Blended Workflow Access Control Perspective

Using Alloy Specifcation

Frederico Madeira
Técnico Lisboa
Lisbon, Portugal
frederico.m.madeira@tecnico.ulisboa.pt

ABSTRACT

The Blended Workflow is a project that combines different representations for the same workflow specification allowing the end user to follow the standard activity-based view when doing his work, but to change that view to a goal-based when an unexpected situation occurs. This work has the objective to enrich the blended workflow model with the access control perspective. Therefore, in this work i model and validate the access of end users to application resources, in order to guarantee that only proper accesses are made. This validation will be achieved using a tool called Alloy Analyzer.

KEYWORDS

Blended Workflow, Access Control, Permission, Alloy, Validation

1 INTRODUCTION

Nowadays every organization has computer systems where all resources that in the past centuries existed only in paper format are now in that computer systems, this collection of resources can be numerous things such as: clients information, past transactions details, organization inventory, work schedules and all other types of data. With all that information on computer systems, there was a normal tendency to use that computer systems to everything support the organization's business processes. A business process is a set of tasks or activities related within themselves that as a whole are a mean to achieve the goal or product that the organization wants to achieve, normally those tasks have an order, for example in the case of an application, those tasks can be, specify, model, develop, test, correct bugs, and deliver to the costumer[1].

With all this business process information in computer systems, a demand of tools to manage that information appeared, thus workflow systems were created. Their primal focus is to facilitate the management of the business process providing an infrastructure to define, manage and monitor the business process. Additionally they provide the users a better view of their business process work, so the user can easily know what has been made, what is left to do and the current state of the business process. There numerous tools for manage the workflow, normally they only support one view of the workflow specification, but there is a lack of tools that support different views for the same workflow specifications[4].

The Blended Workflow[3] project is an approach that supports four different representations of the same workflow: one data-based, condition-based, goal-based, and activity-based. With the possibility of different representations there are different ways that the workflow instances can execute. The goal and activity-based provide views for the execution of blended workflow instances, the activity view shows the user the sequence of activities or tasks they have to preform to achieve the business goal, on the other hand, the goal view allows the user to proceed in a different way from the normal behavior and sequence of the activity view in order to adapt to unexpected situations that may occur and change the course of the business process.

Workflow management systems tend to use resources, which access must be controlled. A common solution is access control which is composed by a set of permissions that allow users to access only the information that they are authorized in order to prevent unauthorized accesses to sensible information or to information that the the user has no rights to.

Implementing an access control is a very sensitive task due to the fact that the access control is the primary barrier that ensures the security of the data of an application, in order to make a correct implementation first there is a need of a correct specification, model and validation of that model.

The tool used in this work for the purpose of the validation of the model is Alloy Analyzer[1]. This tool uses alloy language which is used for numerous applications such as finding holes in security mechanism, Alloy language[2] is based on relations which Alloy uses to represent every types of data such even sets, tuples, scalars. In the Alloy Analyzer models can be made and add constraints that can be solved by the tool, additionally the tool has the ability to generate a graphic visualization for the model by the user defined and to check user defined properties, generate a counterexample and it's correspondent graphic visualization.

2 MOTIVATION

In computer systems security is a main concern, and when there is data in an application, access control restrain everyone from directly access that information. The focus here is that all users cant have the same permissions to access a resource that is part if the application data and this is the main challenge of the access control implementations, some data may be only accessed by people who preform specific task, other can only be accessed for some user that has a specific role, or can only be accessed with other user authorization[5]. This set of constraints and restrictions are what define the access control.

With the knowledge that the access control has a very important role in a workflow management system, it is essential that its specification be as correct and accurate as possible in order to

[1] http://alloy.mit.edu/alloy/

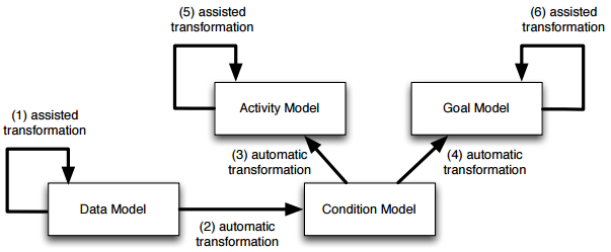

Figure 1: Blended Workflow Design Process [3]

prevent possible incoherences and bugs. The specification is one of the most important part of the access control because a wrong specification may lead to breaches in the application security and allow unauthorized access to sensible information, to validate the correct specification all the constraints defined must be evaluated with the use of the Alloy Analyzer. What makes Alloy Analyzer such a great tool is the ability to visualize the partial model of the representation of the specification which makes it easier to represent incrementally the model, the graphical visualization also allows to navigate the model to better understanding of possible specification mistakes and the structure itself, additionally the user can test conditions cases that should occur as stated in the defined specification and Alloy Analyzer has the ability to generate counterexamples if they exist, which allows the user to better understand the faults on the specification.

3 OBJECTIVES

The Objectives of this work is to define the specification of the access control for blended workflow, prove the correct specification through validations of the defined access control.

4 BLENDED WORKFLOW

The Blended Workflow is a project that combines different representations for the same workflow specification allowing the end user to follow the standard activity-based view when doing his work, but to change that view to a goal-based when an unexpected situation occurs. This work has the objective to enrich the blended workflow model with the access control perspective. Therefore, in this work i model and validate the access of end users to application resources, in order to guarantee that only proper accesses are made. This validation will be achieved using a tool called Alloy Analyzer. This processes based on the data model, which can be modified through the transformation (1). There are three possible operations: add, remove or update over 4 possible resources: entities, attributes, relations and dependencies. These operations do not preserve the models equivalence and every time the data model is changed the other three models are automatically regenerated through transformations (2), (3) and (4). The condition model is generated automatically and is the only one that does not support any transformation, this model contains the conditions regarding the workflow instances. The condition model generates two different models,the activity model through transformation (3) and the goal model through transformation (4). Both activity and goal model support transformations but this transformations preserve the equivalence of all models. In the activity model it is possible to perform the operations: rename, merge and split to an activity and the operations: add and remove to a sequence and in the goal model it is possible to perform operations: rename, merge and split to a goal.

