Authorisation Management in Business Process Environments

An Authorisation Model and a Policy Model

by

Khalid Adnan Alissa
Bachelor of Computer Engineering, King Saud University, Riyadh, Saudi Arabia, 2004
Masters of Information Technology, Queensland University of Technology, Brisbane, Australia, 2009

Institute for Future Environments
Faculty of Science and Engineering
Queensland University of Technology

A dissertation submitted in fulfilment
of the requirements for the degree of
Doctor of Philosophy

2015

Keywords: Authorisation management, Business process, Authorisation, workflow, Authorisation model, Policy language model, Authorisation policy language.

In accordance with the requirements of the degree of Doctor of Philosophy in the Faculty of Science and Engineering, I present the following thesis entitled,

Authorisation Management in Business Process Environments
An Authorisation Model and a Policy Model

This work was performed under the supervision of Emeritus Professor Ed Dawson, and Dr. Jason Reid. I declare that the work submitted in this thesis is my own, except as acknowledged in the text and footnotes, and has not been previously submitted for a degree at Queensland University of Technology or any other institution.

Khalid Adnan Alissa

QUT Verified Signature

8/10/2015

To my mother and father, and to my wife Arwa.

*To those who seek knowledge for the satisfaction of the mind,
and understanding of the world.*

Acknowledgements

I would like to thank my wife, Arwa Alumran, for her constant support throughout my PhD experience; my 5-year-old son, Suliman, and my 1-year-old daughter Lina for their patience.

I would also like to extend my gratitude to my parents in Saudi Arabia, my father Adnan Alissa and my mother Sawsan Alomran, for their moral support and guidance in my postgraduate studies and their continuous patience while I live far away from them. This goes to my brothers as well, Eisa, and Omar, and my one and only sister Lulu.

Special thanks to Dr. Farzad Salim for his help and professional support throughout my studies.

I would also like to extend my thanks to Dr. Michael Adams, and Dr. Raffaele Conforti for their help with YAWL. Also I would like to thank Nishchal Kush and Nicholas Rodofile for their help with the SPCC implementation.

I would like to acknowledge the contribution made by many stakeholders involved in the Airports of the Future project (www.airportsofthefuture.qut.edu.au). Part of this research was undertaken as a part of the Airport of the Future project (LP0990135), which is funded by the Australian Research Council Linkage Project scheme.

Thanks to everyone in QUT who was there for me throughout my pleasant PhD journey, especially my supervisors: Emeritus Professor Ed Dawson and Dr. Jason Reid. I am very grateful to have Ed and Jason as my supervisors and mentors. Thanks also to Professor Arthur ter Hofstede and Dr. Chun Ouyang for their guidance through my first year.

Thanks to my colleagues in the information security discipline, and the research student centre. Thanks to my colleagues Raphael Amoah and Nimalaprakasan Skandhakumar. I would like to extend my thanks and gratitude to all of my friends in Brisbane: thank you for always being there for me.

Abstract

Information security is concerned with protecting information systems and information from unauthorised access, unauthorised use, unauthorised disclosure, unauthorised disruption, unauthorised modification, or unauthorised destruction. Therefore authorisation is a key concept in information security, and authorisation management is an important aspect in the protection of an organisation and its information systems. Authorisation management is concerned with writing the authorisation policies, storing the policy, managing the policies, and enforcing the policy. This thesis is concerned with the language for writing authorisation policies, and the model, which enforces them.

Business process management (BPM) is a growing, maturing domain, which today's organisations are increasingly adopting. Business processes are implemented using a special type of information system called Process-Aware Information Systems (PAIS). BPM is a key concept for organising and improveing an organisation's activities. These activities, which are called tasks, can be performed by individuals or by systems. Task performance is governed by authorisation policies, which are enforced using the authorisation model.

Authorisation models for business process environments require specific functionalities that current authorisation models (such as RBAC) do not support. This thesis identifies these functionalities and uses them to show that none of the available authorisation models is suitable for organisations that use business processes. It also identifies the characteristics of an authorisation policy language for business process environments, and uses these characteristics to show that there is a need for a BPM authorisation policy language.

This thesis offers two main contributions. The first one is BP-TRBAC, a unified authorisation model that can support legacy systems as well as business process systems. BP-TRBAC uses RBAC as its base and extends it to support specific features that are required by business process environments. After identifying the characteristics of a business process authorisation model, we used these characteristics to introduce BP-TRBAC, which supports all the identified characteristics. BP-TRBAC is designed to be used as an independent enterprise-wide authorisation model, rather than having it as part of the workflow system. It is designed to be the main authorisation model for an organisation.

The second contribution is the proposal of BP-XACML an authorisation policy language that is designed to represent BPM authorisation policies. It extends XACML to support business process environments. The characteristics of such a language were identified, then they were used to identify how to extend XACML to support business processes policies. BP-XACML as a language is able to represent authorisation policies for business processes, as well as already available authorisation policies that are used for legacy systems. Besides the language, the contribution also includes a policy model for BP-XACML.

Using BP-TRBAC as an authorisation model together with BP-XACML as an authorisation policy language will allow an organisation to manage and control authorisation requests from workflow systems and other legacy systems.

A third contribution made in this thesis is an implementation of a use case of BP-TRBAC. The use case is a subset of BP-TRBAC, called the 'Security Policy Compliance Checker' (SPCC). It was implemented in a workflow system called 'Yet Another Workflow Language (YAWL)'. SPCC is designed to overcome a shortcoming in YAWL, as YAWL allows the process modeler, who is not a security expert, to assign users/roles to perform tasks. SPCC provides a method for checking if the role-task assignments are in compliance with the authorisation policy.

Contents

List of Figures

List of Tables

Listings

Acronyms

List of acronyms used in this document

- BPM: Business Process management
- BPMN: Business Process Modelling Notation
- BPMS: Business Process management Systems
- BoD: Binding of Duties
- CC: Compliance Checker
- CH: Context Handler
- IR: Instance-Level Restriction
- IS: Information Security
- ORM: Object-Role Modelling
- PDP: Policy Decision Point
- PAP: Policy Administration Point
- PAIS: Process-Aware Information Systems
- PEP: Policy Enforcement Point
- PIP: Policy Information Point
- PL: Performers List
- RBAC: Role-Based Access Control
- REA: Role Enablement Authority
- ROPE: Risk-Oriented Process Evaluation
- SEPL: Security Enhanced Process Language
- SoD: Separation of Duties
- SPCC: Security Policy Compliance Checker

- TA: Task Authority
- TPL: Task-Permissions List
- UML: Unified Modelling Language
- XACML: eXtensible Access Control Markup Language
- YAWL: Yet Another Workflow Language

Previously Published Material

The following paper has been published and was presented at the ACISP 2015 Conference:

- Khalid Alissa, Jason Reid, Ed Dawson, and Farzad Salim. BP-XACML: an authorisation policy language for business process. In Douglas Stebila and Ernest Foo, The 20th Australasian Conference on Information Security and Privacy (ACISP), Brisbane. LNCS 9144, pp. 307-325. Springer International Publishing. Switzerland. 2015.

The following paper was submitted to the International Journal of Cooperative Information Systems on June 2nd 2013. The paper went through the review phase and has been resubmitted awaiting final decision:

- Khalid Alissa, Farzad Salim, Jason Reid, and Ed Dawson. Business Process Task-Role-Based Access Control Model (BP-TRBAC). submitted to the International Journal of Cooperative Information Systems. World Scientific Publishing Company.

The candidate has presented the following:

- Khalid Alissa. Integrating Identity and Access Management in Business Processes in an Airport Environment. QUT-ISI consortium. Brisbane, Australia. November 2011.

- Nimalaprakasan Skandhakumar and Khalid Alissa. Identity Management in an Airport Environment. IATA PEMG06 Conference. Brisbane, Australia. January, 2012.

The candidate also presented 2 posters in the:

- Airports of the Future Grand Showcase, Brisbane, Australia. May, 2012.

Chapter 1

Introduction

In the modern business world the design and management of business processes is a key factor for large organisations to compete effectively [43]. While business process management (BPM) and process aware information systems are now attracting more attention [93], existing BPM methodologies barely consider addressing information security and authorisation management requirements [38]. Business process management and authorisation management have been developed separately and often do not follow a coherent strategy [66]. Information security considerations are typically overlooked in business process models [38]. It is, however, crucial to ensure the security of corporate business processes for the success of an organisation [23]. To this end, it is important to consider information security aspects such as authorisation management and authorisation policies when developing and managing business processes.

To be able to enforce authorisation policies on business processes, it is important to define a method that represents authorisation policies in a structured language that process-aware information systems can interpret. A machine-readable representation enables computational methods that can locate conflicts and inconsistencies between policies. There have been a number of studies on languages that translate authorisation policies from a natural language to a machine-readable language (*e.g.* [74]). As will be shown in Chapter 3, most do not consider business processes. It is important to have an authorisation model that enforces these policies in a business process environment.

This research explores the current state of the art of authorisation management in BPM, with an aim to develop an approach that integrates authorisation management into BPM. This research focuses on two major aspects. Firstly, with a focus on authorisation policies, it introduces an authorisation model that enforces authorisation policies in business process environments. Secondly, it provides a machine-readable and enforceable policy language that is designed to represent authorisation policies for business processes.

The rest of the chapter is organised as follows. Section 1.1 will identify the area of research that this thesis belongs to. Section 1.2 will identify the problem that this thesis is aiming to address. Section 1.3 will list the aims and objectives of this thesis. Section 1.5 will identify the contribution of this research. Finally, Section 1.6 will provide the layout for the rest of the thesis.

1.1 Research Area

The domain of Business Process Management (BPM) is an important and maturing domain. A survey by Gartner [32] showed that BPM is the number one concern for many senior executives. This has motivated research in a variety of directions. For example *security-aware BPM* has become a new trend in research in the past few years. It deals with integrating the domain of information security into the domain of BPM. Information security means "protecting information and information systems from unauthorised access, use, disclosure, modification, or destruction" [40]. Authorisation management is the process that defines and enforces what is authorised or allowed in terms of access, use, disclosure, modification, and destruction. An authorisation policy language provides the means by which the rules are expressed and an authorisation model provides the method by which the rules are enforced. This thesis is focused on *authorisation management for business process environments*. With the increasing reliance on information systems and computer networks, authorisation management has become a critical part of business operations across different domains [49]. A working BPM system that does not have an effective authorisation management system could present a significant vulnerability for an organisation [58].

Information systems cannot understand the nuances of natural language or recognise its inconsistencies. It is therefore important to have a structured representation of the authorisation policies that can be interpreted by computer systems [34]. The domain of *machine-readable representation of authorisation policies and access control* is not a new domain. There have been several investigations of a machine-readable representation of authorisation policies that have explored the application of structured languages for authorisation policies (*e.g.* see [22, 74]).

Having the policies in a machine-readable language is not sufficient by itself if there is no mechanism to enforce these policies. An authorisation model provides the method of enforcing the *authorisation policies*. The domain of authorisation models is not a new domain. Role-Based Access Control (RBAC) [77] is a widely known authorisation model. There have been numerous proposals for other authorisation models (*e.g.* see [86, 95]).

Figure 1.1 shows the research area that this thesis is concerned with. The intersection between the business process management domain and the authorisation management domain produced the domain of '*Authori-*

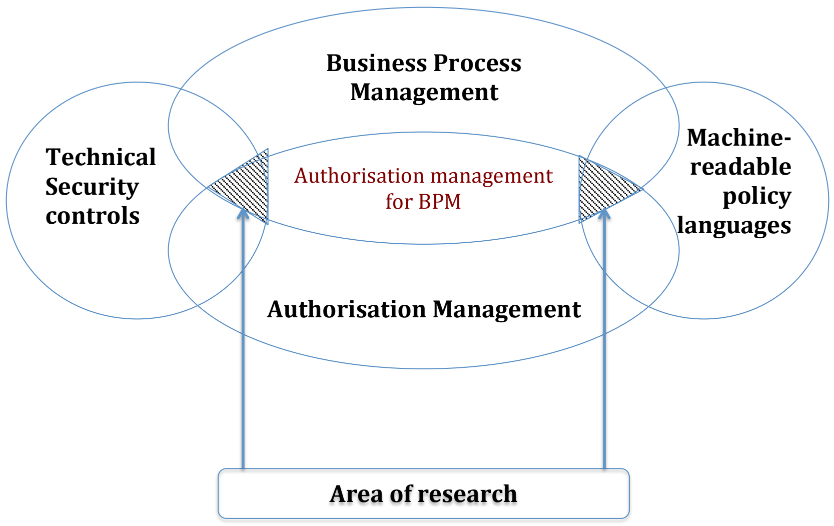

Business Process Management
Technical Security controls
Authorisation management for BPM
Machine-readable policy languages
Authorisation Management
Area of research

Figure 1.1: Area of research

sation management for BPM', which is the concern of this thesis. To be more specific, the diagram shows which parts of the *authorisation management for BPM* domain that this thesis is concerned with. The domain of structured, machine-readable policy languages involves languages that are designed to represent policies in a way that machines and systems can interpret and enforce. These languages seek to remove the ambiguity that is often found in natural language. The intersection of this domain and authorisation management gives the domain of *Machine-readable authorisation policies languages*. In this thesis we are specifically concerned with *'machine-readable business process authorisation policies languages'*. The other concern is the enforcement control. *Technical security controls* are measures that are concerned with enforcing security policy requirements. Access control systems and firewalls are two examples of technical security controls. Technical controls in the authorisation management domain are concerned with authorisation models. From the technical controls side, this thesis is focused on *Authorisation models for BPM*.

1.2 Problem

The past two decades have seen an increasing level of adoption of the principles of business process management within organisations [93]. In line with this trend, more and more organisations are seeking to make their information systems 'process-aware' [26]. A Process-Aware Information System (PAIS) can be defined as "a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models" [26]. We have already identified that it is crucial to ensure the security of the organisation's systems for the success of the organisation [23], that authorisation is a key concept in information security, and that authorisation management is an important aspect in the protection of an organisation and its information systems, including the process-aware information systems. In authorisation management, authorisation models provide the method of enforcing authorisation policies, while the authorisation policy language provides the means of expressing these authorisation policies.

In business process environments, authorisation models require specific functionalities that current authorisation models (such as RBAC) do not support. Some proposed authorisation models have been designed for business processes, but some of these models are designed to be part of the workflow system and only work in the context of workflow authorisation requests. As will be argued in Chapter 3, to ensure consistent application of an organisation's authorisation policy, the authorisation system should be an independent organisation-wide system, rather than having multiple authorisation systems, each of which must be separately maintained. Other proposed models are independent, and not part of the workflow system, but they can deal only with workflow requests. Again, this means that the organisation will require another authorisation system for non-workflow requests. Other models cannot support specific requirements such as instance-level restrictions (which are discussed in more detail in chapter 3). Therefore, there is a need for a unified authorisation model that supports business process systems and other legacy systems at the same time.

During the process of writing authorisation policies for business processes it will be noted that there are specific requirements that appear only in the business process context, and that the policy language should be able to express the policy. For example, in business process it is expected that the policy constraints will be expressed in terms of 'tasks', 'processes', and 'instances' (more details are presented in chapter 3). Although there are several authorisation policy languages, they do not consider business processes and cannot express business process components. On the other hand, there are structured, machine-readable languages that are designed for business processes (*e.g.* business process compliance languages), but these languages are not designed to express authorisation policies. Therefore, there is a need for a *structured, machine-readable business process authorisation policy language.*

To summarise, business processes are widely used nowadays, and it is important to make sure they are supported by authorisation management systems. Yet, as will be shown in Chapter 3, the current authorisation models do not satisfy the necessary specific characteristics for business process environments. Moreover, to the best of the author's knowledge there is no authorisation policy language that supports business process policies. Therefore, there is a need for a BPM authorisation model and for a BPM authorisation policy language.

1.3 Aims and Objectives

This research aims to enable unified authorisation management in business process environments, where processes run across multiple systems and deal with different policies. To reach this main aim, this research has two sub-aims:

- To provide an authorisation model for business process environments that controls authorisation requests and enforces authorisation policies.
- To provide a structured, machine-readable language that has the ability to represent authorisation policies for business processes.

This research provides a case study from a security-sensitive environment that uses business processes. The case study helps in understanding both the problem and the contribution. The case study also provides a context within which, the necessary characteristics for an authorisation model and an authorisation policy language for BPM can be identified.

To reach the research aims, this research will have three main objectives:

- First Objective: A preliminarily analysis that includes:
  - Provide a case scenario from a real-life security-sensitive environment that uses business processes.
  - Identify the characteristics of an authorisation model for business process environments. Establish the characteristics that a language should satisfy to be able to represent authorisation policies in a business process environment.
  - Evaluate the suitability of available authorisation models for business process environments, and evaluate the suitability of available authorisation policy languages for business process environments.
- Second Objective: Provide an authorisation model for business process environments. The model is used to control the authorisation requests by enforcing the authorisation policies. The model is developed based on the characteristics identified in the first objective. A use-case implementation of the model using a workflow system is also part of this objective.
- Third Objective: Provide a structured, machine-readable language that has the ability to represent authorisation policies in business processes. The characteristics identified in the first objective (the language characteristics) are used to assess the suitability of the language.

Each objective is performed as a study. The results of the first study (Preliminarily Study) are presented in Chapter 3. The results of the second study (Main Study 1) are presented in Chapter 4. The results of a use-case implementation can be found in Chapter 5. The results of the third study (Main Study 2) are presented in Chapter 6.

1.4 Approach

After we identified the need for an authorisation model for business process environments, the first step was to identify the required characteristics for such a model. To do so, we interviewed experts from a security-sensitive environment to provide a case scenario as well as conducting in-depth study of the relevant literature. Using this case scenario and the literature, we performed an analysis to identify the required characteristics. After that we used these characteristics to propose a new authorisation model. A use-case implementation for a subset of the model was performed using an open source workflow system. The implementation showed that there is a need for an authorisation policy language that has the ability to represent business process authorisation policies to be used with the model proposed in this thesis. Then we proposed BP-XACML: a machine-readable authorisation policy language for business process environments. The language is to be used with the authorisation model.

1.5 Contributions

This thesis provides two major contributions. The first contribution is the proposal of a new authorisation model (BP-TRBAC). This model was based on the BPM authorisation model characteristics identified in this thesis. BP-TRBAC uses RBAC as a base and extends it to support specific features that are required by business process environments. It is designed to be a unified authorisation model suitable for business process environments. BP-TRBAC is designed to be used as an independent enterprise-wide authorisation model, rather than having it as part of the workflow system. Having the authorisation model as part of a workflow system that is designed to deal only with workflow authorisation requests is not sufficient. It means that the organisation cannot use this model by itself, and will need another authorisation model to handle authorisation requests that do not originate in the context of a workflow. That will cause duplication in functionalities and policies. Moreover, having a single unified enterprise-wide authorisation model helps in having a single unified policy that governs all the organisation's authorisation requests.

The summary of this contribution is available in the following submitted paper:

- Khalid Alissa, Jason Reid, Ed Dawson, and Farzad Salim. BP-XACML: an authorisation policy language for business process. In Douglas Stebila and Ernest Foo, The 20th Australasian Conference on Information Security and Privacy (ACISP), LNCS 9144, pp. 307-325. Brisbane. Springer International Publishing. Switzerland. 2015.

The second contribution is the proposal of BP-XACML an authorisation policy language that is designed to represent BPM authorisation policies. First we identified the characteristics needed for a business process authorisation policy language. Based on these characteristics we extended XACML to support business process environments. BP-XACML as a language is able to represent authorisation policies for business processes as well as standard RBAC authorisation policies that are used for non-workflow systems. The contribution also includes a policy model for BP-XACML.

The summary of this contribution is available in the following accepted paper:

- Khalid Alissa, Jason Reid, Ed Dawson, and Farzad Salim. BP-XACML: an authorisation policy language for business processes. In Douglas Stebila and Ernest Foo, The 20th Australasian Conference on Information Security and Privacy (ACISP). Brisbane. Springer. 2015.

Besides the two major contributions, a third contribution was made in this thesis: an implementation of a use case of BP-TRBAC. The use case is a subset of BP-TRBAC that is called 'Security Policy Compliance Checker' (SPCC). It was implemented in a workflow system called 'Yet Another Workflow Language (YAWL)' [90]. SPCC is designed to overcome a shortcoming in YAWL, as YAWL allows the process modeller, who is typically not a security expert, to assign users/roles to perform tasks. SPCC provides an automated method of checking if task-role assignments are in compliance with the authorisation policy.

1.6 Thesis Layout

This thesis is organized as follows. Chapter 2 provides background information about business process management and authorisation management. It provides information needed to understand the rest of the thesis. Chapter 3 provides an example case scenario from a security-sensitive organisation. The scenario helps in understanding the need for authorisation management in business process environments. Chapter 3 also provides a review of the literature, showing the need for an authorisation model and an authorisation policy language for business process environments. In order to perform the review, the chapter first performs a systematic analysis to identify the authorisation model characteristics. Based on these characteristics it identifies the authorisation policy language characteristics. The characteristics are used to compare the models and the languages in the literature.

Chapter 4 introduces a new authorisation model (BP-TRBAC) that is designed for business process environments. It is designed to address the characteristics identified in Chapter 3. After that Chapter 5 provides a use-case implementation. This implementation includes a subset of BP-TRBAC concerned with checking if task-role assignments made by the process modeler during design time of the business process are in compliance with the organization's authorisation policy. The subset is called 'Security Policy Compliance Checker' (SPCC). Chapter 5 describes a proof of concept implementation for the use case (SPCC).

