Supporting Human-Intensive Systems

Lori A. Clarke
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003
+1 413-545- 1328
clarke@cs.umass.edu

Leon J. Osterweil
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003
+1 413-545- 2186
ljo@cs.umass.edu

George S. Avrunin
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003
+1 413-545- 0510
avrunin@cs.umass.edu

ABSTRACT
Executing critical systems often rely on humans to make important and sometimes life-critical decisions. As such systems become more complex, the potential for human error to lead to system failures also increases. In the medical domain, for example, sophisticated technology has been introduced in the last decade without adequately considering the impact and role of the medical professionals. This is just one of many domains, where human agents, hardware devices, and software systems must interact with each other, and where humans are expected to make important, and sometime life-critical, decisions. This position paper argues that human-intensive systems should be a major concern of software engineering in the future and describes some of the research issues that need to be addressed.

Categories and Subject Descriptors
D.2 [Software Engineering]: Requirements/Specifications, Design Tools and Techniques, Verification; H.1 [Models and Principles]: User/Machine Systems.

General Terms
Documentation, Design, Verification.

Keywords
Human-intensive systems, process improvement, life-critical systems.

1. INTRODUCTION
Systems where the human contributions require considerable domain expertise and have a significant impact on the success or failure of the overall mission, are referred to here as *Human-Intensive Systems*. We believe that the role of humans in such systems needs to be taken into account as a first class concern. As such, human behavior should be modeled and evaluated during the earliest stages of development, carefully considering the interactions and constraints between humans and their collaborating hardware and software components. Moreover, we believe that development and deployment environments must include support for representing and reasoning about human behavior, for monitoring and guiding human behavior, and for accumulating data about erroneous behaviors, past failures, and near misses. Based on such accumulated data, process improvements can be recommended, requirements and test cases enhanced, and probabilistic analysis updated. Thus, we are proposing a new paradigm for the development and improvement of human-intensive systems that is driven by a detailed understanding and evaluation of the coordination among human agents, software systems, and hardware devices. Since these coordinating process models are software too [11], we believe that the study of the development, evaluation, evolution, and execution of human-intensive systems is an important area for future software engineering research.

FoSER 2010, November 7–8, 2010, Santa Fe, New Mexico, USA.

Numerous systems that are the backbone of our societal infrastructure are human intensive systems. As these systems become more complex, there is increased opportunity for human-errors to lead to serious system failures. Perhaps some of the most prominent examples of this come from the healthcare domain. The news media frequently report on medical mishaps, where lives are lost, healthy organs removed, or other adverse and avoidable negative outcomes occur. An Institute of Medicine report [6] estimated that there were about 98,000 deaths each year in the US due to avoidable medical errors, making this one of the leading causes of death. Although human error is often involved, the real culprits are often the complex processes being applied. As stated in a recent US National Research Council report about healthcare [13]:

> "persistent problems do not reflect incompetence on the part of health care professionals - rather, they are a consequence of the inherent intellectual complexity of health care taken as a whole and a medical care environment that has not been adequately structured to help clinicians avoid mistakes or to systematically improve their decision making and practice."

These findings should not be surprising. Medical processes are excellent examples of rapidly changing, complex systems involving many different types of human agents (e.g., doctors and nurses with different specializations and roles, pharmacists, lab technicians, and support staff), hardware devices (e.g., infusion pumps, radiation therapy machines, and patient monitoring devices), and software systems (e.g., computerized physician order entry systems, decision support systems, and electronic healthcare records). Coordination is particularly key in these systems, as humans are often participating simultaneously in several different processes at any given time, and their participation in each process may entail the parallel performance of many different subtasks as well as interaction with several different hardware and software components. In performing these tasks, it is common for exceptional conditions to arise, requiring specialized actions that may vary considerably depending upon the circumstances. Continual change is also a key issue in

medical processes. Changes may result from the introduction of new devices, new software systems, new personnel, or even personal preferences. New research findings may lead to new guidelines or standards of care. Still other changes come about as reactions to errors encountered in local practice. Regrettably, changes are sometimes made based only on informal analysis of poorly understood processes, leading to ill-advised changes that may slow the delivery of care, fail to address the root cause of the error, or lead to poor outcomes.

