Considerations for Online Deviation Detection in Medical Processes

Stefan C. Christov
School of Computer Science
University of Massachusetts
Amherst, MA 01003
christov@cs.umass.edu

George S. Avrunin
School of Computer Science
University of Massachusetts
Amherst, MA 01003
avrunin@cs.umass.edu

Lori A. Clarke
School of Computer Science
University of Massachusetts
Amherst, MA 01003
clarke@cs.umass.edu

***Abstract*—Medical errors are a major cause of unnecessary suffering and even death. To address this problem, we are investigating an approach for automatically detecting when an executing process deviates from a set of recommended ways to perform that process. Such deviations could represent errors and, thus, detecting and reporting deviations as they occur could help catch errors before something bad happens. This paper presents the proposed deviation detection approach, identifies some of the major research issues that arise, and discusses strategies to address these issues. A preliminary evaluation is performed by applying the approach to a part of a detailed process model. This model has been developed in an in-depth case study on modeling and analyzing a blood transfusion process.**

I. Introduction

In 1998, an Institute of Medicine (IOM) report estimated that preventable medical errors cause the death of 98,000 people each year in the U.S. [1]. Despite considerable work to reduce errors and their consequences, human errors are still a major concern for many medical processes. More than a decade after this IOM report, a 2009 US National Research Council (NRC) report [2] indicated that the problem with errors still persists and that "it is widely recognized that today's health care ... suffers substantially as a result of medical errors". Moreover it states that "Making environments safer means looking at processes of care to reduce defects in the process or *departures from the way things should have been done*" (italics ours).

To address this problem, we are investigating an approach for automatically detecting when an executing process deviates from a set of recommended ways to perform that process. Such deviations could represent errors and, thus, detecting and reporting deviations as they occur could help catch errors before something bad happens. In complex and time-sensitive medical processes, simply informing process performers that deviations have occurred might not be sufficient for identifying the errors in a timely manner and for deciding how to recover from them before harm is done. Thus, we are also investigating techniques for providing information that could be useful for identifying the errors and deciding how to recover from them.

We adopt the definition of *error* from the IOM report [1]: "Error: Failure of a planned action to be completed as intended (i.e., error of execution) or the use of a wrong plan to achieve an aim (i.e., error of planning)". The proposed deviation detection approach targets primarily planning errors. Examples of planning errors are omitting an activity that should have been done (error of omission) or performing an activity that should not have been done (error of commission).[1]

The deviation detection approach presented in this paper is part of an overall framework for supporting process monitoring, deviation detection, and process guidance [3]. As envisioned for this framework, the steps performed by process performers are recorded in real time, resulting in a sequence of steps that grows as time passes. The recording mechanism is not the focus of this research, but it presents, of course, a difficult research problem on its own. We expect the increasing use of information technology in healthcare to facilitate the automatic recognition and recording of performed steps, hopefully leaving only a small number of steps that need to be manually logged by humans.

A detailed, formal process model is a key component of this framework. The process model captures the different recommended sequences of steps to perform the process. Every time a step is performed, the sequence of steps performed so far is compared to the process model to detect deviations. The detection of deviations could in turn enable the detection of errors before harm is done.

To support such deviation detection, the process model needs to be written in a notation with formal semantics. This facilitates automatically comparing the sequence of performed steps to the recommended sequences of steps specified by the model and applying various kinds of formal analyses, such as model checking and fault-tree analysis, to increase confidence in the validity of the model. The notation used for the process model also needs to have rich semantics that support the representation of complex process behaviors, such as exception handling and concurrency. Exceptional situations and concurrent process execution, which are known to often be the cause of errors [4], occur frequently in medical processes. Catching errors in such circumstances would be facilitated if exception handling and concurrency are adequately represented in the

[1]The distinction between execution and planning errors often blurs when decomposition is taken into account. For instance, the error of executing activity A incorrectly could be represented as a planning error if performing A consists of sequentially performing subactivities B and C, and the reason why A was executed incorrectly is that B was omitted.