Chapter 6 introduces BP-XACML, an authorisation policy language that is designed for business process environments. It is a proposal to extend XACML (a structured authorisation policy language) to be able to support representation of business process concepts in authorisation policies. Chapter 7 summarise the thesis contribution, provides possible directions for future research, and concluding remarks.

Chapter 2

Background

Business process management (BPM) is a key concept for organising and improveing an organisation's activities [99]. It is inherently concerned with defining *what* should be done, by *whom* and *when*, to achieve a business goal. As the name indicates, its primary organising principle lies in defining repeatable, step-by-step processes. Authorisation management shares a related concern: it seeks to define and enforce rules about *who* can do *what* with an organisation's assets and information systems in order to protect those resources from misuse and loss. Its primary organising principle is resource-centric as opposed to process-centric. In order to ensure effective protection, the authorisation rules must be observed in all interactions with the resources: both those that occur in the context of a structured workflow and those that might be considered more ad-hoc. The authorisation rules and associated enforcement mechanisms that protect a resource will often be in place before a business process is even defined. Therefore, to ensure the related *who* and *what* concerns are addressed consistently, business process management and authorisation management need to be coordinated. As the following chapters will argue, this has not yet been done effectively in published business process authorisation models.

This chapter explores the relationship between BPM and authorisation management. It starts with a review of business process management, then discusses access control and authorisation management, including policy languages. After that it reviews the domain of authorisation management for business processes focusing on authorisation models and authorisation policy languages.

2.1 Business Process Management

The domain of Business Process Management (BPM) is a growing and maturing domain [93]. BPM focuses on "aligning all aspects of an organisation with the wants and needs of clients, [and is considered as] a holistic management approach" [94]. BPM includes "concepts, methods, and techniques to support the design, administration, configuration, and analysis of business processes" [99]. As is the common convention in the literature, we use the terms *workflow* and *business process* interchangeably.

Products and services that an organisation is aiming to deliver are the result of a number of activities. BPM is a key concept to organise and improve these activities [99]. These activities can be performed by individuals or by systems [99]. In BPM these activities are known as 'tasks'. A task is an atomic unit of work with clearly defined inputs and outputs. Having these tasks in a structured flow that supports the business goals will form a *business process* [37]. Weske [99] says business processes "consist of a set of activities that are performed in coordination in an organisational and technical environment". A business process is a process of transferring business objects (inputs) to business goals (outputs), where control-flow is the core of the process [108]. A business process therefore consists of a set of tasks that are performed in a controlled order that may include complex conditional branching among tasks. For an organisation to reach its business goals in an efficient and effective manner, it is required that the enterprise resources cooperate in performing the *business processes* [99]. BPM denotes the function that defines, governs, administers and maintains an organisation's *business processes*, including whatever is needed to achieve this. [94].

So, BPM is considered to be important for today's organisations as it "helps organisations to gain higher customer satisfaction, product quality, delivery speed and time-to-market speed" [47]. BPM is an ongoing activity whose separate phases researchers have described as the *BPM life cycle* [92]. There are different views about the BPM life cycle but the one proposed by van der Aalst *et al.* seems to be the most concise [46]. The authors state that the BPM life cycle consists of: "process design, system configuration, process enactment, and diagnosis" [92]. Figure 2.1 shows the business process lifecycle.

The first stage, 'process design', is about modelling and designing the business process [92], which starts with collecting data about the process. This data collection happens using methods such as interviewing stakeholders. This data will help the business process analyst understand how the process is being done now, which is called the "as-it-is" process [99]. Designing the *as-it-is* process will allow for a systematic analysis to identify the strengths and weaknesses of the current process [99]. After identifying the weaknesses and issues with the current process, the business process is redesigned to produce the 'to be' process model [99].

The design of the process model can be done using one of many process modeling languages such as 'BPMN (Business Process Modeling Notation)', 'YAWL (Yet Another Workflow Language)', and 'UML (Unified Modeling Language)'. While these graphical notations for process modeling may differ in their details, their fundamental characteristics are quite

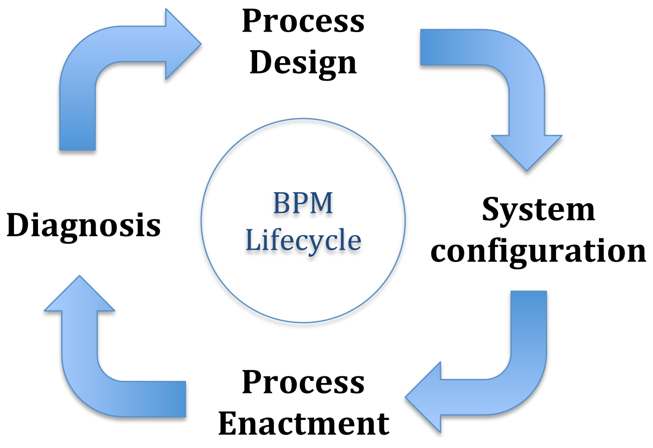


Figure 2.1: BPM lifecycle

similar [99]. BPMN is a business process flows modeling standard proposed by the Business Process Management Initiative (BPMI) [56]. BPMN provides a notation that is understandable by all business users, including business analysts and technical developers [56]. One of the notations that BPMN uses is 'swim lane'. BPMN uses one pool for each organisation involved in the business process and then uses one swim lane for each subject, within this organisation, that is involved in the business process. For example, a BPMN process model can have a swim lane for the role 'coordinator' and place all tasks for this role in his swim lane [56].

Although BPMN is one of the most popular modeling languages [56], it does not have a formal foundation and does not provide automation and simple implementation [90], which is part of the second stage of the BPM life cycle, 'System Configuration'. BPMN can be useful in the first stage (process design); if BPMN was used in the first stage, in the second stage the technical developer needs to understand the model and then build a system based on that model. On the other hand, YAWL is based on a formal foundation, which provides an automated transformation of the process model (first stage) into a working system (second stage). YAWL will be discussed in more detail in Section 2.1.2. This discussion about the process automation and process implementation leads to the second stage of the BPM life cycle, 'System Configuration'.

System configuration is about implementing the design by configuring the Business Process Management System (BPMS) and the infrastructure (*e.g.* synchronisation of roles and organisation charts) [92]. A BPMS is a "generic software system that is driven by explicit process representations to coordinate the enactment of business processes" [99]. After designing and verifying the process model in the first stage, in this stage the business process is implemented using the BPMS. This is done through several steps. First, an implementation platform is chosen, then the technical implementation details for the process are specified to facilitate the process's enactment by the BPMS. Then the system needs to be configured according to the organisational environment and the business process that needs to be performed. The configuration includes specifying the employees' interaction with the BPM system and the integration of the other systems with the BPMS. For these systems to provide support for the process and deal with the BPMS, they have to be 'Process Aware' [92].

Process-aware information systems (PAIS) are a type of information system that is specifically designed to provide support for business process functionalities [51]. PAIS also allow for separating the process logic and the application code [98] (see Section 2.1.1 for more information about PAIS). In the configuration phase, the process aware information systems are configured to support the enactment of the business process, which happens in the next stage of the BPM life cycle.

After configuration, the enactment stage starts, which involves operating and executing the model from stage one into the system that was configured in stage two [92]. It is about the actual run time of the business process, where the BPMS configured in stage two is in control of enacting the business process defined in stage one [99]. Monitoring and data collection, important aspects of the 'enactment stage', allow for gathering valuable information about the process execution, which is stored and logged [99]. These logs and the information they contain are the basis for the evaluation in the next stage.

The diagnosis stage is about analysing and enhancing the process in all aspects using monitoring tools and analysis (process mining) [92]. In this stage data gathered from the previous stage is used to evaluate the process model and the implementation [99]. The information is evaluated using process mining techniques on log files that contain data about the processes execution [17]. These process mining techniques are designed to identify "the quality of business process models and the adequacy of the execution environment" [99]. Evaluation can lead to better understanding of the issues with current processes, and how to modify and improve the process. Process mining mainly focuses on discovery analysis, showing how the processes are actually being executed, and on the organisational model for a given process [17].

The BPM literature does not consider 'information security' in any detail. None of the lifecycle stages has information security as an explicit concern. Although information security should be part of the configuration stage, it has generally been an afterthought in BPM, which might create significant issues, as will be shown and discussed in 2.3. The following subsection will explain more about PAIS and BPMS, focusing on YAWL as a BPMS example.

2.1.1 Process Aware Information Systems

As previously defined, PAIS are a type of 'information system' that supports business processes. So, as a first step let us discuss the term 'information systems'. Alter [4] defines the term information system as a "particular type of work system that uses information technology to capture, transmit, store, retrieve, manipulate, or display information, thereby supporting one or more other work systems" [4]. This definition uses two main terms: 'information technology' and 'work system', which are defined by Alter as follows: Information technology includes all "the hardware and software used to store, retrieve, and transfer information" [4], and a work system is "a system in which human participants perform a business process using information, technology, and other resources to produce products for customers" [4]. Based on these definitions we can depict the entities of an information system: customers, products, business process, participants, information, and technology [26].

In this chapter we consider a special type of information system that is process-aware: these systems link information technology to business processes. PAIS is a "software system that executes operational parts of a business process involving people, applications, and/or information sources on the basis of process models" [26]. Therefore a 'process-aware' system is one that has the ability to understand and that is aware of the process's existence. Some systems might be used to execute a process although they are not process-aware. For example, an email client is usually not a process-aware system, though it may be part of one. Although an execution of a task within the process might lead to sending an email, the email client itself is unaware of the business process. A process-aware system may include components that are not themselves process aware. Other orchestration components call these unaware components at the appropriate time when they are required to perform a task.

According to Dumas *et al.* [26], there are a number of advantages in shifting to process-aware information systems. The first advantage is that the use of 'process models' provides an easy way to communicate between business analysts and IT architects [26]. Also, having a system that is working on the basis of the process model rather than on code, makes it simpler for the system to be changed by just changing the process model, without the need to change the code [48]. Moreover having the process as the base of the system's operation increases the organisation's efficiency and optimises the time and resource usage. The system itself will know where to route information using the instruction from the business process [26]. Finally, process-aware systems provide information and logs on how the process was performing, which helps management to enhance the way the work is done [26].

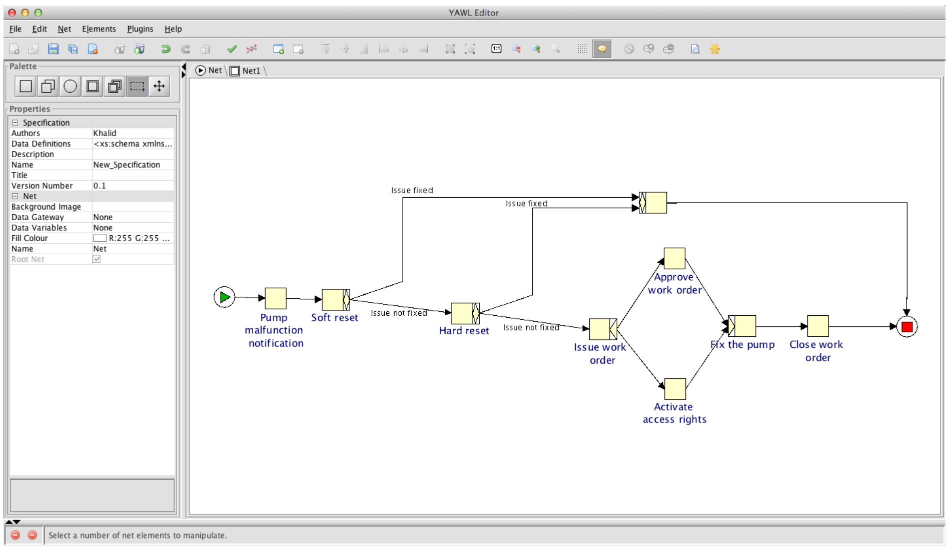
Business process management systems (BPMS) are an example of PAIS. They are a special type of PAIS as they are not only aware of the process but also control the process. At the first stage, at design time, processes are designed using an explicitly defined logic, then at run-time the BPMS "orchestrates the processes according to the defined logic and allows for the integration of users and other resources"[98]. BPMS also "enable the automated support of business processes execution that are carried out by a combination of human actors and systems" [51].

2.1.2 YAWL : a BPMS

Business Process Management systems (BPMS), also known as Workflow management system (WfMS), are used to control the enactment of the business process. YAWL (Yet Another Workflow Language) is a workflow management system that provides a modeling language of its own [90]. YAWL has an Engine module that can translate the process model into a working system using a web interface [90]. The YAWL modeling Language is based on Petri nets [72]. Petri nets, used as a formal method since the 1960s [64], have been extended to include the notion of time [55] to improve expressiveness. Then Color PetriNets (CPN) [42] were introduced where each color represents different data types. This also helped in improving Petri nets' expressiveness. The YAWL modeling language provides a graphical representation to model the business process, and maintains the formal base of Petri nets.

The YAWL language is implemented by a software system consisting of a worklist handler, an execution engine, a resources service, and a graphical editor. The graphical editor is used to design the process model using the YAWL language. In Figure 2.2 the YAWL Editor shows an example process model. As can be seen, a process model has one start and one end, with multiple tasks in between. A process might also include splits (AND, OR, XOR) and joins of the same type. A split is used if the process sequence needs to branch. For example, after doing 'issue work order' the two tasks 'approve work order' and 'activate access rights' can be done at the same time, as neither depends on the other one. In this case the split was 'AND' as both tasks need to be performed but they do not depend on each other. If the situation was that only one task needed to be performed, but not both, the split will be 'XOR'. The third type of split 'OR' is used if either or both tasks may be performed. For each split there has to be a join of the same type exactly. The process describes the flow of the work and the task order from the start point to the end point.

The YAWL editor allows the process modeler to choose a resource to perform each task. The resource can be either human or another service. Assigning a human resource can be by assigning a specific user or by assigning a specific role, so any user with that role is allowed to perform the task. Within the YAWL system the resource service handles all resources, and the organizational model. Logging into the resource service allows the administrator to manage the organisational structure and to manage users' data. Figure 2.3 shows how to use the resource service to manage roles and users. Although the resource service helps the modeler to choose only from

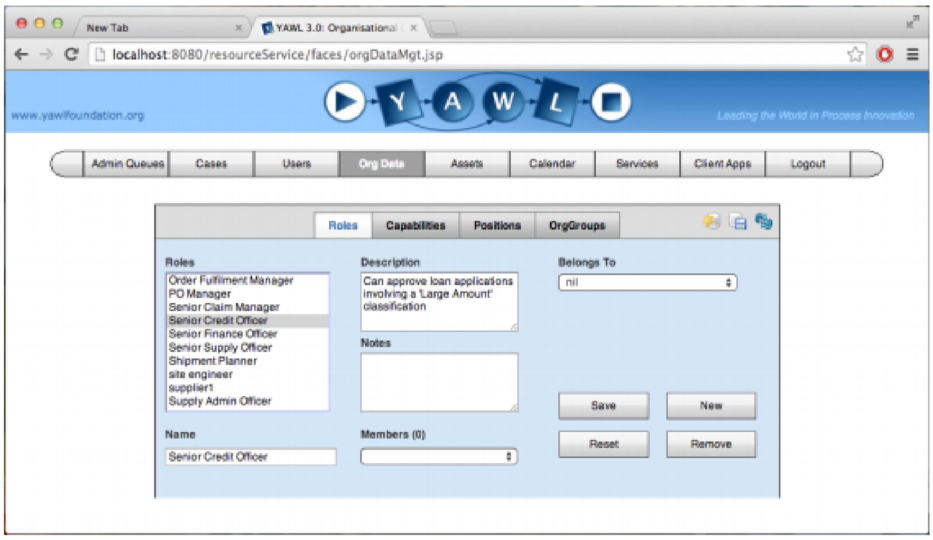


⊟ Specification	
Authors	Khalid
Data Definitions	<xs:schema xmlns...
Description	
Name	New_Specification
Title	
Version Number	0.1
⊟ Net	
Background Image	
Data Gateway	None
Data Variables	None
Fill Colour	R:255 G:255 ...
Name	Net
Root Net	✓

Select a number of net elements to manipulate.

Figure 2.2: YAWL editor

available users and available roles, there is no mechanism within YAWL to make sure that the user assignment complies with the organisation's authorisation policy.



localhost:8080/resourceService/faces/orgDataMgt.jsp

Roles
Order Fulfilment Manager
PO Manager
Senior Claim Manager
Senior Credit Officer
Senior Finance Officer
Senior Supply Officer
Shipment Planner
site engineer
supplier1
Supply Admin Officer

Description: Can approve loan applications involving a 'Large Amount' classification

Notes:

Belongs To: nil

Name: Senior Credit Officer

Members (0):

Figure 2.3: YAWL resource service

When the model is ready with all the required resources and data, it can be saved as a process specification. Then the specification can be uploaded. The YAWL engine will then take control of enacting the business process. The engine is the runtime component. It is responsible for determining which tasks need to be performed and when, but it is not responsible for actually performing the tasks themselves [85].

YAWL was chosen as an example BPMS to be used for a use-case implementation in the context of this research (see Chapter 6), because YAWL software is available as open source software and is service oriented, which allows for replacing existing components with one's own and extending the current environment with new components [85]. Moreover, it can handle complex data transformations, full integration with organisational resources and external Web Services [85]. YAWL has a powerful and expressive process specification language that captures control flow dependencies. It can also automatically transfer the process model into a working system. YAWL is a modeling language and business process management system all in one open-source software.

2.1.3 Declarative Workflow

The YAWL language is a procedural language. Such languages are most suitable for processes that require tight control, and can be repeated without modification. The drawback of such languages is that they lack flexibility, which is important for processes that are driven by user decisions, rather than system decisions [71]. The high degree of flexibility of these processes gives users the ability to control task ordering and coordination while executing the process. Pesic *et al.* [71] proposed a new declarative language, based on constraints, to describe the workflow model instead of the known procedural approach.

The basic idea of the declarative language is that anything is allowed unless explicitly forbidden. Procedural languages describe exactly how the process should be executed stating all possible alternatives. On the other hand, declarative languages aim to specify constraints, and state that the process can be performed in any way as long as the constraints are not violated. Pesic *et al.* [71] proposed constraints templates that can be used to express execution constraints and some business rules. For example, the *precedence* templet specifies that a certain task can be performed only after performing another specific task, bearing in mind that other tasks can be performed in between, *e.g. precedence(A,B)* means that task B can be executed only after task A is executed.

In reality declarative languages cannot replace procedural languages. Declarative languages can add value to procedural languages “where users can quickly react to exceptional situations and execute the process in the most appropriate manner” [71]. Procedural languages are still important as most business process are “repeatedly executed in the same manner, and are too complex for humans to handle and where human mistakes must be minimised” [71].

2.2 Information Security

Information is an asset, and like any asset it needs to be protected. The focus on protecting information is becoming more important with increasingly interconnected business environments [41]. Information can be found in many forms, and is now exposed to a growing range of threats [41]. Information security aims to protect information from this growing number of threats, which means minimising risk [41]. Identifying the required protection is based on the assessment of risk. Knowing the risks will also help to specify the security policies and the controls required to enforce these policies [41].

Once the risks are identified, suitable security policies need to be put in place to mitigate these risks. Policies are written with respect to the organizational decisions on the level of risk that is accepted, and the risk treatment approach. It also should take into consideration the relevant national and international regulations [41].

As stated in [88] there is no simple way to define policy. There are different points of view and themes to look at in the definition of policy [88]. In this research we choose one of the Merriam-Webster dictionary [59] definitions, which defines policy as "a definite course or method of action selected from among alternatives and in light of given conditions to guide and determine present and future decisions" [59]. A policy can be typically seen as a "principle or rule to guide decisions and achieve rational outcome" [8]. The objective of the information security policies is "to provide management direction and support for information security in accordance with business requirements and relevant laws and regulations" [41]. Policies can be written with the help of existing standards such as the ISO/IEC 27002, a code of practice for information security management [41].

Policies are usually written in an 'information security policy document' using plain English (or any other human language). This document should be published and communicated to all relevant parties [41]. Enforcing these security policies needs systems that implement controls. Controls can be categorised as administrative or technical. Administrative controls guide and constrain the actions of employees in their conduct of an organisation's business. They typically specify a procedure that requires an employee or employees to perform particular actions. Technical controls are a combination of hardware and software implemented to address a threat to an organisation's information assets. Access control is a vitally important technical control. A firewall is another example. Access control consists of 'authentication' and 'authorisation'. The aim of authentication is to prove that a person (or resource) is really who is he claiming to be [25], while authorisation is granting permission to the authenticated entity to access, do, or have something [25]. It can be clearly seen that both aspects are important in enforcing access control security policy [41]. Authentication is the first step for making sure that it is the right user and to retrieve all needed information, then authorisation is more involved with enforcing the actual policies in allowing or denying the user from performing the requested action. More detail about access control and authorisation management is presented in Section 2.2.2. Access control is only one system of many that can be used to enforce security policies. In this thesis we focus on the authorisation aspect of access control.

In order for these systems to be able to enforce the security policies, the first step is for the system to understand the policies. Policies should be translated into a language that is machine readable. That is why several machine readable policy languages such as XACML [62] were proposed. The language should have the ability to represent different policy components and also have a formal base to be translated to a machine language. Policy languages are discussed in Section 2.2.1.

We have already noted that controls can be categorised as technical or administrative. In the past it was common for the steps required by an administrative control to be known to employees but not written down. If these knowledgeable employees left the organisation, the undocumented procedure could be lost or incorrectly applied. Business process systems helped in shifting administrative controls to being enforced using systems. Designing a business process with a specific sequence of tasks and having the system depend on the business process to move forward in performing the tasks makes it easier to enforce administrative controls. The gap that we are focusing on in this thesis is the need to integrate the systems that enforce administrative controls (*i.e.* business process management systems), and the systems that are used to enforce the technical control (*i.e.* access control).