5 CONTEXT

The access control specification is achieved through the enrichment of the Alloy specification of the Blended Workflow. Currently the specification is divided in two parts: the generic to all models and the specific to each model. In this chapter the Blended Workflow specification is described to provide a proper context for a better comprehension of this work.

5.1 Generic

The generic part contains the Alloy specification that does not change independently of the specific model. In here there is the Blended Workflow specification which represents the structure for the three models. Additionally there are three specification which represent the conditions of each model.

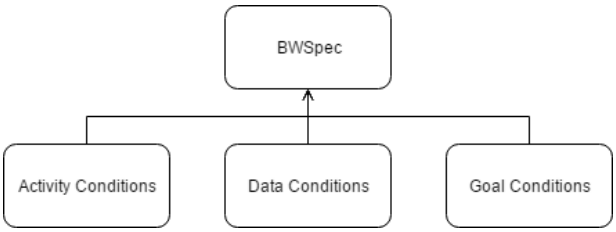

Figure 2: Blended Workflow Generic Structure

The `BWSPec` on the Figure 2 represents the generic Blended Workflow specification. In order to define the Blended Workflow it is necessary to specify the state, which is used to represent the execution of the Blended Workflow . The state is represented by the signature `AbstractState`, which is used in the three models. Each state has the resources already created. In order to represent the resources, this specification introduces two concepts. These concepts are: `Obj` which represents an entity and `FName` which represents an attribute or association between entities. Although association and attributes are both represented as `FName` they can easily be distinguished in the state. The state has objects and fields, the objects represent the entities through a set of signatures of the type Obj and the fields represent the attributes and associations through a function that maps `Obj` x `FName` x `DefVal` and `Obj` x `FName` x `Obj` respectively. In the Blended Workflow models it is only relevant to represent whether the attribute is defined being irrelevant what is the concrete value. Additionally there may exist dependencies between resources. These dependencies are defined as signatures which contains `sourceObj`, `sourceAtt`, `sequence`, `targetAtt`. Although there are four field it is possible that some of them have no value.

Although the invariants and rules are only used in the specific part, their definition is generic. The state transition invariants is the set of invariants that must be ensured whenever a transition between states occur. The completion rules a a set of rules

that must be achieved in order to obtain a meaningful complete state, the workflow finishes. In the states transition invariants there are four invariants that must be ensured. The first one is the `noExtraFields` which ensures that an Entity only has as fields its attributes or associations and noting else. The secound one is the `noMultiplicityExceed` which is used during the execution to ensure that the maximum multiplicity is always respected. Then there is the `bidirectionalPreservation` which ensures that during the execution process either an association is bidirectional and each `Obj` instance of the association is in the fields of the other or it is unidirectional and it is possible to create the inverse link of the association in order to make the association bidirectional. The last invariant is used to represent the dependencies. This invariant has a source object, a source attribute, a path form the source object to the target attribute object and a target attribute. Whenever a dependence exists the target resource and path from the source to the target must already be defined in order to define the source of the dependence. The dependence. When the dependencies do not have source attribute nor target attribute it is a dependence between two objects, if there is no source object it is a dependence between an object and an attribute, if there is no target attribute it is a dependence between an attribute and an object and if there is no path then it is a dependence between two attributes of the same object.

The completion rules are composed by four rules. The first, called `attributesDefined`, ensures that for each entity their attributes must be defined to be consider complete. The second rule is the `multiplicityRule` that guarantee the that multiplicities restrictions, both min and max, are respected. The third one is the `bidirectionalRule` which ensures that the associations are bidirectional and each `Obj` instance of the association is in the fields of the other which is needed to achieve a meaningful complete state. The last one is the same representation of the dependencies that appear in the state transition invariants.

5.1.1 Data Conditions. In the data model there are three operations which are the condition which cause a transition between states. These operations are represented as predicates. They are the creation of an entity, an attribute and an association between objects represented respectively as `defObj`, `defAtt`, and `linkObj`. In order to create an entity, it is necessary that the entity does not exists already in the model. For the definition of an entity attribute it is necessary that the entity already exists and that the same entity does not have already that attribute defined. In both attribute and entity definition, the resource that is being defined may have a dependence, whenever that occurs the resources on which it depends must already be defined. The relation between two objects is unidirectional and to ensure that all relations are bidirectional there is the need to execute the operation two times.During the execution of the operation both objects must exist in the model, and the target object should not be already assigned to the source object.

5.1.2 Activity Conditions. The activity model uses the same state as the data model but it may contain more complex operations for state transition. An operation associated to a state has two predicates which represent their pre-conditions and post-conditions. The pre-condition predicate has as parameters the entities and attributes that must be defined in order to initiate the activity. Although there are two parameters the predicate may receive only entities, only attributes or none of them. The post-condition predicate has as parameters every resource that will be created with the execution of the activity. The post-condition has three parameters which represent each of the three resource types.

5.1.3 Goal Conditions. The goal model also uses the same state as the activity model and data model. Like the activity model an operation also has two predicates that represent two model conditions. One is the activation condition which represent the resources that must be defined in order to activate the goal, and other is the success condition which represent the resources that must be created in order to achieve the goal.