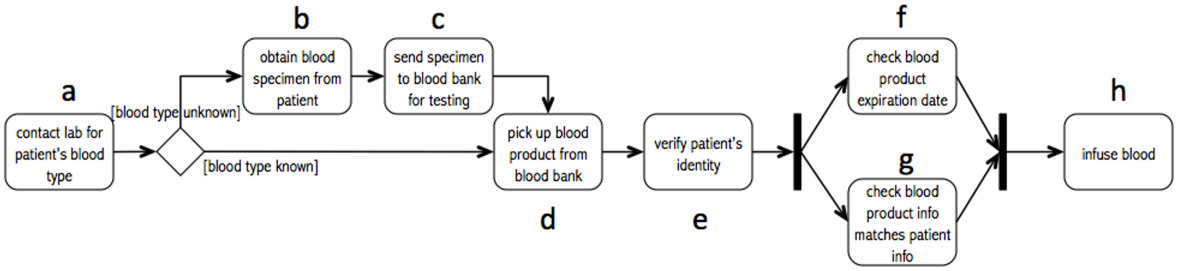


Fig. 1: Simplified blood transfusion process.

process model. In our investigation of the deviation detection approach we use the formal and semantically rich Little-JIL process modeling language [5]. For simplicity and to save space, however, in this paper we only show simple, mostly sequential process flow and thus use a control flow graph notation, based on UML activity diagrams.

In prior work, we constructed Little-JIL models of several medical processes [6]–[8]. We also constructed a framework and a set of tools for analyzing such models to detect various safety problems and evaluate proposed changes (e.g., [8]). We are now beginning to examine the potential for using such models, validated by the analyses, for the monitoring and guidance of executing processes.

There are existing approaches that aim to reduce the number of medical errors or the harm that might result from them by encouraging conformance with some specification of the recommended ways to perform a process (e.g., process aids such as checklists [9]–[11] and care sets [12]). Such process aids, however, tend to specify only the major steps during normal flow, omitting important details such as exceptional scenarios and concurrent process execution [6], [7]. Some of these process aids focus on training, but provide no support during the actual performance of the process. The ones that could be used while a process is being performed, such as checklists, add to the workload of already heavily-burdened medical professionals. The use of checklists, for example, often requires medical professionals to check what needs to be done, to remember what steps they completed, and to decide what the appropriate checklist is to use in a given context.

To remove some of the burdens that process aids, such as checklists, place on medical professionals, there have been attempts to create systems that automatically check the compliance of a process that is being performed with a specification of that process. For example, Fitzgerald et al. designed and deployed a process guidance system in a trauma center to guide medical professionals during the first 30 minutes of trauma resuscitation [13]. This system increased compliance with the underlying medical algorithms and reduced error rate, but these algorithms did not support complex process behaviors, such as concurrency and exception handling. The framework we are proposing, including the work reported in this paper on deviation detection and analysis, is intended to go beyond these limitations.

The rest of the paper is organized as follows. Section II presents an example application of the proposed deviation detection approach in a medical process and section III discusses the approach and some of the issues that arise. Section IV presents some preliminary experimental results before section V summarizes the contributions and future directions.

II. Example

Figure 1 shows a model of a simplified blood transfusion process as a UML activity diagram. The figure focuses on some of the steps performed by the nurse and, in the interest of space, leaves out the tasks of others (e.g., blood bank, physician) as well as most of the exceptional situations that could arise (e.g., the patient does not speak the language or is unconscious).[2] According to this model, to carry out a physician's order for blood transfusion, the nurse needs to first contact the laboratory to check whether the patient's blood type is known. If the blood type is not known, the nurse needs to obtain a blood specimen and send it to the blood bank for testing. Once the patient's blood type is known and the blood bank has prepared the blood product, the nurse can pick up the blood from the blood bank.

After picking up the blood and before infusing it into the patient, the nurse needs to first verify the patient's identity and then verify the blood product to ensure that the correct blood product will be given to the correct patient. Verifying the patient's identity involves asking the patient for identification information, such as name and date of birth, and making sure that this information matches the patient's ID band and the patient chart. Verifying the blood product involves checking its expiration date and checking that the information on the product (patient name, date of birth, and blood type) match the same information on the patient ID band and in the patient chart.

