A Workflow-based Access Control Framework for e-Health Applications

Giovanni Russello, Changyu Dong, Naranker Dulay
Imperial College London
{g.russello, changyu.dong, n.dulay}@imperial.ac.uk

Abstract

In this paper, we present a framework where access rights are provided to entities on the basis of the actual task that the entities must fulfill as part of their duties. For capturing the requirements of entities' duties we use the notion of workflow. Our main aim is to provide an access control mechanism that is able to balance the competing goals of flexibility and security. As the main beneficiary of our approach we consider e-Health Applications*, where flexibility and security are major requirements. We also provide an implementation of a medical case study to illustrate the framework.*

1 Introduction

In the last decade, the integration of software systems in healthcare environments has increased, leading to so called *e-Health Applications*. Building and deploying such applications in real-world medical environments has identified several research issues. One of the important issues is that of security. Healthcare systems can be thought of as collaborative systems where access to sensitive medical resources should only be provided to authorised entities.

Classical access control mechanisms are too rigid for capturing the dynamic behaviour typical of healthcare applications. As discussed in [32], such mechanisms fail in providing the appropriate flexibility that is necessary when dealing with unexpected situations that are typical of healthcare applications. A typical case is that of a patient admitted into an Emergency Department (ED) before the system administrators have provided to carers the necessary access privileges. As a result, carers cannot check the patient's medical history since they do not have the appropriate access rights for the patient's medical record stored in the hospital database. Another common scenario is that of a doctor that asks for the second opinion of another doctor in charge in another ward. Because the doctor is assigned to a different ward he may not be permitted to access the patient in order to provide an opinion.

The main drawback of current access control mechanisms is that the granting of access rights requires statically binding a subject (i.e., a doctor) to a target (i.e., a patient's medical record), where the subject and the target must be known in advance. A reasonable solution would be to define access rights in more general terms. For instance, we could provide access rights to all doctors working in the ED to the medical records of the patients admitted in the same department. However, there is still the problem of letting an entity from another ward access the resource. This type of situation can be dealt with by allowing accesses to resources to unauthorised entities as exceptional conditions. When an exceptional access is made, such an event must be logged for future analysis. Log analysis can be used for identifying whether the exceptional mechanism is misused. However, as pointed out in the study conducted by Røstad and Edsberg in[24], in most cases the information provided in logs is not sufficient for correctly identifying situations where the mechanism was misused.

The problem that we are facing is that of balancing the tradeoff between the restrictions that regulate the use of information and resources in medical environments and the usability of a system in such dynamic applications. If the access control mechanism is too restrictive then it could become a burden for medical staff that have to use it; whereas, if access control is too loose then protection of information and resources cannot be guaranteed. A precise access control mechanism is required that is able to deal with the level of unpredictability typical of healthcare applications. On the one hand, access rights must be granted only to entities that require access to a given resource and just for the amount of time that such access is necessary. On the other hand, entities and resources must be dynamically identified as the needs for accessing a

978-0-7695-3096-3/08 $25.00 © 2008 IEEE
DOI 10.1109/WAINA.2008.131

resource arise.

What is required is an access control mechanism that is able to manage access rights on the basis of the actual situation in which entities are acting. In a medical environment, carers have to perform tasks as part of their duties. If accessing a patient's record is the task that a doctor has to perform to fulfill his duty, then the access control mechanism should grant the appropriate access right to the specific patient's record to that doctor.

In this paper, we introduce a framework to realise *Workflow-based Access Control* (WBAC), where the notion of workflow is used to capture the tasks that entities have to perform as part of their duties [16]. Workflow system are well established within medical environments since they have been used for diagnosis [3], therapy/treatment [12, 23], and hospital administration [11, 21].

The major contributions of this paper can be summarised as follows. First of all, our approach provides a fine-grained control on the granting of access rights. An entity will be granted access only to the specific resource instance that is bound to the execution of the workflow task. Secondly, the WBAC mechanism automatically adapts the access rights of workflow entities to the requirements of their duties expressed by means of a workflow. This approach corresponds to the principle of *least privilege*, where an entity is granted the minimal access rights for carrying out their duties. Thirdly, our WBAC mechanism ensures that access rights are *dynamically* adapted to the actual needs of entities. An entity can access the resources associated with a workflow task but only while the workflow task is active. Once the task is completed the access rights is invalidated. Finally, we propose an architecture for a concrete realisation of a WBAC mechanism and describe its details in this paper.

This paper is organised as follows. We start with Section 2 where the background of our research is given in more details together with a comparison with related work. In Section 3 we introduce the components that realise our framework. Section 4 focuses on the details of the actual implementation of our framework. For evaluating the feasibility of our approach in Section 5 we discuss the implementation of a case study based on a real medical scenario. Conclusion and future directions are provided in Section 6.

