Leveraging UML for Security Engineering and Enforcement in a Collaboration on Duty and Adaptive Workflow Model that Extends NIST RBAC

S. Berhe[1], S. Demurjian[1], S. Gokhale[1], J. Pavlich-Mariscal[2,3], R. Saripalle[1]

[1] Department of Computer Science & Engineering, University of Connecticut, U-2155, 371 Fairfield Road, Storrs, CT, USA
{solomon.berhe,steve,ssg,rishikanth}@engr.uconn.edu
[2] Pontificia Universidad Javeriana, Cra 7 N 40-62, Bogota, Colombia
[3] Universidad Catolica del Norte, Angamos 0610, Antofagasta, Chile
jpavlich@ucn.cl, jpavlich@javeriana.edu.co

Abstract. To facilitate collaboration in the patient-centered medical home (PCMH), our prior work extended the NIST role-based access control (RBAC) model to yield a formal collaboration on duty and adaptive workflow (COD/AWF) model. The next logical step is to place this work into the context of an integrated software process for security engineering from design through enforcement. Towards this goal, we promote a secure software engineering process that leverages an extended unified modeling language (UML) to visualize COD/AWF policies to achieve a solution that separates concerns while still providing the means to securely engineer dynamic collaborations for applications such as the PCMH. Once defined, these collaboration UML diagrams can be utilized to generate the corresponding aspect oriented policy code upon which the enforcement mechanism can be applied to at runtime.

1 Introduction and Motivation

Over the last five years there has been a dramatic shift towards collaborative computing in multiple domains. One such application domain is the patient-centered medical home (PCMH) where a primary physician coordinates care for a patient across a range of providers, who all must interact with one another across distance and time [1]. Our prior work in this regard has been a formal model for obligated collaboration on duty and adaptive workflow (COD/AWF) [3] that extends the National Institute of Standards and Technology (NIST) Role-Based Access Control (RBAC) model [11]. This COD/AWF model adds capabilities to NIST RBAC that include: secure collaboration to control access to data; obligated collaboration which denotes individuals that must participate and how they interact; team-based collaboration, which defines the collaboration

with multiple individuals; and coordinated collaboration, which characterizes the way that individuals are allowed to interact with one another. These four components are grouped together into the following definition:

Def. 1 *$COD = [\mathcal{COD}_{ID}, \mathcal{COD}_{NAME}, \mathcal{TEAM}, \mathcal{CODC}, \mathcal{P}, \mathcal{CW}]$ is a uniquely $\mathcal{COD}_{ID}$ named collaboration $\mathcal{COD}_{NAME}$ with a team of role types $\mathcal{TEAM}$, a set of collaboration constraints $\mathcal{CODC}$, a set of permissions $\mathcal{P}$, and a collaboration workflow $\mathcal{CW}$ composed out of collaboration steps.*

The next logical step is to explore its integration into a software process that includes security engineering from design through enforcement. Towards this goal, we promote security engineering that leverages an extended unified modeling language (UML) to visualize COD/AWF policies to separate concerns while still providing the means to securely engineer collaborations for applications such as the PCMH. Defining collaborative security for PCMH will require role teams, obligations, collaboration steps, and workflows, resulting in requirements that are tangled with one another. The contribution of this work is two-fold, first we propose a set of new UML slice diagrams for COD/AWF that extends prior work on UML with roles, delegation, and user authorization diagrams [10]. Second, these COD/AWF diagrams will be utilized to generate the corresponding policy code upon which the enforcement mechanism can be applied to. There has been a myriad of related work with regard to UML and access control and workflows [4,7,8,10,12,14,15], however none of it considers an integrated Collaboration on Duty (COD) approach which integrates the four components (Obligation, Access Control, Workflow and Teams) into a single formal model. Moreover, to the best of our knowledge no MDA-based approach has been done which extends UML with the four components facilitating separation of concerns and generating COD policy code. The remainder of this paper contains three sections. Section 2 introduces an approach for security engineering of COD/AWF with UML and proposed extensions. Section 3 presents the code templates that are collected together followed by concluding remarks in section 4.