A common error reported in the medical literature, and one that can cause severe harm to patients, is not fully following the procedure for verifying the patient's identity [7]. A possible instance of this error is omitting the step *verify patient's identity* altogether. Consider the situation where in a busy emergency department patient A is wearing the incorrect ID band—that of patient B. Perhaps a registration clerk had to

[2]This information is included in the Little-JIL definition of the blood transfusion process benchmark described in [14] and is necessary if the detection of deviations that occur in such exceptional situations is of interest.

place ID bands on several patients and inadvertently switched the ID bands; or there was a shift change and due to a miscommunication the new clerk placed the ID band for patient B on patient A. Suppose that patient B is the one for whom a blood transfusion was ordered. Since patient A is wearing patient B's ID band, if the nurse does not check the identity of patient A prior to infusing the blood, patient A might receive the blood ordered for patient B. Note that the nurse might still have successfully performed the blood product checks since they are done against the ID band and not against the patient's real identity.

Potential harm as a result of this error might be avoided if the nurse is warned that the process is being performed incorrectly before the infusion is started. One way to achieve this is by comparing the sequence of steps the nurse has performed against the process model. A possible sequence of steps when the nurse forgets to verify the patient's identity is *contact lab for patient's blood type, pick up blood from blood bank, check blood product expiration date*. As soon as the nurse starts the step *check blood product expiration date*, the sequence of steps is no longer a sequence allowed by the model specified in Figure 1[3]. Informing the nurse about such a deviation might help the nurse recover from the error before harm is done (i.e., infusing blood into the wrong patient).

Depending on the level of expertise of the process performer and the complexity of the error, just a warning that an error might have been committed might be sufficient to identify the error and to recover from it. In the example above, it might be fairly easy for an experienced nurse to determine what went wrong. In more complicated situations, perhaps involving a less experienced process performer or involving multiple process performers working concurrently and dealing with exceptional cases, additional information might help determine what the error(s) was and how to recover from it.

For instance, a hypothesis about the location(s) in the sequence of performed steps where the error was committed could be presented. In the current example, the nurse could be told that an error might have occurred when the third step in the sequence, *check blood product expiration date*, was performed. In a more complex situation, the actual error might have occurred earlier than when it was detected (an example is discussed in section III-A). Pointing the process performers to that earlier location in the performed sequence of steps could provide them with the necessary context to determine what the error is and how to recover from it.

In the interest of space, for the rest of this paper we refer to the steps in Figure 1 by using the single letters next to them.

[3]Note that checking the blood product expiration date before verifying the patient's identity might not be problematic by itself. The hospital might have designed the process the way it is shown in Figure 1, however, based on experience that when the blood product checks are done before verifying the patient's identity, the verification of the patient's identity is more likely to be omitted, or for efficiency reasons.

III. Proposed deviation detection and error localization approach

The proposed approach consists of four phases: *deviation detection*, *trace selection*, *alignment computation*, and *error localization*.

The deviation detection phase checks whether the sequence of performed steps has deviated from the recommended ways to perform the process. We define *process trace* (or, for brevity, a *trace)*, as a prefix of a step sequence from the process model. For example, based on the process model in Figure 1, **ade** is a trace. We define a *deviant sequence* (or just a *deviant*) as a step sequence that is not a trace, for example **aed** is a deviant.

The trace selection phase selects a set of traces that are likely candidates for the recommended sequence of steps the process performers had planned to carry out. Intuitively, the more similar a trace is to the deviant, the more likely it is that the process performers planned to perform that trace. The notion of *similarity* between two sequences is essential to the proposed approach. We use the *edit distance* [15] (described in more detail in section III-B) between two sequences as a measure of similarity.

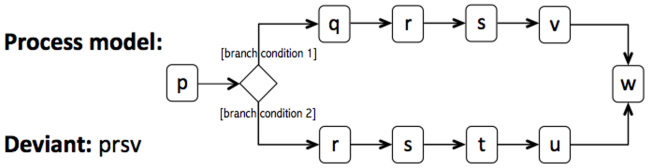
The differences between the deviant and the selected traces could suggest potential planning errors. To identify such differences, the alignment computation phase finds *alignments* (defined in section III-C) between the deviant and each of the selected traces that minimize the edit distance between the deviant and each trace.

Finally, the error localization phase interprets the differences between the deviant and each of the selected traces to hypothesize locations in the sequence of performed steps where an error might have occurred. These locations are ranked based on the edit distances between the deviant and the selected traces.