2 Background

To better illustrate the problem that we are tackling we will use the following scenario. A patient with acute abdominal pain is admitted into the Emergency Department (ED) of a hospital. The patient is assigned to a doctor that will perform the Acute Abdominal Pain Diagnosis (AAPD) procedure. The diagnosis procedure requires the doctor to access the patient history, then to carry out a physical exam, and finally to ask for some lab and imaging exams. Optionally, the doctor can ask the opinion of one or more colleagues, depending on the nature of the patient's symptoms.

The basic assumption is that patients' medical records are stored in a database and accessible from any computer in the hospital. However, since the records contain sensitive information, the medical staff of the hospital should have specific restrictions on accessing the records. For instance, a doctor can only access records of patients assigned to his ward. Such a restriction requires that a patient is admitted to the ward, possibly requiring that the patient is physically present in the ward and that access can only be made from terminals in the ward. However, when a doctor from another ward needs to access a patient record the above mechanism will not allow the access unless authorisation is provided, for example, by the first doctor for a limited time.

As shown by the study conducted by Røstad and Edsberg [24] the situation described above is typical when an access control mechanism is used to protect the privacy of patients' records. In their case, the access control mechanism is based on the Role-Based Access Control (RBAC) model and deployed in Norwegian hospitals[1]. RBAC [15, 26] is motivated by the observation that in the real-world most access control decisions are based on the subject's job functions. It is natural to use roles to reflect the various responsibilities in organisations. Users that perform the same job function are assigned into the same role. Permissions are assigned to roles, not directly to the users. The authorisation decisions are based on the role membership of the users. Roles can be organized hierarchically for representing an organization's lines of authority and responsibility. RBAC has attracted a lot of attention from e-healthcare application developers [8, 14, 6] and standards are being developed in the United States by the Healthcare RBAC Task Force [17].

Although RBAC is considered as a state-of-the-art of access control mechanism, it is not flexible enough for coping with the above situation. To circumvent such a limitation, a mechanism was designed to allow accesses to specified entities as exceptional situations. This requires some extra logging to provide trials that can be used for identifying misuses [13]. This idea comes from the notion of *optimistic security* introduced

[1]The system studied was DocuLive, a product of Siemens Medical Solution

in [22]. The basic assumption of the optimistic security model is that in the case of misuse the system has a fallback mechanism that can revert the system to the state before the misuse was made. Such a property does not always hold. In certain situations, once the misuse is done it is not possible to recover. For instance, if the confidentiality of a patient's record is compromised such a loss is not always recoverable. Moreover, Røstad and Edsberg found that the generated logs are not so informative. Therefore, identifying misuses can be harder than expected. Another interesting finding of their study is that such exceptional accesses are not a rarity but the norm in situations similar to the one presented in our scenario. This confirms that there is a real need for a more flexible access control mechanism.

As discussed in [32], similar conclusions can be drawn for most of current access control mechanisms. The starting point of our approach is the observation that classical access control mechanisms fail in capturing the *responsibilities* of the entities acting in the system. This gap can be filled if an access control model is augmented with the notions of tasks and workflows. Workflows can capture the responsibilities of entities and the execution flow, that is the sequence in which the entity must execute the workflow's tasks. When an action on a resource is specified as a task in a workflow, the entity executing the workflow must have the appropriate access rights for fulfilling its responsibilities. For example, if a task in the workflow specifies that *"a doctor must read the patient's medical record"*, then the access right for reading the record must be granted to the doctor. The justification for such access control to exist is that the entity requires it for performing its duty. Workflow management systems are particular relevant in medical situations where failure in performing an action could lead to serious consequences. In recent years, workflow management systems have been applied to medical environments. Several experimental workflow systems have been proposed for helping in carrying out diagnosis, treatment/therapy and for management in hospital administration.

Simplifying, a Workflow-based Access Control (WBAC) mechanism should consist in deriving access rights from tasks specified in workflows. There is a correspondence between tasks and access rights. A task can be thought of as a (subject, target, action)-tuple, where the subject is the entity that must execute the task, the target is the resource required by the entity and the action is the activity that the entity needs to perform on the resource. Similarly, an access right can also be specified in terms of a (subject, target, action)-tuple (as pointed out in [29]). However, if this mapping is statically realised during the specification phase then the flexibility of the mechanism is going to be compromised. The Task-based access control (TBAC) proposed in [30, 31] suffers exactly from this issue. TBAC extends the traditional subject-target models using task-based contextual information. This model is therefore more active which allows activating and deactivating permissions in accordance with emerging context associated with progressing tasks. Unfortunately, in TBAC, the permissions needs to be statically bound to target instances. This means that the specific instance of a target must be known in advance to be specified. Therefore, contrary to our approach, the handling of access control for unspecified target instances is not supported. Moreover, to the best of our knowledge, such an approach has never been implemented yet.