To summarise, it can be said that information security is achieved by specifying appropriate security policies, and by implementing a suitable set of controls that enforce these policies. Controls include both technical systems and administrative controls [41]. According to the ISO/IEC 27002 standard, these policies and controls need to be "established, implemented, monitored, reviewed and improved, where necessary, to ensure that the specific security and business objectives of the organization are met. This should be done in conjunction with other business management processes" [41].

2.2.1 Authorisation Policy Languages

ISO/IEC 27002 [41] identifies access control as one of the main areas of an organisation's information security management framework. Authorisation policies are initially authored in plain, human language. For example, 'only human resource employees can view employee records'. With the increased reliance on computer systems comes the need to express these policies in a machine-enforceable language to make sure that all policies are applied according to specifications [22]. To automate the enforcement of a policy, it needs to be translated from natural language to a machine-enforceable language, so systems are able to interpret the policy [57].

To be able to automate policy integration into a technical control system, the first step is to provide a structured representation of these policies, in the form of a language that information systems can interpret. In the past decade there have been several research efforts aimed toward forming methods or languages to represent policies in an enforceable way. Some policy languages were focused specifically on access control policies. In this thesis we want the language to also support business process terminology and to represent the policies in a structured way.

Before going into details, it is important to explain and distinguish between several types of languages: business process modelling languages, business rules modelling languages, business process compliance languages, and security policy specification languages. Business process modelling languages are languages that use the analysed customer requirements to create a process model. This process model represents the sequence of tasks to be performed to achieve the business goal [99]. Business rules modelling languages are languages used to derive a model of the rules and regulations that a business has to comply with. It focuses only on representing the rules and regulations [107]. Business process compliance languages are languages that try to link the previous two domains. They are languages that try to represent the rules and regulations that a business should comply with and integrate these into the business process, to make sure that each process of this business is in compliance with the rules and regulations [34]. A security policy specification language is a language that represents security policies in a structured way that is machine readable [22].

A security policy is a policy that defines information security principles or rules that have been adopted by an organisation. It defines what it means for a system or an organisation to be secure. For an organisation, the security policy addresses the constraints on the behaviour of its members and its adversaries [28]. For systems the security policy addresses functions and their flow, and addresses constraints on access to the system or its data by humans or other systems [28].

eXtensible Access Control Markup Language

eXtensible Access Control Markup Language (XACML) is a standard issued by the Organisation for the Advancement of Structured Information Standards (OASIS) [62]. XACML provides a XML-based language for expressing authorisation policies and a syntax and evaluation method for access request messages and responses. The goal of XACML is to propose a common language that provides the organisation with the ability to manage all the elements of its authorisation policies across all the information systems [16].

In XACML the basic elements are rules that specify whether to allow or deny a specific action on a specific resource by a specific subject [20]. A rule contains a 'target', which identifies or filters the set of requests that the rule applies to. It may also contain a condition, and an 'effect', which is to either 'allow' or 'deny' [68]. Rules are gathered in a policy, which also contains a target and a rule combination algorithm. Policies are gathered in a policy set, which contains a target, a combination algorithm, and may also contain other policy sets [68].

XACML also provides a method for evaluating a policy [20]. It defines a standard component called a Policy Decision Point (PDP), whose function is to select those policies that are applicable to an access request and to evaluate them in the requester's context to arrive at a decision as to whether the request should be allowed or denied [16]. Figure 2.4 shows the major components in XACML and the data flow between these components. A Policy enforcement point (PEP) initiates a request in response to a user's attempt to access a controlled resource and enforces the decision returned by the PDP [20]. A context handler (CH) is the entity that translates between the XACML components (PDP, PEP) and other systems [16]. CH receives the request from the PEP and translates it before forwarding it to the PDP. The policy administration point (PAP) is responsible for administering and managing the policies [16]. The policy information point (PIP), is responsible for communication with other systems, in case some extra information is needed to make a decision [20].

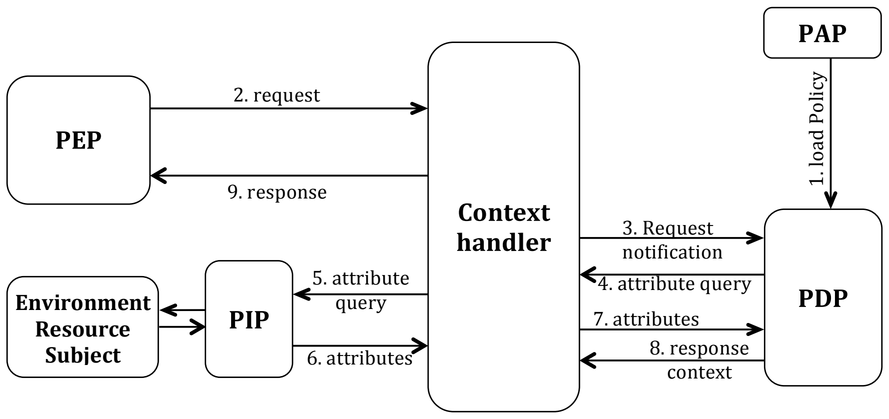


Figure 2.4: XACML major actors

XACML can serve as a structured policy language, but it also provides a framework and mechanisms to make authorisation decisions [20]. For example, XACML defines a number of policy-combining algorithms. These algorithms will help in making a decision in case of conflict between policies [68]. For example, 'Permit-overrides' is a combining algorithm. It will override all decisions if one of the policies permits the action. Other combining algorithms include deny-overrides and first applicable, which returns the result of the first matching rule or policy.

XACML has become the defacto standard for enterprise-wide, policy-based access control [52]. On the other hand, XACML does not natively support all types of access control models. For example, role-based access control (RBAC), which bases the policy on the role rather than on the user, was not supported in XACML, as there is no support for the notion of role. OASIS proposed that it is more efficient to have an extension to XACML than to build a new language. Therefore, the 'XACML Profile for Role Based Access Control (RBAC)' was proposed by OASIS [7]. It extends XACML with the notion of 'role', and provides a new authority for role assignment [5]. OASIS argued that role assignment should be out of the scope of the PDP, and therefore a new authority called 'Role enablement authority (REA)' was introduced to handle role assignment [5]. REA uses the role assignment policy set to decide whether the user is allowed to have the role or not [5]. The new extension also supports role hierarchy by including a reference to policy sets that belong to a junior role in the policy set of the senior role [5]. The RBAC profile supports both core and hierarchical RBAC, but it does not support separation of duty (SoD) constraints [7].

2.2.2 Access Control

The definition of access control is "exerting control over who can interact with a resource" [67]. It is a combination of *authentication* and *authorisation* [25]. Authentication aims to make sure that an entity has the right to claim a unique identity within a domain. A username is a common example of an identity and an organisation's directory of users is an example of a domain. Authorisation is a process that involves granting or denying permission to an authenticated entity to access a resource in a particular way (*e.g.* reading or writing a file) [25].

The specific access control method that an information system uses is defined by the access control *model* that it is based on. There are three fundamental access control models recognised in the literature: Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role Based Access Control (RBAC) [25]. RBAC, widely used in commercial and business settings, is the dominant model underlying access control proposals for process-aware information systems [69].

Role-Based Access Control

In RBAC, permissions are assigned to roles and users acquire permissions indirectly through being assigned as a member of a role. Thus, the approval for accessing a resource depends on the role that the subject is playing. For example, if an organisation's security policy states that only operation officers are allowed to access an operation room, then only users with the role 'operation officer' will be able to access the door of the operation room. RBAC supports user to role assignment, and permission to role assignment, where these assignment relations can be many-to-many. It also supports the inheritance of permissions via a role hierarchy, the user sessions concept through which roles can be selectively activated and deactivated, and the constraints to enforce the concept of separation of duties (SoD) to avoid fraud. Figure 2.5 shows the concept of RBAC.

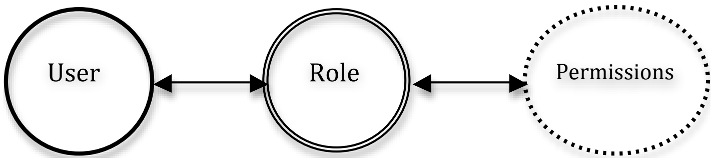


Figure 2.5: RBAC concept

RBAC received broad support as an authorisation model, and so many commercial products and academic research were based on the RBAC concept, but there was no standard that specified what an RBAC implementation should contain [30]. It was hard to provide a standard model for RBAC at the time as several requirements had been added, so the idea was to have different levels of RBAC, each with its own model [78]. Several papers were produced in order to try to standardise RBAC. NIST provided three RBAC standard models: RBAC0, RBAC1, RBAC2 [30]. RBAC0 (also known as Flat RBAC and Core RBAC) supports basic RBAC requirements. It provides user-to-role assignments, and permissions-to-role assignments, where these assignment relations can be many-to-many. It also supports the concept of users' sessions, which allow selective activation and deactivation of roles. Finally, users are able to simultaneously exercise permissions of multiple roles [78]. RBAC1 (also known as Hierarchical RBAC) supports RBAC0 and the concept of role hierarchy. RBAC2 (also known as Constrained RBAC) supports RBAC1 and the concept of separation of duties (SoD) [30].

RBAC implements the traditional notion of Separation of Duties (SoD) by making it possible to prevent incompatible roles from being assigned to or activated by the same user. SoD can be static or dynamic. With static SoD, a pair of roles are defined as incompatible and therefore a user may hold either one but not both. A common banking-based example is that no person can be assigned the roles of teller and branch manager at the same time. With Dynamic SoD, incompatible roles can be assigned to a user but the user cannot activate them both simultaneously. RBAC uses sessions to enforce this constraint, where for each user all activated roles exist within a session [79].

Task Based Access Control

Task Based Access Control (TBAC) [86] uses the notion of *tasks* to assign permissions to users. Each user is assigned a set of tasks he/she can perform, and each task is assigned a set of permissions [86]. Task-Based Access Control (also known as Activity-Based Access Control (A-BAC)) was introduced to provide the notion of "just-in-time permissions" [45]. Thomas and Sandhu [86] specify that active access control models are required for workflow management, and that TBAC establishes the foundation for research into the active access control models [45].

Just-in-time access control, (also known as *active access control*) means that access right assignment is separated from activation of access rights [86]. For example, in a business process if a task is assigned to the user to execute (assignment of access rights), the user will not be able to execute the task (activation of access rights) until the task prior to it in the business process is completed. So even if a user or a role is allowed to perform a specific task, the ability to activate the access right will be provided 'just-in-time'. Perhaps more importantly, the right will be deactivated immediately after the completion of the task. To achieve this, the access control model must work in conjunction with the workflow engine that regulates the progression of each executing process, from task to subsequent task. RBAC supports only passive access control, where once a role is assigned to the user and the role is activated, the associated permissions will be available [70]. This is appropriate for tasks that do not belong to any business process but the ability to selectively activate and deactivate permissions according to the execution state of a process instance is an important capability in workflow systems [45].

Figure 2.6 shows the concept of TBAC and how, even though permissions are assigned to users, they are activated only as the user begins to perform the relevant task. In this thesis we define task as it is identified in TBAC: a collection or a set of permissions.

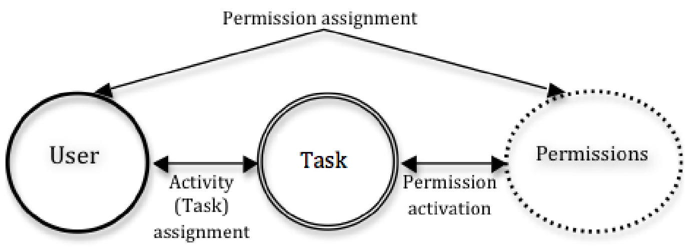


Figure 2.6: Task-based access control concept

Although TBAC provides access control based on task, and supports active access control, it does not provide support for passive access control, and does not support role hierarchy or permission inheritance [70].

2.3 Information Security and Business Process

After describing the domain of business process and having discussed the concepts of information security, this section will investigate the intersection between these two domains.

An important observation is that the professionals and specialists in the business process domain are usually not experts in information security [58], and neglect the integration of security [38]. Business process management (BPM) and information security management are developed and prasticed separately and often do not follow a coherent strategy [66]. Similarly, security goals are often not presented as part of the business process models [100].

Not having the security aspects in coherence with BPM could cause harm to the organisation. McCoy in his article “BPM and Security: Not Feeling So Good” [58] explained the problem of BPM without security, and showed that a business process needs security restrictions to avoid malicious users misusing it. So even if the process was designed to satisfy all customer requirements, a process without security could be used against the business. This is one of the reasons why security should be part of BPM. Tjoa *et al.* [87] state that security threats could be harmful to the business if these issues are not addressed. It is clear that both security experts and business process domain experts need to be able to identify a common abstract level where they can define their security goals together.

Rodríguez et al. [75] found that security policies are usually defined before the process modelling stage starts, which supports the idea of Wolter and his colleagues [100] that integrating security should start from the modelling level. Integrating security policies into the process model will help in forming ‘security aware BPM’, where security policies are more transparent for those who interact with these processes [100]. D’Aubeterre *et al.* [23] say that security should be considered from the requirements and data collection phase, and should be seen as a functional requirement. Enforcement of security policies can be considered a major factor for a business’s success [100]. In conclusion, BPM experts and security experts must work together to establish a platform that can link business processes with security on all aspects, including authorisation management and authorisation policies [100].

As the focus of this thesis is authorisation management, the following chapter will provide a more detailed review of the area of authorisation management in BPM, focusing on authorisation models for BPM as well as authorisation policy languages for BPM.

2.4 Summary

Business process management as a field is growing, and more systems are becoming process-aware. Business processes need to be security-aware. Security policies are an important factor to mitigate risk. In this thesis the security focus is on authorisation management. So, as a first step toward having security-aware BPM, we need a policy language that can represent authorisation policies related to BPM.

Enforcing the security policies requires control systems such as access control. The authentication part identifies the user; then the authorisation makes sure that the user would perform only what is allowed, based on the policy. It is basically making sure that all actions are in compliance with the authorisation policy. Authorisation management models need to be developed to support PAIS. Business process environments and process-aware systems have unique characteristics, so access control dealing with PAIS has specific requirements that need to be investigated. Although RBAC is a widely used model, it is not designed for business processes.

The next chapter will show a case study from a real-life situation to provide a motivating example scenario. It will investigate and identify the required characteristics that an authorisation model for BPM needs to support, as well as the characteristics needed for a BPM authorisation policy language. The chapter will also review the literature on authorisation models and authorisation policy languages. Based on these characteristics we will propose our new BPM authorisation model (Chapter 4), and our BPM security policy language (Chapter 6).

Chapter 3

Authorisation Management for Business Processes: Characteristics Analysis and Literature Review

In the previous chapter it was established that there is a need for authorisation management in business process environments, including an authorisation model and an authorisation policy language. As a first step, it is necessary to perform an analysis to define the characteristics needed for such a model and language. A review of the literature is also needed to identify if a suitable authorisation model and authorisation policy language exists based on the characteristics identified. Moreover, an example case scenario will help throughout the thesis in understanding the need for the model and the language. It also simplifies the understanding of the new features introduced in this thesis.

This chapter will start by introducing an example *case scenario* developed using a real-life example from a complex, security-sensitive environment that uses business processes across multiple systems. We conducted interviews with experts and stakeholders to gather information that is used to build the case scenario. We interviewed the buildings facilities asset manager, the operational manager, and the security compliance manager of this security-sensitive and complex environment.

Based on the literature and the case scenario this chapter then presents a *characteristics analysis* for the needed authorisation management system. This includes a characteristics analysis for the authorisation model to be used in a business process environment, as well as the characteristics needed for an authorisation policy language in business process environment.

The chapter then provides a literature review on business process authorisation management using the identified characteristics. It will first review the literature for the authorisation models that are designed for business process management environments. The review establishes that none of the current models support all the needed characteristics. It will then review the literature for the authorisation policy languages, showing that none of the available languages are suitable to represent authorisation policies for business process management environments.

3.1 Example Scenario

In order to illustrate the need for authorisation management in the business process environment, and to help identify the requirements for an authorisation model for workflow, this section describes an example business process that runs across a number of different application systems. The scenario deals with the process of fixing a pump malfunction in an air-conditioning system in a high-security facility such as an airport, chemical factory or prison. In such complex environments, a single process might run across multiple inter-networked systems, and each of these systems will typically have its own security policies and associated access restrictions. Coordinating these in a coherent way according to an overarching process model is difficult. Processes that run across multiple systems are an increasingly common feature of the modern business landscape and they represent a challenge for workflow security and access control in particular.

The example process is triggered by receiving an electronic pump malfunction notification, and it ends with the closing of the work order. The organisation's security policy and business rules require separation of duty and binding of duty constraints to be observed in the execution of the process to minimise the possibility of fraud and other security violations. There are three systems involved in this scenario. The first system is the asset management system, which is used to manage the building's assets including tracking maintenance jobs and the warranty status of equipment. The second system is the enterprise building integrator, which is used to actively control and manage a range of electronic systems in the building, such as heating ventilation and cooling systems, lighting, and elevators. The third system is the physical access system, which is used to enforce physical access restrictions through the building's doors. The roles involved are, the Coordinator (building facility officer), the Manager (coordinators' supervisor) and the Contractor (who is contracted to fix any malfunction when needed).

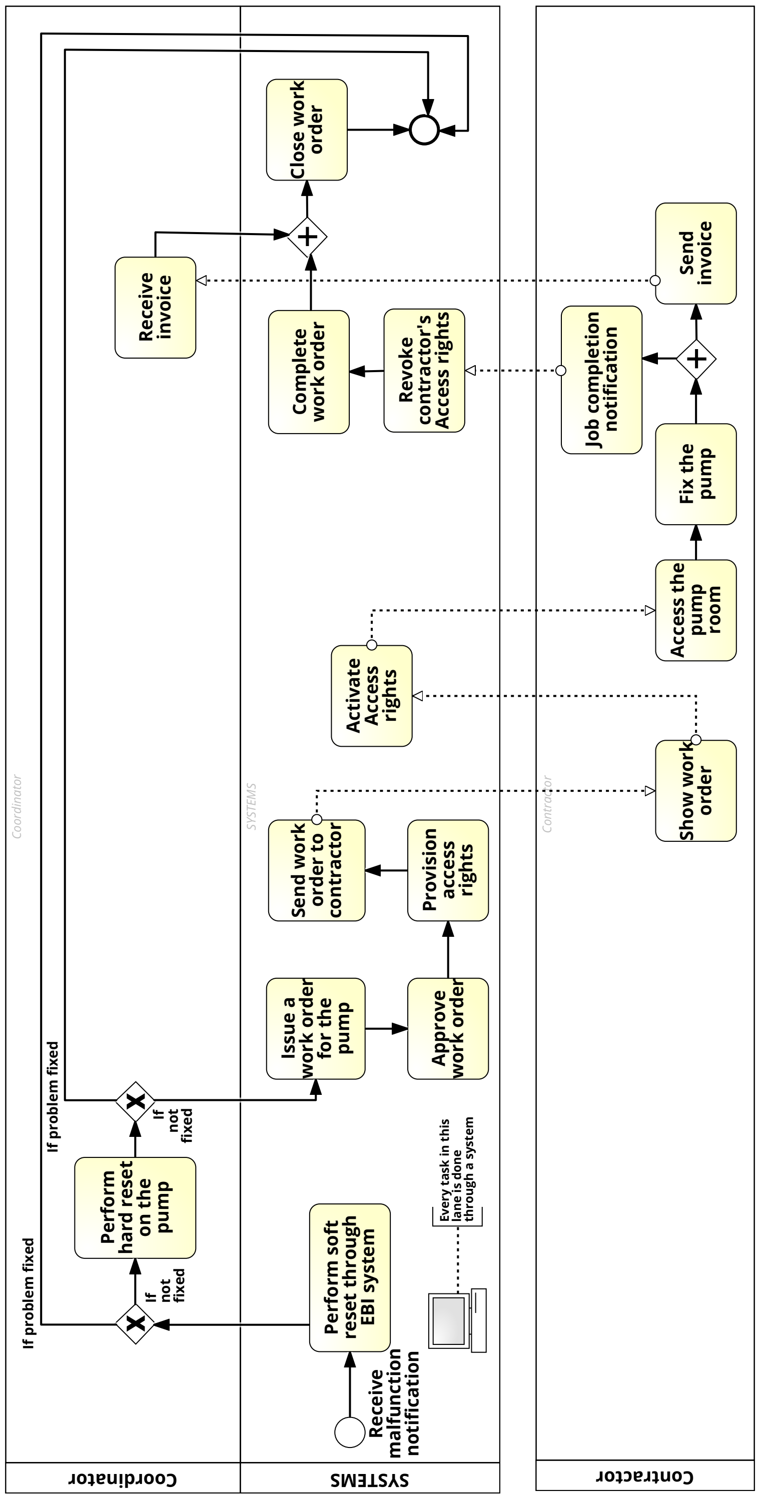

Coordinator
SYSTEMS
Contractor
Receive malfunction notification
Perform soft reset through EBI system
Every task in this lane is done through a system
If problem fixed
If not fixed
Perform hard reset on the pump
If problem fixed
If not fixed
Issue a work order for the pump
Approve work order
Provision access rights
Send work order to contractor
Show work order
Activate Access rights
Access the pump room
Fix the pump
Job completion notification
Send invoice
Revoke contractor's Access rights
Complete work order
Receive invoice
Close work order

Figure 3.1: Business process model for fixing pump malfunction

When a pump malfunction notification is received through the enterprise building integrator, the coordinator responds to the notification. First, he would investigate the nature of the fault and attempt to electronically reset the pump through the enterprise building integrator system. If this does not correct the fault, he will physically inspect the pump and perform a hard reset. If the problem is still not fixed then the coordinator needs to take the pump offline, create a maintenance job in the asset management system and issue a work order. The work order then needs to be approved by a manager and sent to the contractor, again via the asset management system. The physical access permissions that the contractor will require to access the pump room are then provisioned in the physical access system, but without activating them.