A. Deviation Detection

Every time a step is performed, the sequence of steps performed so far is compared to the process model to check whether that sequence is a trace or whether it has become a deviant. This check is fairly straightforward. The method that we currently use explores the process model in a breadth-first manner from the start, advancing one step at a time as steps are being performed and recorded. During this exploration, only traces that match the sequence of performed steps so far are kept and if at some moment there are no traces left that match the sequence of performed steps, a deviant is reported.

An interesting issue that arises during deviation detection is the issue of *delayed deviant detection*. In some situations, it is possible for process performers to deviate from the recommended ways to perform a process, but this deviation cannot be detected until a later time. For example, consider the process model in Figure 2(a) and the deviant **prsv**. Suppose that during that particular execution of the process, the process performers planned to carry out the sequence of steps in the top branch but forgot to perform step **q**. In this situation, the deviation has occurred after **p** was performed, but will not be detected until **v** is performed. It will be interesting to investigate how often the problem of delayed deviant



(a) Example process model and deviant.

Process trace	Dist. to deviant
pqr	3
prs	1
pqrs	2

Process trace	Dist. to deviant
prst	1
pqrsv	1
prstu	2

(b) Distances between the deviant **prsv** and some process traces.

p	p
r	r
s	s
-	v

p	p
r	r
s	s
t	v

p	p
q	-
r	r
s	s
v	v

(c) Alignments between the deviant (2^{nd} column) and the closest process traces (1^{st} column).

Fig. 2

detection occurs with realistic process models and sequences of performed steps and also what structures in the process model tend to be associated with this problem. A preliminary investigation of this question is discussed in section IV.

B. Trace Selection

After the sequence of performed steps has been detected to be a deviant, the trace selection phase starts. This phase involves choosing a subset of all the traces specified by the process model. The comparison of the deviant to the traces from this subset will then be used to hypothesize the locations of potential errors. For a simple process model, like the one in Figure 1, it is feasible to obtain all possible traces and compare them to the deviant. A more realistic process model, however, can specify a very large, or even infinite, number of traces and comparing all of them to the deviant may be infeasible. Thus, criteria for selecting a subset of traces are needed.

Based on the intuition given earlier, we expect that traces that are more similar to the deviant will be more useful for obtaining information about errors. We use the *edit distance* between two sequences [15] as the measure of similarity, where the edit distance is a function of the costs of the operations (often called *edit operations*) needed to transform one sequence into the other. Other measures of similarity could be chosen for the purposes of trace selection, but how this choice is made requires further investigation.

Computing edit distance is expensive (the worst case complexity of the algorithms is usually quadratic in the length of the two sequences [15]) and computing the edit distance between the deviant and a large number of traces could certainly be infeasible. There are techniques, however, for computing the edit distance incrementally and discarding a large number of sequences before they need to be compared in full length to the deviant. One such technique is to keep track of traces similar to the sequence of performed steps during the breadth-first exploration of the process model in the deviation detection phase (this technique is used in [16] for process model validation). In addition to traces that match the sequence of performed steps so far (i.e., traces that are exactly the same as the sequence of performed steps), other traces that are within some edit distance of the sequence of performed steps can also be kept under consideration.

There are different kinds of edit distances, depending on the kinds of edit operations allowed. Edit operations could be used

Process trace	Dist. to deviant
a	2
ab	2
ad	1
abc	2

Process trace	Dist. to deviant
ade	1
adef	1
adefh	2

(a) Distances between the deviant **adf** and selected process traces.

a	a
d	d
-	f

a	a
d	d
e	f

a	a
d	d
e	-
f	f

(b) Alignments between the deviant (2^{nd} column) and some process traces (1^{st} column).

Fig. 3: Example application of the deviation detection approach.

to encode different kinds of errors (e.g., deletion of a single step vs. deletion of multiple steps could encode omission of a single step vs. omission of an entire subprocess respectively). Different edit operation costs could be used to represent domain knowledge about errors, such as the likelihood or the severity of an error.

Figure 3(a) shows the edit distances between the deviant **adf** and the traces from the model in Figure 1 within edit distance of 2. In this example, for simplicity, the Levenshtein distance [17] is used as a measure of similarity between the deviant and the process traces. The Levenshtein distance is the minimum sum of the costs of the edit operations needed to transform one sequence into another. The Levenshtein distance is defined in terms of the edit operations deletion, insertion and substitution of a single step. Again, for simplicity, in this example all edit operations have an equal cost of 1.

