Analyzing Regulatory Conformance in Medical Research Systems Using Multi-Paradigm Modeling*

Phillipa Bennett[1], Wuliang Sun[1], Ted Wade[2], Geri Georg[1], Indrakshi Ray[1], Michael Kahn[3]

[1]Computer Science Department, Colorado State University, Fort Collins, CO, USA
[bennettp | sunwl | iray | georg] @cs.colostate.edu

[2]Biostatistics and Bioinformatics, National Jewish Health, Denver, CO, USA
WadeT@njhealth.org

[3]Research Informatics, Children's Hospital Colorado Research Institute Aurora, Colorado
Michael.Kahn@ucdenver.edu

Abstract. Many of today's partially or fully automated medical research systems are rule, or constraint-driven systems that must be analyzed with respect to their rule compliance. We propose a new method called Multi Modeling Constraint-Driven System Analysis to explore and analyze rule conformance in these systems. Specifically, our method takes advantage of existing system documentation in the form of operational flow charts, and refines and transforms the information in them to into more formal system representations that allow automated analysis and reasoning, including reasoning about rule conformance. The method uses UML, Alloy, and protocol state machine models, and includes trace linking between informal and formal diagram and model elements, to allow analysis reporting on more informal diagrams and also 'what if' exploration to address atypical settings or problems found during analysis. Sometimes models are large and not amenable to automated analysis so we demonstrate how slicing techniques can be used to extract subset models relevant to particular analyses. We demonstrate our method using an existing medical research system.

Keywords: *Index Terms*—Multi-Paradigm Modeling, Medical Research Systems, Rule Conformance Analysis, Formal Models.

1 Introduction

Many of today's partially or fully automated medical research systems are subject to a multitude of regulatory requirements, resulting in complex constraint-driven systems. One way these systems can be abstractly described is in terms of the actions that need to be taken and decisions that must be made to accomplish various end user goals. For example, in systems falling under regulatory constraints, a specific set of actions and decisions must occur to achieve an end user goal such as accessing a particular medical record or performing analytical studies across a large set of patient data in a compliant

* This work is based on research sponsored in part by NSF under award number CCF-1018711, by NIST under agreement number 70NANB14H264, an internal grant from Colorado State University, and funding from Dr. Wade and Dr. Kahn. Dr. Wade was supported by The Colorado Clinical and Translational Science Institute, grant UL1RR025780 from NIH.

manner [4]. The set of actions and decisions needed to achieve such a goal comprises a flow of work through the system.

Regulatory compliance is a dynamic process: regulations and institutional policies may change over time or system functionality may be enhanced and become subject to new regulations. In order to demonstrate that the system enforces rules and to provide compliance evidence, both the actual flow that is followed through the system and the values of data along the goal path and when the goal is reached may be important. However, we have found that system documentation may not reflect the constraint enforcement details required for these compliance assurances, and instead may adopt an operational view of system functions. One such representation is an abstract flow chart that only includes the users' most common and useful work flows.

This paper describes a multi-modeling approach we call Multi Modeling Constraint-Driven System Analysis (MMCDSA) that takes advantage of existing system documentation in the form of operational flow charts, and refines and transforms the information in them to more formal system representations that allow automated analysis and reasoning. The method uses UML class and activity diagrams, Alloy models, and protocol state machine models. We use UML as it is the de facto specification language used in the software industry. Moreover, most domain experts use flow charts and E-R diagrams in their system which are similar to UML activity diagrams and UML class diagrams. The motivation for using Alloy is that the Alloy Analyzer allows us to explore predicates specifying desired or undesired system behavior. We use protocol state machines because they express valid interaction sequences and their additional formal semantics help us verify system properties of interest.

We add details to the operational information in flow charts to create UML activity diagrams and related UML class diagrams, and from these we construct an intermediate annotated activity diagram which is transformed into a protocol state machine. We use the state machine to reason about paths through the system and about properties that need to exist along paths, for instance, all paths ending at some particular state must pass through a specific sequence of prior states. We use a slicing technique to subset the annotated activity diagram and associated class diagram to semi-automatically create Alloy models. A series of Alloy models, each representing a different operational view and associated system state, can be used to reason about details related to a particular analysis. Our method also includes trace linking between informal and formal diagrams and model elements, which allows analysis reporting on more informal diagrams, and also 'what if' exploration to address problems found during analysis.