Upon the arrival of the contractor he/she needs to show the work order before the required access rights are activated in the physical access system. The contractor is issued with an electronic access token which he uses to pass through controlled access doors on his way to the pump room. The contractor will fix the pump and then notify the coordinator, who issued the work order, that the issue has been resolved. The coordinator will then need to revoke the access rights from the contractor in the physical access system, and will fill in and complete the work order through the asset management system. The contractor's company will then send the invoice to the coordinator, which he will include in the work order, then close it. Figure 3.1 shows the process model for fixing the pump malfunction.

As can be seen in Figure 3.1, a coordinator cannot perform a 'soft reset' on the pump unless a malfunction notification is received. If both 'soft reset' and 'hard reset' fail to solve the problem, a work order is issued. Only users with the role 'coordinator' can issue the work order. The approval of that work order should be done by a different user with the role 'Manager'. To avoid fraud, no one can perform both 'issue work order' and 'approve work order' for the same work order (same instance). So, users can issue work orders (by activating the role coordinator), and can approve work orders (by activating the role manager), but no user can perform both tasks for the same instance. A user with the role 'contractor' needs to show the work order to gain access to the pump room. Once the issue has been resolved the user who fixed the problem will notify the user who issued the work order. This notification will result in revoking the access rights granted to the contractor. The user who created the work order is the only one allowed to close it, and will be able to do so only after receiving the notification and the invoice. To avoid fraud it is important to have a SoD between the roles 'coordinator' and 'contractor'. So, it is not allowed to have both roles assigned to the same user.

The business rules and authorisation policies associated with this process are:

1. The task ‘soft reset’ should not be performed unless a malfunction notification was received.
2. Only the role ‘coordinator’ is allowed to issue work orders.
3. Contractors should be granted access only after showing the proper work order documentation.
4. A work order can be closed only after receiving both work-order completion and an invoice.
5. Only the person who issued a certain work order is allowed to close it.
6. No person is allowed to perform ‘issue work order’ and ‘approve work order’ for the same ‘work order’.
7. No person is allowed to have both roles ‘coordinator’ and ‘contractor’.

3.2 Characteristics Analysis

In this section we will analyse and identify the characteristics needed by the authorisation model to be able to support business process environments. Based on these characteristics we will then identify the characteristics of a policy language to be used with a business process authorisation model. In order to identify these characteristics, we will use the literature and supporting examples from the case scenario in Section 3.1.

3.2.1 Authorisation Model Characteristics

The provisioning of access rights based on roles has a number of advantages in a workflow setting. Most notably, it reduces administrative effort in allocating access privileges to users because an administrator can grant a complex and carefully selected set of rights to a user simply by granting them membership of appropriate roles. This is more efficient than directly granting each user the same set of privileges. If a process changes and new access rights are required, these need to be granted only once, to the appropriate role, rather than individually to each affected user. Thus, it is practical to use a role-based resource allocation pattern with workflows [86, 12]. Moreover, the use of role-based provisioning in the access control model allows the system to reflect the organisational structure and the delegation of authority to various roles and positions, as indicated by the organisational chart [70].

Separation of duties (SoD) and binding of duties (BoD) are necessary aspects to consider in an access control model for business processes [9]. RBAC supports SoD on a role level, but in a business process context SoD can also apply on a task level (which is not supported by RBAC), where different tasks within a business process instance cannot be done by the same person. Sometimes the restriction is the other way around, where two or more tasks should be done by the same person. In this case it is called 'binding of duties' (BoD) [14]. For example, the scenario states that 'only the person who issued a certain work order is allowed to close it'. This is a binding-of-duties restriction. SoD and BoD are used to avoid fraud and misuse, and to make sure that no conflict of interest appears using the access control privileges [70].

In RBAC, SoD is addressed using sessions [39]. In BPM, special SoD and BoD constrains are required that apply only within the same instance. To solve this issue the authorisation model should support 'instance-level restrictions' (IR). In the example from 3.1, it was necessary to prevent the same user from executing the tasks of 'issuing work order' and 'approving work order' for the same 'fixing pump malfunction' process instance (or in terms of roles, one would like to avoid that the same user be the issuer and the approver of the same instance). So Adam may be the approver for Carol's work order, and may himself be the issuer of a work order for a different process instance, which must be approved by another coordinator. This cannot be enforced through session-based restrictions because Adam may be at the same time issuing a work order for a process instance and approving Carol's for another process instance, and that is acceptable provided these roles are being played in different maintenance job instances (cases) [95]. So, standard RBAC and the notion of session are not enough when there is a SoD or BoD restriction on a process instance level.

The example scenario shows an example of a role-level SoD, where no person is allowed to have both roles 'coordinator' and 'contractor'. To support this restriction, the model should support SoD on role-level. The example scenario also shows examples of an instance-level restriction where no user is allowed to perform both tasks 'issue work order' and 'approve work order' for the same instance. Another instance-level restriction is where only the person who issued the work order is allowed to close it *i.e.* BoD on an instance-level. To be able to satisfy such rules, it is necessary that the access control model support separation of duties and binding of duties on a process instance level.

Using 'tasks' in the task-based access control (TBAC) model helps to group permissions and to reduce the need to interact with each permission by itself, thereby reducing the administrative overhead [86]. Similar to how RBAC made it easier to deal with roles rather than users, TBAC makes it easier to deal with tasks rather than permissions, particularly since a bundle of permissions are often required to perform a unit of work at an application level. Therefore it is useful to use task-based allocation, to allocate tasks to authorised subjects in an access control model for a business process environment [9]. A task groups a set of related permissions, where each permission is a pair made up of an action and a resource. For example, the task *close work order* includes the set of permissions (access, asset management system), (read, work order), (read, invoice), and (edit, work order).

Combining task-based assignment in conjunction with role-based will help make access control more efficient and easier to use [70, 101]. For example, if a set of tasks imply a potential risk of fraud if performed by the same user, authorisation to perform such tasks can be assigned to different roles at workflow design time [101]. Therefore, a "task-based assignment of tasks according to a user's role enforces separation of duty security requirements as long as a user is restricted to a single role within a workflow instance" [101]. However, although having the combination of task-based and role-based reduces management overhead, it is not enough for business process environments, as in real life business process environment users are allowed to have multiple roles active at the same time. Therefore, to avoid fraud, instance-level restrictions are needed to support the SoD of such type. Tasks that imply a potential risk of fraud if performed by the same user will have an IR stating that they cannot be performed by the same user for the same instance, regardless of the role that the user has active at the time.

An access control model for business processes should support active access control as well as passive access control [86, 45]. In the example scenario, although the access right to perform the task 'soft reset' for the pump is assigned to the role 'coordinator', we do not want the coordinator to be able to reset the pump, unless there is a valid reason. So the permission is not activated until the pump malfunction notification is received, and the right is revoked as soon as the task is over. Without active access control, permissions will be turned on either too early or too late, and might remain available long after the task is over [86]. Another example from the same process is the technician's authorisation to access the pump room, which is activated only once a work order is received; otherwise, technicians are not allowed to access this room. In other words, authorisations on tasks depend on the outcome of another task [9].

Organisations are confronted with difficult challenges in managing the security of their information systems due to new rules and regulations imposed by governments. These rules aim to address a diverse range of issues including improving privacy protection, ensuring greater accountability for business practices and addressing new threats linked to terrorism. The management of information security used to be primarily the concern of the IT Department. It is now an enterprise-wide issue and organisations are recognising the need for enterprise-wide approaches to information security management. In particular, this means that an authorisation system that can regulate access to all of an organisation's information systems and resources according to a consistent set of security policies is required [65]. Against this backdrop, we argue that an authorisation system for workflow should be conceived as part of a larger enterprise-wide authorisation system, not one that is internal to the workflow system itself. As a consequence, it is important that the model be able to support both workflow and non-workflow tasks. Workflow tasks means tasks that are part of a business process, while non-workflow tasks are tasks that are not part of a business process and performed outside of the workflow system. This approach will assist by not adding further to the already large number of proprietary authorisation systems that an organisation must manage. Instead, it offers a path to consistent access enforcement based on a single set of organisation-wide security policies.

In summary, a business process access control model should support the following characteristics:

- Role-based access control.
- Static and dynamic separation of duties.
- Active access control.
- Instance-level restrictions.
- Task-based access control.
- Supports workflow and non-workflow tasks.

NIST-RBAC [30] supports the first two characteristics. So, an access control model for workflow should extend RBAC to support the other characteristics. Moreover, in a typical organisation that uses process-aware information systems, some tasks will not belong to any workflow. Therefore, it should be an enterprise-wide access control model that caters for the non-workflow tasks as well as the workflow tasks.

3.2.2 Authorisation Policy Language Characteristics

With its focus on tasks and their controlled, sequenced execution, an authorisation model for business processes introduces a range of capabilities not found in the standard RBAC model [84]. Similarly, an authorisation policy language for business processes requires specific characteristics. This section will identify the important concepts and constraints that such a policy language should be able to express, by describing the functionalities that business process authorisation models support. This section will use supporting examples from the scenario in Section 3.1. The authorisation policy language has to express the specific types of rules and constraints that the authorisation model allows; therefore the analysis of the language characteristics is based on the characteristics that the authorisation model should support. We will use the authorisation model characteristics identified in the previous section, and for each characteristic we will identify the characteristic that the language should support in order to express the authorisation model characteristic.

As explained in Section 3.2.1, RBAC is an important concept to be supported in a business process authorisation model [50]. Support for RBAC among the published business process authorisation models including [95] and [70] is widespread. So, for a policy language to be able to express RBAC concepts, it should support the notion of user, role, and permission. As can be seen in the example from Section 3.1, any user with the role 'coordinator' can perform the task 'soft reset'. The decision is based on the role this user has activated. For a policy language to be able to express such a policy, it should be able to represent the user's role (*i.e.* coordinator), the operation (*i.e.* soft reset), and the permission (*i.e.* permit). Moreover, the policy language should have the ability to represent Separation of Duties (SoD) constraints as a tool to assist in preventing fraud [29].

Tasks are a fundamental concept in business process management. They are the building blocks of business processes, so business process authorisation models such as [70] and [95] typically focus on extending RBAC with the notion of a task. Reflecting this, a policy language for business process access control should have the ability to express the notion of 'tasks'. The example in Section 3.1 shows that the requests are to perform a 'task', rather than to acquire a permission. So it is important to be able to express tasks as well as permissions.

An important functionality that is supported by the more expressive business process authorisation models (such as [95]) is history-based restrictions between tasks on the instance-level, which we refer to as 'Instance-level Restrictions (IR)'. For the policy language to be able to express instance-level restrictions, the first step is to be able to distinguish between instances. Some authorisation models such as [95] and [96] support the notion of 'task instance', which allows different execution instances to be distinguished. So, a policy language should be able to represent a 'task instance'. The language should also have the ability to represent the actual 'instance-level restrictions'. The example in Section 3.1 states that only the user who requested a work order is allowed to close it. The restriction should be applicable only within the same work order. Moreover, in order to enforce this condition (an IR restriction), the language should have the ability to retrieve history-based information on an instance level [96], such as who issued a specific 'work order'.

In summary, the key concepts that an access control policy language for business processes should be able to represent are:

- Users.
- Roles
- Operations.
- Tasks.
- Tasks instance.

- Instance-level restrictions.
- Role-level-SoD restrictions.

The RBAC profile of XACML supports representation of the first three characteristics. Therefore, the proposed policy language need to extend RBAC-XACML to be able to express the other four characteristics.

3.3 Authorisation Management for BPM: Literature Review

Access control as a major security requirement has been a concern for system engineers from an early stage. The issue is although it is an important aspect, it is often neglected in BPM, and typically comes as an afterthought. Wolter *et al.* [100] talked about 'Access Control' and tried to map it to process modelling elements as part of their effort to integrate access control to process modelling. They defined three elements in the access control decision: "The subject that wants to access a target, the target itself, and an operation that can be performed on this target" [100]. Subject and target can both be represented as the object in process modelling, while operations are represented as the interaction within the model itself. This solution only provided a way of including the authorisation policies in the process model, but did not provide any enforcement mechanism.

As explained earlier, access control consists of 'authentication' and 'authorisation'. According to [33] it is safe to say that authentication has been largely solved with single sign-on architectures. Ghalimi [33] states that authorisation (also known as entitlement) is not yet solved in terms of BPM. Authorisation management in general is concerned with: writing an authorisation policy, storing the policy, managing the policy, and enforcing the policy [13, 76, 81]. In this thesis our focus is on the language for the authorisation policy (writing the policy), and on the authorisation model that will enforce the policy.

The following section will review authorisation models that are based on RBAC and designed for a business process environment. As explained in Section 3.2, it is important that the model supports RBAC, but the RBAC model by itself is not enough to support access control for business process systems. Section 3.3.2 will review authorisation policy languages for a business process environment and will investigate the suitability of current policy languages and their extensions.

3.3.1 Authorisation Models for BPM

RBAC is widely used and accepted as an authorisation model. Using the 'role' concept in RBAC minimises management overhead. Unfortunately, as explained in Section 2.2.2, RBAC was not designed to deal with workflows. It does not support, for example, the idea of active access control. This is a significant shortcoming, as Alturi and Huang argue that "a suitable authorisation model for workflows must ensure that authorisation is granted only when the task starts and revoked as soon as the task finishes" [9]. TBAC [86] was proposed as a solution that provides active access control, but TBAC lacks the support for passive access control, and does not support the notion of 'role', which, as explained, is an important concept to minimise management overhead.

The Workflow Authorisation Model (WAM) [9], was also introduced as a model that supports active access control. According to [9], temporal authorisation models are not suitable because they only provide a static fixed start time and end time, which may not be in synchronisation with the business process. To achieve active access control, WAM depends on objects moving from one task to another where access control is affected by granting or revoking a subject's access to this object [9]. WAM is similar to RBAC in that it provides support for role-based authorisation including separation of duties. It is also similar to TBAC in that it supports active access control [9]. On the other hand, unlike TBAC, WAM does not deal with task-based authorisation, which is an important aspect in workflows, since tasks are the building blocks of workflows. The Flexible Workflow Authorisation Model (FWAM) [105] was introduced as an enhancement to WAM. Both WAM and FWAM still lack the support for task-based authorisation. Moreover, they both lack the ability to deal with non-workflow task authorisation. In a normal organisation the authorisation system usually handle a mix of workflow and non-work flow requests.

The Service-Oriented workflow access control model (SOWAC) [104] is similar to WAM in the way that it supports both role-based access control and active access control [104]. SOWAC also supports task-based authorisation, but it still suffers from the shortcoming of not supporting non-workflow tasks [104]. Secure Role-Based Workflow Model (SRBWM) [45] another model introduced to extend RBAC [77] with active access control. In addition to supporting RBAC and active access control, SRBWM also supports task-based authorisation. However, SRBWM does not cater for non-workflow tasks.

Another shortcoming shared by all the models we have mentioned is the lack of the ability to differentiate between task instances. As briefly explained in 2.2.2, in BPM each task execution is considered as an instance of the task. Not distinguishing between task instances means that a permission to execute a task would imply the permission to execute any instance of the task. In BPM we might allow Adam to perform a task (*e.g.* approving a work order), unless he also performed another task for the same instance (*e.g.* issuing a work order). The RBAC SoD, which uses session, is not useful in this situation.

In workflow applications, it can be difficult to apply the SoD concept through the notion of sessions. SoD restrictions are more commonly defined within the context of a business process instance rather than as simple rules concerning role assignment or activation 'at the same time' [95]. To solve this issue, 'instance-level restrictions' (IR) are needed. A process instance is an individual execution of the process (also known as a *case*). Consider a business rule that states that the person who approves a work order must be different from the one who issued it, but it is permitted for a person to be both an issuer and approver of work orders as long as they do not perform both functions for the same process instance. This rule can not be enforced through RBAC's traditional notions of static or dynamic SoD, since a user may need to execute multiple process instances within a session and thus must have the roles that hold the incompatible permissions active simultaneously. Therefore, in an access control model for workflow, it should be possible to authorise a user to perform a task in one process instance but deny the same user the permission to perform that task in another, within the same session.

Bertino *et al.* [11] claim that they are the first to discuss the issue of SoD on an instance-level. They propose a solution for SoD in workflow applications. It does not use history based restrictions, but rather identifies the set of all possible assignments of roles and users that do not violate the SoD policy and other constraints, prior to workflow execution, and only permits assignments from this set. This solution requires prior knowledge of the workflow specification and its tasks. Furthermore, it focuses solely on SoD within workflows, while some examples of SoD are not related to workflows. This solution cannot cater for such an issue. Chadwick *et al.* [18] proposed Multi-session SoD (MSoD), which focuses on solving two issues: the issue of long-term SoD and the issue of instance-level SoD. It uses a new concept called 'business context', rather than user sessions. This solution solves the issue of instance-level restrictions. On the other hand, it requires pre-definition of business contexts and identification of SoD policies that belong to each business context. MSoD does not provide an active access control mechanism, and does not provide task-based authorisation. WSession [14] also requires a pre-identification of all conflicting roles, users, tasks, and privileges. WSession does not support active access control, does not support task-based authorisation, and does not cater for non-workflow tasks.

W-RBAC [95] was introduced as an access control model that extends RBAC and is specifically designed to support workflow systems. As can be seen in Figure 3.2, W-RBAC extends RBAC with the notion of a case (an instance of executing a business process) and a three-way relationship called 'doer' between a user, permission and case. The model does not differentiate between 'tasks' and 'permissions'. Comparing W-RBAC with previously reviewed models such as TBAC, SOWAC, and SRBWM shows that W-RBAC has the advantage of supporting and capturing the notion of instance-level restrictions. On the other hand, W-RBAC does not support task-based authorisation, and does not support 'active access control'. Another shortcoming is that W-RBAC depends on the workflow engine to control the process, while W-RBAC will only provide the list of users allowed to perform the task.

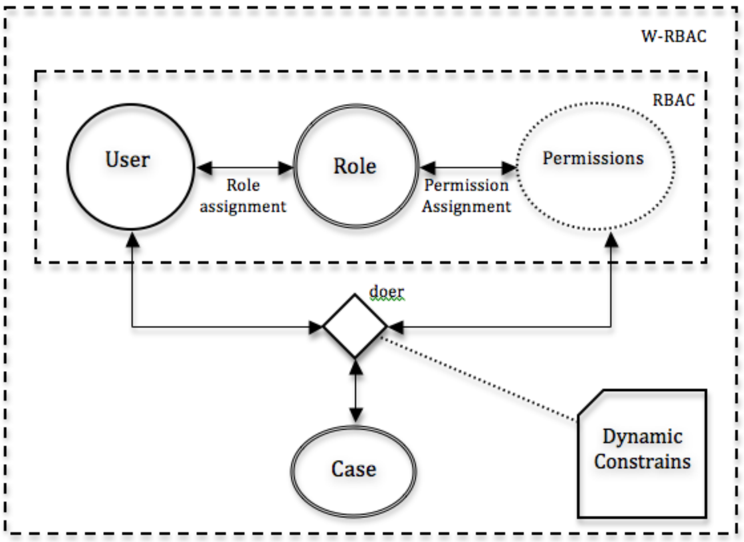


Figure 3.2: W-RBAC concept

The Task-Role-Based access control (T-RBAC) model [70] was introduced in 2003. It is similar to W-RBAC in supporting instance-level restrictions. It has the advantage over W-RBAC by supporting active access control and task-based authorisation. It also supports role-based access control and caters for workflow and non-workflow tasks. As can be seen in Figure 3.3, T-RBAC uses the concept of RBAC and adds the concept of a task. T-RBAC supports SoD, and the notion of sessions. On the other hand, T-RBAC does not support certain types of instance-level restrictions because it does not record task execution history and therefore cannot support history-based restrictions. For example, T-RBAC cannot support SoD on instance-level where one of the tasks (in the SoD policy) has already been completed. It can support SoD only for task instances that are still active. Moreover, it does not support BoD at an instance-level. For example, a rule stating that 'only the person who issued a certain work order is allowed to close it' can not be enforced using T-RBAC.

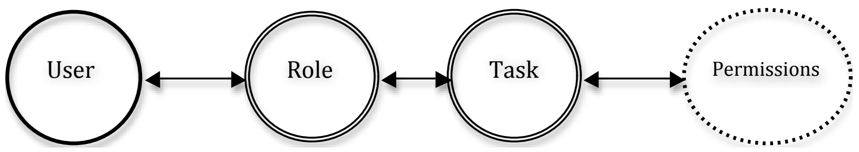


Figure 3.3: T-RBAC concept

Other access control models designed to support workflow systems are based on RBAC. For example, the proposal by Weber *et al.* [97] extends the core RBAC model with features to support workflow. The focus of their work is on providing a balance between security and flexibility in workflow systems, with an emphasis on how a workflow can be dynamically resequenced or adapted at runtime. In their work they have stated that they did not focus on dynamic SoD or dynamic constraints in general. AW-RBAC [50] also extends RBAC and focuses on adaptive workflow systems, but does not focus on task level constraints, and does not provide support for instance level restrictions.