To circumvent this issue, several approaches have been proposed [7, 10, 19, 33] where the notion of *class of data* are used for grouping targets. In the following, we review the work done by Wu *et al.* in [33] based on the METEOR WMS [27]. The same conclusions can be drawn for the other approaches. Wu *et al.* propose an access control mechanism where privileges p for accessing an object o are assigned to a role r when r is executing a task t. However, the object o represents not a single instance of a target but a more generic group of data that in METEOR are called *datagroups*. During execution, the specific instances of targets populate the different datagroups. The main shortcoming of this approach is the granularity for identifying targets since a role obtains privileges for a set of similar targets. For our scenario, this means that a doctor would be allowed to access any patient's records contained into the datagroup.

As for workflow security, most of the previous research has been focused on designing methods for task assignment constraints, inter-workflow security, and multilevel secure workflow systems, as surveyed in [5]. To the best of our knowledge, only two research efforts discuss some ideas similar to ours. The first to conceive an authorisation mechanism based on workflow were Atluri and Huang in [4] where they introduced the Workflow Authorisation Model (WAM). In WAM, authorisation constraints for data and resources are synchronised with the execution of workflows. Basically, the interval in which an authorisation for a given task is valid is changed according to the actual execution of such a task. Although in the paper a model of an implementation using Petri nets is given, no concrete implementation of such model has ever been presented. The main reason behind this is that a concrete realisation of this model is impractical due to the temporal synchronisation that the model requires.

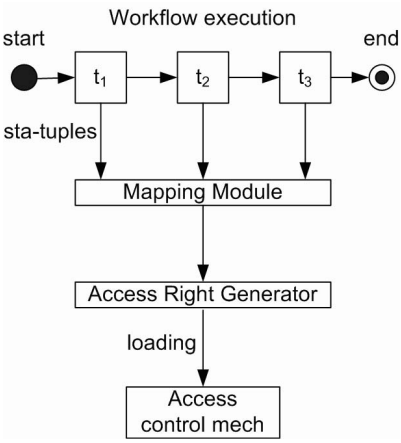

Figure 1. A conceptual overview of our approach.

Knorr describes in [18] an approach where access control matrices are used for regulating access control of data used in the execution of a task. The access rights of subjects specified in the access matrix change according to the execution of the workflow. The main disadvantage of this approach is that the whole set of subjects and data targets must be known in advance to be specified in the matrix.

In light of this, we propose a framework that addresses the issues left open in previous approaches. In particular, we are interested in providing a flexible access control mechanism without compromising the granularity of access rights. A conceptual overview of our framework is shown in Figure 1. During the execution of a workflow, the (subject, target, action)-tuples of each task are mapped to specific instances of subjects and targets. These instances will be used for generating access rights that are subsequently loaded into the access control mechanism. Dynamically generating and loading access rights makes sure that our framework provides enough flexibility for entities to complete their tasks. On the other hand, in our framework, each target represents a specific instance of a resource. Therefore, entities are granted access to exactly the target resource that is needed.

In the rest of this paper we provide more details on how our framework can be realised. Moreover, to illustrate the feasibility of our approach, we will return to the scenario presented at the beginning of this section, providing an implementation based on our framework.

3 Framework Components

To realise our framework two main components are required: a Workflow Management System (WMS) and an Access Control Module (ACM). Our framework is independent of the WMS used for defining and executing workflows. As for the ACM, we use a policy-based access control mechanism. The use of policies for specifying access rigths proved to be more flexible than other access control mechanism and best-fits in our framework. In the following, we provide details on each of the two components and how such components are integrated in the framework.

3.1 Workflow Management System

The WMS that we use in our framework is based on the YAWL environment [1, 2]. The YAWL environment provides a very powerful workflow language together with a workflow execution engine, and an editor for workflow specifications.

The YAWL environment can be customised to export to external components certain events that occur in the life-cycle of workflow instances. On receiving a task-enabled event, a component may decide to 'check-out' the task from the engine. On doing so, the engine marks the task as *executing* and effectively passes operational control for the task to the component. When the component has finished executing the task, it will check it back in to the engine, at which point the engine will mark the task as *completed*, and proceed with the workflow execution.

It should be realised that our framework is independent of the specific workflow language/engine used as long as the workflow environment provides means for interacting with our framework.