Deciding what set of edit operations and what associated costs to use depends on factors such as the availability of domain knowledge (e.g., common errors and their frequency) and the richness of the information in the process model. For instance, if it is known that process performers omit subprocess A as frequently as they omit the single step x and the process model contains information (such as hierarchical decomposition) to determine what steps are part of subprocess A, then deletion of the single step x and deletion of all the sub steps of A could be used as edit operations. Furthermore, these edit operations could be given equal cost since the two corresponding errors are known to be equally likely. Empirical evaluation could also be used to choose the set of edit operations and associated costs, but how to make this choice is certainly an issue that requires further research.

C. Alignment Computation

Once a set of traces is selected, each trace in that set is compared to the deviant to examine how that trace differs from the deviant. This could be done by computing *alignments* between each selected trace and the deviant. An alignment of two sequences is a list of ordered pairs (a, b) such that *(i)* a is an element of the first sequence or is the "blank" element "–", *(ii)* b is an element of the second sequence or is "–", *(iii)* the pair $(-, -)$ does not appear in the list, and *(iv)* the order of the non-blank elements in the first and second slots of the pairs in the list is the same as the order of elements in the first and second sequences, respectively. Figure 3(b) shows some alignments. An alignment indicates how one sequence could be transformed into the other, where the blank elements indicate that elements were inserted in one sequence or deleted from the other at the corresponding places.

Optimal alignment(s) (i.e., the alignment(s) that minimize the edit distance between two sequences) are computed by the sequence comparison techniques for computing edit distances [15]. There could be more than one alignment between the same two sequences depending on the choice of edit operations and their associated costs. In fact, there could be more than one optimal alignment between two sequences for a fixed set of edit operations and costs. As mentioned earlier, this choice of edit operations and associated costs depends on various factors, such as domain knowledge, and approaches to make this choice require further research.

D. Error Localization

Once alignments are computed, they can be used to obtain information about the location(s) of potential error(s). We define a *potential error index (PEI)* as an index in the sequence of performed steps, such that the performance of the step at that index is suspected to be an error. The intuition behind using alignments to obtain PEIs is that non-matching alignment pairs (such as the shaded pairs in Figure 3(b)) may represent locations where an error has been committed.

For example, based on the three alignments in Figure 3(b), it could be hypothesized that the nurse adhered to the recommended ways to perform the process while performing the first two steps, **a** and **d**, but not after that. Thus, 3 could be a PEI as the nurse might have committed an error when performing the third step. This seems to be a reasonable hypothesis in this example as the nurse performed **f** (*check blood product expiration date*) when the nurse should have performed **e** (*verify patient's identity*) instead, which would have kept the sequence of performed steps a process trace.

An alignment can have more than one non-matching alignment pair and, thus, more than one PEI could be identified based on these pairs. A strategy that we currently use is to identify a single PEI per alignment. This PEI is based on the first non-matching alignment pair. One reason for this strategy is the assumption that if the first non-matching alignment pair represents the location of deviation, then subsequent non-matching alignment pairs might be less informative about possible errors. This is because after the deviation, the process performers might not have "returned to" the trace in the alignment under consideration, assuming this was the trace they were following before the deviation. Thus, comparing the suffixes of the deviant and the trace after the first non-matching alignment pair might not reveal useful error information. Deciding how PEIs should be identified from an alignment is subject to further research.

	Del.	Ins.	Subst.
Number of generated mutants	1762	1926	1926
Percentage of mutants where deviation is detected after the mutation index	11.5%	0.9%	0.9%
Average deviation detection delay (number of steps)	1.5	1.4	1.0
Minimum deviation detection delay (number of steps)	1	1	1
Maximum deviation detection delay (number if steps)	2	3	1
Percentage of mutants with single PEI	74.3%	97.5%	98.9%
Number of mutants with single PEI that does not match the mutation index	0	0	0
Percentage of mutants with multiple PEIs	25.7%	2.5%	1.0%
Number of mutants with multiple PEIs that do not contain the mutation index	0	0	0
Average number of PEIs for mutants with multiple PEIs	2.1	2.0	2.0
Maximum number of PEIs for mutants with multiple PEIs	3	2	3

Fig. 4: Results from the preliminary experiment. The averages for deletion and insertion mutants are obtained over 10 replications of the experiment.