2 Security Engineering of COD/AWF with UML

This section proposes four new UML diagrams that are utilized to constrain and define permissions associated with collaboration, namely: the positive and negative role slice diagrams and the team slice diagram in Section 2.1; the obligation slice diagram and the collaboration workflow slice diagram in Section 2.2. By differentiating between these four diagrams, we essentially separate the concerns to allow the different aspects of permissions to be characterized in different diagrams as illustrated in Figure 1.

2.1 UML Role and Team Slice Diagram

The role slice diagram in Fig. 1a defines permissions [6] for the Emergency Room Collaboration (ERC) has CS Triage (others not shown). Triage negates permis-

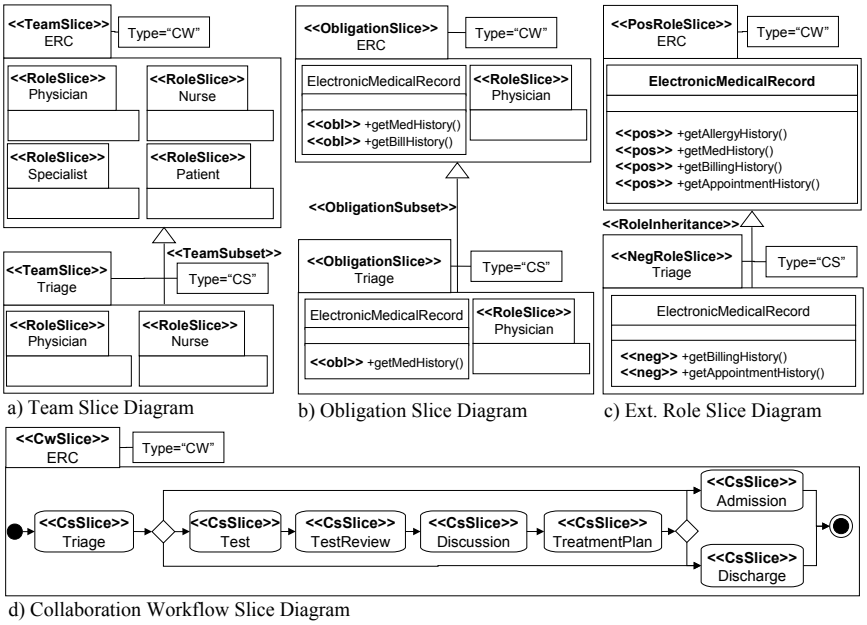

Fig. 1. COD UML Slice Diagrams.

sion getBillingHistory and getAppointmentHistory. During the collaboration, all activated permissions must be a subset of $\mathcal{P}$ (Def. 1), which is modeled using $<< RoleInheritance >>$ stereotype, such that the assigned $\mathcal{P}$ to the collaboration workflow (CW) is represented as the root role slice (Fig. 1c). This CW type is tracked through the use of UML tagged values (Type="CW" and Type="CS"). It's used to match role slices with the corresponding CW, CSs, and roles in the remaining slices (Figs. 1a, 1b, and 1d). All collaboration steps that are activities in the CW are only allowed to activate a subset of $\mathcal{P}$. The current inheritance semantics allows adding additional positive permissions to any role slice [10]. Our objective is to capture COD role slice inheritance semantics in which CSs are only permitted to activate the set of permissions which is not specified as negative and is present in the parent role slice [5]. To enforce this semantic, we extend this notion of role slice with two new annotations: $<< PosRoleSlice >>$ which only allows the specification of positive permissions and is used in the root role slice to set the scope of allowed privileges throughout the collaboration; and, $<< NegRoleSlice >>$ which only allows the specification of negative permissions which is utilized to further restrict privileges in a particular collaboration step (CS).