3.2 Policy-based Access Control

The other component in our framework is the policy-based access control module based on the Ponder2 policy language and interpreter developed at Imperial College London [20]. The language supports the specification of policies. Policies are rules governing the choices in the behaviour of a system [28]. Ponder supports the specification of authorisation policies and obligation policies (event-condition-actions). The policy interpreter organises the entities and resources on which policies operate in hierarchical domains of managed objects. A managed object has a management interface that the object has to implement in order to be managed by the interpreter. Domains allow the classification and grouping of managed objects in a hierar-

chy. Furthermore, domain paths can be used to address managed objects in policy specifications. Domains can be used to represent resources (e.g., data repositories, printers, X-ray machines, etc.), devices (e.g., sensors), and people (i.e., nurses, doctors, GPs, etc).

Authorisation policies are used for controlling the rights that entities have on the resources managed in a domain structure. In the case of a medical scenario these resources could be represented by the patient's medical data, the medical sensors that the patient wears, and any other application/devices that could be used to assist the patient. Authorisation policies are defined on (subject,target,action)-triples, where the subject is the entity that wants to execute the action on the target. The language also supports negative authorisation policies, that when applied negate the execution of the defined action. Moreover, the authorisation model supports the specification of authorisation policies for independently controlling the subject and the target of an action [25]. In case of authorisation conflicts, that could happen when authorisation policies of different signs apply to the same triple, the interpreter is also able to autonomously resolve those conflicts.

Ponder2 obligation policies are used to capture event-condition-action rules. Such policies are used to dynamically adapt the system to changes of either context or behaviour of applications. Events are trigged by such changes and are propagated using an event bus. Obligation polices capture events and execute actions for adapting the system. For example, obligation policies can change the domain structure adding/removing domains and managed objects, can invoke action on managed objects, can enable/disable other policies and can trigger other obligation policies by sending more events.

The policy access control model allows the insertion and activation of new authorisation policies at runtime. This feature allows us to dynamically adapt the authorisation rights of an entity to the actual needs expressed in the task that is executing. However, it is still necessary to capture which are the access rights required by such an entity and convert them in an authorisation policy. In the following section, we describe how such conversion is performed and provide more details on how the interaction between the WMS and the policy enforcement mechanism is realised.

4 Architecture

In this section, we present the architecture for realising our framework. The architecture consists in integrating the two components presented in the previous section. The integration of the WMS and the policy access control module of the policy interpreter is shown in Figure 2.

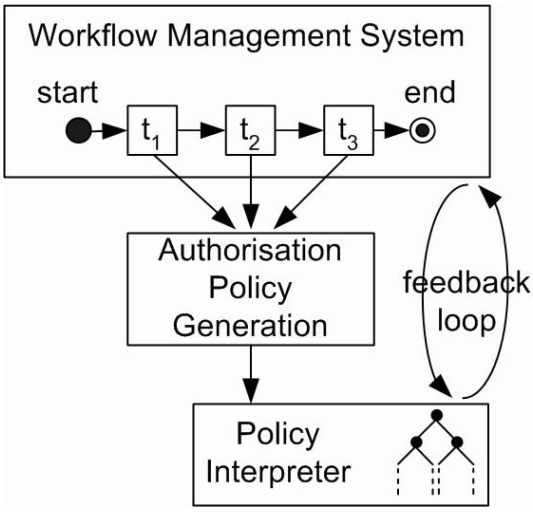

Figure 2. Workflow and policies interactions in our framework.

The workflows executed in the WMS are used for deriving authorisation policies. For each task in a workflow, an authorisation policy is defined in order to provide access rights during the workflow execution. The workflow is enacted by the workflow engine and the authorisation policies are enforced by the policy interpreter.

Although authorisation policies for a given workflow are specified, we want to enable a specific policy only when the corresponding task is executed. This requires the WMS to communicate to the policy interpreter which is the next task to be executed in order to have the appropriate authorisation policy enabled. On the other hand, after the task execution is completed, the authorisation policy must be disabled. Moreover, the WMS needs to be informed that the actual execution of the task is completed to move on with the workflow execution. In order to achieve this, we implemented a feedback loop between the WFM and policy interpreter. The feedback loop is realised using obligation policies. Obligation policies are used for enabling the relevant authorisation policy for a task, intercepting when an task is concluded, and disabling the authorisation policies after task completion.

There are however the following points to address before realising a complete integration:

- convert the notions of entity and resource specified in workflows into managed objects allocated in the domain structure.

- the generation of authorisation policies from the information contained in the workflow specification.
- interfacing the WFS with the policy interpreter for the interaction via the feedback loop.

Each of these points will be discussed in the following.