The activation condition predicate has the resources that must already be created in order to activate the goal as parameters. The entities in the activation condition only need to exist in current state. The attributes must have their entities already created and must exist either as a attribute that maps a `DefVal` or an entity that already exists in the current state. In other hand the success condition entities can not exist in the current model and must be possible to define the attributes in the already existing entities or in the entities that are in the post condition. For the associations in the success conditions it must be possible to associate the source and target entities. The execution of the operation adds the entities, attributes and association to the model and ensure that other already existing fields do not suffer changes.

5.2 Specific

The specific part contains all Alloy specification that is specific to each Blended Workflow specification. It contains the Blended Workflow specification which has the model representation, a meaningful complete state, and the invariants to ensure a proper model construction until it achieves the complete state. There is a specification of the model conditions according to the Blended Workflow representation and a representation of the execution of that conditions for each of the models.

5.2.1 Blended Workflow Specification. As said in the beginning of this section the Alloy specification is based on a specific Blended Workflow specification. Using a simple example, as shown in Figure 3, where there are two entities one with two attributes and other with one and there is an association between them.
The Alloy specification is created based on the model shown in Figure 3. In this specification all entities are represented as extensions of objects and the attributes and association are represented as extensions of `FName`. The `FName` which represent an association

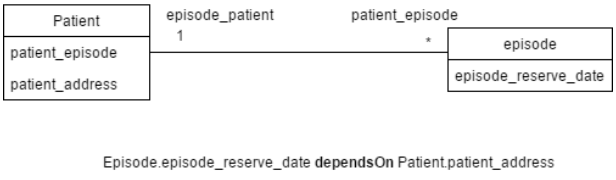

Figure 3: Blended Workflow Example

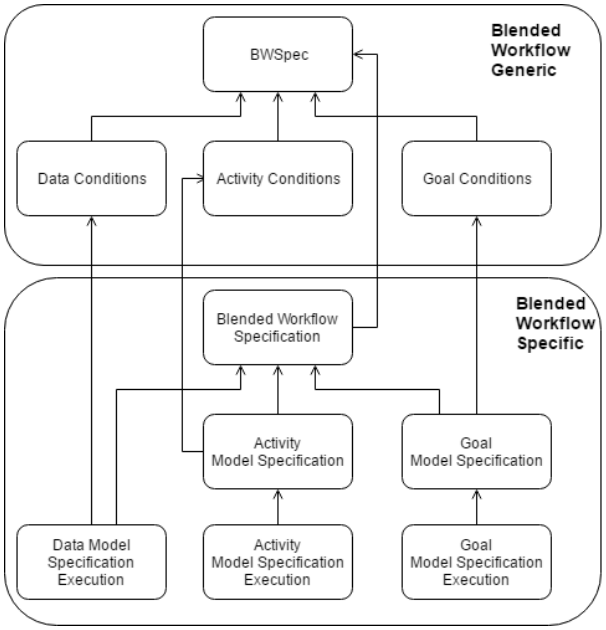

Figure 4: Blended Workflow Specific Structure

have inverse, minimum and maximum multiplicity defined. The inverse represents the inverse link between the two entities. Every entity should have a unique name, the same should happen in the fields where are represented both associations and attributes. Due to space limitations only part of the specification in Alloy is shown in the listings of this work.

,

Listing 1: Example Model Representation

```
sig Patient extends Obj {}
one sig patient_name extends FName {}

fact relations {
        patient_episode.minMul = 0
        patient_episode.maxMul = 10
        patient_episode.inverse = episode_patient
        episode_patient.minMul = 1
        episode_patient.maxMul = 1
        episode_patient.inverse = patient_episode
}

fact dependencies {
        reserve_date_dependence.sourceObj = Episode
        reserve_date_dependence.sourceAtt = episode_reserve_date
        reserve_date_dependence.sequence =  0 -> episode_patient
        reserve_date_dependence.targetAtt = patient_address
}
```

In addition to the resources, the Blended Workflow specifications also contains a predicate that represents a complete state and some conditions that must be met in order to achieved it. This predicate contains the total number of entities instances in the model as well as the total number of each entity type. It also contains predicates that verify that the attribute values are defined, the multiplicity is respected an the bidirectionality of each association is ensured. It is also defined the existent dependencies between model resources.

5.2.2 Data Model Specification and Execution. In the specification of the data model the possible transitions between states are supposed to be defined, instead the operations specified in the data model conditions are used, therefore the data model transitions does not need be specified for each Blended Workflow specification. In order to represent the execution there is the necessity of creating a signature `State` that extends `AbstractState` to create instances that will be used in the execution. Then it is ensured that the first state has no resources and that the only way to transition between states is performing one of the three possible operations of the data model. The definition of an attribute cannot be done without the attribute object already defined and a relation cannot be defined without both objects defined and the possibility to define the inverse relation with the same instances. To represent a proper execution, it is ensured that a complete state can be achieved based on what is defined in the achieve file. Additionally it is tested through the use of asserts that every operation performed in all possible scenarios maintain the invariants stated in the invariants file.

Listing 2: Data Model Specification

```
sig State extends AbstractState {}

pred init (s: State) {
        no s.objects
        no s.fields
}

fact traces {
        first.init
        all s: State - last | let s' = s.next |
        some p: Patient, e: Episode |
        defObj [s, s', p] or
        defAtt [s, s', p, patient_name] or
        defAtt [s, s', p, patient_address] or
        defObj [s, s', e] or
        defAtt [s, s', e, episode_reserve_date] or
        linkObj [s, s', p,  episode_patient, e, patient_episode] or
        linkObj [s, s', e, patient_episode, p, episode_patient]
}

run complete for 4 but 8 State, 5 Int
```

5.2.3 Activity Model Specification and Execution. Contrary to the data model the activity model need a specification for each possible transition between states. These transition are represented using the predicates `preCondition` and `postCondition` specified in the activity model conditions. In this particular case, there are three activities which are `registerPatient`, `registerPatientAddress`, `createEpisode` and `bookAppointment`. The `registerPatient` has no pre-condition and has the `Patient` entity and the attribute `patient_name` in the post-condition. The `registerPatientAddress` has the `Patient` as pre-condition and has `patient_address` as post-condition. The `createEpisode` has the `Patient` as pre-condition and the `Episode` and the association between the `Patient` and the `Episode` as post-condition. The `bookAppointment` has the `Episode` as pre-condition and the attribute `episode_reserve_date` as post-condition.