Strembeck and Mendling [84] present a generic metamodel that can be used to extend workflow languages to support access control requirements in workflow systems. The metamodel is designed to be used to extend workflow languages. Although it supports design time and run time constraints (including instance level restrictions), it is designed to build on top of an available workflow system and becomes part of the workflow system. This model is not intended to be used as an independent enterprise-wide authorisation system as it is not designed to cater for non-workflow tasks. It means an organisation cannot use this model by itself: It will also need another authorisation model to handle authorisation requests for tasks that are not in a workflow and not managed by the workflow system.

Table 3.1 compares the most relevant authorisation models from the literature. The comparison criteria are the characteristics identified in Section 3.2.1. As can be seen in the table, none of the models we have reviewed support all the needed characteristics. The table shows the shortcomings of each of these models. It shows the need for a unified BPM authorisation model that is designed to satisfy the required characteristics. The following chapter will introduce a new authorisation model that is designed to satisfy the required characteristics.

Table 3.1: Comparing different authorisation models

	Compression criteria	RBAC*	WAM	FWAM	SRBWM	T-RBAC	W-RBAC	MSoD	WSession	SOWAC	Str. & Mend.	AW-RBAC
1	An independent access control model	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	No	Yes
2	Supports Role-based access control	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
3	Supports Task-based authorisation	No	No	No	Yes	Yes	No	No	Yes	Yes	Yes	No
4	Supports Active access control	No	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	No
5	Supports SoD on instance-level	No	No	No	No	Yes	Yes	Yes	Yes	No	Yes	No
6	Supports History-based SoD on IL**	No	No	No	No	No	Yes	Yes	No	No	Yes	No
7	Supports BoD on instance-level	No	No	No	No	No	Yes	Yes	Yes	No	Yes	No
8	Support non-workflow tasks	Yes	No	No	No	Yes	No	Yes	No	No	No	Yes
9	Support workflow tasks	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

* The RBAC model used in this table is NIST RBAC2

** IL= Instance-Level

3.3.2 Authorisation Policy Languages for Business Processes

In the past few years there have been a number of investigations and studies on finding a formal language for authorisation policies. The formalism of languages differs. For example, some languages are logic-based languages, while others are rule-based languages.

Logic-based languages such as SecPAL [10] and OPL [3] are usually expressive. However, “the trade-off between expressiveness and efficiency seems to be strongly unbalanced” [76]. For example, most of the logic-based languages do not provide a conflict resolution solution. Some languages such as Rei [44] can detect conflicts only during run time, and do not offer a design time conflict resolution method, which is more practical. Moreover, according to [76] “often the people specifying the security policies are unfamiliar with logic based languages” [76]. For that reason recent proposals tend to express authorisation policies as rule-based documents [76].

In rule-based languages a set of rules are used to describe the content. It is a “declarative language, which binds logic with rules” [106]. In rule-based languages policies and regulations are abstracted into a set of rules, usually in ‘condition’ ‘action’ rule statements [1]. Policy Description Language (PDL) [54] and Ponder [21] are examples of rule-based languages. Both languages use the event-condition-action paradigm, and they both map a series of events into a set of actions. PDL can be used to express any type of policies, not only authorisation policies. It describes policy rules as actions that can take place if the condition was true. PDL cannot detect or solve conflicts between policies. PDL “do not support the composition of policy rules into roles, or other grouping structures” [21].

Ponder distinguishes itself by pre-defining the type of policies to be used, and provides a structure for each of these policies. This pre-definition method makes it hard to add to Ponder and to widen the language capabilities. According to Phan *et al.* “the main problem with the Ponder language is its lack of generality. The language itself is more like a collection of different groups of features rather than a well-designed set of constructs that can be used to describe any behaviors in general. That is, instead of having a common set of language constructs that can be used to describe different types of policy, Ponder has five basic policy types and four composite policy types each with different syntax” [73]. Ponder policies can be mapped into XML representation. Ponder is easy to enforce, but it is complex to extend, and does not provide the ability to “analyse policies relating to entities described at different levels of abstraction” [19].

XACML [62] is another rule-based language. As explained in Section 2.2.1, XACML supports conditional authorisation policies and policies with external post-conditions [20]. XACML provides the ability to be extended. One can use attributes, as well as defining new policy sets and new specialised PDP to extend XACML for a specific purpose.

Unfortunately none of these languages are suitable for business process authorisation policies. The languages, as they are, do not provide a representation for task instances, nor do they have the ability to represent instance-level restrictions. For this reason it was required either to propose a new language, or to extend a language that already exists, to be able to support authorisation policies for business processes. As an example, standard XACML does not support RBAC. Later OASIS provided a new profile as an extension to XACML, to be able to support RBAC. In our case we need an extension to support business process authorisation policies.

To the best of the author's knowledge, currently there is no published work that aims to extend XACML to support business process policies. There are several published works that extend XACML to support different models but none of them focus on 'business processes'. For example, Wolter *et al.* in [103] developed a XACML customised profile that supports the RBAC concept, mandatory access control, and permission-based separation of duty policies. The work does not take into consideration the special requirements that 'business process' policies need such as 'tasks', and it does not extend XACML to support business process policies. The main focus of the paper was to propose a model transformation framework that focuses on deriving security policies from the process model. The framework, which uses "seBPMN" (introduced as a process modeling language enriched with access control annotations) as the modeling language, and uses XACML as an example policy language, is specific to "seBPMN" and it does not extend XACML.

Another work that also focuses on translating business process components into an XACML terminology is [102]. It proposes an approach that automatically derives XACML authorisation policies from BPMN model annotations. It provides a model-driven extraction of the policies based on a mapping between BPMN (a policy modeling language) and XACML meta-models. The method requires extending the BPMN semantics. It is limited to BPMN only. For example, they derive the XACML subject role attributes of the policy target from lanes and nested lanes, which is a feature of BPMN. Sinha *et al.* [82] also propose a method of translating security requirements into XACML, by making use of its obligation feature. The paper focuses on web services only.

Achim *et al.* in [15] presented "SecureBPMN", and Stackelberg *et al.* in [63] presented "BPMS". They are both process modeling languages that extend BPMN with security annotation to represent security constraints for business processes. Both papers propose methods for translating their extended security requirements into XACML security policies. Both proposals are limited to the BPMN language. Instead of enriching XACML to support business process, they aim to translate the BPMN extensions to the already known XACML syntax. Vijayant *et al.* in [24] focus on extending XACML to deal with decentralised policies of web services, rather than focusing on ‘business process’ requirements.

All the reviewed proposals are limited to the specific process modeling language they propose. Moreover, none of the reviewed work presents an authorisation policy language (or extends one) that can be used in a business process environment.

3.4 Summary

The example scenario, developed from a real-life complex environment, showed the need for authorisation management in business process environments. It showed how such environments are complex and require specific models that are designed to support process-aware systems.

Authorisation management requires both an authorisation model and a policy language. In order for an authorisation model to be sufficient in business process environments and to support process-aware systems, it should support specific characteristics. Section 3.2.1 identified these characteristics. Moreover, for a policy language to work coherently with an authorisation model, it needs to be able to represent and support specific characteristics, which were identified in Section 3.2.2.

Section 3.3.1 reviewed the BPM authorisation models in the literature. The review showed that there is no authorisation model currently available that supports all the needed characteristics. Thus there is need for a new authorisation model designed for BPM. RBAC, being widely accepted and used, can form a good base on which to build the authorisation model.

Section 3.3.2 reviewed the literature on authorisation policy languages with regard to the needed characteristics. The review showed that there is no current security policy language that is designed for BPM. XACML, being a widely used and accepted policy language, is a good base to build on as a language that suits the need.

The gap that needs to be addressed is the lack of a security policy language that is designed to support business processes and the policies related to them, for example, having the ability to support policies that include instance-level restrictions. Moreover, there is a need for a unified authorisation model that is designed to control the authorisation requests of a business process system. The system should have the ability to support different types of instance-level restrictions, including history-based, and at the same time to not be part of the workflow system, but be an independent authorisation system. Therefore, there is a need for a new authorisation model and a new policy language. The new model and the new language should be designed to satisfy the identified characteristics.

The following chapter will introduce a new authorisation model that is designed to satisfy all the characteristics identified in this chapter. Chapter 6 then introduces a new authorisation policy language that can be used with the BPM authorisation model. The new language is designed to address the characteristics identified in this chapter.

Chapter 4

Business Process Task-Role-Based Access Control Model (BP-TRBAC)

In the past few years there has been an increased adoption of business process systems [26]. As explained in Chapter 3, for an authorisation model to be able to work with business process systems it requires the ability to support a specific set of characteristics, of which instance-level restrictions are a notable example. This has led to the need for an authorisation model that is specifically designed to work with process-aware information systems (PAIS) [80].

RBAC is a widely accepted and adopted access control model [69], but when used in the context of process-aware information systems in its standard form (*i.e.* NIST Standard RBAC [30]) it has an number of shortcomings. For example, RBAC's use of sessions to enforce dynamic separation of duty is a poor match for PAIS because a process execution instance (which may span multiple user sessions) is a more applicable context in which to enforce constraints over which user can perform what task. The dynamic nature of process orientation demands an access control model that is similarly dynamic. So-called 'active access control' allows full advantage to be taken of the contextual information that is available in a process model in order to enforce security constraints through the various steps or *tasks* in a process. To do this it is necessary to differentiate between task assignment and privilege activation [86], because even though a user may be authorised to perform a particular type of task, it is necessary that he/she performs only a specific instance of that task within an executing process when it is the right time to do so, and not before. This means that privileges should be activated 'just in time' and then deactivated. In contrast, the standard RBAC model activates a role's complete set of privileges as soon as the user activates the role [70].

Because of the shortcomings in RBAC there have been a number of access control proposals that are designed to work in a business process environment. Some of these models support active access control such as TBAC [86], while other models can differentiate between process instances when enforcing constraints such as W-RBAC [95]. However, to the best of the author's knowledge there is no model that supports all the required characteristics that we identify in Section 3.2. This chapter will introduce the 'Business-Process Task-Role-Based Access Control model (BP-TRBAC)', a new unified authorisation model that is specifically designed to provide authorisation services to business process systems. It works cooperatively with a PAIS but it is an independent system that is also intended to perform authorisation control for applications and functions that are not process-aware. The model is unified in the sense that it combines important characteristics and capabilities that have been proposed separately in other workflow authorisation systems. However, prior to this proposal, these capabilities have not been brought together in a single independent model.

BP-TRBAC is a new unified model that extends NIST RBAC [30] to support the needs of a business process environment that included task-based authorisation, active access control, and instance-level restrictions (including SoD and BoD on an instance level). It also caters for workflow and non-workflow tasks (as distinguished in Chapter 3), and is designed to be an independent authorisation system that can interact with the workflow system.

The rest of the chapter is organised as follows: it starts with an overview of the new BP-TRBAC model in Section 4.1, then it describes the conceptualisation and annotation of the model in Section 4.2. Section 4.3 provides the formal description of the model showing how BP-TRBAC achieves all of the identified characteristics. In Section 4.4, the example scenario from Chapter 3 is revisited to show how the business and security rules are enforced using BP-TRBAC. A discussion concerning the points of strength and weakness of the BP-TRBAC model is presented in Section 4.5. This is followed by a review comparing BP-TRBAC to other models from the literature in Section 4.6. The chapter then concludes in Section 4.7.

4.1 BP-TRBAC

Role-based access control (RBAC) was introduced to minimise administration overhead. RBAC defines roles for users, granting access rights based on the roles the user possesses. In RBAC, users acquire access rights from the roles they activate. Figure 4.1 shows the basic RBAC concept. It can be seen that users are assigned membership of roles and that each role has permissions, instead of having permissions assigned directly to users.

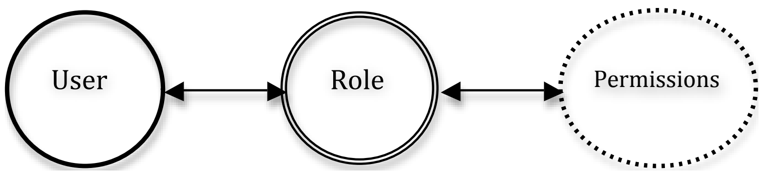


Figure 4.1: RBAC concept

BP-TRBAC uses the RBAC concept as a base, and extends it. Figure 4.2 shows the model of BP-TRBAC, which includes the main concepts of the core NIST-RBAC [30]: user, role, permission, and session. The model also supports the notion of constraints.

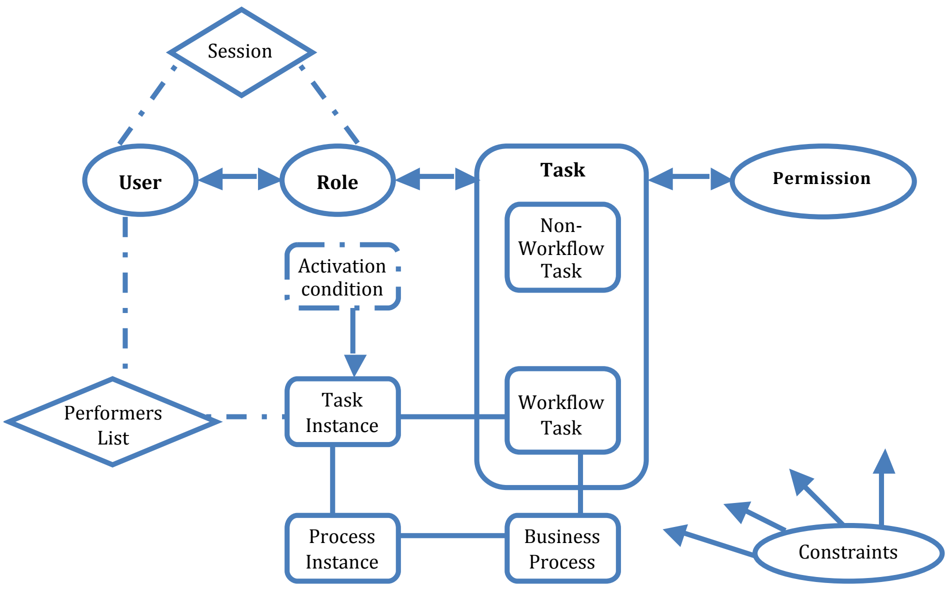


Figure 4.2: BP-TRBAC model

In BP-TRBAC, instead of having permissions assigned directly to roles, permissions are assigned to tasks, which are assigned to roles. Grouping permissions in tasks simplifies administration. It also helps minimise the storage for history-based instance-level restrictions, which are discussed in Section 4.3.5. Moreover, in business processes specifically, it is more important to deal with tasks instead of permissions, as restrictions usually apply to tasks.

BP-TRBAC expands on the notion of task to support workflow and non-workflow authorisation requests. BP-TRBAC supports two types of task: non-workflow and workflow. For workflow tasks it distinguishes execution instances, which have an activation condition, and performers list.

BP-TRBAC also supports instance-level restrictions by making use of task-instances, their status, and the 'Performers List' (PL) function. Having a unique identifier for each instance gives BP-TRBAC the ability to satisfy the restrictions on the instance-level. The PL function in particular is designed to help with history-based restrictions.

BP-TRBAC achieves active access control by using the 'activation condition' and the task-instance's status. Activation conditions are designed to ensure that prior tasks in the business process sequence are completed, which means it is time for this task to be performed.

More details about BP-TRBAC, its components, and the way they are integrated are discussed in the following sections. The rest of the chapter explains the concept and provides a formal description of BP-TRBAC, showing how it supports the necessary characteristics that were identified in Chapter 3.

4.2 Conceptualisation

In this chapter we use 'Object-Role Modeling (ORM)' [27] as a conceptualisation modeling language. This section briefly explains ORM and why it was chosen as the conceptualisation method, then describes the BP-TRBAC conceptual ORM model.

4.2.1 ORM

ORM is "primarily a method for modelling and querying an information system at the conceptual level" [36]. It pictures a concept in terms of objects (entities or values) that play roles (parts in relationships). For example, the object *'Org. role'* plays the role of *'can do'* a *'task'*, and the object *'task'* can play the role of *'done by'* an *'Org. role'*. It makes no explicit use of attributes [36]. For example, instead of using 'Can be an Accountant' as an attribute of a *'user'*, it uses the relationship type *'user'* *'can be'* an *'org. role'* (*e.g.* an accountant). ORM's formalism helps in identifying how each object is represented and its relation to other objects. For further details on ORM please refer to [36].

4.2.2 BP-TRBAC ORM Model

Figure 4.3 shows a conceptualisation of the BP-TRBAC model using ORM. From the ORM model, we can see that a *'user'* can play an *'org. role'*, and an *'org. role'* can do a *'task'*. If the *'task'* is a *'workflow task (W-Task)'*, then each execution of it will be considered a *'task-instance'*, which has an *'activation condition'* and a *'status'*.

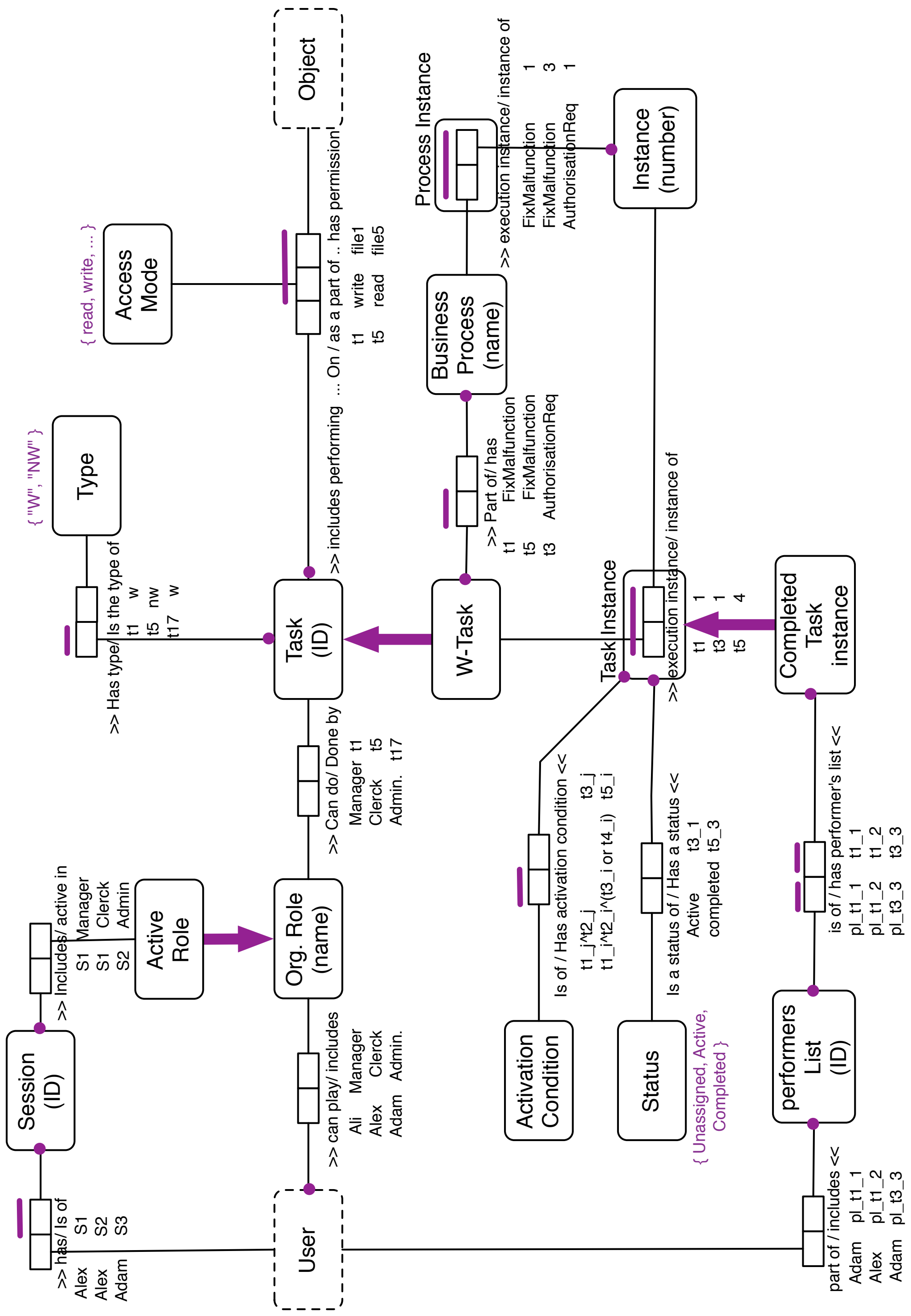


Figure 4.3: BP-TRBAC concept using ORM

The model shows that *active roles* are a subset of *org. roles* (indicated by the heavy arrow), *w-tasks* are a subset of *tasks*, and *completed task instances* are a subset of *task instances.* A permission is an *access mode* on an *object.* So the ORM model shows the relation between the *task* and the permission included in it by showing the *access mode* type and the *object* that it applies to. The combination of the *object* and the *access mode* (a permission) is unique (indicated by the line on top of the role boxes for these two objects).

The ORM model provides example 'population' under each object to further illustrate what it is supposed to be. The model also shows that each *task* must have at least one *type* (the dot connecting the object *task* to the line indicates that it should have at least one), and that it should have no more than one type (the uniqueness line on top of the role box indicates that for each task the type is unique, *i.e.* the task should have no more than one type). Only tasks of type *w-task* are part of a *business process* (the line connected only to the subset *w-task* and not to the object *task*).