Entities and Resources An entity executing a workflow is represented in the framework by an `Entity` managed object. We refer to such an entity as the *subject* of the workflow. An `Entity` object is associated with a *type*, that is the role that such an entity has in the domain structure, i.e. nurse, doctor, intern, etc. To associate an `Entity` with the workflows that it has to execute, we define the `EntityDefinition` managed object. For a given type of entity, the corresponding `EntityDefinition` contains the workflows that the `Entity` will execute as part of the entity's responsibilities. In addition to the workflow association, the `EntityDefinition` includes other specific information associated with the type of an entity. First of all, the `EntityDefinition` contains the set of credentials that must be presented in order to instantiate and activate an `Entity` object. An `EntityDefinition` object can be thought of as a passive structure that holds the specification of an entity type, while the `Entity` object is an active instance of such an entity type.

Resources are represented as standard managed objects. We refer to resources in a workflow as *targets*. Several targets can be associated with the specification of a workflow. When the workflow execution is started each of its targets must be associated with the corresponding managed object in the domain structure.

Authorisation Policy Generation Generating authorisation policies is a two-step process. The first step is executed at specification time, when a workflow specification is bound into a `EntityDefinition`. For each of the tasks in the workflow, an authorisation policy template (`APT`) is created and stored in the `EntityDefinition`. Although the `APT` captures the access rights of a method it is not a concrete authorisation policy. This is because the subject and the target are not instantiated managed objects, hence the references to such objects cannot be specified.

The second step is executed when the `Entity` object starts the execution of the workflow. At this stage, the subject and the targets are allocated in the domain structure, therefore references can be provided. The `APT` objects stored in the `EntityDefinition` are used to generate the corresponding authorisation policies that will be loaded in the policy interpreter. Although the authorisation policies are allocated in the domain structure, they are disabled. Each authorisation policy is activated only when the corresponding task is to be executed (more details on this in Section 5).

The use of `APT` objects might seem redundant, since the authorisation policies could be derived directly by analysing the workflow specification at instantiation time. However, this can be inefficient since it requires the analysis to be performed each time an `Entity` instance is created. For large workflows, such an analysis can be a lengthly process to perform at each instantiation.

Feedback Loop The feedback loop between the WMS and the policy interpreter can only be realised if the WMS can send events to the event bus and receive notifications from obligation policies. This is implemented using a WMS managed object (`WMSMO`). The `WMSMO` is a managed object that exposes the WMS interface to managed objects and obligation policies in the domain structure. Obligation policies can be used to notify the WMS when events are propagated in the event bus. On the other hand, the WMS can also send events via the `WMSMO`. Such events capture situations related to the workflow execution. For instance, when a given task need to be executed then the corresponding authorisation policy must be activated. Another type of event that can be sent could capture the case of a task timeout. In this way, obligation policies can invoke appropriate actions for notifying entities. Moreover, the `WMSMO` provides methods for loading, starting, and aborting workflows that can be used by `EntityDefinition` and `Entity` objects.

5 Case Study: Acute Abdominal Pain Diagnosis

In this section, we provide a detailed description of how the Acute Abdominal Pain Diagnosis (AAPD) [9] workflow is specified and executed in our framework.

In the workflow, a patient with abdominal pain is admitted to the Emergency Department (ED) of the hospital. The patient provides to the receptionist his health insurance number (HIN) or other data that will be used to identify the patient. The receptionist will then assign the patient to an intern on duty at the ED. We assume that the hospital is using our framework for executing workflows and enforcing access control to protect the patients' assets.

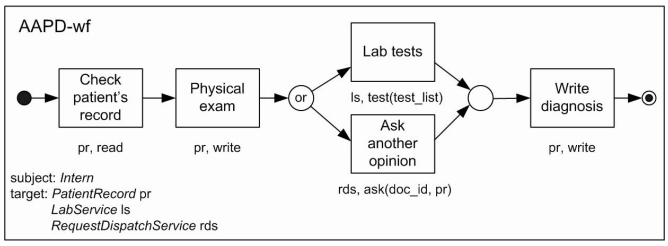

Figure 3. AADP workflow specification.

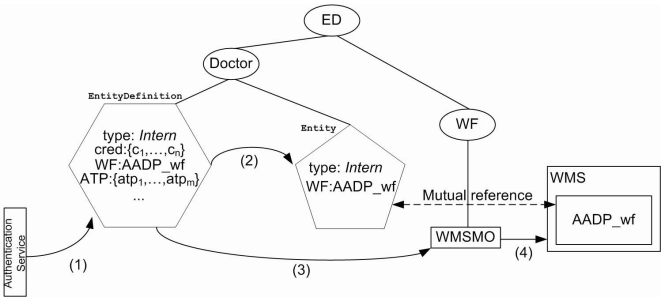

Figure 4. Instantiation of an `Entity` for an intern.