The team slice diagram in Fig. 1a depicts a separate concern to capture permissions for the entire team. In the ERC example, each team contains the specific role slices that are needed; the latter is inclusive of all roles (entire team of four roles), the former limited to roles within a step. Using the sub-

set << $TeamSubset$ >> relationship for the team slice diagram, the root slice represents $\mathcal{TEAM}$ (Def. 1) and all CSs subset team members from this root team slice. A team slice is depicted as a UML package with the stereotype << $TeamSlice$ >>. This package contains a set of role slices. Permissions are not specified - they are given in the role slice diagram in Fig. 1c - and the focus for team slices is to specify the participants of each step. For permission activation, team membership allows a role to be authorized to permissions.

2.2 UML New Obligation and Workflow Slice Diagram

The obligation slice diagram in Fig. 1b defines the set of permissions that are required to be activated and roles that must participate. These complement RBAC constraints and model the obligation requirement (who is allowed to perform which method at which time) [9]. In Fig. 1b, for the ERC team, a physician is a role that is obligated to participate. For example, during "Triage" CS, the physician must participate. In COD/AWF, obligated participation implies that a role must activate at least one of its permissions. With regard to obligated permission activation, getMedHistory *must* be activated before the collaboration terminates. The obligated activation of a permission requires its activation of any authorized role in the collaboration before it can terminate. Permissions from Fig. 1c are used to constrain the role slice elements within the obligation slice. Permission activation requirements are modeled as classes along with their obligated permissions that are elements of the obligation slice marked using the << *obl* >> stereotype. Similar to the team slices, the root obligation slice represents the set of obligations that must be activated during the entire collaboration, while each collaboration step only must fulfill a subset of it. This is depicted using the << $ObligationSubset$ >> stereotype.

The collaboration workflow diagram leverages and extends the UML activity diagram and allows the security engineer to focus only on the design of the healthcare coordination requirements. In Fig. 1d, the ERC package is composed of 7 collaboration steps into a workflow. The annotation << $CwSlice$ >> is in charge of matching the collaboration steps in the other COD slices with the corresponding collaboration workflow $\mathcal{CW}$ (Def. 1). Access control, obligation and team requirements are unified in this diagram by essentially linking across the four diagrams (1a-1d); while the concerns are separate, they are tied with one another though naming convention and are linked through the unique identifier where matching CS identifiers are located in the previous three slices.

3 Mapping to Enforcement Policies for COD/AWF

Section 2 visually specified COD/AWF via extended UML, and using that as a basis, this section explores the generation of enforcement code that exactly meets the COD/AWF requirements as defined in the UML slices (see Fig. 1a

to 1d). Specifically, this section presents the mapping of the four new/extended COD/AWF UML diagrams to a policy code-based model, which are interfaces/ templates from which actual collaboration domain application can then be enforced at runtime. Accompanying these policy code templates is an authorization enforcement algorithm which checks if a user in a particular collaboration is permitted to activate a permission in a workflow at a particular step (not shown). Our intent in this section is to demonstrate the generated policy code model (templates) for the example as given in Section 2. Note that the COD/AWL UML new/extended diagrams and the code model are extensions to the formal UML Class meta model (not shown)[13]. Finally, this work uses Java-like code templates to illustrate the code mapping of the COD/AWL diagrams.

The remainder of this section is organized as follows. Section 3.1 presents the code template for the role slice and the team slice diagram. Section 3.2 details the code template for the obligation slice and the collaboration workflow slice diagram.