Our research differs from existing approaches for analyzing rule conformance in several ways. First, conformance is often tied to auditing existing implementations such as event logs and searching for either desired or undesired events [8]. Our method uses various models of a system, and while our example system is an existing implementation, our method can be used during system development with models at higher levels of system abstraction, such as requirements or design. Second, conformance analysis using models is most often accomplished using model checking for path property analysis [2, 5]. Model checking requires abstract models in order to avoid state explosion. As a result, the detail needed to actually demonstrate rule conformance may be lost. We include this detail in our models, and use slicing to extract model elements needed for specific analyses in order to create Alloy models. We can therefore analyze a system with respect to rules that require detailed system information.

Third, our focus is on automation as far as possible, in order to achieve repeatable results quickly and to explore as many situations as possible efficiently. This is in contrast to approaches that are based on flow charts and checklists that require human expertise. One such approach uses goal modeling to guide human expert system analysis of audit data [6]. Finally, by maintaining trace links across models we expect that analysis results can be presented on the related portions of more abstract models and diagrams. This may help system administrators to more easily determine the effect that any necessary system modifications will have on end users. The links will also allow administrators to explore "what if" scenarios by changing abstract models to, e.g. disallow unwanted behavior, and by propagating changes to the more formal models and re-running analyses, to verify that the changes have the desired effects.

We use an example medical research system in this paper to demonstrate our method and show how the results of model checking and Alloy-based formal analysis can allow reasoning about regulatory compliance. We also show how analysis can identify unexpected potential operational situations, so that they can be dealt with in a proactive manner. The rest of this paper is structured as follows. Section 2 describes our method to create formal models, and how we can analyze them. Section 3 demonstrates our method using an example drawn from an existing medical research system. Section 4 discusses our results to date and plans for future work.

2 The MMCDSA Method

Creating Formalisms. While there may be informal, abstract flow chart representations of a constraint-driven system that are created as documentation for a variety of users, these often do not follow any particular syntax, and are necessarily incomplete since they only document the flows of immediate interest to users. However, the paths documented are important in the system. Decisions along the flows are based on human goals or on the constraints that must be applied by the system as those goals are achieved. Flow chart decisions allow us to identify the data in the system that is used to make a decision, and by the location of the decision in the flow we can determine the necessary state of that data prior to the decision and how it is changed. We can use this information to specify a partial structural view of the system and its data, but again, it is incomplete since the flow chart does not cover all aspects of the system.

We must add information to these representations in order to formalize them. We do this through the following steps. In every step we model the system from the view of the decisions that occur, since these decisions reflect rule enforcement and ultimately rule conformance. We perform these steps manually for the example in this paper, and plan to use existing approaches and automations wherever possible in our on-going research.

1. Add determinism to the flow chart, and then to use this detail to make the structural view more complete. This step is very interactive with the system designers and administrators. We have chosen to create a UML activity diagram that represents the deterministic flow chart, and a UML class diagram that represents the corresponding system structural view.
2. Refine the activity diagram so that it is syntactically and semantically correct. This involves transforming portions of the activity diagram to include concurrent activities, activities that are nested, etc. These changes may in turn require modifications to the class diagram.

3. Annotate the activity diagram with information from the class diagram about the data used in each activity. The annotations take the form of OCL pre- and post-conditions to operations that actually perform an activity.
4. Transform the annotated activity diagram into a protocol state machine [3]. The system activities and decisions are included as part of the state transition notation so that we can use paths to trace decisions made as part of rule enforcement.

Analyzing Models. We use the protocol state machine, annotated activity diagram, and class diagram to perform a variety of analyses. For example, to explore rule enforcement, we can use the decisions shown in state transition guards to define a path of states that enforce a rule. We can then analyze the complete state machine with a model checker to find out if there are any alternate paths to a final state that should include this path but do not. However, model checkers can fail if the search space is too large, so it is important to restrict a state machine analysis to only the states required to explore some particular system property of interest, and to restrict the system data that needs to be considered during the analysis. Similar to other researchers [2, 5], we restrict models by removing states and data not essential to the analysis, simplifying the analysis, and abstracting states and/or data. However, since many rules are enforced through specific data manipulation along a specific path through system, we may need to show that some data of interest changes appropriately as the system moves through a particular set of states. Techniques other than simplification and abstraction may be needed in these cases. One such technique is slicing, which partitions a model based on user-specified criteria.