The figure shows that the combination of a *w-task* and its *instance* is unique, as well as the combination of a *business process* and a *process instance.* Each *performers list* belongs to one *completed task instance*, and each *completed task instance* has only one *performers list* (indicated by the two separate lines on top of the role boxes). On the other hand, the *performers list* might contain one or more *users.* The model also shows that each *task instance* must have one *activation condition* (the combination of the dot on the object box (at least one) and the line on top of the role (maximum of one) indicates that it must have one and only one *activation condition*). On the other hand, the same *activation condition* might be used for more than one *task instance* (there is no uniqueness line on top of the role).

BP-TRBAC supports all the characteristics identified in Section 3.2. BP-TRBAC provides instance-level restrictions in an efficient way. A 'performers list' in BP-TRBAC ties the user to the task performed in a specific instance, where a task is tied to a set of permissions. By introducing the notion 'task' between role and permissions, the model supports task based authorisation. It also caters for both workflow and non-workflow tasks, as can be seen in Figure 4.3, where tasks are categorised and labelled as either a workflow task or a non-workflow task, and each type is dealt with in a different way. BP-TRBAC achieves 'active access control' using activation conditions.

4.3 Formal Representation

This section provides a formal representation of the BP-TRBAC model. Using set theory it will start by explaining the standard components including roles, users and permissions. Then it describes the new components: task, and task instance. After that it describes the activation conditions. Finally it describes the role-level SoD restrictions and the IR restrictions.

4.3.1 Roles, Users, and Permissions

This model is based on RBAC, so the formal description of all the components that are available in RBAC (*i.e.* user, role, permission, session, and user-role assignment) will follow standard RBAC terminology [77]. Let 'O' be the set of resources that are subject to access control, and 'A' be the set of actions that can be performed on these objects (*e.g.* read, write). The set of all possible actions on objects is referred to as permissions: P = A×O.

$$p = (a, o) : a \in A, o \in O \tag{4.3.1}$$

Let 'U' be the set of all users, where a user is an authenticated subject, and 'R' be the set of all roles. A role is a job function within the context of an organisation with some associated semantics regarding the authority and responsibility conferred on the user assigned to the role [30]. User-role assignment (URA) is a many-to-many mapping user-to-role assignment relation.

$$(u, r) \in URA \subseteq U \times R \tag{4.3.2}$$

A user may be authorised to play one or more roles. The set of roles that can be played by a specific user (*e.g.* u') can be found in URA.

$$\forall r \in (u, r) : u = u' \tag{4.3.3}$$

'S' is the set of all sessions, where a session ($s \in S$) is a function that returns the set of active roles for a specific user.

$$s(u) = \{\text{the active roles for user u}\} \tag{4.3.4}$$

$$s(u) = \{r\} : \{r\} \subseteq R \wedge r \text{ is active for u} \tag{4.3.5}$$

Since a user can activate only the roles that he is assigned, the set of activated roles (*i.e.* a user's session) must be a subset of the assigned roles.

$$s(u') \subseteq URA : u = u' \tag{4.3.6}$$

A user can ask for permissions only through activating roles. So, for a user (u) to ask for a permission, $s(u) \neq \varnothing$.

4.3.2 Task and Task-instance

Task and task-instance are part of the extension to RBAC introduced in this chapter. To be able to cater for both workflow and non-workflow tasks, we identified a field called ‘task type’ (ty). A task could be either a workflow task : (ty=w), or a non-workflow task : (type=n).

$$Ty = \{w, n\} \tag{4.3.7}$$

A task (t) is a tuple of $id \in \mathbb{N}$ (where $\mathbb{N}$ is the set of natural numbers), task type ty ∈ Ty, and a set of permissions p ⊆ P. The set of all tasks in the system is referred to as ‘T’.

$$t = (id, ty, p) \in \mathbb{N} \times Ty \times P \tag{4.3.8}$$

A task is a defined unit of work; as formally identified, it contains a set of permissions. So it will be easier dealing with tasks (collection of permissions) than dealing with each permission by itself. As can be seen in Figure 4.2, a task connects permissions to roles. So instead of assigning permissions to each role (as in RBAC), we assign permissions to tasks and then assign tasks to roles.

Permission-task assignment (PTA) is a many-to-many mapping permissions-to-tasks assignment relation.

$$PTA \subseteq P \times T \tag{4.3.9}$$

Task-role assignment (TRA) is a many-to-many mapping tasks-to-roles assignment relation.

$$TRA \subseteq T \times R \tag{4.3.10}$$

A user can perform a task only if the task is assigned to a role activated by this user. So, a user (u) can perform task (t) only if:

$$(t, r) \in TRA : r \in s(u) \tag{4.3.11}$$

A task of type (w) is part of a business process. We identify a business process (bp) by its name (n), where n ∈ N, and N is the set of names of all business processes in the organisation. Each bp is a tuple of the name (n), and a set of tasks t ⊆ T. The set of all business processes in an organisation is referred to as BP.

$$bp = (n, t) : t \subseteq T, n \in N, bp \in BP \tag{4.3.12}$$

Task-process assignment (TPA) is a one-to-many mapping tasks-to-business process assignment relation. Each business process can have one or more tasks, while each task is assigned to one business process.

$$TPA \subseteq T \times BP \tag{4.3.13}$$

A business process can be executed more than once, where each execution is called a process instance (pi). Here we denote 'PI' as the set of all process instances in an organisation. Further, each process instance ($pi \in PI$) is a tuple of (bp, n), where n $\in \mathbb{N}$ and bp $\in$ BP.

$$pi = (bp, n), pi \in PI \tag{4.3.14}$$

Each 'pi' includes the executions of all tasks that belong to this bp. An execution of each task is called a task instance (ti). A task instance belongs to a pi if they have the same instance number. A 'ti' also has a status, where the status of ti can be either: unassigned, active, or completed. A task instance (ti) is a tuple of st $\in$ ST : $ST = \{unassigned, active, completed\}$, task t $\in$ T, and a number n $\in \mathbb{N}$. The set of all task instances is 'TI'.

$$ti = (st, t, n) : \forall(bp, n'), n = n' \tag{4.3.15}$$

'Performers list' is a function mapping between completed task instance and a subset of users that performed this ti.

$$pl : ti \rightarrow u \subseteq U,\ ti \in TI \wedge ti(st) = completed \tag{4.3.16}$$

For every task instance the function will return a list of all the users that performed this 'ti'. This should include one or more user.

4.3.3 Active Access Control

In BP-TRBAC we are enhancing the 'activation condition' method from T-RBAC [70]. In BP-TRBAC the 'activation condition' is attached to each task-instance. This is part of the extension to RBAC introduced in this chapter. It is simply a boolean function that returns either true of false.

$$AC : ti \in TI \rightarrow v \in \{true, false\} \tag{4.3.17}$$

The task will be activated only if the condition returns 'true'. The Boolean function is a set of the immediate previous tasks instance's status that are required to be performed prior to this task (given that $t_0, \cdots t_n$ are a series of tasks such that t_{n-1} is the immediate prior task to t_n). If the status of the previous task-instance is 'completed' then it returns true, otherwise it returns false.

$$AC(ti_n) = true \text{ iff } St(ti_{n-1}) = completed \tag{4.3.18}$$

Note that so far, given the BP definition, we have not put any constraints on the order in which these tasks are to be executed. In the workflow literature the order in which one task precedes another task can take one of the following primary forms [91]. First is the sequential tasks, where t_n follows a single immediate prior task t_{n-1}. In such cases the prior definition of 'AC' applies directly. Second, it is also possible that t_n has more than one immediate prior task. In such cases, the ‘AC’ will be a set of all the immediate prior tasks with an ‘OR’ or ‘AND’ between them. For instance, if the situation is as shown in Figure 4.4, the activation condition for t_n will be:

$$AC(t_n) = true \text{ iff } (St(t_1) = completed \wedge St(t_2) = completed) \qquad (4.3.19)$$

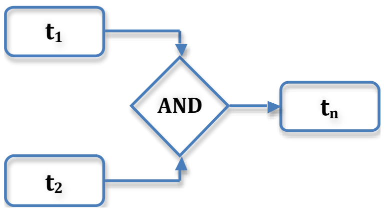


Figure 4.4: An ‘AND’ join

while if the situation is one of those shown in Figure 4.5, in both cases the activation condition for t_n will be:

$$AC(t_n) = true \text{ iff } (St(t_1) = completed \vee St(t_2) = completed) \qquad (4.3.20)$$

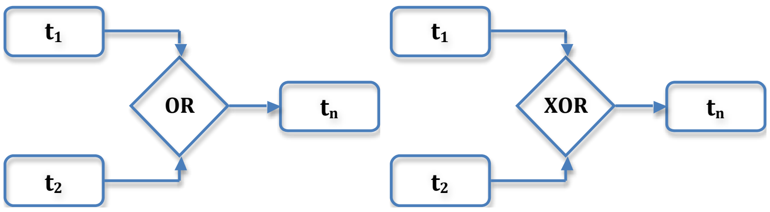


Figure 4.5: An ‘OR’ join and an ‘XOR’ join

So, in the case of having more than one immediate prior task, the activation condition will depend on the set of these two or more immediate prior tasks. The activation condition should have an ‘AND’ relation between the status if the execution relationship between these tasks was ‘AND’. Otherwise, it will be an ‘OR’ between these immediate prior tasks. In the case of the first task of the process, where there is no prior immediate task the activation condition will always be ‘true’. In the motivating scenario presented in chapter 3, the task ‘perform soft reset’ will not be active until the task ‘receive malfunction notification’ is completed. This will guarantee that, even though the ‘coordinator’ has the ability to perform soft reset, he will not be able to do so unless he receives a malfunction notification. In the same scenario the task ‘close work order’ will not be activated unless both tasks ‘complete work order’ and ‘receive invoice’ have been completed.

Activation condition as a concept is used to achieve the idea of active access control. Active access control is an important concept when dealing with business process. As a practical implementation matter in a process-aware system, active access control could be part of the authorisation system or the workflow system. In the system design we present in Chapter 6, the authorisation system and the workflow system work cooperatively to implement active access control. The authorisation system requests confirmation from the workflow system that the activation condition is met.

4.3.4 Separation of Duties

Role-level separation of duties (SoD), as specified in NIST-RBAC [30], identifies pairs of roles that cannot be assigned to (static), or simultaneously activated by (dynamic) the same user. SoD is represented as a tuple of the roles and the type of SoD (static or dynamic).

$$SoD = (r_1, r_2, \text{sod-type}) : r_1, r_2 \in R \text{ and sod-type} \in \{static, dynamic\} \tag{4.3.21}$$

If the SoD-type was 'static' this means that no user is allowed to be assigned these two roles (r1 and r2).

$$(r_1, u) \in URA \wedge (r_2, u') \in URA : u \neq u' \tag{4.3.22}$$

If the SoD-type was 'dynamic' this means that even though a user might be assigned both roles (r1 and r2), no user is allowed to activate these two roles in the same session.

$$if\ r_1 \in s(u) \rightarrow r_2 \notin s(u) \tag{4.3.23}$$

Static-SoD is achieved statically at design time, where as dynamic-SoD is achieved using sessions.

4.3.5 Instance-level Restrictions

The concept of *instance-level restrictions* is part of the extension to RBAC introduced in this chapter. These instance level restrictions (SoD and BoD on an instance level) are enforced via the task instance and the performers list. Every 'ti' is unique; the combination of the task id and the instance number is unique; and each 'ti' has a status (*i.e.* unassigned, active, completed). The function 'performers list' (PL) is also an important component for achieving instance-level restrictions (IR).

An 'ir' rule is written as a tuple of the two task instances and the type of the restriction (type). The set of all instance-level restrictions in a system is referred to as 'IR'.

$$ir = (ti_1, ti_2, type) : ti_1 \wedge ti_2 \in TI \text{ and } type \in \{SoD, BoD\}, ir \in IR \tag{4.3.24}$$

As shown in Figure 4.2, the PL connects the task-instance with the user who performed the task for that specific instance. This is used to make sure that assigning a specific task to a user does not violate any constraints. In BP-TRBAC every user is assigned to one or more roles, and every role has a number of tasks assigned to it. So, for every task there is a set of users who are allowed to perform the task, based on the role they belong to. Before allowing the user to activate a 'ti', BP-TRBAC's authorisation engine checks the 'ir's that apply to this 'ti'. It then uses the PL function to identify if the requesting user would violate an 'ir' if the task were performed by them.

For example, Adam and Anna are both assigned to the role 'coordinator', which makes them eligible to perform the task 'approve work order'. Adam asked to perform the task 'approve work order' for the instance 3. But the pre-defined rule (ir) states that 'no one is allowed to issue a work order and approve it for the same process instance'.

$$ir = (t1_i, t4_j, SoD) \rightarrow u \in pl(t1_i) \text{ and } u \in pl(t4_j) \text{ iff } i \neq j \tag{4.3.25}$$

Using the PL function it was found that Adam performed 'issue work order' for instance 3, so Adam is not allowed to perform 'approve work order' for instance 3. On the other hand, if Anna asked to perform the task, she will be allowed since this will not violate the 'ir' rule. At the same time Adam was allowed to perform 'approve work order' for instance 5, because Adam did not perform 'issue work order' for instance 5.

So, BP-TRBAC is able to satisfy the SoD and BoD on an instance-level, since the SoD rules are applied within the same process instance instead of being applied within the same time or session. A user can perform 'issue work order' and 'approve work order' tasks at the same time as long as they do not belong to the same process instance.

4.4 Example Scenario Revisited

This section revisits the example scenario from Chapter 3, using the BP-TRBAC model, to illustrate the detailed operation of the model and to show the benefit of using BP-TRBAC. The example scenario in Chapter 3 included a list of business and security rules, derived from the scenario that require access restrictions for their enforcement. To show how BP-TRBAC operates, listed below are the rules, together with their expression using the formal representations introduced in Section 4.3:

1. **The task 'soft reset' should not be performed unless malfunction notification was received.**
   (AC(soft_reset)= true iff st(receive_malfunction_notification)=completed).

2. **Only the role 'coordinator' is allowed to issue work orders.**
   ($(issue\_work\_order, coordinator) \in TRA$)

3. **Contractor's access rights should be activated only after showing the proper clearance (the work order).** (AC(Activate_access_rights)= true iff st(show_work_order)=completed).

4. **No work order can be closed until receiving both a 'work order completion' and an invoice.** (AC(close_work_order)=true iff (st(complete_work_order)=completed $\wedge$ st(receive_invoice)=completed)).

5. **Only a person who issued a certain work order is allowed to close it.** (ir=(issue_work_order, close_work_order, BoD)). $u \in pl(close\_work\_order_x)$ iff $u \in pl(issue\_work\_order_x)$.

6. **No person is allowed to perform 'issue work order' and 'review work order' for the same 'work order'.** (ir=(issue_work_order, review_work_order, SoD)). if $u \in pl(issue\_work\_order_x) \rightarrow u \notin pl(review\_work\_order_x)$.

7. **No person is allowed to have the role coordinator and the role contractor at anytime.** (SoD=(coordinator, contractor, static)). if $(contractor, u) \in URA \wedge (coordinator, u') \in URA \rightarrow u \neq u'$.

We will now describe the operation of each constraint in more detail. Assume that an instance of the 'fix pump malfunction' process is starting after receiving the starting trigger (the malfunction notification) and it is called instance number 7 (*i.e.* n=7). When Adam wants to log in as a coordinator, the system will first make sure that Adam is allowed to do so (*i.e.* the role 'coordinator' is statically assigned to him),

$$(Adam, coordinator) \in URA),$$

and that there is no violation of any 'SoD' rules by Adam activating the role 'coordinator'. Only then will the system allow Adam to activate the role 'coordinator'.

$$s(Adam) = \{coordinator\}.$$

After receiving the notification of a pump malfunction through the enterprise building integration system, Adam (as a coordinator) wants to perform 'soft_reset' (t(id)=soft_reset) for instance number 7(n=7). First the system identifies if this role is allowed to perform this task.

$$(soft\_reset, coordinator) \in TRA.$$

The system then identifies the instance number, and knows that Adam wants to perform task-instance:

$$ti = (unassigned, soft\_reset, 7).$$

Checking the IR of this 'ti' shows that there are no restrictions ($i.e.\ ir(soft\_reset) = null$).

Then it will check the status of this 'ti', which is 'unassigned', meaning that no one has performed this 'ti' yet. The activation condition of this task instance is that the task instance 'receive malfunction notification 7' is done:

$$AC(soft\_reset_7) = true \text{ iff } st(receive\_malfunction\_notification_7) = completed,$$

which is the case for this example, so the activation condition will be true ($AC(soft\_reset_7) = true$), and 'soft_reset$_7$' is ready to be activated (performed by a user). So the BP-TRBAC system will now allow Adam to perform 'soft reset for instance number 7'.

The second restriction can be achieved using BP-TRBAC because it uses the concept of roles, and assigning tasks to roles. So, if Adam did not activate the role 'coordinator' ($coordinator \notin s(Adam)$), he will not be able to perform the task 'issue work order', as this task is assigned to the role 'coordinator' ($(issue\_work\_order, coordinator) \in TRA$) and only a user with this role can perform it.

$$((issue\_work\_order, coordinator) \in TRA) \wedge (coordinator \notin s(Adam))$$

$$\rightarrow Adam \notin pl(issue\_work\_order, coordinator)$$

The third restriction is achieved because BP-TRBAC uses the concept of active access control. The activation condition of the task 'Activate access rights' will only be true if the task 'show work order' has been completed. Otherwise the activation condition will be false and no one can perform the task yet. So, contractors will be granted access only after showing the 'work order'.

$$AC(Activate\_access\_rights) = true \text{ iff } (st(show\_work\_order) = completed).$$

The same property (active access control) is used to achieve the fourth restriction. The activation condition for the task 'close work order' will be true only if both tasks 'complete work order' and 'receive invoice' are completed. So, the task of 'closing the work order' will not be activated unless the work order is completed and the invoice is received. Otherwise the activation condition will be false and no one can perform the task yet.

$$AC(close\_work\_order) = true \text{ iff } (st(complete\_work\_order) = completed \wedge st(receive\_invoice) = completed).$$

The fifth and sixth restrictions are achieved in BP-TRBAC because of the feature of instance-level restrictions (IR). There will be an IR on the task 'close work order' saying that it should be done by the same user who performed 'issue work order' for the same instance (ir=(issue_work_order$_x$, close_work_order$_x$, BoD)). When a user is about to perform the task 'close work order', the system will check the IR. To satisfy the restriction, it will identify the user who performed the task 'issue work order' for this instance using the 'performers list' function:

$$pl(issue\_work\_order_x) = \{Adam\},$$

The system will allow only the same user (Adam, or any other user in the list) to perform the task 'close work order'.

The sixth restriction will be the other way around, as the restrictions state that it should not be the same person (ir=(issue_work_order$_x$, review_work_order$_x$, SoD)). The system will check the performers list of the task 'issue work order':

$$pl(issue\_work\_order_x) = \{Adam\}$$

The system will not allow the user (Adam, or any other user in the list) to perform the task 'review work order'.

The seventh restriction can be achieved using role-level SoD, which is based on the standard RBAC SoD.

4.5 Discussion

BP-TRBAC is a unified model designed to support all the characteristics identified in Chapter 3. To the best of the author's knowledge, of the authorisation models that are designed for business process aware systems and that support some of the characteristics, none support all the characteristics. Combining all the characteristics to be supported by one model and maintaining the concept of RBAC to be able to support non-workflow requests is the point of strength for BP-TRBAC. Workflow authorisation has its own unique specifications, which do not apply to non-workflow authorisation. That is why the model needs to be designed in a way that is aware of different types of requests and that has the ability to deal with different types of requests, as they have different requirements.

The concept of active access control is an important aspect of workflow authorisation models. Some of the BPM authorisation models in the literature assume that active access control can be ignored; depending on the fact that workflow systems are aware of the task sequencing, and it can be assumed that all workflow requests are requested only at the right time. In this thesis we wanted BP-TRBAC to be a unified model that is in charge of all authorisation request decisions, and that does not rely on any other systems to make the decision. This model can make use of information retrieved from another system but does not depend on the other system to make the decision for it. For that reason we have used the idea of activation conditions.

The activation conditions concept is proposed to make sure that the tasks are performed only at the right time. In terms of implementation, activation conditions can be part of the authorisation system itself, or delivered through a cooperative interaction between the workflow system and the authorisation system. Since workflow systems are designed to be aware of task sequence and instance execution, the authorisation system can make use of the workflow system to retrieve information to enforce the activation condition. In this way, we can make sure that the authorisation system is still responsible for enforcing the constraint, but we also do not need to duplicate a feature that is already available in the workflow systems.

Instance-level restrictions as a concept has been addressed in different ways by different proposals [14, 18, 70]. We have found that most of the proposed solutions in the literature are structured in a way that cannot support history-based instance-level restrictions. Some of the proposals that support history-based restrictions require predefining of the context in which the restriction will be applied [11]. Maintaining history to support history-based restrictions is an important functionality to be able to evaluate instance-level restrictions that concern completed tasks. Otherwise, it will be possible to evaluate instance-level restrictions only if the task is still being performed, as in [11]. In BP-TRBAC we proposed the idea of 'Performers List' (PL) to be able to retrieve information about the users who performed a specific completed task instance.

Supporting all these characteristics and maintaining the features of the NIST-RBAC concept makes BP-TRBAC a unified model that can support legacy systems and also can support workflow systems. A simple authorisation request that is not part of a workflow will be processed according to how it is processed in the NIST-RBAC model.

4.6 Review

BP-TRBAC is designed to be an independent authorisation system that interacts with the workflow system and cooperatively enforces the enterprise's authorisation policies in a consistent way across both workflow and non-workflow systems. With its architecture based on the XACML standard, BP-TRBAC is designed to form the basis of an enterprise-wide authorisation system that is capable of integrating with existing applications, not all of which need to be process-aware.