3.1 Policy Code Template for the Role and Team Slice Diagram

The negative and positive role slices allow us to define the set of allowable permissions during the Emergency Room Collaboration (ERC) at the root slice node. In this context, we utilize role slices to define the specific privileges that are associated with the ERC and each of its collaboration steps (e.g. Triage, Admission, etc.). The permissions assigned to the collaboration step/workflow are specified as interfaces which can be implemented by specific classes (e.g. Triage interface can be implemented for an ERTriage or RegularTriage class); this is shown by the code template a) for the slice of Fig. 1c. This allows this COD/AWF framework to be generic enough to adapt to the particular sub-domain (e.g., CDC, Hospital, Clinic, Family Practice, etc.). We utilize the ElectronicMedicalRecord (EMR) class to specify all of the privileges that can be performed against this patients' clinical data. The annotations @PosRoleSlice and @NegRoleSlice are applied to interfaces and enforce sub-interfaces to only specify positive (@pos) or negative (@neg) permissions. This requirement can be verified at runtime using meta programming. In this example, every class that implements the Triage collaboration step interface in the context of ERC is not allowed to activate both permissions getBillingHistory but only getMedHistory (see Code Template b).

```
Policy Code Template a)
@PosRoleSlice
public interface ERC{
  public interface EMR {
    @pos getMedHistory();
    @pos getBillingHistory();
  }
}
```

```
Policy Code Template b)
@NegRoleSlice
public interface Triage ext ERC{
  public interface EMR {
    @neg getBillingHistory();
  }
}
```

In the code template for the collaboration team slice diagram, the root team slice specifies the entire team (from Triage to Admission/Discharge); this is shown by the code template c) for the slice of Fig. 1a. Each team is marked using the @TmSlice annotation. A particular collaboration step further restricts the participation of roles depending on the context using the subset relationship. In the policy code, this relationship is expressed through the @TmSubset annotation. Both annotations can only be applied to interface, which allows the specification of generic teams which can be customized in a particular domain through specific implementation. In this example, the ERC team is composed out of all roles depicted in Fig. 1a. During the Triage collaboration step, only users with the Physician is allowed to participate (policy code template d); all other roles are prohibited to participate in this collaboration step. Fig. 1a only contains a partial representation of who can participate in which steps; for a full collaboration, the diagram would have additional TeamSlice definitions for all collaboration steps.

```
Policy Code Template c)
@TmSlice
public interface ERC{
  public interface Roles {
    public interface Nurse();
    public interface Physician();
  }
}
```

```
Policy Code Template d)
@TmSlice
  @TmSubset(name=TmSlice, val=ERC)
  public interface Triage {
    public interface Roles {
      public interface Physician();
    }
  }
```

3.2 Policy Code Template for the Obligation and Workflow Slice Diagram

The obligation slice policy defines the permissions that must be activated and roles that must participate during a particular collaboration step. The root node defines the obligations that can be specified throughout the ERC collaboration workflow; this is shown by the code template e) for the slice of Fig. 1b. This is denoted using the @CodcSlice annotation. The @CodcSubest annotation further subsets the obligation requirements for a child collaboration step. All of the required roles and permissions are marked using @obl annotation. For example, during the Triage step (policy code template f), it is required to review the patients' medication history but not to read the billing. In terms of participation, Triage requires the Physician to participate. Again, the policy code templates e) and f) only presents a partial definition of the obligation slices.

```
Policy Code Template e)
@CodcSlice
public interface ERC{
  public interface Roles {
    public interface Nurse();
```

```
Policy Code Template f)
@CodcSlice
@CodcSubest(name=CodcSlice, val=EMC)
  public interface Triage ext ERC{
    public interface Roles {
```

```
    public interface Physician();    public interface Physician();
  }                                  }
  public interface EMR {             public interface EMR {
    @pos getMedHistory();              @pos getMedHistory();
    @pos getBillingHistory();        }
  }                                  }
}
```

The final part of the COD/AWF policies specifies all of the collaboration steps and the order in which they must be activated. The << *CollabSlice* >> marks an interface as a collaboration step and the << *NextCollabSlice* >> states the subsequent collaboration steps; this is shown by the code template g) for the slice of Fig. 1d. The ERC interface name along with its collaboration step names are utilized to link them to the code as given in Figs. 1a-1c. The collaboration workflow is annotated using @CollabWorkflowSlice, and each of its collaboration steps with @CollabSlice. Moreover, each collaboration step contains the information about the subsequent collaboration steps using @NextCollabSlice. Again, the code template g) only shows the first two steps of the collaboration in Fig. 1d; the full code template would have all of the steps and represent the entire needed workflow for each collaboration.

```
Policy Code Template g)
@CollabWorkflowSlice
public interface ERC{
  @CollabSlice
  @NextCS(name=CollabSlice value="Test, Admission, Discharge")
  public interface Triage();

  @CollabSlice
  @NextCS(name=CollabSlice value="TestReview, Admission, Discharge")
  public interface Test();
}
```