The slicing technique we propose uses patterns as the slicing criteria to describe the elements of interest and thus determine the elements that are included in slices. This reduces the size of the model and makes the analysis more efficient [1, 10]. Model slicing techniques typically proceed in two steps. First, the dependency between model elements satisfying a slicing criterion and the rest of the model is analyzed using heuristics related to a model's properties such as its structure. Second, a fragment of the model is extracted, which consists only of elements satisfying the slicing criterion and their dependent elements. We use OCL pre- and post-conditions from the annotated activity diagram to specify the slicing criteria.

We use slice results to semi-automatically generate an Alloy model [7], which then is used by the Alloy Analyzer to generate instances of the system according to predicates also defined in the model. These instances are used to find counterexamples of properties such as discovering whether system data indicating that a rule has been enforced can also exist when the rule has not been enforced. An example of our proposed method is given in the following section. We also demonstrate how instance generation can be used to discover situations where particular paths are followed and the final system state conforms to the specification, but where policy problems may still exist.

3 MMCDSA Example

The example system we present is a medical research system that consists of a data warehouse with health information from multiple sources. The institution has also developed an honest broker service (*HBS*) to provide medical information stored in clinical databases outside of the warehousing system. In these cases someone outside of the HB organization (a data collector, *DC*) must actually extract the information from the clinical database and provide it to the HBS for further processing. In many cases this

means de-identifying (*De-id'd*, or *DeID* in the figures below) the data prior to delivering it to the researchers who requested it, per HIPAA (Health Insurance Portability and Accountability Act) rules. The HBS requires that the person who actually retrieves the data from the clinical database comply with a data collector policy. The policy has several provisions, one of which is that the data collector cannot be directly or indirectly supervised by a member of the research team.

Initial Activity Diagram. Figure 1 shows a portion of an activity diagram that describes the system for researchers, in particular the portion dealing with requests for de-identified data that is not catalogued in the warehouse, but instead resides in an external clinical database. The activity diagram adds more detail than exists in the original flow chart by making each decision and all of its possible results explicit.

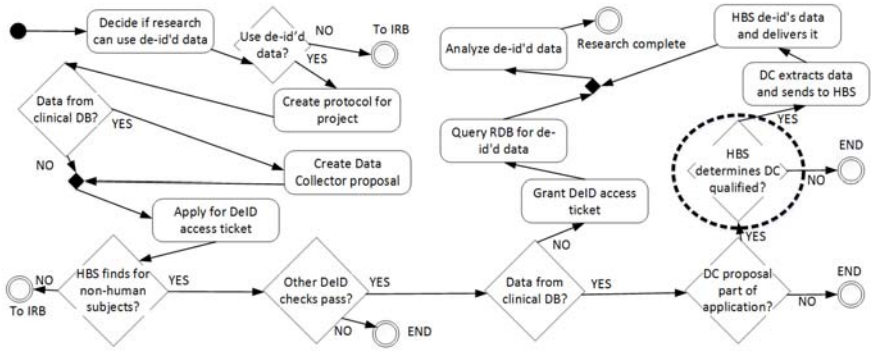

Fig. 1. Portion of activity diagram dealing with non-warehouse databases.

Our example will focus on the decision of whether the proposed data collector is qualified. This decision is circled with a thick dotted line in Figure 1.

UML Class Diagram. Using the diagram in Figure 1, we refined our initial UML class diagram that was based on the organization's initial flow chart. The portion of the class diagram that deals in particular with the decision of whether or not a proposed data collector is qualified is shown in Figure 2. A Certified Honest Broker (*CertifiedHB*) makes the decision on a data collector proposal (*DataCollProposal*). A data collector proposal is associated with an Application that is related to a Protocol for the Project. The Protocol references the data sources (*DataSource*) that will be used to obtain data. These can be databases in the warehouse or external databases (*ClinicalDB*) that require a data collector to actually extract the data. The *CertifiedHB* uses two Rules to make the decision, '*ProjStaffNotDCSupervisor*' and '*DCAuthorizedForClinicalDB*'. The first rule ensures that no one on the research project team can directly or indirectly supervise the proposed DC, and the second ensures that the proposed DC is authorized to extract data from the clinical database. The supervisor policy is enforced by making sure that the transitive set of the DC's supervisors (*Personnel* class, *supervisor* role) does not include any members of the project team associated with the proposal (*Researcher* class, *projMember* role).