Once the patient is assigned to the intern, the intern starts the execution of the AAPD workflow. The specification for the AADP workflow is presented in Figure 3. An intern first needs to access the patient record to check the patient's medical history. Then a physical exam should be performed. During this task, the intern can write some preliminary notes on the patient's medical record. Afterwards, the intern may ask for some lab tests or he may ask for second opinion from another doctor. The workflow concludes with the intern writing a diagnosis on the patient's record. The targets required for the execution of the tasks in this workflow are specified as workflow's parameters. In particular, the targets of the AADP workflow are the patient's record `pr`, the service `ls` for requiring tests from the laboratory, and the service `rds` for acquiring the opinion of another doctor. In this last case, the `rds` will start the execution of another workflow that is executed by the `Entity` object specified in the `doc_id` parameter. The patient's record is also passed as a parameter and will be provided to the other workflow.

Because this workflow should be executed by an `Entity` object of type `Intern` the reference to the AAPD workflow is added to a `EntityDefinition` object for type `Intern`, as shown in Figure 4. Adding the AADP workflow to the intern's `EntityDefinition` will trigger the parsing of the workflow specification tp provide:

1. the APTs for each of the workflow's tasks. Once the ATPs are created, they are also added to the `EntityDefinition`.

2. the generation of code that is included in the `Entity` object. This code contains a method for each of the tasks in the workflow. The fulfillment of a task in a workflow corresponds to the execution of the corresponding method by the `Entity` object.

To start the execution of the AAPD workflow, the `Entity` object corresponding to the intern must be instantiated. This instantiation is the result of an authentication phase, as depicted in Figure 4. The intern needs to provide valid credentials for authentication. For instance, the intern's digital credentials can be stored in a SmartCard and validated through the PC in the visiting room assigned to the patient. After the credentials are verified the authentication service finds the `EntityDefinition` with matching credential set (1). The `EntityDefinition` instantiates the `Entity` object (2) and loads the workflow in the WMS using the `WMSMO` object (3 and 4). Now the intern is ready to execute the workflow.

The execution of a workflow requires references to the target resources of the tasks. For instance, these references could be provided by another workflow that is executed at the admission desk. When all the required references are available, the authorisation policies are generated from the ATPs in the `EntityDefinition` and loaded in the policy interpreter but are disabled. During the execution of the workflow, the access control adapts the access rights to the current needs of the entity executing the workflow. This means that if the intern wants to write the diagnosis before completing the other tasks, then the system will not authorise such an action.

An authorisation policy is dynamically activated when the relevant task in the workflow is to be executed. Important here is the notion of the *current task*, that is the task that the entity has to execute before moving on in the workflow. When a new current task is activated, the WMS sends an event providing the details that identify such a task. This event is captured by an obligation policy (event-condition-action rule) that checks-out the task from the workflow engine and enables the corresponding authorisation policy. The execution of a task in the workflow corresponds to the execution of a specific method in the `Entity` object. When the method is executed the action on the target resource will be granted since the corresponding authorisation policy is enabled.

When the method concludes another event is sent. This event triggers two obligation policies. One obligation policy disables the authorisation policy used for performing the task. The second obligation policy informs the WMS that the current task is concluded by checking back the task in the engine. The WMS selects the next task in the workflow, sets it as the current task and sends an event to announce the new current task.

Another interesting aspect of our model is that access rights can be delegated to other entities as long as such action is contemplated in the actual workflow. For instance, in our case study the AADP workflow allows the intern to ask for the second opinion from another colleague. Such an action corresponds to starting another workflow executed by the doctor entity that the intern selects. For instance, let us assume that the intern wants to ask for the opinion of doctor Ross. The intern specifies in the request doctor Ross's id and the patient's record. The request is forwarded to the `Entity` object corresponding to doctor Ross that triggers the execution of a new instance of the AADP workflow. While doctor Ross executes the workflow, our framework grants the appropriate access rights for the patient's record. The main advantage of this approach is that new entities can dynamically acquire access rights without the need of capturing the exact details regarding the subject and the target during the specification phase.

When the intern completes the workflow, all the access rights will already have been revoked. This ensures that when the workflow is terminated there are no valid access rights for the entity to access a resource. However, it could be the case that an entity erroneously (or maliciously) leaves the execution of a workflow suspended on a task that is not completed, leaving the access right enabled. To prevent such situations, an expiration time can be associated with the given workflow. After the time expires, the workflow is automatically terminated and all the access rights will be disabled. Moreover, a message will be logged to record such a situation and trigger some further investigation if the situation is logged too often for a given entity.