In this position paper, our examples are drawn from the healthcare domain, but the problems are prevalent in many domains. For example, there are similar concerns about complex military systems and the recent oil spill in the Gulf coast has been, at least partially, attributed to human errors in a complex, ill-understood, computer-supported process.

2. RESEARCH ISSUES

Some of the issues that need to be addressed to support the development, maintenance, evaluation, and deployment of such human-intensive systems are discussed in the ensuing subsections. Specifically, we look at modeling support to capture the human element in such systems; static analysis approaches that consider this human behavior in their reasoning; requirements engineering approaches that address the boundaries between software, hardware, and human agents; simulation approaches that evaluate effectiveness, and on-line guidance to assist human participants and help reduce human errors.

2.1 Modeling Human-Intensive Processes

The specification of how such a system coordinates human agents, hardware devices, and software components, if it is provided at all, is usually expressed only informally, in natural language documents, in hard-coded interfaces, and in the behavior of the hardware and application software components. Nonetheless, it is only through the coordination of the human participants, hardware devices, and software systems that the overall goals of the system are achieved. This suggests the value of treating coordination as a separate and key part of such systems.

We propose an approach to the development and improvement of human-centered systems in which coordination is explicitly separated out from the other aspects of the system and is precisely modeled in an executable language. Such an executable model of coordination has been called a *process definition* and, in prior work, it has been shown that rigorous analysis of such models can detect defects and vulnerabilities, as well as mismatches with the hardware and software components [2]. Based on such analyses, modifications and improvements in the coordination model, hardware devices, and software components of the system can be proposed and evaluated. In addition, such validated process definitions can be used to drive simulations, train agents, and proactively provide real-time, on-line guidance, thus amortizing the cost of developing and maintaining such models.

The process definition languages used to specify such coordination will need to provide rich semantic capabilities. In addition to the capabilities that programming language research has demonstrated should be available in programming languages, such as support for abstraction, composition, typing, and restricted, well-formed control flow constructs, process languages need to support the rich control models needed for human behavior.

The handling of exceptional situations is one area where such support is needed. Most current workflow and process languages tend to focus on normative flow. Since the response to exceptional situations is the source of most errors, process languages need to provide rich semantic mechanisms for handling the detection and responses to non-normative circumstances [7].

Concurrency is another area where richer semantic models are needed. Since humans often multi-task and want to have flexibility in deciding where to focus their attention, complex concurrency control needs to be provided, including support for selective pre-emption (e.g., a medical emergency pulls doctors away from their current task) and resumption or reassignment.

Resource management is another area of concern. Processes involving humans often have to manage resources, entities that may be under contention. Deciding how best to specify the resources that are available and how to select the one(s) used to respond to a request are challenging issues. A medical process, for instance, may involve the participation of many types of resources, such as devices and people, with each type consisting of instances that have different capabilities, titles, skill levels, etc. Thus, doctors would be viewed as resources, but doctors have many different specialties. Further, different doctors have different skills and skill levels. Complicating matters is the fact that, especially in emergencies, medical professionals may sometimes be allowed to perform certain tasks that would ordinarily require more specialized training, and therefore context must be taken into consideration. Thus, process languages will need to support the declaration of available resources and their usage constraints and, during execution, manage resource requests, and perhaps incorporate resource scheduling and task planning into the execution framework.

All processes that execute in the real world have constraints on the speed with which they must execute. In medicine, for instance, these constraints are particularly important, as failing to meet one may mean the difference between life and death. It is interesting to note that very few process and workflow languages currently incorporate timing constraints, although such capabilities are clearly needed for human-intensive systems.

As is often the case in language design, there needs to be serious consideration given to human understanding. Since process definitions should be reviewed by the domain expert, process definitions or views of those definitions must be comprehensible by non-computer scientists. Unfortunately, there is often a tension between powerful language mechanisms and understandability that needs to be taken into consideration.