Once a DC is qualified an association is created between the DC and the project (*Personnel* class, *dcCollector* role). There are several system invariants that are not shown in Figure 2, three of which are relevant to this example: (1) there may only be one Director object, and that is the only object of type Personnel without a supervisor, (2) no object of type Personnel can supervise itself, and (3) for all objects of type Personnel other than the Director, the supervisor association must include the Director (in fact, if the supervisor association is viewed as a tree, the Director must be the root).

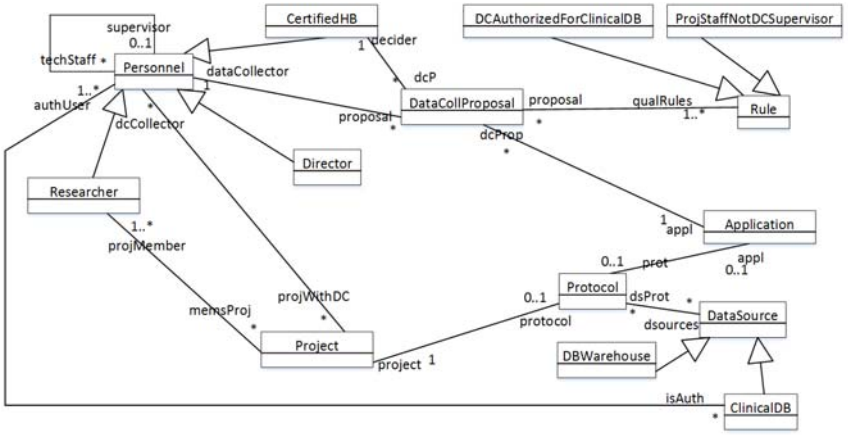

Fig. 2. Portion of class diagram associated with non-warehouse databases.

Annotated Activity Diagram. Figure 3 shows a portion of the annotated activity diagram for the example system. The figure only shows the specific activities and decisions that are needed to decide the qualification of the proposed data collector for a ClinicalDB data source. The activities (rounded rectangles) are labeled A1, A13, and A14. The decisions are labeled D1, D10, and D11.

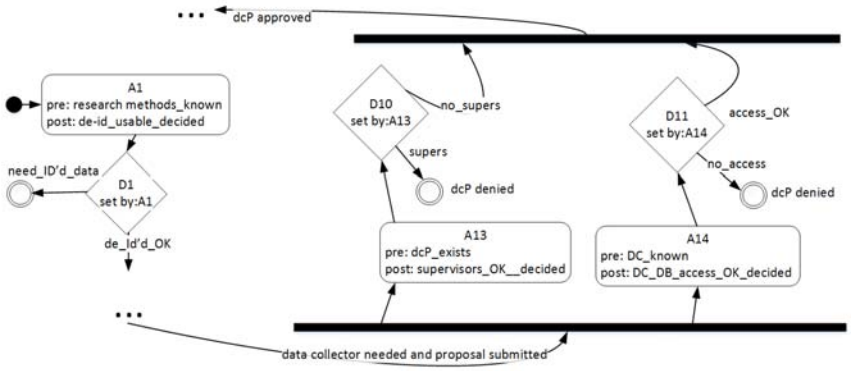

Fig. 3. Portion of annotated activity diagram showing data collector decision.

The solid circle on the left represents the start of the activity diagram, and basic flow through the diagram is shown with directed arrows. Double lined empty circles represent final conditions, and the 3 ellipsis (**…**) represent activities and decisions that have been removed from Figure 3. Each activity element shows the pre- and post-conditions for the activity, and each decision element shows the activity that sets it. The two horizontal thick solid lines represent fork and join points in the activity diagram. There is a fork line at the bottom of the figure, from which activities A13 and A14 execute concurrently. The join line shown at the top of the figure is where their approval results are collected and the activity diagram proceeds. Note that for the proposal to be approved both activities must approve it.

