas C. Analyzing Processes for E-Government Development: The Emergence of Process Modeling Languages. J of E-Government. 2004;1(4):63-89.
10. Cass AG, Lerner BS, McCall EK, Osterweil LJ, Stanley M. Sutton J, Wise A. Little-JIL/Juliette: A Process Definition Language and Interpreter. Intl Conf on Software Engineering; 2000; Limerick, Ireland; 2000. p. 754-8.
11. Wise A. Little-JIL 1.5 Language Report: Department of Computer Science, U. of Massachusetts, Amherst (UM-CS-2006-51); 2006.

12. Chen B, Avrunin GS, Henneman EA, Clarke LA, Osterweil LJ, Henneman PL. Analyzing Medical Processes. Intl Conf on Software Engineering; 2008 May; Leipzig, Germany; 2008. p. 623-32.
13. Shahar Y, Miksch S, Johnson P. The Asgaard Project: A Task-Specific Framework for the Application and Critiquing of Time-Oriented Clinical Guidelines. Artif Intel in Medicine. 1998:29-51.
14. Tu SW, Musen MA. A Flexible Approach to Guideline Modeling. Am Medical Informatics Assoc Symp; 1999; 1999. p. 420-4.
15. Molino G, Terenziani P, Montani S, A. Bottrighi, Torchio. M. Glare: A Domain-Independent System for Acquiring, Representing and Executing Clinical Guidelines. Am Medical Informatics Association Symp Supplement. 2006.
16. Peleg M, Boxwala A, Ogunyemi O, Zeng Q, Tu S, Lacson R, et al. GLIF3: The Evolution of a Guideline Representation Format. Am Medical Informatics Association Symp. 2000:645–9.
17. Sutton DR, Fox J. The Syntax and Semantics of the PROforma Guideline Modeling Language. J of Am Medical Informatics Association. 2003:433-43.
18. Protocure II. 2006 [cited; Available from: http://www.protocure.org]
19. ten Teije A, Marcos M, Balser M, van Croonenborg J, Duelli C, van Harmelen F, et al. Improving Medical Protocols by Formal Methods. Artif Intel in Medicine. 2006;36(3):193-209.
20. Baumler S, Balser M, Dunets A, Reif W, Schmitt. J. Verification of Medical Guidelines by Model Checking—A Case Study. SPIN 2006, Springer-Verlag LNCS. 2006;3925.
21. Balser M, Reif W, Schellhorn G, Stenzel K, Thums A. Formal System Development with KIV. Fundamental Approaches to Software Engineering, Springer-Verlag LNCS; 2000; 2000. p. 363–6.
22. Cimatti A, Clarke E, Giunchiglia E, Giunchiglia F, Pistore M, Roveri M, et al. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. Computer Aided Verification, Springer-Verlag LNCS. 2002;2404: 359–65.
23. Noumeir R. Radiology Interpretation Process Modeling. J of Biomedical Informatics. 2006;39(2):103-14.
24. Ruffolo M, Curio R, Gallucci L. Process Management in Health Care: A System for Preventing Risks and Medical Errors. Business Process Mgmt; 2005; 2005. p. 334-43.
25. Voak D, Chapman JF, Phillips P. Quality of Transfusion Practice Beyond the Blood Transfusion Laboratory Is Essential to Prevent ABO-Incompatible Death. Transfusion Medicine. 2000;10:95-6.
26. Battles JB, Kaplan HS, van der Schaaf TW, Shea CE. The Attributes of Medical Event Reporting Systems for Transfusion Medicine. Arch Pathology Laboratory Medicine. 1998;122:231-8.
27. Galel SA, Richards CA. Practical Approaches to Improve Laboratory Performance and Transfusion Safety. Am J of Clinical Pathology. 1997;107 (Suppl 1):S43-S9.
28. van der Gaag LC, Renooji S, Witteman CLM, Aleman BMP, Taal BG. Probabilities for a Probabilistic Network: A Case-Study in Oesophageal Cancer. Artif Intel in Medicine. 2002;25(2):123-48.
29. Vesely W, Goldberg F, Roberts N, Haasl D. Fault Tree Handbook Washington, D.C.: U.S. Nuclear Regulatory Commission; 1981 January.
30. Chen B, Avrunin GS, Clarke LA, Osterweil LJ. Automatic Fault Tree Derivation from Little-JIL Process Definitions. 2006 Software Process Workshop and 2006 Process Simulation Workshop; 2006 May 20-22; Shanghai, China: Springer-Verlag LNCS; 2006. p. 150-8.
31. Raunak MS, Osterweil LJ. Effective Resource Allocation for Process Simulation: A Position Paper. Intl Workshop on Software Process Simulation and Modeling; 2005 May 14-15; St. Louis, MO; 2005.