2.2 Static Analysis

There has been considerable success lately in using static analysis techniques to analyze hardware systems, software systems, and process definitions (e.g., [3, 5]). We believe that this success can be pushed even further so that mixed models, involving hardware, software, and human process definitions can be evaluated to assure that these various aspects are working together to assure that important safety requirements are met. This will require improvements to compositional approaches as well as to cross model analysis.

In addition, there is considerable synergy forming around using probabilistic model checking approaches and safety analysis techniques, such as Fault Tree Analysis (FTA) and Failures Mode and Effects Analysis (FMEA).

In probabilistic model checking [14], the property specification language is typically extended to allow probabilistic path quantifiers (and sometimes additional features, such as costs) and the underlying model of the system is a Markov chain or Markov decision process. Where a standard model checker might check that a particular fault never occurs, a probabilistic model checker checks that the probability of a fault occurring is less than some value. The techniques currently used extend standard model checking techniques with methods from numerical linear algebra and Markov chains. For human-intensive systems, this checking will need to be expanded to consider contexts, since as noted above, expected responses can often be impacted by the context in which a situation occurs.

In safety engineering, a *hazard* is a state or set of conditions of a system that, together with certain other conditions in the environment of the system, will lead inevitably to an accident causing loss. FTA [15] is a technique used to identify the possible causes of a hazard. Once a fault tree has been constructed, techniques from Boolean algebra can be used to identify *minimal cut sets*, minimal sets of basic and undeveloped events that are sufficient to cause the hazard to occur. If sufficient information about the probabilities and independence of the various faults is available, the tree can also be used to derive quantitative information about the probability of the hazard. For large systems, fault trees can be extremely complex. Further, they typically are constructed by human experts who must identify all possible causes of a fault; if the experts fail to think of a possible cause, the fault tree will be incomplete. One of the advantages of a process definition is that FTA representations can be automatically generated for hazards that can be identified in the process definition. Although omissions or errors in the process model will also lead to incomplete or inaccurate fault trees, careful analysis of the process can help eliminate those problems. Importantly, a single, carefully developed process model can be used to derive many fault trees.

Instead of going from a potential hazard to the causes of a hazard, FMEA [12] traces the impact of an individual failure on the overall system. For each way in which an entity could fail (a *failure mode*), FMEA inductively identifies the resulting entity and system failures that could be caused by that failure mode, using a forward search based on the underlying dependency between entities. As with fault tree information, FMEA tables can be automatically generated from process definitions.

An interesting research direction is to explore how these different analysis approaches can be combined to leverage each other. For example, FMEA analysis could be used to suggest hazards to be subsequently explored via FTA. The FTA analysis could then use the probabilities from probabilistic model checking to predict the probabilities of component failures or the severities of system failures. Model checking could then be applied to determine the violations that might arise if failures occur. Although there has been some work in this direction (e.g., [8-10]), it needs to be further developed and carefully evaluated, especially for real applications involving human participants.

2.3 Generating Requirements

Model checking is concerned with verifying that important properties or requirements hold for any possible trace or execution of a system. Although we may know the overall requirements for a system, it may not be clear what the requirements are for the individual components of that system that must work together to satisfy the overall system requirements. Thus, when using a device in a medical procedure, it might not be fully known how that device must be applied to guarantee that the overall system properties are satisfied.

As a motivating example, consider a “smart” infusion pump responsible for intravenously administering medications. Such pumps employ a drug library to obtain information about the recommended dosage limits. These limits depend on the procedure and location of the pump. For example, larger dosages are usually allowed in operating rooms than in recovery rooms. One important requirement for medical procedures is that a patient is never administered a drug overdose. This might be reflected as a requirement on the infusion pump stating that if a selected dosage is within the dosing range provided by the configured drug library, then the pump administers that dosage; otherwise it reports a dose alert. For hospitals that share pumps among different areas, it is necessary to also include a requirement that the pump must be reconfigured whenever it is moved from one area to another. This requirement might be overlooked unless that usage scenario is explicitly considered.