Activity A1 and decision D1 represent the first activity and decision shown in Figure 1. Activity A13 and its related decision D10 apply the Rule '*ProjStaffNotDCSupervisor*' from the class diagram shown in Figure 2. The decision is either that no one on the project team either directly or indirectly supervises the proposed DC (*no_supers* arrow from D10), or that there is some supervision relation (*supers*, leading to the final state of the DC proposal being denied). Similarly, activity A14 and decision D11 apply the Rule '*DCAuthorizedForClinicalDB*'.

Protocol State Machine. We transform the annotated activity diagram into a protocol state machine where the state transition notation is the pre-condition of the related activity pre-condition and any required decision value, the activity operation, and the activity post-condition. A portion of the protocol state machine for the example system is shown in Figure 4.

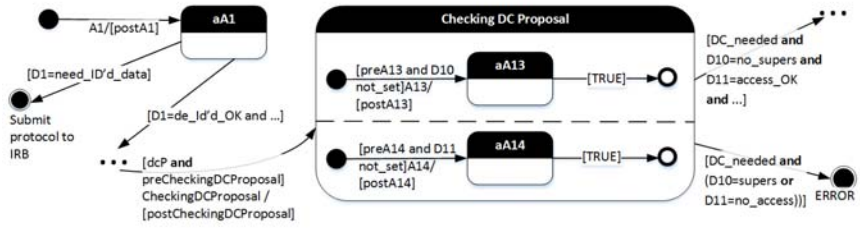

Fig. 4. Annotated protocol state machine.

The states are named with the name of the activity that precedes them, e.g. A1, with a prefix 'a', which means 'after'. Thus, the state 'aA1' means the state of the system after activity A1 has executed. The arrow leading into a state is labeled with UML state transition notation (e.g. '[postA13 and D10 not_set] A13/[postA13]'). The dashed line in the state '*Checking DC Proposal*' separates this state into two regions. This state is an orthogonal state, and the regions are executed concurrently. In general, an orthogonal state can exit if either region is complete. However, the exit transitions of this state both include decisions D10 and D11, thus both must be set via A13 and A14 respectively, before the state can exit.

The solid circle is the initial state, and the next state, *aA1*, has no pre-condition. This state represents a state where the user has decided whether or not de-identified data can be used in the project research. If so, then the transition with the guard '[*D1=de_id'd_OK and …*]' is taken. Otherwise the transition labeled '*[D1=need-*

ID'd_data]' is taken to a final state shown as a double lined solid circle. The '*Checking DC Proposal*' state is entered if there is a data collector proposal that needs to be decided ('dcP') and the other pre-conditions for this state have been met. This state has two exit transitions, one taken when aA13 and aA14 have decisions that allow the data collector proposal to be approved (guard on upper exit line), and the other taken when one or both of them results in a decision that leads to an error termination where the proposal is denied. Ellipsis (…) are used to indicate states that are not included in Figure 4.

Analysis. We present two analyses of the example system. In the first, a slicing technique was used to extract portions of the annotated activity diagram and class diagram related to the DC proposal decision. Figure 5 shows a portion of an Alloy Analyzer output for decision D10 that is related to activity A13, where the rule checking supervisors is applied. The Alloy Analyzer creates instances of the model objects, and makes relations between them according to the constraints in the specification. For this model, a predicate was specified to generate an instance where the decision D10 is '*no_supers*', that is, one where the proposed DC is not supervised directly or indirectly by any member of the project team. The Alloy model defines a sequence of snapshots: one before the activity and one after it. In this paper, a snapshot represents a system state at a particular time. Two snapshots have been combined in Figure 5 as follows. The before snapshot identifies some of the objects with the prefix '$test_' (*test* is the name of the predicate that the Alloy Analyzer is running). Thus, *Application1*, *Protocol1*, *Project1*, and *DataCollProposal1* are the objects that are identified as objects to be used in the predicate in the before snapshot. In addition, the decision D10 is associated with a *null* object in the before snapshot, and the other relations and objects exist as shown in Figure 5. In the after snapshot, the only change is that the decision D10 is now associated with the *no_supers* object.