The representation of the execution of the design Blended Workflow activity model is where the `State` signature is defined. Like the data model, the activity model has a fact that ensures that the first state does not have any resource and that the state will only transition when one of the activities occur. Besides the representation of the execution of the activities defined in the previous section, there

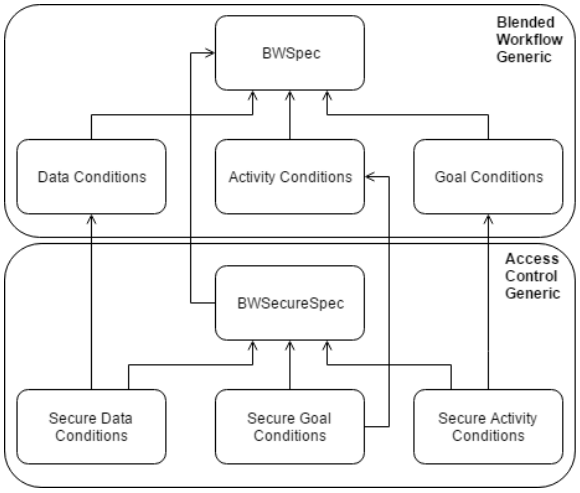

Figure 5: Access Control Generic Alloy Structure

is an assertion that ensures that for every state and any activity that occurred the invariants are always respected.

The goal Model is similar to the activity model and to avoid repetition it is not referenced from this point on but its implementation of the access control is similar to the implemented in the activity model.

6 APPROACH

The approach taken to create the specification of the base access control is focused in the already existing Alloy specification of the Blended Workflow. This specification is enriched with the access control formal specification which is divided in two parts: the generic and the specific. Note that both activity and the goal model have a similar specification and so their access control. For that reason only the activity model is described from here forward.

6.1 Generic

In generic part the base access control contains three new specifications, one for each of the Blended Workflow models and they contain their enrichment with the access control pattern specification. This has a similar structure to the Blended Workflow current specification in which the data, activity and goal models are associated to the Blended Workflow Specification. The three new specifications use the same secure state which is an extension of the Blended Workflow Specification `AbstractState`.

6.1.1 Blended Workflow Secure Specification. The `BWSecureSpec` enriches the already existing specification with the access control specifics. These specifics are `Subject`, `Transition`, `Rights`, `User` and `AccessControlRules`. The `User` is the representation of the person that executes the operation in the Blended Workflow. The `AccessControlRules` is a singleton which acts as a global variable that do not change with the execution of the Blended Workflow. It has fields with all specific elements that represent the access control rules. Common to all security patterns are the `resources` and `permissions` fields. Some security patterns add additional fields with their specification. The `resources` contains all existing resources of the Blended Workflow specification. This resources representation may vary depending on the pattern. The `permissions` is a function that maps `Right` x `Subject` x `resources` which represents the permissions. The state used in this specification has a new field called `log` which records every state transition previously made in a ordered way. For that purpose there is the signature `Transition`. This is useful in order to verify that the operations occurred while respecting the permissions and the invariants.

Listing 3: Blended Workflow Secure Specification

```
abstract sig Subject{}
abstract sig Rights{}
abstract sig User{}
one sig Def, Read extends Rights{}
abstract sig Transition{}

one sig AccessControlRules {
        resources: set {Obj + FName},
        permissions: Rights -> Subject -> resources,
}

abstract sig AbstractSecureState extends AbstractState {
        log: seq Transition
}
```

6.1.2 Secure Data Conditions. The secure data model has three possible transitions. The `defObjTransition`, `defAttTransition` and `linkObjTransition` which stores the information related to that operation. Whenever an entity definition occurs it is stored the `Obj` that represents the defined entity and the user that defined that `Obj`. In the case of an attribute definition it is stored the `FName` that represents the attribute defined, the `Obj` that represents the attribute entity and the user. When an association between entities occurs it is stored the `Obj` entities of the association and the `FName` that represents that association. There is no need to store the `FName` that represents the inverse association because it exists as a field of the `FName`.

Listing 4: Secure Data Model Transitions

```
sig defObjTransition extends Transition{
        dO_obj: Obj,
        dO_usr: User
}

pred addObjToLog (s, s': AbstractSecureState, o: Obj, usr: User){
        some d: defObjTransition |
        d.dO_obj = o and d.dO_usr = usr and s'.log = s.log.add[d]
}
```

The operations specified in the secure data model are `secureDefObj`, `secureDefAtt` and `secureLinkObj`. Note that the predicate that ensures a subject has permission will vary from pattern to pattern. These operations use the same parameters as the operations already defined in the data model and additionally receive a user.

The `secureDefObj` which is shown in Listing 5 ensures that the user that is performing the operations exists in the access control rules and that he has permission to execute that operation through the predicate `hasDefObjPermissions`. Then the operation uses the `defObj` operation and add the operation information to the log through the `addObjToLog`.

The `secureDefAttuses` the same approach as the `secureDefObj` and ensures that the user exists in the access control rules and has permissions to define the attribute. In order to verify the user

permissions the hasDefAttPermissions ensures that the user has permission to define the attribute and permissions to read the object of the attribute. Then it uses the defAtt operation and adds the operation information to the log through the predicate addAttToLog.