Although this is a simple example, it is easy to envision situations where user processes might change or vary, where new versions of software are applied, or where different hardware is employed. Although we might expect the overall requirements to remain the same in such situations, do these changes impose new or modified requirements on the human agents, hardware devices, or software components used in that process? And if so, what are those requirements? One area of future work is to explore the generation of component requirements based on the context in which they might be used. This work would most likely build upon recent advances in automatic assumption and interface specification generation techniques which use a combination of model checking and learning algorithms [1, 4].

Returning to the infusion pump example, a learned requirement might be that the pump must be initialized after it is moved. This could be represented in the device by requiring initialization after any movement of the pump, or in the human process by requiring that the user always checks for the appropriate drug library before proceeding, or in both the pump and the process to provide some level of fault tolerance. Areas of future research include exploring how to systematically develop requirements based on process definitions of the users' behaviors and how to better understand the implications of process, device, or software modifications.

2.4 Process Definition Driven Simulations

Simulation languages and engines have been in existence as far back as the 1960s. There is a thriving marketplace of languages and systems aimed at supporting the simulation of various domains, including healthcare. Most of these systems, however, suffer from the lack of flexibility in specifying the process to be simulated, the resources to be used, or both. As a consequence these systems are often difficult to use to explore such questions as how best to design an emergency department process and a resource scheduling approach that complement each other to

produce optimal utilization of resources. Interesting research question are whether process definitions can serve as drivers for discrete event simulation and will the added detail lead to predictions that are more accurate.

In addition to using more detailed and carefully verified models on which to base the simulations, simulation validity depends on the accuracy of the input probabilities. By providing an environment in which execution results can update our understanding of the probability of events, simulation results should be improved. In addition, simulation results can then be used to influence resource allocation decisions and process scheduling during process execution, improving both run time performance and our understanding of runtime performance. As is always the case with simulations, there are also difficult questions to be addressed about simulation validity that are magnified here if simulation results are used to impact runtime performance, which will then be used to impact simulation inputs, and so on.

2.5 Process Guidance

Process guidance can be thought of as the provision of information that helps human agents to be more effective while they are actually participating in the process. Work needs to be done to investigate how to provide information that can help agents to perform their tasks better in real time.

One concern is how to provide participants with a view of the executing process so that they understand the relevant past history, current context, pending tasks, and options available to them. Clearly, this must be done in a way that provides a highly summarized view of this *process state*. This view should be relevant to the participants with respect to their role in the process. For example, a nurse's sphere of concern may be quite different from one for an anesthesiologist. For life critical situations, there should be support for ensuring that participants remain well within the envelop of safety. Determining what this envelop is and when its boundaries are being approached, as well as support for safely returning to its fold, if it is ever breached, are important issues.

In some cases, on-the-fly analysis results might be extremely helpful in making decisions. For example, scheduling and planning analyses might suggest possible advantages or disadvantages of selecting certain alternatives in the performance of a task. In some cases, we anticipate that human participants may be empowered to request such analyses. In other cases, relevant analyses may be implemented as part of the defined process to ensure that the results will be presented to human participants proactively. These cases will require linkages between the analyzers and the execution framework and careful evaluation of when such guidance will be beneficial.

Monitoring process progress and maintaining an accurate representation of process state can be extremely difficult. Participants often do not explicitly indicate when they are done with a task or announce the cognitive activity they undertaking. Although sophisticated capabilities using various kinds of sensors have been proposed for detecting process progress, we expect that most progress can be captured by low-tech means, such as a click on a screen or the entry of data. The ways in which these proactive activities are to be initiated and the ways in which their results are to be presented to participants will undoubtedly have an impact upon the acceptance of process guidance. Thus, these issues need to be studied in cooperation with experts in dealing with human behavior, perhaps drawing upon research in such areas as human factors, human computer interface design, industrial engineering, and psychology.