Given that the set of selected traces could be large (especially for a realistic process model and deviant), that there could be multiple alignments between each selected trace and the deviant, and that there could be more than one PEI per alignment, the number of PEIs could be large. Providing all PEIs to process performers upon deviation detection, however, could be overwhelming rather than useful. Thus, a strategy may be needed to rank the possible PEIs in terms of usefulness for error localization.

The ranking strategy we currently use is based on the edit distance between the deviant and each of the selected traces. A PEI is ranked according to the minimum edit distance to a trace with an alignment suggesting that PEI. Using this PEI ranking strategy and the strategy discussed above for identifying a PEI from an alignment, 3 would be the single top-ranked PEI in the sequence of performed steps **adf** in the example in Figure 3.

In general, there could be multiple top-ranked PEIs, however. For example, in Figure 2 the alignments based on aligning the deviant to the closest traces suggest two PEIs—2 and 4. It will be interesting to explore how often there are multiple equally top-ranked PEIs and how many PEIs there are in such cases in realistic process models and sequences of performed steps. We are currently investigating this issue, which is also related to the issue of delayed deviant detection since multiple highly-ranked PEIs seem to arise in situations where the deviant is detected with delay (as is the case in Figure 2). A preliminary investigation of this question is discussed in section IV.

IV. Preliminary Experiment

To perform an initial evaluation of the proposed deviation detection approach, we applied it to a model of a blood transfusion process and deviants created by mutating traces from that model.

A. Experiment Description

We used the Little-JIL process model from the blood transfusion benchmark [14], but replaced each of the *verify patient's identity* and *specimen labeling* subprocesses by a single step to make the model smaller. The resulting model was still of significant size. It contained 144 Little-JIL steps (53 leaf steps) and it specified 18 exception handling situations. All traces consisting of up to 15 leaf steps (a total of 164 such traces) were generated[4]. These generated traces were then mutated to represent sequences of performed steps where the process performers have deviated from the recommended ways to perform the process.

In this preliminary experiment, we used three kinds of mutations to represent simple errors of omission, commission, and substitution: deletion, insertion, and substitution, each of a single step. Errors, however, could also involve multiple steps, such as the omission or commission of sequences of steps, or even entire subprocesses. Thus, future evaluation will need to incorporate more complex mutations to represent such errors.

In this experiment, a deletion mutant was created by deleting a step from a trace. A step from every index (except the last index) was deleted from each of the original 164 traces. A deletion of the last step in a trace was not performed because that would not make the trace a deviant. An insertion mutant was created by inserting a step, chosen uniformly and at random from all leaf steps, into a trace. A step was inserted between every two steps (including before the first step, but not after the last step) in each of the original traces, unless the inserted step was the same as the step before which it is to be inserted (because, in this case, the resulting mutant would always exhibit the delayed deviant detection issue). A substitution mutant was created by substituting a step in a trace with another step chosen uniformly and at random from all leaf steps. A substitution mutation was done at every index of each of the original traces. For all three mutation kinds, mutants that remained traces were discarded.

The deviation detection approach was then applied to each mutant and statistics related to delayed deviant detection and PEI determination were collected. The results are shown in Figure 4. A deviation detection *delay* is defined as the number of steps between the index in the sequence of steps where the mutation was done and the index where that sequence was recognized to be a deviant. The breadth-first method mentioned in section III-A was used for deviation detection.

To compute PEIs, all traces from the process model up to 17 steps long were selected to be compared against the mutants. The Levenshtein distance was used as a sequence similarity measure. The edit operations were deletion, insertion, and substitution of a single step with equal cost of 1. The PEIs for each mutant were based on the alignments between that mutant and the traces that have minimum distance to it. For each such alignment, a single PEI was produced based on the first non-matching alignment pair.

[4]Only leaf steps were included in traces because in Little-JIL, the leaf steps are the ones that agents perform and are thus most likely the steps to be recorded. Non-leaf steps are used primarily to provide abstraction and specify control flow among leaf steps in a Little-JIL process model.