The secureLinkObj approach is similar to the previous operations. It is ensured that the user that is trying to perform the operation exists and has permissions to link the objects having permission to define the FName that represents the association and to read both association objects. Then the linkObj operation is used and the operation information is added to the log through the predicate addLinkToLog.

Listing 5: Secure Object Definition

```
pred secureDefObj(s, s' : SecureState, o: Obj, usr:User) {
        hasDefObjPermissions[s, o, usr]
        defObj[s, s', o]
        addObjToLog[s, s', o, usr]
}
```

6.1.3 Secure Activity Conditions. The secure activity model uses the same state as the other models, but adds to the log the information about the execution of activities. That activity log contains all the resources existing in the pre-condition and post-condition.

As it happens in the secure data model, all the information is stored in the log whenever a transition between states occur, but in the secure activity model only one type of Transition is used. This information is used to make a subsequent verification that the user who executed the activity had permissions to do it. Whenever an activity occurs the predicate addActivityToLog stores in the log the user that executed the activity, the pre-condition entities and attributes, and all the post-condition resources that were defined through the execution of this activity.
Unlike the activity model the secure activity model groups both pre-conditions and post-conditions in one model. These ensures that all permissions verifications and log recording occurs properly. In the next paragraphs the structure of the secure activity model is represented. Note once again that the permissions verification will change from pattern to pattern.

The pre-condition contains the resources that must already be defined and the post-condition contains the resources that are created with the execution of the activity. In order to execute an activity the user needs read right over all resources in the pre-condition and def rights over all resources in the precondition. This is ensure through a predicate which use the predicates defined in the secure data model to verify the permissions.

In order to ensure that the user has permissions to execute an activity, there is a predicate called hasActivityPermission. This permissions can be granted by different means as is described in the next chapter. In this work simpler patterns of the access control the user needs to have permissions, with a subject that represents that user, over all resources of the activity. The hasActivityPermission verifies that the subject has the permissions. Additionally the secureActivity uses the addActivityToLog in order to store all resources and the user that executed the activity.

Listing 6: Blended Workflow Secure Activity Model

```
pred secureActivity(...){

    hasActivityPermissions[s, s', pre_entDefs, pre_attDefs,
        post_entDefs, post_attDefs, post_muls, usr]
    preCondition[s, pre_entDefs, pre_attDefs]
    postCondition[s, s', post_entDefs, post_attDefs, post_muls]
    addActivityToLog[s, s', pre_entDefs, pre_attDefs,
    post_entDefs, post_attDefs, post_muls, usr]
}
```

6.2 Specific

In the specific part exists a secure Blended Workflow specification which adds the base access control to the already existing Blended Workflow specification.

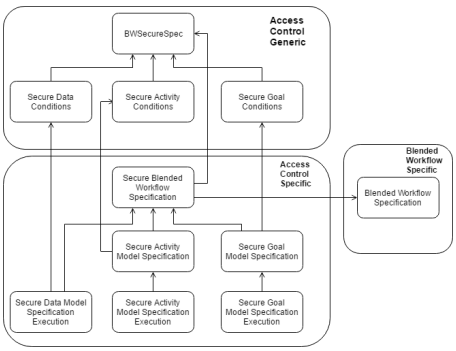

Figure 6: Access Control Specific Alloy Structure

6.2.1 Secure Blended Workflow Specification. The secure Blended Workflow specification uses the already defined specification adding only the access control specifics such as the subjects and the resources. Depending on the security pattern the subject and resources may be represented by different signatures. In some patterns domain objects are added and this requires adding new invariants and completion representations. The access control rules are also specified in this section. It is possible that the specification use on or multiple patterns, which it is the case of the majority of the examples created to ensure proper behavior of the defined patterns. The utimate goal is to create a specification in which all patterns can be used together.

6.2.2 Secure Data Model Specification and Execution. Like the specification of the data model without access control this one does not need to specify the operations of each different Blended Workflow specification. The representation of the execution has a signature that extends AbstracSecureState which is only used in this execution. First this initial secure state is defined where there is no objects, fields or log. There is a fact that ensures that the transition between states only occur when of the defined operations occur. It is also specified an execution where the model achieves the complete state defined in the Blended Workflow specification and verifications which ensures that every operation respect both invariants and secure invariants. Although the data model specification and execution is similar across all patterns the operations: secureDefObj, secureDefAtt and secureLinkObj have different ways to ensure that a user can execute the operation and the secureDMInv must verify the correct execution of the operation according to each pattern verification. Additionally, in patterns that

add domain objects, the invariants and complete state have different verifications from patterns that do not add domain objects.

6.2.3 Secure Activity Specification and Execution. The secure activity model specification is very similar to the activity model one, but instead of using the predicates for each condition it is used a predicate that groups both conditions and all verification related to the activity.

Listing 7: Secure Activity Model Specification example

```
pred secureRegisterPatient(..) {
        secureActivity[s, s', none, none -> none,
        p, p -> patient_name, none -> none -> none, usr]
}
```

The Blended Workflow specific secure activity model is where the `SecureState` signature is defined. Like in the secure data model execution, there is a fact that initiates the first state without `objects`, `fields` or `log`. Besides the representation of the execution of the activities defined in the previous section, there is an assertion that ensures that for every state and any activity that occurred the invariants and secure invariants are always respected.

7 IMPLEMENTATION

In this section the security patterns and their specifications are described. Each subsection of the chapter represents a different pattern which follows the same approach of the base access control and is divided in generic and specific parts.

7.1 User Based Access Control

The user base access control that is used give permissions directly to the user. This access control is the most simple and granular access control in this work. When defining a permission with this access control the `Subject` can be represented as a `User` signature.

In order to define permissions using users it is necessary to specify the `User` signature as an extension of `Subject`. The `Rights` of the permission can either be `Def` or `Read` and the resources can either be `Obj` or `FName` signatures. Additionally the specification of the `AccessControlRules` has a field where the users of the Blended Workflow are represented.