Table 4.1: Comparing BP-TRBAC to other authorisation models

	Compression criteria	**RBAC***	**WAM**	**FWAM**	**SRBWM**	**T-RBAC**	**W-RBAC**	**MSoD**	**WSession**	**SOWAC**	**Str. & Mend.**	**AW-RBAC**	**BP-TRBAC**
1	An independent access control model	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	No	Yes	Yes
2	Supports Role-based access control	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
3	Supports Task-based authorisation	No	No	No	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes
4	Supports Active access control	No	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	No	Yes
5	Supports SoD on instance-level	No	No	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes
6	Supports History-based SoD on IL**	No	No	No	No	No	Yes	Yes	No	No	Yes	No	Yes
7	Supports BoD on instance-level	No	No	No	No	No	Yes	Yes	Yes	No	Yes	No	Yes
8	Support non-workflow tasks	Yes	No	No	No	Yes	No	Yes	No	No	No	Yes	Yes
9	Support workflow tasks	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

* The RBAC model used in this table is NIST RBAC2

** IL= Instance-Level

Table 4.1 compares BP-TRBAC to the models reviewed in Chapter 2, using the main characteristics identified in Section 3.2: RBAC, task based authorisation, active access control, instance-level restrictions, and support for both non-workflow tasks and workflow tasks. The table shows for each model which characteristics are supported or not supported. To improve clarity, the instance level restriction capability is divided into three different characteristics. The first characteristic is SoD on an instance level, which means SoD on active task instances. Second is history-based SoD, which means supporting SoD restrictions that include a task instance that has been completed. The third is BoD on an instance level. This distinction has been made because some of the reviewed models support only a subset of these three characteristics.

As previously stated, it can be seen in the table that all reviewed models are RBAC-based and support workflow tasks. MSoD, W-RBAC, Strembeck and Mendling, and BP-TRBAC are the only models that support all three types of the instance-level restrictions (Characteristics 4, 5, and 6).

MSoD [18] was proposed as a new SoD idea, rather than a complete model. It is an idea to solve long-term SoD and instance-level SoD. MSoD proposes the use of 'business context' instead of session, where SoD constraints will be defined within the 'business context'. The MSoD concept requires a pre-definition of business contexts and identifying SoD policies that belong to each business context. Moreover, it does not provide task-based authorisation, so the instance-level restrictions will be on permissions rather than on tasks.

W-RBAC [95] defines a new notion called 'case', which is an instance of a business process execution. The model also defines 'doer', a three-way relation between the user, permission and the case, which makes it easier to deal with instance-level restrictions. The doer relation connects the user's executions of permissions for certain cases. W-RBAC deals with permissions rather than tasks. So even though it is possible to identify instance-level restrictions, these will be with regard to permissions and not tasks.

Strembeck and Mendling's proposal [84] uses a formal method to define the instance-level restrictions. They define three new relations, one called SME (Static Mutual Exclusion); the second DME (Dynamic Mutual Exclusion); and the third RB (Role Binding) the instance-level binding of duties. For example, a DME of task 'T' will show all tasks that are in a dynamic SoD with task 't' (*e.g.* DME(t1)=t2, which means that t2 is in a dynamic SoD with t1). In BP-TRBAC we define a relation called 'IR' (Instance-level Restriction). If there is a SoD relation or a BoD relation between two tasks on an instance level they will be included in one rule including the type of the restriction whether it is a SoD or BoD (*e.g.* IR=(t1,t2, SoD) this is an SoD on an instance level between t1 and t2). While achieving effectively the same result, Strembeck and Mendling's proposal for each mutually excluded task two rules needs to be identified in the system. In BP-TRBAC one rule will include both tasks. Moreover the BP-TRBAC rules, by having a type, can support both SoD and BoD, while in Strembeck and Mendling's proposal, a specific function is proposed for each type.

BP-TRBAC supports instance-level restrictions on tasks rather than on permissions. It also has the ability to deal with SoD, BoD, and history-based restrictions on an instance-level. In addition, it deals with non-workflow tasks and is designed to be a unified system. The following paragraph discusses other shortcomings of related models and why it is necessary to have a model such as BP-TRBAC.

A shortcoming of W-RBAC is that it depends on the workflow engine to control the process, while W-RBAC provides only the list of users allowed to perform the task. It does not support task-based authorisation and it does not cater for non-workflow tasks. It was designed to focus on workflow tasks and to be part of the workflow system. A similar shortcoming can be found in the Strembeck and Mendling proposal, which was also designed to be part of the workflow system and does not cater for non-workflow tasks. This would require the organisation to have more than one authorisation system. Strembeck and Mendling's proposal has the advantage over W-RBAC that it actually supports task-based authorisation. The MSoD proposal is the only one beside BP-TRBAC that caters for non-workflow tasks. Unfortunately, MSoD does not support task-based authorisation. BP-TRBAC is the only one of these models that supports all types of IR as well as task-based authorisation, and caters for non-workflow tasks at the same time.

BP-TRBAC is designed to be the authorisation system in control, avoiding duplication of functionalities. It is an independent unified authorization system. It can be an enterprise-wide authorisation system that deals with workflow and non-workflow systems.

4.7 Conclusion

In this chapter we proposed BP-TRBAC, a new unified access control model. The model builds on the NIST RBAC model, extending it to support process-aware systems. BP-TRBAC supports task-based authorisation, with both passive and active access control. One of the main contributions is that BP-TRBAC supports instance-level restrictions, including history-based instance-level restrictions. Another advantage of BP-TRBAC is that it caters for both workflow tasks and non-workflow tasks. It was designed to be the main authorisation system for the whole organisation, rather than a specialised model to be embedded in a workflow system.

This chapter provided a conceptual representation and a formal description of the introduced model. It showed how to represent tasks and task instances, and showed the formal representation of the instance-level restrictions and the separation of duties restrictions. The formal description helps in understanding the mechanism of the characteristics and the way they are supported.

In comparison to other related models, BP-TRBAC introduces a unified model in which the necessary characteristics of an authorisation model for a business process environment are supported. It is conceived as an independent, enterprise-wide authorisation system that can provide an access control decision making service to other systems including, but not limited to, workflow systems, based on a consistent set of organisational security policies.

In the following chapter we will show an example 'use case' of BP-TRBAC. The chapter will show how to implement a subset of BP-TRBAC as a use case showing implementation results. Chapter 6, then presents a proposal for a new XACML-based policy language to be used with this authorisation model.

Chapter 5

SPCC for YAWL: a Use-Case of the BP-TRBAC

In the previous chapter we introduced BP-TRBAC as an authorisation model. In this chapter we will demonstrate an implementation of a subset of BP-TRBAC as a use case. We have identified a shortcoming in YAWL as a workflow system. The current structure of YAWL allows the process modeler to specify and assign users/roles to perform tasks [2]. Having this privilege under the control of the process modeler may lead to authorisation policy breaches. There is no method for making sure that these assignments are in compliance with the organisation's authorisation policies. This can pose serious threats in a real-world business process environment, as it may lead to breaching the company's authorisation policies. We have identified a subset of BP-TRBAC to produce a security policy compliance checker (SPCC).

SPCC is a use-case of BP-TRBAC, which works as a compliance checker. BP-TRBAC is a complete authorisation model that is used to check run time authorisation requests, while SPCC is using only a subset to perform design time checking. It was designed specifically to overcome the shortcoming we identified in the YAWL system. SPCC takes in the process modeler's assignment of a role to perform a task, checks if that assignment is in compliance with the authorisation policy, then sends back a result stating whether this assignment is allowed or not. So, SPCC provides design time checking of role-task assignments with regard to the authorisation policy, and assists the process modeler in not breaching the authorisation policy.

In this chapter we describe how SPCC has been implemented as a plug-in for YAWL. YAWL was chosen because it is an open source workflow system that has its own modeling language. Moreover, the new YAWL Editor 3.0 supports extension through a pluggable interface [2].

This chapter begins by giving an overview of key technical aspects of YAWL, showing the shortcoming that YAWL has now. Then it explains SPCC as an application, showing its structure and design. After that it discusses implementing SPCC as a plug-in for YAWL, explaining technical details. Finally, it shows results from the implementation.

5.1 YAWL

The YAWL system uses the YAWL language as a business process modeling language [2]. The YAWL language distinguishes itself from other languages by having a formal foundation. This formal foundation allows for automatic translation of the model to build a workflow system that works based on the designed process model [90]. In YAWL the *Editor* allows the process modeler to design the business process model (also known as a process specification) [60]. The model uses the YAWL language, which has a formal foundation. The YAWL *Engine* is able to create a workflow system that is based on this model. The system is a web-based system, which uses apache as a web server. The YAWL *resource service* manages all resources including users/roles and applications. Using the resource service, the administrator can add, remove, or modify resources [60]. In the Editor the modeler can use these resources, available from the resource service, to assign them to perform tasks [2].

The YAWL system consists of an extensible set of YAWL services, with each having a unique set of addresses and end points and one or more interfaces [60]. Some of these services offer functionality to end users, while some interact with other services and applications and some can do both. YAWL was developed using Java as a programing language. The communication between YAWL services is done through HTTP GET and POST operations and the data documents that are transferred are in XML format [60].

The YAWL system has several services, but in this section we discuss only the relevant aspects of YAWL. Figure 5.1 shows the most relevant components of the YAWL system and their relationships. In the following subsections we will give an overview of these components/services.

5.1.1 YAWL Engine

The YAWL Engine manages the execution of instances or cases [2]. The specified data mappings between the case and its tasks are performed as required by the Engine and each case is executed according to its current state and control-flow description. The Engine is crucial for execution as it determines whether a work item should be offered or announced to the environment at each stage of the process. Generally a YAWL service is explicitly associated for each task in a process instance during design time [2]. In this chapter we will not be dealing with the engine, as our focus will be on design time only.

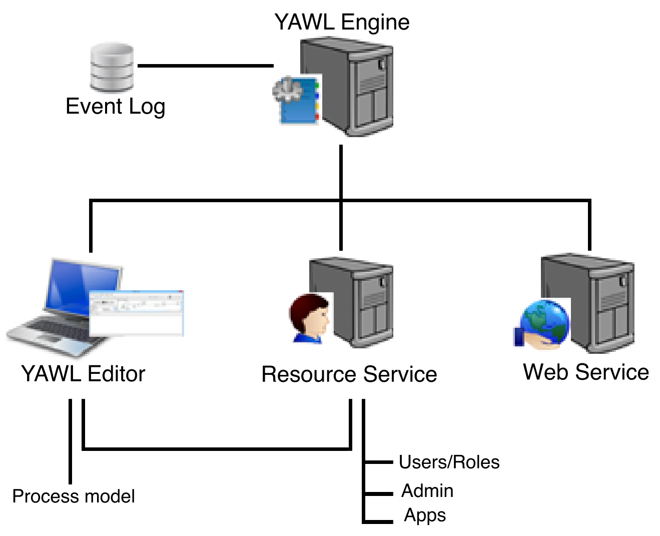


Figure 5.1: YAWL architecture

5.1.2 YAWL Resource Service

The YAWL resource service is completely separate from the Engine and provides the resource perspective for specifications [60]. The main function of the Resource Service is to allocate work items to resources for processing. A resource can be a person, an application, or a service. YAWL allows assignment of roles to users, and also allows assignment of positions to users [60]. This allows for better control and flexibility by assigning a role to perform a specific task rather than assigning a specific user. The resource service also allows each participant to possess a number of capabilities [60]. So, resource assignments to tasks can be based on the user's capabilities rather than his role. The resource service is used to identify and manage the resources, while the 'Editor' can be used to specify the resourcing requirements for tasks at design time to be used at run time [2].

5.1.3 YAWL Editor

The YAWL Editor can be used for creating, editing, configuring, validating, and analysing workflow specifications [60]. The first step in executing a workflow model is defining it and this is done using the Editor. In the Editor the process modeler can also assign tasks to predefined users/roles [2]. Graphical representation allows solution architects and developers to capture workflow models and automatically detect potential errors early on with the help of the graphical editor. Other functionalities such as the verification functionality help in detecting errors. The verification functionality allows for detecting syntax errors such as having a task that does not have a successor task. Another functionality, validation, allows for identifying design errors such as closed loops [60]. Unfortunately, there is no function that checks if resource assignments are in compliance with the organisation's authorisation policy. YAWL's editor is the most related aspect of YAWL to the plug-in that will be presented in this chapter.

5.1.4 The Problem

During the process design time, a process modeler has the ability to assign task performance to user's roles, specifying the role that has the ability to perform this task. However, process modelers are not security experts and may not be aware of the organisation's authorisation policy for certain resources or applications. Therefore, the assignments might not be in compliance with the policy. Task performance sometimes requires interaction with other applications that have their own authorisation policy. If the modeler's assignment is not in compliance with these policies, the task will not be performed, as the user will be denied during run time, and therefore, the process will not be completed. If it was chosen to disable the run time policy enforcement method (*e.g.* authorisation model) in order to make the process work, then the protected asset or system is at risk of misuse or loss with the associated potential for undesirable consequences for the organisation (*e.g.* regulatory fines for unauthorised use of personal information).

One method for making sure that these assignments are in compliance with the authorisation policies is having the modeler manually check each assignment against the authorisation policies. This manual process is expensive and time consuming. Moreover, human actions are comparably more prone to errors. Therefore, there is a need for an automatic method to check if the assignments are in compliance with the policy. As explained earlier, YAWL provides a functionality for checking the context, and the design, but there is no functionality to check the resource assignment.

5.2 SPCC

To solve the shortcoming in YAWL, the Security Policy Compliance Checker (SPCC) was designed to work in the YAWL environment to check the compliance of the assignments. SPCC is intended to check the compliance of an assignment of tasks to a roles with the organisation's authorisation policy during design time. This is achieved through integrating the SPCC software module with YAWL so that the SPCC service can be called through the editor. The request includes the process modeler's assignments of roles to tasks. Then SPCC checks if these assignments are in compliance with the authorisation policy, and sends the results for all assignments at once.

The following subsection will provide details about the design, architecture, and structure of SPCC.

5.2.1 Architecture

This section explains the architecture of SPCC by itself. Figure 5.2 shows the architecture of SPCC without showing the connections to the workflow system. In SPCC, *Compliance Checker* receives the request from an external system (*i.e.* workflow system). The request contains a list of role-task assignments, each of which should be checked for compliance with the authorisation policy. For this purpose, the *Compliance Checker* might need to get extra information from log files (or the workflow system). The Compliance Checker then passes a request for each assignment and the needed information to the *Policy Decision Point* (PDP).

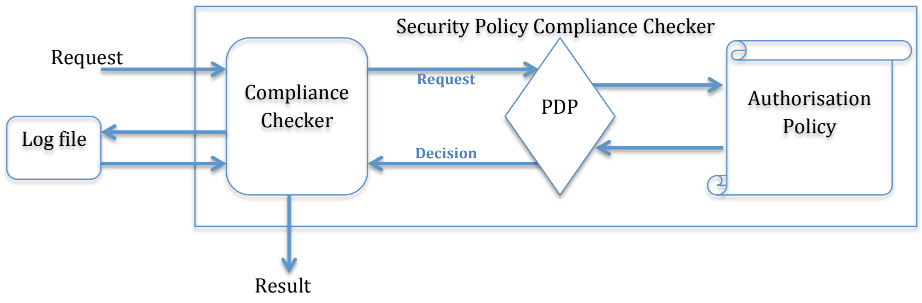


Figure 5.2: SPCC architecture

The PDP then checks if the task-role assignment is in compliance with the policy file. For this purpose, PDP reads the policy file. If the assignment complies with the policy, PDP will send *'Allow'* output to the *Compliance Checker*. If the assignment violates the authorisation policy, PDP sends *'Deny'* output to the Compliance Checker. If there is no matching policy for the assignment, PDP sends *'No match found'* to the Compliance Checker.

5.2.2 Software

The software module for SPCC consists of two major components, the *Compliance Checker* (CC) and the *Policy Decision Point* (PDP).

Compliance Checker

The 'ComplianceChecker' class is used for receiving the request to check compliance with the policy and for framing the request to the required format. It also gathers extra information and passes it along with the formatted request to the (PDP) class. It then receives the output from PDP and interprets the output into an appropriate response. The 'ComplianceChecker' class contains three methods: 'main', 'format' and 'readFile'.

Main Method: performs the entire execution of the SPCC software module.
Format Method: frames the request into a specified format when the elements are passed as arguments by the workflow system.
ReadFile Method: receives data from the workflow system and returns the result.

In short the 'ComplianceChecker' class does the following:

- Reads request and gathers extra information
- Sends requests to PDP
- Receives decision from PDP
- If result was *No match found*, CC makes a default decision (currently configured as *'allow'*)

Policy Decision Point (PDP)

The PDP class checks the compliance of the received request with the policy. For this purpose, it reads the policy file and checks the request against the policy. The PDP sends back the output to the compliance checker informing whether the request complies with the policy or not; it also informs the compliance checker if no matching policy is found. This class consists of one method, which is 'PDP'.

PDP Method: is called by the main method in *ComplianceChecker* class. It reads the policy file, checks if the request matches with any policy, then assigns the appropriate value to variable response. If the request does not match the policy, it assigns *No match found* to the variable response.

In short 'PolicyDecisionPoint' class does the following:

- Receives request from CC
- Reads policy file
- Searches policy file for all matching policies
- If matching policy found, retrieves rule from policy file and sends to CC
- If multiple results found, and at least one of them is *deny*, then sends *deny* to CC
- If no result found meeting the request, the PDP returns *no match found*

5.2.3 Data Structure

Since the aim of the implementation is to investigate the use case, we have defined our own simple data structure. We used a specific structure for the policy file and the request. In this section we will explain the data structure used in the software module of SPCC.

Policy File Structure

The policy file includes the policies that the role/task assignments need to be in compliance with. Each policy contains a subject, an object, an action, and a resulting rule. The structure of the policy says if this subject wants to perform this action on this object, then the resulting rule should apply. For example, a policy will look like this: Subject: *coordinator role*, Object: *pump*, Action: *hard reset*, Rule: *allow.* Such policy means a user with the role *coordinator* can perform *hard reset* on the *pump.* For each policy, the subject element comes first, object element second, action element third, and rule element last.

We have identified our own structure to test the PDP and the CC of the SPCC. The structure is simple, and intended to test only if SPCC is doing what it is supposed to do and can communicate with YAWL. Building this simple policy structure helped us to recognise the need for a structured language. This opportunity helped us to understand the elements and the structure of such a language. In general the policy file should be unambiguous, up-to-date, and complete. Having a clear naming convention and using a structured machine-readable language helps in making sure that the policy file is unambiguous. Having a single organisation-wide authorisation policy file makes it easier to make sure that the policy is always up-to-date. If there were multiple copies of the file, it will be harder to make sure that all copies are up-to-date. Finally, for a policy file to be complete it has to cover policies for all assets and systems in the organisation. In order to do so, the language should have the ability to represent policies for different systems. For example, the language should have the ability to represent policies for business process system.

Request Structure

An authorisation request is a request by a subject to perform a specific action on a specific object. Therefore, each request should contain a subject, an object, and an action. For example, if a request contains the following: Subject: ‘coordinator’ role, Object: ‘pump’, Action: ‘soft reset’, the request is asking if the user with the role ‘coordinator’ is allowed to perform ‘soft reset’ on the ‘pump’.

Decision

Decisions sent from PDP to CC will be any of the following:

- Allow
- Deny
- No match found

Conflict Resolution

If more than one policy is found matching with the specified request, and at least one of the decisions for the matching policies is ‘deny’, then ‘deny’ will be the final decision. If all matching policies decisions are ‘allow’, then it will return ‘allow’.

If ‘no match found’ output was received from the PDP, then the ‘Compliance Checker’ should make a decision based on the business requirement. Currently, ‘Compliance Checker’ will return ‘Allow’ as the response to the workflow interface if there was no match found.

5.2.4 Flowchart

This section describes the flowchart of the SPCC software and how it works. Figure 5.3 shows the steps SPCC performs to reach an authorisation decision. SPCC works as follows:

1. Compliance checker (CC) receives the request and needed information and passes them as parameters to PDP.
2. PDP then reads the policy file from specified address.
3. PDP checks each request for matching statement in policy file.
4. If a policy match is found then the PDP saves the decision stated at the end of the policy, and keeps searching for matching policies, until the end of the policy file.
5. If there is more than one matching policy statement, then the output to ‘Compliance Checker’ will be ‘allow’ when all the decisions for each of the matching policy statements are ‘allow’; otherwise ‘deny’ will be the final output.
6. If there is no matching policy found, PDP sends ‘No match found’ to ‘Compliance Checker’.
7. If the result ‘Compliance Checker’ received from PDP is ‘allow’, then ‘Compliance Checker’ prints ‘allow’ as output.
8. If the result ‘Compliance Checker’ received from PDP is ‘deny’, then ‘Compliance Checker’ prints ‘deny’ as output.
9. If the result ‘Compliance Checker’ received from PDP is ‘No match found’, then ‘Compliance Checker’ prints ‘allow (No match found)’ as output.