B. Discussion

Delayed deviant detection occurred rarely for insertion and substitution mutants—for less than 1% of the mutants the deviation was detected after the mutation index. The main reason for the delay was the presence of what we call *shuffle regions*, which are subsequences in the original traces where steps are allowed to occur in any order based on the process model. For example, the fork-join structure in the model in Figure 1 would be responsible for shuffle regions in traces from that model, because the steps **f** and **g** are allowed to occur in any order with respect to each other. When a step from such a shuffle region is inserted or substituted in that same region in a trace, the resulting mutant sometimes cannot be recognized as a deviant until several steps into the shuffle region, because the step that was inserted or substituted in, followed by several of the other steps in the shuffle region often are allowed to occur in that order based on the model.

Shuffle regions cause even more cases of delayed deviant detection for deletion mutants. When a step is deleted from a shuffle region (except for the last step in that region), the deviation is detected with delay, because the steps remaining in the shuffle region are allowed to occur before the deleted step. Given that deletion mutants are created by deleting a step at every index of each of the original traces, every time a step is deleted from a shuffle region (except for the last step in such a region), the resulting mutant is detected as deviant *after* the index of mutation. Insertion and deletion mutants are also generated by inserting/substituting at every index of each of the original traces, but a step from a shuffle region is inserted/substituted into that same region less often since steps are chosen uniformly at random from the set of all leaf steps. This is the main reason the deviation is detected with delay less often for insertion and substitution mutants than it is for deletion mutants.

Branching in the process model was also the cause for delayed deviant detection. For example, given the model in Figure 1, if the trace **ade** is mutated by inserting b after the **a**, so that the mutant is **abde**, then the deviation will not be detected until **d**, which is one index after the index of mutation.

For all three kinds of mutants, the deviation detection delay was small—at most 3 steps after the mutation index and less than 1.5 steps on average.

For most of the insertion and deletion mutants (more than 97.5%), a single top-ranked PEI was identified. For almost 75% of the deletion mutants a single top-ranked PEI was identified. The percentage for deletion mutants is lower, again, mostly due to mutations in shuffle regions. For all three kinds of mutants with a single top-ranked PEI, that PEI was the same as the mutation index.

In the cases when there were more than one top-ranked PEIs, the number of PEIs was small—maximum 3 and less than 2.1 on average—and the mutation index was always among the top-ranked PEIs.

V. Conclusion and future work

This paper outlines an approach for online deviation detection that can be used within a larger framework for process monitoring, deviation detection and process guidance. The goal of the proposed approach is to support performers of medical processes by detecting errors before harm is done as a result of these errors. Major research issues with this approach are identified and potential strategies to tackle them are discussed. A preliminary evaluation of the approach is performed by applying it to part of the blood transfusion process included in a benchmark for evaluating software engineering techniques for improving medical processes.

In the future, we plan to pursue the research issues discussed in section III. In particular, we will investigate strategies for selecting a set of traces to be compared against the deviant upon deviant detection. This selection influences both the efficiency of the approach as well as its usefulness for error localization. We plan to leverage information from applying static analysis to the process models. For instance, information about process structure, such as forward and backward dominator steps, could be used to focus the search for traces to be compared against the deviant.

We plan to evaluate the proposed deviation detection approach with larger and more realistic models and to use real sequences of performed steps from medical processes. We plan to investigate how often and under what circumstances the issue of delayed deviant detection arises as well as the usefulness of PEIs for error localization. We are currently considering other kinds of information that could help process performers localize and identify potential errors. For example, if process performers are comfortable reading some representation of the process model, then, upon deviation detection, they could be given potential locations in the process model where an error might have occurred. Unlike PEIs, which are locations in the sequence of performed steps, locations in the process model can also provide information about the recommended ways to perform the process, which could further facilitate process performers with identifying errors and deciding how to recover from them.