The verification of the user based access control is done by a predicate called `userBasedPermission`. This predicate has the current state, two resources, which can be either an `Obj` or an `FName`, the right and a user as arguments. When defining or reading a resource such as an `Obj` or an `FName`, the predicate must receive the user that is defining or reading the resource and the respective right. The first resource must be the resource that is being defined or read. The second resource argument should always be empty and represented as `none` with the exception of the association between two `Obj`. In that case the second argument must be the inverse of the `FName` that represents one side of the association. The predicate ensures that an permission exist where the `Subject` is a `User` and the resource over which the operation is being performed is the resource of that permission and the `user` performing the operation is the `Subject` of the permission.

Listing 8: User Based Access Control Specifics

```
abstract sig User extends Subject{}

one sig AccessControlRules {
        users: set User,
        resources: set {Obj + FName},
        permissions: Rights -> Subject -> resources,
}
```

7.2 Dynamic Access Control

The dynamic access control also gives permissions directly to a user, but it depends on the association of his domain representation. The user is explicitly represented in the domain through an object. Although these type of objects are not entities, they are also extensions of the signature `Obj`. A single user can have several different representations in the domain, for instance, a user can be both a doctor and a patient. On the other hand, different domain paths may bring to the same user representation, for instance, the doctor may be reached from the Episode or from the Data, where the first situation corresponds to a permission associated with the Episode resource or one of its attributes, whereas the second one corresponds to a similar permission for Data or one of its attributes. This pattern overcomes the limitation of the base access control where there was no way to define permissions based on the domain state.

In order to store their domain representations the `User` signature now has a field represented as a set of `Obj` which contains all his domain representations. Another concept added to support this security pattern is the `DomainSubject` signature. This signature has a field that represents the path, which is a sequence of `FName` between the resource where the operation is being performed and the representation of the `User` in the domain. The `DomainSubject` signature extends `Subject` and it is used in order to define permissions depending on the domain association of the `User`. Like in the previous security pattern, the target resources of this pattern can be either `Obj` or `FName` and the rights can be either `Def` or `Read`.

Listing 9: Dynamic Access Control specifics

```
abstract sig User extends Subject{
        usr_obj: set Obj
}
abstract sig DomainSubject extends Subject{
        path: seq FName
}
```

The verification of the permissions represented with the `DomainSubject` signature are made by the predicate `dynamicResourcePermission`. This predicate arguments are the current state, two `Obj`, an `FName`, the right and the user performing the operation. When the operation is the definition or reading of an `Obj` the second `Obj` and the `FName` in the arguments are empty and represented as none, if the operation is on an `FName`, the predicate does not receive a second `Obj` and in the case where the operation is the definition of an association between two `Obj`, the predicate receives the `Obj` signatures of both ends of the association. The verification ensures that a permission exists where the `Subject` is a `DomainSubject` and the resource is the object that is being defined and, in order to fulfill this rule, one of the user's representations in the domain must be referred by the path associated with the `DomainSubject`, applying this path from the resource instance for which the permission applies. Since the object to be defined does not exist in the model, the permission should be verified in the resulting state, in order

to be possible to evaluate the path. The permissions for the definition of attributes follow a similar approach but the path is applied from the objects the attribute is going to be defined in. The same happens in the definition of an association but there must exist a path associated to a DomainSubject must refer the user domain representation when applied from either end of the association. The predicates for verify if a user can read an object or an attribute it is similar to the definition object but the permission must have Read right instead of Def. When performing an operation the user must have either the permissions defined in this pattern or the ones defined in the base access control.

7.3 Role Based Access Control

The role base access control grants the permission to roles instead of directly to users. These roles create a level of abstraction that allows the definition of permissions before the existence of users in the domain. The role main purpose is to group permission common to many users, knowing that, there is representation of the roles each user has. In order to define permissions the Subject can now be represented as a role. This permission grants the right over a resource to all users that have the role in that permission.

In this security pattern it is added the concept of role. This concept is represented by a signature called RoleSubject.The AccessControlRules signature now have a field represented by a set of RoleSubject which contains all the roles in the domain and another field which is a function that maps users to their respective roles.

Listing 10: Role Based Access Control specifics

```
abstract sig RoleSubject extends Subject{}

one sig AccessControlRules {
        users: set User,
        roles: set RoleSubject,
        u_roles: users -> set roles,
        resources: set {Obj + FName},
        permissions: Rights -> Subject -> resources,
}
```

The verification of the role based access control is made by the predicate roleBasedPermission. This predicate is similar to the one used in user based access control. It has the current state, two resources, which can be either an Obj or an FName, the right and a user as arguments. When defining or reading a resource such as an Obj or an FName, the predicate must receive the user that is defining or reading the resource and the respective right.In order to allow a user to perform an operation there must be a permission where the Subject is a RoleSubject and the target the resource over which the operation is being made. Additionally the user that is performing the operation must have the role existing on the permission. The verification is the same except for the definition of an association where there must be a permission for each FName signatures that represent the association.

7.4 Access Control Over Model Operation

The three patterns represented until this point all use basic resources as the target of their permissions. The access control over model operation change that. Instead of permission over basic resources, they are over operations. These operations are the goals and activities existing in the Blended Workflow. For example, when registering a patient, the hospital worker, instead of having a permission to define the Patient, another to define his name and another to define his address he has only one that allow him to perform the activity register patient. This pattern can be used in conjunction with the three previous security patterns. The permissions can have as resource an operation and the Subject represented as a User, a DomainSubject or a RoleSubject. The operations are now identified by their name.