4 Conclusion

Collaboration applications such as the patient-centered medical home (PCMH) require individuals to interact with one another towards a common goal (treat a patient) across time and under certain limitations; such applications must provide a means to facilitate access and interaction across a sophisticated workflow that is adaptable. The work reported herein extends our prior work on adding collaboration on duty and adaptive workflow (COD/AWF) to NIST RBAC by considering security engineering for collaborative applications that can leverage existing, extended, and new UML diagrams, thereby elevating security to a first class citizen in an integrated software process. Towards this objective, the paper: proposed four new collaboration diagrams that extend and augment UML

to separate concerns for the COD/AWF model in Section 2; presented policy code templates a-g for the four new UML diagrams (Fig. 1a-d) of Section 2. Overall, we believe this work is a crucial step forward for both collaborative security and security engineering, particularly in applications like PCMH.

References

1. American Academy of Family Physicians (AACP): http://www.aafp.org/pcmh
2. G. Ahn and R. Sandhu. " Role-based authorization constraints specification." *ACM Transaction on Information and System Security. Vol. 3, pp. 207-226 (2010).*
3. S. Berhe, S. Demurjian, T. Agresta. "Emerging Trends in Health Care Delivery: Towards Collaborative Security for NIST RBAC." *23rd Annual IFIP Working Conference on Data and Applications Security. Springer-Verlag, Berlin (2009).*
4. E. Bertino, E. Ferrari, V. Atluri "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems." *ACM Trans. Inf. Syst. Secur. 2(1): 65-104 (1999).*
5. T. Budd "An Introduction to Object-Oriented Programming." *Addison-Wesley (1997).*
6. P. Centonze, G. Naumovich, J.S. Fink, M. Pistoia "Role-Based access control consistency validation." *Proceedings of the International Symposium on Software Testing and Analysis (2006).*
7. D. D'Amour, L. Goulet, L. Jean-Francois, S.L. Martin-Rodriguez, P. Raynald "A model and typology of collaboration between professionals in healthcare organizations." *BMC Health Services Research (2008).*
8. J. Juerjens "Secure Systems Development with UML." *SpringerVerlag (2003).*
9. N. Li, M. Tripunitara, Z. Bizri "On mutually exclusive roles and separation-of-duty." *ACM Transaction of Information System Security (2007).*
10. J. Pavlich-Mariscal, S. Demurjian, D. M. Laurent "A framework of composable access control features: Preserving separation of access control concerns from models to code." *Special issue on software engineering for secure systems, vol.29, pp.350-379. Science Direct (2010).*
11. R. Sandhu, D.F. Ferraiolo, R. Kuhn "The NIST Model for Role Based Access Control: Toward a Unified Standard." *Proceedings of the 5th ACM Workshop on Role Based Access Control, pp.47-63, Berlin (2000).*
12. Y. Sun, X. Shijun, P.L. Peng "Flexible Workflow Incorporated with RBAC." *Proceedings of the 9th International Conference on Computer Supported Cooperative Work, pp.525-534 (2005).*
13. A. Teilans, A. Kleins, U. Sukovskis, Y. Merkuryev, I. Meirans "A Meta-Model Based Approach to UML Modelling." *Proceedings of the 10th International Conference on Computer Modeling and Simulation, pp.667-672 (2008).*
14. K.R. Thomas "Team-based access control (TMAC): a primitive for applying role-based access controls in collaborative environments." *Proceedings of the 2nd ACM workshop on Role-based access control (1997).*
15. J. Zarnett, M. Tripunitara, P. Lam "Role-based access control (RBAC) in Java via proxy objects using annotations." *Proceedings of the 15th ACM symposium on Access control models and technologies. (2010).*