Even though the preliminary investigation of the proposed process guidance approach seems promising, there are many research issues that need to be tackled before the approach can be applied in clinical settings. In addition to the deviation detection issues described in this paper, the success of the proposed approach also depends on other aspects of the overall framework, such as the quality of the sequence of performed steps captured by the recording mechanism. Furthermore, even if deviations and the location of errors that cause the deviations can be identified with great accuracy, there are significant human factors issues that need to be addressed, such as when and how information about potential errors should be presented to process performers. We believe, however, that the use of a detailed process model in a formal and semantically-rich notation combined with the increasing use of electronic devices that can capture performed steps in real-time, could enable the proposed deviation detection approach to address some of the limitations of current process aids in terms of detecting medical errors before harm is done.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Award(s) IIS-1239334 and CNS-1258588. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

The authors gratefully acknowledge the contributions of Lee Osterweil, Heather Conboy, Elizabeth Henneman, and Jenna Marquard.

References

[1] L. T. Kohn, J. M. Corrigan, and M. S. Donaldson, Eds., *To Err is Human: Building a Safer Health System*. Washington DC: Nat. Acad. Press, 1999.

[2] W. W. Stead and H. S. Lin, Eds., *Computational Technology for Effective Health Care: Immediate Steps and Strategic Directions*. Nat. Acad. Press, 2009.

[3] G. Avrunin, L. Clarke, L. Osterweil, et al., "Smart checklists for human-intensive medical systems," in *42nd Intl. Conf. on Dependable Systems and Networks (DSN-W), Workshop on Open, Resilient, Human-aware, Cyber-physical Systems, IEEE/IFIP*, 2012, pp. 1–6.

[4] N. Leveson, *Safeware: System Safety and Computers*. Addison-Wesley, 1995.

[5] A. G. Cass, B. S. Lerner, J. Stanley M. Sutton, et al., "Little-JIL/Juliette: a process definition language and interpreter," in *ICSE '00: Proceedings of the 22nd Intl. Conf. on Softw. Eng.*. ACM, 2000, pp. 754–757.

[6] B. Chen, G. S. Avrunin, E. A. Henneman, et al., "Analyzing medical processes," in *ICSE '08: Proceedings of the 30th Intl. Conf. on Softw. Eng*. ACM, 2008, pp. 623–632.

[7] E. A. Henneman, G. S. Avrunin, L. A. Clarke, et al., "Increasing patient safety and efficiency in transfusion therapy using formal process definitions," *Transfusion Medicine Review*, vol. 21, no. 1, pp. 49–57, 2007.

[8] G. S. Avrunin, L. A. Clarke, L. J. Osterweil, et al., "Experience modeling and analyzing medical processes: Umass/Baystate medical safety project overview," in *Proceedings of the 1st ACM Intl. Health Informatics Symposium*, ACM, 2010, pp. 316–325.

[9] B. M. Hales and P. J. Pronovost, "The checklist: a tool for error management and performance improvement," *Journal of Critical Care*, vol. 21, pp. 231–235, 2006.

[10] J. M. Wilkinson and K. V. Leuven. Procedure checklist for administering a blood transfusion. [Online]. Available: http://davisplus.fadavis.com/wilkinson/Procedure_Checklists/PC_Ch36-01.doc

[11] World Health Organization, "Surgical safety checklist," 2008. [Online]. Available: http://www.who.int/patientsafety/safesurgery/tools_resources/SSSL_Checklist_finalJun08.pdf

[12] W. C. Mertens, D. E. Brown, R. Parisi, et al., "Detection, classification, and correction of defective chemotherapy orders through nursing and pharmacy oversight," *Journal of Patient Safety*, vol. 4, no. 3, pp. 195–200, 2008.

[13] M. Fitzgerald, P. Cameron, C. Mackenzie, et al., "Trauma resuscitation errors and computer-assisted decision support," *Archives of Surgery*, vol. 146, no. 2, pp. 218–225, 2011.

[14] S. C. Christov, G. S. Avrunin, L. A. Clarke, et al., "A benchmark for evaluating software engineering techniques for improving medical processes," in *Proceedings of the 2010 ICSE Workshop on Softw. Eng. in Health Care*, ACM, 2010, pp. 50–56.

[15] D. Sankoff and J. Kruskal, *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison*. Addison-Wesley Pub. Co., Advanced Book Program, 1983.

[16] J. E. Cook and A. L. Wolf, "Software process validation: quantitatively measuring the correspondence of a process to a model," *ACM Transactions on Softw. Eng. and Methodology*, vol. 8, pp. 147–176, 1999.

[17] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Doklady Akademii Nauk SSSR*, vol. 163, no. 4, pp. 845–848, 1965.