This security pattern adds the concept of operation. This concept is represented by the signature Operation. With the permission now being defined over operations, the AccessControlRules field where the resources are represented now contains the Operations signatures in addition to the Obj and FName. The addition of the signature Operation causes a modification in the predicates secureActivity and secureGoal which in addition to all the arguments received previously, now receives an Operation signature which is the signature identifier. The arguments added to these two predicates have no impact in the previous security patterns which can use these new predicates with the operation argument empty. This pattern causes another modification in the Transition extensions for both activity and goal which now have an extra field that stores the operation. The predicates addActivityToLog and addGoalToLog starts to add the operation to the log. This is used for the assertions that demonstrate the correct execution of the operations.

The permission verification in this pattern is done by the predicate hasExecOperationPermission. This predicate uses the already defined predicates for the user based access control and the role based access control in order to verify the cases where the subject is either a User or a RoleSubject with the difference that the arguments that represent resources in those predicates can now be Operation. When verifying the permission of user based or role based there must exist a rule where the subject is respectively a User or a RoleSubject and the resource is the operation being performed. Like in their respective security patterns if the permission subject is a User the user performing the operation must be the one in the permission, if the permission subject is a RoleSubject the user performing the operation must have the role in the permission. For the permissions where the subject is a DomainAssociation a new predicate is introduced. This predicate is called dynamicOperationPermission and besides the current state and the user performing the operation it also receives the Operation that is being performed and a set of Obj. This set contains all objects and attributes objects of the conditions of the operation. In order to execute an operation there must exist a permission where the subject is a DomainSubject which path leads from at least one object of the set of objects of the operation to the user representation in the domain.

7.5 Privilege Propagation

The privilege propagation allows a user to share the permissions he has to another user under certain circumstances. The propagation depends on a rule which is represented by a function that maps Operation to RoleSubject. The idea is that only when performing certain operations can occur the propagation and only for users

that have certain role. When propagating privileges it is not required that the source user has permissions over all resources of the operation. There may be cases where the source user has only part the permissions necessary to perform an operation and the target user of the propagation has the rest.

This pattern do not add new signatures to the access control but causes changes in already existing predicates. First of all it is added a `User` argument to the `secureActivity` and `secureGoal` predicates. This new user is the user to whom the privileges are propagated. The signature `ActivityTransition` now has an additional user and so the `addActivityToLog`. The same happens for the goal model.

The permission verification in an operation is made with the already existing predicates. In order to perform an activity with privilege propagation either the `hasExecOperationPermission` predicate is satisfied for the source user or the target user. It is not mandatory that the permissions to perform the operation target be the `Operation`. It is possible that the source user has permission over each resource, the second has the permissions or the combined permissions of the two are allow to perform the operation. This is achieve through use `hasActivityResourcesPermissions` which now receives two users as arguments. The modification to the predicate ensures that for each resource either the source user has permissions or the target user has permissions. This permission verification to the basic resources can be satisfied with any of the first three security patterns.

8 EVALUATION

8.1 Results Demonstration

The execution of the example uses a scope of seven states. These seven states are the minimum required in order to be possible to execute all six activities at least one time. The execution of these six activities leads to the achievement of the complete state in which all entities must have one instance and all its attributes and associations defined. Although the example complete state has a instance of each entity, it is possible to define a complete state with two or more instances for an entity. The change in the complete state definition leads to a change in the scope of the execution of the workflow.

Before executing the workflow in order to achieve the complete state, the initial state must be defined. In the example the initial state has only one entity in the domain. This entity is `DoctorAlice` which is a representation of the user `Alice`, which has the role `Doctor`, in the domain. There are no fields in the domain in the initial state. With the initial state defined, the execution of the activities can now occur according to the access control rules defined previously. Each execution causes a transition between states. The number of transitions until achieves the complete state plus one is equal to the scope provided for state. The six activities that cause transition can be executed in any order taking into account that all dependences and activity pre-conditions are respected. The correct execution of the example achieved the specified complete model. The execution generated an instance of the model where the user `Carl` executed the activity Register Patient, then `Carl` executed the activity Book Appointment, the third activity, Collect Data, was executed by `Alice` and `Bob`, after that `Carl` executed the Check In activity, then `Alice` executed the activity Write Report and the last activity, which was Checkout, was executed by `Carl`. Reviewing all activities one by one, the correct execution of the model according to the defined permissions can be observed. The first activity was Register Patient and was executed by the user `Carl` who has the role `Receptionist` which grant him permissions to execute the operation. The second activity was also executed by `Carl`. With the role receptionist he has permissions to define all resources that need to be defined in the Book Appointment activity. The third activity was executed by `Alice` with the privilege propagation by `Bob` which respectively have the roles `Doctor` and `Nurse`. The `Nurse` role grant `Bob` the permission to define all resources in the Collect Data activity and there is a rule in the access control rules which allows this specific activity to be performed by a user with the role `Doctor` when another user with privileges share them. The fourth activity was done by `Carl` which has the role that grants him permission to define the `episode_checkin` resource. Note that the Collect Data activity was executed before the Check In. That happened because in any place of the model was stated that the Collect Data activity could only be done after the Check In neither through dependences or activities pre-conditions. The fifth activity was executed by `Alice`. The permissions only allow users with a domain representation associated to the `Episode` to perform the operation Write `Report` which is the case of `Alice`. The last activity was Checkout and was executed by `Carl` which through his role has permission to execute the activity.

When observing different instances of the execution of this model, it can be observed that different users executed some activities. With the exception of Register Patient, Book Appointment and Write Report, which were always executed by the same user in different instances, it was achieved instances of the complete model where the Check In and Checkout where either executed by either `Carl`, which have the role `Receptionist` or `Dan` which has a domain representation associated to the `Episode`. Additionally the Collect Data was either executed only by `Bob` or `Alice` and `Bob`.