3. CONCLUSIONS

This position paper calls on software engineering to broaden its focus on software to include interactions with hardware devices as well as human participants. Software now dominates many of the systems that were once primarily hardware based, such as transportation and communications, as well as many of the processes that were once human-centric, such as medicine and banking. Software engineers have long complained that hardware engineers leave software issues to the end, when it is too late to develop the most effective alliance between the two. Now that software is starting to be seen as the dominant component of such systems, we need to develop the appropriate paradigms for designing and building the appropriate alliances with hardware components and human participants.

Here we argue that human participants are often overlooked; yet, for human-intensive systems, humans play a critical role. Systems that are developed without carefully considering how to support their role may be error prone, with possibly catastrophic consequences. To incorporate support for humans, software engineers will need to work with experts in a number of domains. In this paper, we have highlighted some of the areas in software engineering that will need to be extended.

4. ACKNOWLEDGMENTS

We appreciate the many contributions of those involved in the UMASS Medical Safety Project, including computer scientists Ben Chen, Rachel Cobleigh, Huong Phan, M.S. Raunak, Danhua Wang, and Sandy Wise, and medical professionals Lucinda Cassells, David Brown, Elizabeth Henneman, Philip Henneman, and Wilson Mertens.

This material is based upon work supported by the National Science Foundation under Awards CCF-0820198, CCF-0905530 and IIS-0705772, and by a Gift from the Baystate Medical Center, Rays of Hope Foundation. 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 NSF.

5. REFERENCES

[1] Beyer, D., Henzinger, T. and Singh, V., 2007. Algorithms for Interface Synthesis. In *Proceedings of the Computer Aided Verification*, Lecture Notes in Computer Science, Springer Verlag, 4-19.

[2] Chen, B., Avrunin, G.S., Henneman, E.A., Clarke, L.A., Osterweil, L.J. and Henneman, P.L., 2008. Analyzing Medical Processes. In *Proceedings of the 30th International Conference on Software Engineering*, (Leipzig, Germany, May), 623-632.

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

[4] Giannakopoulou, D., Pasareanu, C.S. and Barringer, H., 2002. Assumption Generation for Software Component Verification. In *Proceedings of the 17th International Conference on Automated Software Engineering,*, (Washington, DC), 3-12.

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

[6] Kohn, L.T., Corrigan, J.M. and Donaldson, M.S. (eds.). *To Err Is Human: Building a Safer Health System*. National Academies Press, Washington DC, 1999.

[7] Lerner, B.S., Christov, S., Osterweil, L.J., Bendraou, R., Kannengiesser, U. and Wise, A. 2010. Exception Handling Patterns for Process Modeling. *IEEE Transactions on Software Engineering*, *99* (2010), 162-183.

[8] Leveson, N.G. 1986. Software Safety: What Why and How. *ACM Computing Surveys*, *18* (2. 1986), 125-163.

[9] Leveson, N.G. and Turner, C.S. 1993. An Investigation of the Therac-25 Accidents. *Computer* (1993), 18-41.

[10] Lutz, R., 2000. Software Engineering for Safety: A Roadmap. In *Proceedings of the 22nd International Conference on Software Engineering*, (Limerick, Ireland, 2000), 215-224.

[11] Osterweil, L.J., 1987. Software Processes Are Software, Too. In *Proceedings of the Ninth International Conference on Software Engineering*, (Monterey, CA, March 30-April 2), IEEE Computer Society Press, 2-13.

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

[13] Stead, W.W. and Lin, H.S. 2009. *Computational Technology for Effective Health Care: Immediate Steps and Strategic Directions. Committee on Engaging the Computer. Science Research Community in Health Care Informatics*. National Academies Press, Washington, D.C.

[14] Vardi., M., 1985.Automatic Verification of Probabilistic Concurrent Finite State Programs. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press, 327-338.

[15] Vesely, W., Goldberg, F., Roberts, N. and Haasl, D. Fault Tree Handbook U.S. Nuclear Regulatory Commission, Washington, D.C., 1981.