Discussion In this section we have presented an implementation of a case study based on a real medical scenario that is realised using our framework. Such an implementation allows us to pragmatically assess the feasibility of our approach and to highlight its advantages. Clearly, our proposed WBAC mechanism is flexible enough to cope well with dynamic environments. At the same time, sensible data and resources are protected from being accessed by unauthorised users. As the case study shows, the patient's record becomes accessible to a carer because that carer is fulfilling his responsibilities in the ED. It is not required to know in advance the arrival of the patient to provide privileges to carers. Responsibilities of carers are captured by means of tasks in a workflow. In this way, the framework provides access rights only to the entity that requires to perform accesses on a specific target resource. It should be noted that the access rights to a specific patient's record are given only to the specific instance of an intern that is performing the diagnosis and not to all carers in the ED that have a role of intern. Moreover, access rights will be dynamically removed after the responsibility is fulfilled. Another advantage of our approach is that access rights can be delegated to other entities, as long as the *delegation action* is part of the responsibilities captured by the workflow that the entity is executing.

6 Conclusions and Future Work

In this paper, we have presented the Workflow-based Access Control (WBAC), a flexible access control mechanism that adapts the access rights of subjects to the actual tasks that they have to fulfil. The requirements of entities' duties are expressed by means of workflows. WBAC ensures that entities can access the resources associated with a workflow task but only while such a task is active. Our framework is built by integrating two components: an authorisation module based on the Ponder language, and the YAWL workflow management system. We also outlined a medical case study using our framework.

The main advantages of our approach can be summarised as follows. Firstly, the approach provides fine-grained control on the granting of access rights. An entity will be granted access only to the specific resource that is bound to the execution of the task.Secondly, our approach provides each entity with the minimal access rights for carrying out their duties. This corresponds to the principle of *least privilege*. Finally,access rights are dynamically granted according to the workflow execution. Using a feedback loop between the authorisation module and the workflow engine, we are able to grant access rights during a task execution. Once the task is completed the access rights are invalidated.

As shown in this paper, the WBAC mechanism is practically realisable. This is proven by presenting an implementation of a case study derived from real case scenario.

As part of our future work, we intend to carry out performance measurements of our prototype. Currently, our framework is in a early prototyping phase that needs further fine-tuning. Another area that

we are exploring is related to security issues raised when workflows are executed by competing organisation (*conflict-of-interest*) or need to be executed by different entities (*separation-of-duties*). To this end, we are planning to introduce constraints in the authorisation policies that are generated and to keep track of the workflows that each entity is executing.

Acknowledgments

This research was supported by the UK's EPSRC research grant EP/C537181/1 and forms part of the CareGrid, a collaborative project between the University of Cambridge and the Imperial College London. The authors would like to thank the members of the Policy Research Group at Imperial College for their support.

References

[1] W.M.P. van der Aalst, L. Aldred, M. Dumas, and A.H.M. ter Hofstede. "Design and implementation of the YAWL system." In A. Persson and J. Stirna, editors, *Proceedings of The 16th International Conference on Advanced Information Systems Engineering (CAiSE 04)*, volume 3084 of LNCS, pp. 142–159, Riga, Latvia, June 2004. Springer Verlag.

[2] W.M.P. van der Aalst and A.H.M. ter Hofstede. "YAWL: Yet Another Workflow Language." *Information Systems*, 30(4), pp. 245–275, 2005.

[3] L. Ardissono, A. Di Leva, G. Petrone, M. Segnan, and M. Sonnessa. "Adaptive Medical Workflow Management for a Context-Dependent Home Healthcare Assistance Service." In *Electronic Notes in Theoretical Computer Science*, Elsevier, 2005.

[4] V. Atluri and W.-K. Huang. "An Authorization Model forWorkflows." In *Proceedings of the Fifth European Symposium on Research in Computer Security*, Rome, Italy, pp. 44-64. Lecture Notes in Computer Science, no. 1146. Berlin: Springer-Verlag.

[5] P. Barthelmess. "Security in Workflow Systems." Available at http://www.barthelmess.net/Survey_Pages/Security/security.html. 2001.

[6] M. Becker and P. Sewell "Cassandra: Flexible Trust Management, Applied to Electronic Health Records." In *Proc. of the 17th IEEE Computer Security Foundations Workshop (CSFW'04)*, pp. 139–154, June 2004.

[7] S. Chaari, F. Biennier, C. Ben Amar, J. and Favrel. "An authorization and access control model for workflow." *First International Symposium on Control, Communications and Signal Processing*, pp. 141–148, 2004.

[8] R. Chandramouli. "A framework for multiple authorization types in a healthcare application system." In *Proc. 17th Annual Computer Security Applications Conference (ACSAC)*, December 2001.