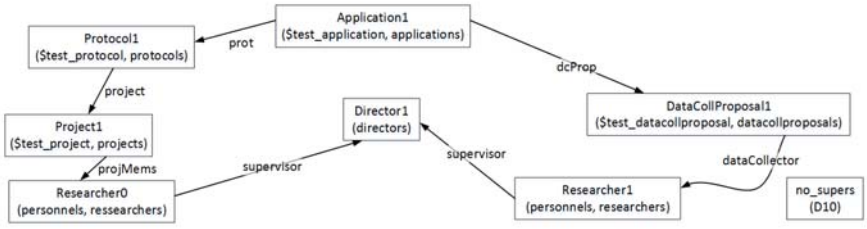

Fig. 5. Portion of Alloy Analyzer instance for A13.

The second analysis we show is an output from the UPPAAL model checker [9]. For this analysis, we abstracted the complete protocol state machine that was derived from the activity diagram shown in Figure 1. The UPPAAL model is shown in Figure 6. This model shows all the decisions made in the protocol state machine with a value of either 0 or 1 since each decision has two possible values. For example, when the system is in state 'aA1', a transition is made to state 'aA2' if decision D1 has the value of 1, and the system transitions to the end state if the value of this decision is 0.

In order to set the value of decision D8, the orthogonal state shown in Figure 4 must be executed as part of another orthogonal state which is called '*chk_deid_app*' in Figure 6. This state is shown in the sub-figure labeled D8, and it consists of broadcasting the

value '*cdcp*' (for **c**heck **d**ata **c**ollector **p**roposal) if D8 and D9 are both 1. These decision values constitute the pre-conditions for the orthogonal state. When the broadcast is received by the states whose diagrams are labeled D10 and D11, each of them starts.

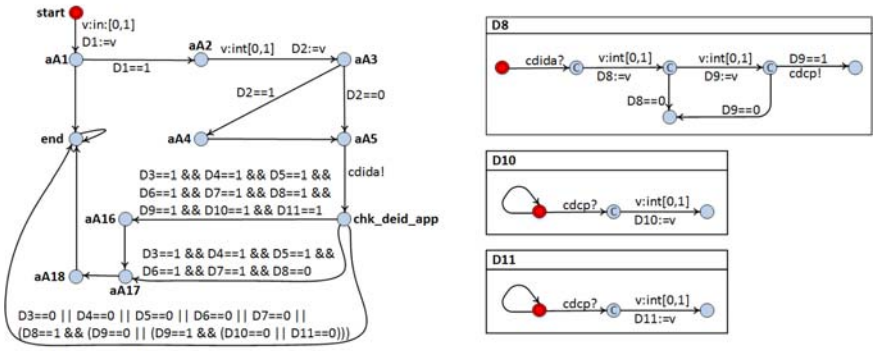

Fig. 6. UPPAAL output for abstracted state machine created from Figure 1.

The 'C' in their second states means that they are *committed*, that is, no interleaving can occur, so they must both run to completion before the orthogonal state can be exited. In each case the decision value is set to 'v', which is either 0 or 1. (The '*chk_deid_app*' orthogonal state uses another broadcast, '*cdida*', for **c**heck **de**-**id**entified **a**pplication, to similarly execute several concurrent states and set the values for D3-D7.) The model shown in Figure 6 was verified to be free of deadlock. An analysis related to rule enforcement is to produce a more complete state model that includes states where a user accesses clinical database information, and show that the accessing states can never be entered unless the decision states for the DC have been traversed.

Another interesting analysis enabled by our method occurs when we have the Alloy Analyzer search for system instances where other relations between a proposed DC and members of a project team besides those of supervision can occur. An example is when a DC for one project, e.g. Proj1, is a team member with other persons on another project, e.g. Proj2, and some of the Proj2 members are also members of Proj1. The Alloy Analyzer can create this object instance which satisfies the DC proposal approval rules, and which also points out an interesting relationship that may in fact be undesired. This situation may be an indication of a missing policy related to data collectors. This example shows that by automatically creating many object instances we can quickly generate situations that are consistent with the class structure, the state machine paths and transition patterns, but may represent undesired behavior and need further exploration.