8.2 Validation

In order to validate the specification created, verification were made to ensure that the operations were executed properly and only by users that had the correct permissions. Additionaly verifications were made in order to study the impacts of changes in the specification of the users, rules and Blended Workflow execution models. The most relevant was the changes in the Blended Workflow.

In this verification different variations of a `Blended Workflow'` were tested. These instances have the same resources of the original specification but have different operations which execution can achieve a complete state. Like the previous one the other two specifications: `User Model` and `Rule Model`, do not change.

The first instance of `Blended Workflow'` has the Register Patient activity divided in Create Patient and Name Patient, and the other instance has the activities Register Patient and Book Appointment merged in one. Given these specification I've tested them with three different instances of `Rule Model` in order to analyse the impact of having the permissions defined over operations. One with permissions over resources, other with half of the permissions over resources and the other half over operation and a third one

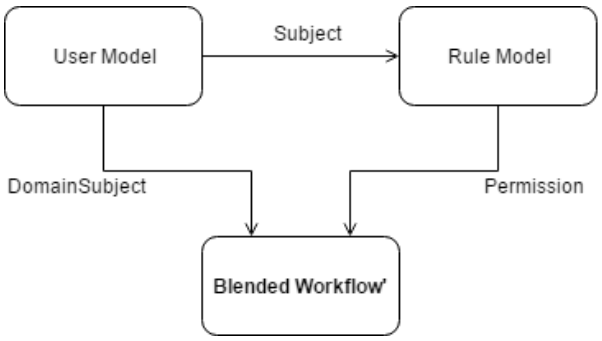

Figure 7: Different Blended Workflow Activity Specification Approach

only where the permissions only apply to the operations. From these three only the one with permissions over resources was able to maintain a execution that achieved a complete state.

Now I analyse what is the behaviour when an operation is split into two new operations. The previous instance of `Rule Model` that achieved the complete state for both instances of the `Blended Workflow'` had only one user with permissions to execute both activities. The `Rule Model` I am going to use has two users, the first user has a role that grants him the permission necessary over all resources of the activity Register Patient, and the second user has a different role which grant him permissions necessary over all resources of the activity Book Appointment. When I split the Register Patient activity into two activities, Create Patient which defines an instance of the `Patient` entity, and Name Patient which defines the *patient_name* of the `Patient`, it is not possible to achieve a complete state, because, it is now necessary to have a permission to read a patient when its name is being defined. From this case we can conclude that the split of activities may, in some situations, impede the complete state to be achieved.

The verification with the instance of `Blended Workflow'` that contains a merge of operations may also impede the achievement of a complete state. Although permissions exist to define the resources, for each one of the users, they to not have alone all the permissions, and so, none of them can execute the merged activity. A solution to this situation is to create a rule stating that the activity resulting from the merge could be propagated from one of the users to the other one. This way the user to which the permission is propagated can execute the activity if the other user allows it.

9 CONCLUSIONS

In this work I proposed and validated, through the use of Alloy language,an access control specification for the Blended Workflow which is comprehensive enough and covers most of the real life application cases. This access control uses most of the common requirements of access control such as user based access control, role based access control, domain association permissions, permissions over operations instead of resources and privileges propagation. The access control specifications are validated in order to find incoherences in the access control such as not being able to achieve the Blended Workflow instance goals due to the lack of permissions to execute some activities.

The validation done in this work allows a better comprehension of the definition of rules and permissions that are associated to basic resources in opposition to associating them to operations and the impact on the workflow execution when the definition of operations is changed. Although the definition of permissions with operations allows the access control to have far less rules it does not have the flexibility to support changes in the way resources are grouped in operations.

The work in this dissertation allowed to identify and validate the foundations of the access control semantics in the blended workflow, which will serve as a basis to its implementation in the blended workflow designer and engine.

10 SYSTEM LIMITATIONS AND FUTURE WORK

This work contains a few identified limitations. The first limitation is the scope used in Alloy execution and assertions. In the largest example used, the execution lasted between a minute and half and two minutes. Although this is not a huge amount of time, the example only has three entities, ten attributes and five bidirectional associations, and the execution only created an instance of each one. For a larger example the execution time would exponentially increase with the number of resources and scope. The other limitation is the lack of a user friendly interface which allows an end user without a technical knowledge to use the Alloy to specify and validate his access control. This specified access control only allows the definition of all rules and permissions in design time and does not address the modification of the control access rules during execution of workflow instances.

The next step of this work would be to implement the access control in the Blended Workflow tool. This implementation would allow the Blended Workflow specification and its access control to be extracted automatically from the data model and access control defined in the Blended Workflow tool eliminating the need for the end user to interact directly with the Alloy tool.

ACKNOWLEDGMENTS

I would like to thank my family and friends for their friendship, encouragement and care over all these years. I would also like to acknowledge my dissertation supervisor Prof. Antonio Rito Silva for his insight, tremendous support, availability and sharing of knowledge that made this Thesis possible. Then I would like to thank to all my friends and colleagues that helped me grow as a person and were always there and with whom i shared great moments. Last but not least I would like to thank to my significant other for everything.

REFERENCES

[1] Marlon Dumas, Marcello La Rosa, Jan Mendling, Hajo A Reijers, et al. 2013. *Fundamentals of business process management.* Vol. 1. Springer.
[2] Daniel Jackson. 2012. *Software Abstractions: logic, language, and analysis.* MIT press.
[3] António Rito Silva. 2012. A blended workflow approach. In *Business Process Management Workshops.* Springer, 25–36.
[4] António Rito Silva and Vicente García-Díaz. 2016. Blended Workflow Designer. Business Process Management Demonstration Track.
[5] Shengli Wu, Amit Sheth, John Miller, and Zongwei Luo. 2002. Authorization and access control of application data in workflow systems. *Journal of Intelligent Information Systems* 18, 1 (2002), 71–94.