[9] K. Cook. "Evaluating acute abdominal pain in adults". http://www.jaapa.com/issues/j20050301/articles/belly0305.htm

[10] G. Coulouris, J. Dollimore and M. Roberts "Role and task-based access control in the PerDiS groupware platform." In *Proc. of the 3rd ACM Workshop on Role-Based Access Control* , pp. 115–121 , 1998.

[11] P. Dadam, and M. Reichert. "Towards a new dimension in clinical information processing." In *Stud. Health Technol. Inform.*, 77, pp. 295–301, 2000.

[12] L. Dazzi, and M. Stefanelli. "A patient workflow management system built on guidelines." In *Proc. of AMIA 97*, pp. 146–150, 1997.

[13] M. Dekker and S. Etalle "Audit-Based Access Control for Electronic Health Records." In *Electronic Notes in Theoretical Computer Science (ENTCS)*, vol. 168, pp. 221–236, 2007.

[14] I. Denley and S. Weston Smith. "Privacy in clinical information systems in secondary care." In *British Medical Journal*, 318, pp. 1328–1331, May 1999.

[15] D. F. Ferraiolo, and D. R. Kuhn. "Role-Based Access Control." In *Proc. of the NIST-NSA Nat. (USA) Comp. Security Conf.*, pp. 554–563, 1992.

[16] D. Georgakopoulos, M. Hornick, and A. Sheth. "An overview of workflow management: from process modeling to workflow automation infrastructure." In *Distributed Parallel Databases*, 3, 2, pp. 119–153, 1995.

[17] The Healthcare Role-Based Access Control Task Force http://www.va.gov/RBAC/.

[18] K. Knorr. "Dynamic Access Control through Petri NetWorkflows." In *Proceedings of the 16th Annual Computer Security Applications Conference*, pp. 159-167, New Orleans, LA, December 2000.

[19] X. Liao,L. Zhang, and S.C.F. Chan. "A Task-Oriented Access Control Model for WfMS." *Information Security Practice and Experience*, pp.168-177, 2005.

[20] The Ponder2 Project http://ponder2.net

[21] M. Poulymenopoulou, and G. Vassilacopoulos. "A Web-based Workflow System for Emergency Healthcare." In *Medical Informatics Europ*, 2002.

[22] D. Povey. "Optimistic security: a new access control paradigm." In *Proceedings of the 1999 workshop on New security paradigms*. ACM Press, Caledon Hills, Ontario, Canada, 2000. ISBN: 1581131496.

[23] S. Quaglini, E. Caffi, A. Cavallini, G. Micieli, and M. Stefanelli. "Simulation of a Stroke Unit Careflow." In *Medinfo*, 2001.

[24] L. Røstad and O. Edsberg. "A Study of Access Control Requirements for Healthcare Systems Based on Audit Trails from Access Logs." In *Proc. of 22nd Annual Computer Security Applications Conference*, Miami, Florida, December, 2006.

[25] G. Russello, C. Dong, N. Dulay. "Authorisation and Conflict Resolution for Hierarchical Domains." In *Proc. of Policy07*. June 2007.

[26] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. "Role-based access control models". In *IEEE Computer 29*, 2 (Feb.), pp. 38-47, 1996.

[27] A. Sheth, D. Worah, K. Kochut, Z.J. Miller, D. Palaniswami, and S. Das. "METEOR Workflow Management System and its use in Prototyping Healthcare Applications." In *Proceedings of the Towards an Electronic Patient Record (TEPR97) Conference*, Nashville, TN, April 1997.

[28] M. Sloman and E. Lupu. "Security and Management Policy Specification." In *IEEE Network*, pp.10–19, Vol. 16, Issue 2, March, 2002.

[29] R.K. Thomas and R.S. Sandhu. "Towards a Task-based Paradigm for Flexible and Adaptable Access Control in Distributed Applications." In *Proceedings of the Second New Security Paradigms Workshop*, IEEE Press, pp. 138–142, 1993.

[30] R. Thomas and R. Sandhu "Conceptual foundations for a model of task-based authorisations." In *Proc. of the 7th IEEE Computer Security Foundations Workshop (CSFW'94)*, pp. 66–79 , 1994.

[31] R. Thomas and R. Sandhu "Task-Based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-Oriented Autorization Management." In *Proc. of the IFIP TC11 WG11.3 11th International Conference on Database Securty XI: Status and Prospects* , pp. 166–181, 1997.

[32] W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong. "Access control in collaborative systems." In *ACM Comput. Surv.*, 37(1), pp. 29-41, 2005.

[33] S. Wu, A. Sheth, J. Miller, and Z Luo. "Authorization and Access Control of Application Data in Workflow Systems." In *Journal of Intelligent Information Systems*, 18(1), pp. 71–94, 2002.