4 Discussion and Future Work

Medical research defines rule and regulatory enforcement as *system governance*. This governance is often designed using best practices from other projects [4]. Best practices may be slow to improve or recover from perturbations caused by changing regulations, such as those caused by public concern regarding medical data security and privacy.

Complete system analysis using MMCDSA could be challenging since the complexity and number of constraints in any governance is high. However, we believe that focusing on subsystems (e.g., data warehouse, honest broker service, etc.) would not only make the analysis more manageable, but would also allow model sharing and modification across institutions and projects. This could enhance rigorous analysis and help accelerate the evolution of better governance to provide faster tracking of the regulatory landscape. A potential limitation of our method exists since models can be analyzed to demonstrate rule enforcement, but there remains a danger that system implementation does not exactly conform to the model. Auditing of the running system may be needed to provide further evidence of conformance: audit results can be used to create an instance of the system that shows what states are actually executed along a path, or what data is manipulated by the activities leading to various states, and the instance can be analyzed with respect to the system models. Our future work includes analyzing such instances for both desired and undesired system properties.

1. K. Androutsopoulos, D. Binkley, D. Clark, N. Gold, M. Harman, K. Lano, and Z.Li. "Model projection: simplifying models in response to restricting the environment". In Proceedings of 33rd International Conference on Software, Engineering (ICSE), pages 291–300. IEEE, 2011.
2. O. Chowdhury, A. Gampe, J. Niu, J. von Ronne, J. Bennatt, A. Datta, L. Jia, and W. H. Winsborough. "Privacy promises that can be kept: a policy analysis method with application to the HIPAA privacy rule". In M. Conti, J. Vaidya, and A. Schaad, editors, 18th ACM Symposium on Access Control Models and Technologies, SACMAT '13, Amsterdam, The Netherlands, June 12-14, 2013, pages 3–14. ACM, 2013.
3. R. Eshuis. "Symbolic model checking of UML activity diagrams". ACM Transactions on Software Engineering Methodology, 15(1):1–38, 2006.
4. J.H. Holmes, T.E. Elliott, J.S. Brown, M.A. Raebel, A Davidson, A.F. Nelson, A Chung, P. La Chance, J.F. Steiner. "Clinical research data warehouse governance for distributed research networks in the USA: a systematic review of the literature". Journal of the American Medical Informatics Association, 2014, Vol 21:4, pages 730-736.
5. L. T. Ly, S. Rinderle-Ma, P. Dadam, and B. Pernici. "Design and verification of instantiable compliance rule graphs in process-aware information systems". In Advanced Information Systems Engineering, Proceedings, volume 6051, pages 9–23, Heidelberger Platz 3, D-14197 Berlin, Germany, 2010. Springer-Verlag Berlin.
6. R. Rashidi-Tabrizi, G. Mussbacher, and D. Amyot. "Legal requirements analysis and modeling with the measured compliance profile for the goal-oriented requirement language". In D. Amyot, A. I. Antón, T. D. Breaux, A. K. Massey, and P. P. Swire, editors, Sixth International Workshop on Requirements Engineering and Law, RELAW 2013, 16 July, 2013, Rio de Janeiro, Brasil, pages 53–56. IEEE Computer Society, 2013.
7. W. Sun, R. B. France, & I. Ray. "Contract-Aware Slicing of UML Class Models". In Model-Driven Engineering Languages and Systems (MODELS 2013). pp. 724-739, Springer Berlin Heidelberg.
8. S. K. L. M. vanden Broucke, J. Munoz-Gama, J. Carmona, B. Baesens, J. Vanthienen, R. Meersman, H. Panetto, T. Dillon, M. Missikoff, L. Liu, O. Pastor, A. Cuzzocrea, and T. Sellis**.** "Event-based real-time decomposed conformance analysis". In On the Move To Meaningful Internet Systems: Otm 2014 Conferences, volume 8841, pages 345–363, Heidelberger Platz 3, D-14197 Berlin, Germany, 2010. Springer-Verlag Berlin.
9. UPPAAL, http://www.uppaal.org/, accessed June 16, 2015.
10. M. Weiser. "Program slicing". In Proceedings of the 5th International Conference on Software Engineering, pages 439–449. IEEE Press, 1981.