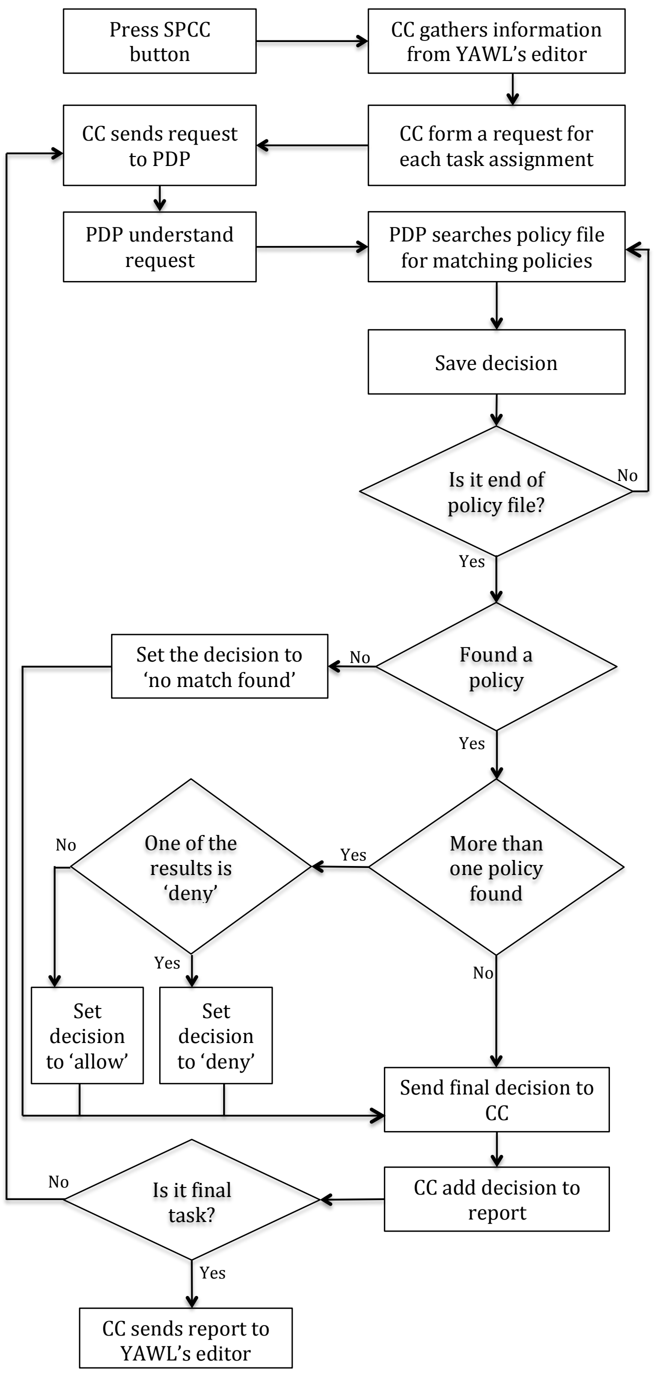

Press SPCC button
CC gathers information from YAWL's editor
CC form a request for each task assignment
CC sends request to PDP
PDP understand request
PDP searches policy file for matching policies
Save decision
Is it end of policy file?
No
Yes
Found a policy
No
Set the decision to 'no match found'
Yes
More than one policy found
Yes
One of the results is 'deny'
No
Yes
Set decision to 'allow'
Set decision to 'deny'
No
Send final decision to CC
CC add decision to report
Is it final task?
No
Yes
CC sends report to YAWL's editor

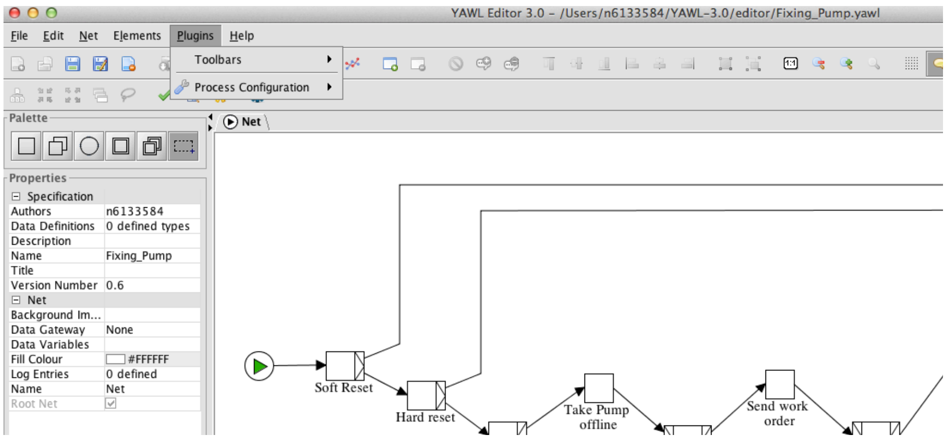
Figure 5.3: Flowchart of SPCC.

5.3 SPCC Implementation in YAWL

The real benefits of SPCC are realised only when it is integrated with a workflow engine. In this chapter we will use the plug-in feature to integrate SPCC as a plug-in in YAWL. This section will outline the implementation of SPCC as plug-in within YAWL, showing the implementation architecture, design, and interface.

5.3.1 YAWL Plug-in Interface

The new YAWL Editor 3.0 supports extension through a pluggable interface. Using the plug-in interface gives the plug-in access to the underlying application framework [2]. A new plug-in will be added in the plug-in menu in the Editor's interface. Through the menu, the plug-in's actions can be executed, so they can provide additional functionality to the editor. The editor also provides the option of calling a plug-in when an event occurs (*e.g.* opening a file) [2]. Figure 5.4 shows the YAWL 3.0 editor showing the plug-in interface.



Specification	
Authors	n6133584
Data Definitions	0 defined types
Description	
Name	Fixing_Pump
Title	
Version Number	0.6
Net	
Background Im...	
Data Gateway	None
Data Variables	
Fill Colour	#FFFFFF
Log Entries	0 defined
Name	Net
Root Net	✓

Figure 5.4: YAWL plug-in interface.

5.3.2 Architecture

Figure 5.5 shows the architecture of the SPCC as a YAWL plug-in. As the architecture shows, the *compliance checker* (CC) part of the SPCC is the one that communicates with YAWL. SPCC will deal only with the *Editor* part of YAWL; it will not be communicating with the *Engine* or the *Resource service*. It is a design time compliance checker for YAWL, aiming to check the modeler assignments of roles to tasks at design time if they are in compliance with the policy or not.

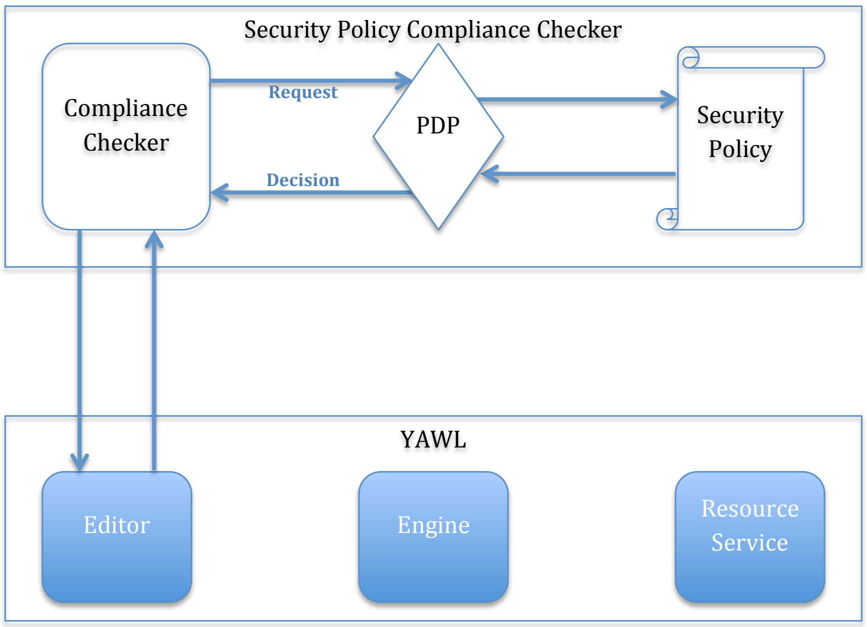


Figure 5.5: SPCC communication with YAWL.

Needed information such as user, role, task, and action will be gathered through the interface and sent to the CC part of the SPCC. CC should form the request into the standard accepted format and forward the request to the PDP. A decision should be made by the PDP based on the policy file. The decision will be sent to CC, which will forward this information to print a message in the YAWL editor saying either *'Role is allowed'*, *'Role is allowed (no match found)'* or *'Role is not allowed'*. The first decision *Role is allowed* will appear if there was a policy stating that this assignments is allowed. The second decision *Role is allowed (no match found)* will appear if there was no matching policy for this assignment. As explained earlier, if there was no matching policy, the PDP will send *No match found* to CC, which makes a decision based on the business environment. In this implementation, for the sake of simplicity, CC is configured to make the decision to allow, if no match found. It will send the following result *Role is allowed (no match found)*. The third decision *Role not allowed* will appear if there was a policy stating that this assignment is not allowed.

Figure 5.6 shows the sequence diagram of SPCC as a YAWL plug-in. As can be seen, the 'Editor' is the only part of YAWL that interacts with SPCC. When SPCC is chosen, CC will send to the 'Editor' gathering information about the roles assignment to all tasks. Then CC sends each role/task assignment by itself as a request. The PDP will assess the request and send the decision back to CC. After completing all requests, CC will send back to the 'Editor' the complete results of all assignments.

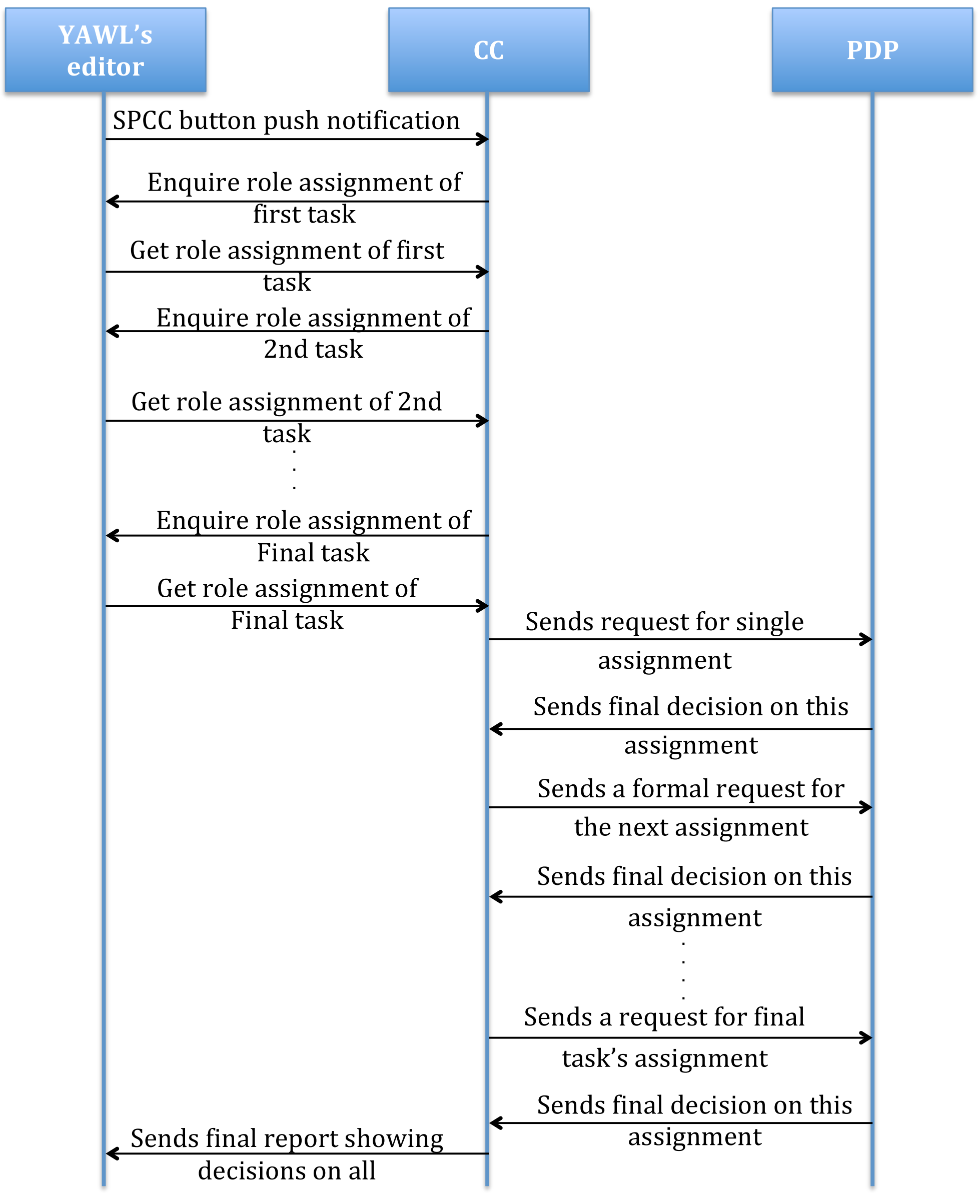
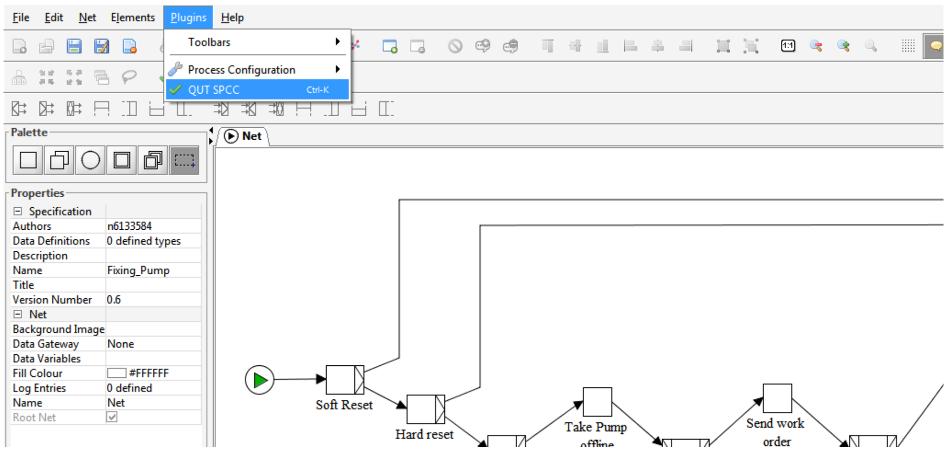

YAWL's editor
CC
PDP
SPCC button push notification
Enquire role assignment of first task
Get role assignment of first task
Enquire role assignment of 2nd task
Get role assignment of 2nd task
Enquire role assignment of Final task
Get role assignment of Final task
Sends request for single assignment
Sends final decision on this assignment
Sends a formal request for the next assignment
Sends final decision on this assignment
Sends a request for final task's assignment
Sends final decision on this assignment
Sends final report showing decisions on all

Figure 5.6: Sequence diagram for SPCC as a plug-in for YAWL.

5.3.3 Interface

SPCC appears in YAWL's plug-in menu interface as shown in Figure 5.7. Once the plug-in is clicked it will trigger the SPCC to check all current task-role assignments.



Properties

⊟ Specification	
Authors	n6133584
Data Definitions	0 defined types
Description	
Name	Fixing_Pump
Title	
Version Number	0.6
⊟ Net	
Background Image	
Data Gateway	None
Data Variables	
Fill Colour	⬜ #FFFFFF
Log Entries	0 defined
Name	Net
Root Net	✓

Figure 5.7: SPCC as a plug-in for YAWL.

When the SPCC plug-in is chosen, a menu will give the modeler the option to choose the policy file that will be used to check the assignments. After that, the SPCC plug-in will check all assigned roles and send back a report for all tasks, showing for each task assignment if it is allowed or not.

5.4 Results and Discussion

After implementing SPCC as a plug-in for YAWL, we tested the plug-in using an example business process as shown in Figure 5.8. It is the process of fixing a pump malfunction. The process includes several tasks with some authorisation constraints. Then we created a policy file that contains all the authorization policies related to this process. The policy file followed the structure explained in Section 5.2.3. After that we assigned different roles to the tasks in this business process. During the role assignments we covered all expected types of results. We deliberately chose roles that are not supposed to perform the task, roles that are authorised to perform the task, and assignments that do not have a matching policy. Finally we used the SPCC plug-in by activating it to test if the assignments are in compliance with the policy file.

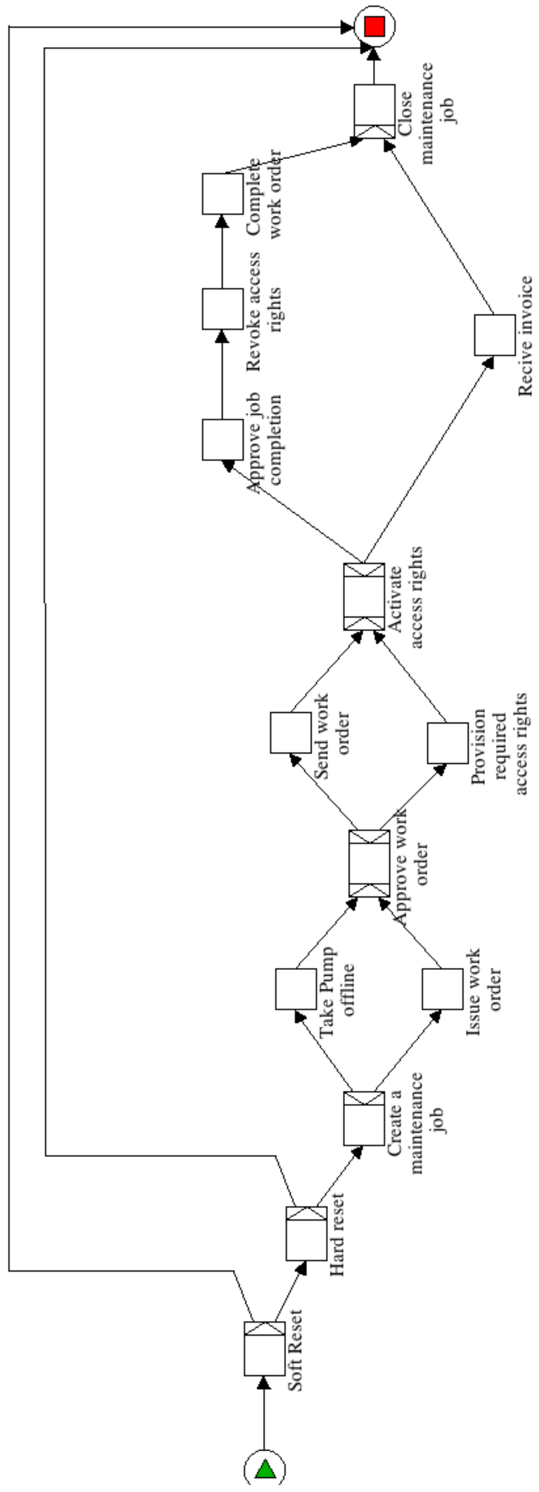
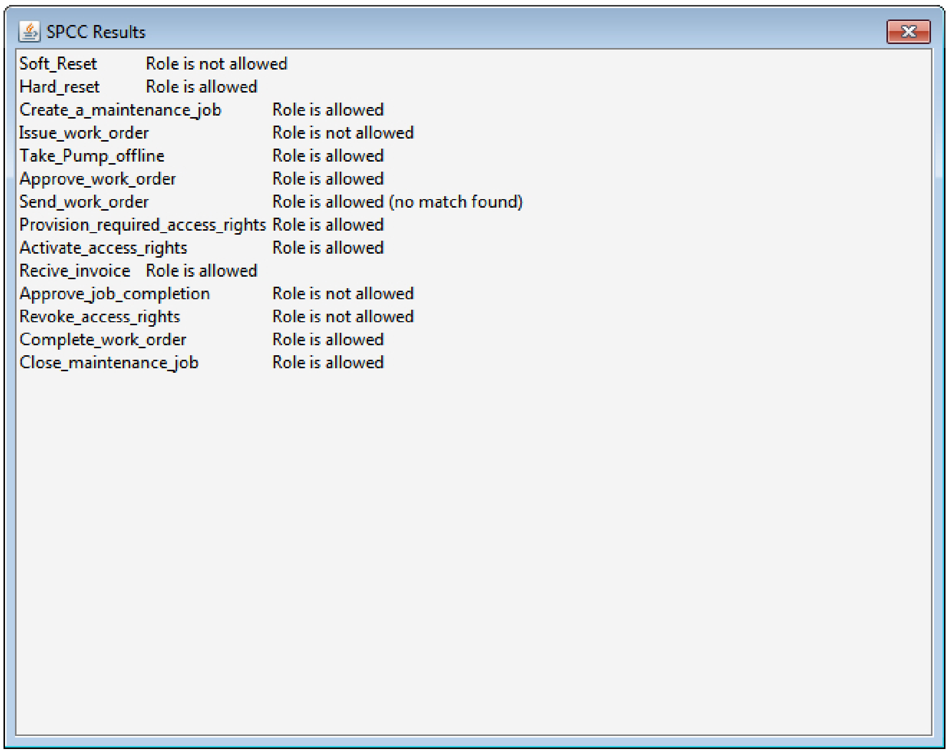


Figure 5.8: An example business process in YAWL.

The final results (see Figure 5.9) shows that SPCC was able to check the assignments, compare them with the selected policy, and send back the result for each assignment showing if it is in compliance with the selected policy or not.



```
Soft_Reset         Role is not allowed
Hard_reset         Role is allowed
Create_a_maintenance_job         Role is allowed
Issue_work_order                 Role is not allowed
Take_Pump_offline                Role is allowed
Approve_work_order               Role is allowed
Send_work_order                  Role is allowed (no match found)
Provision_required_access_rights Role is allowed
Activate_access_rights           Role is allowed
Recive_invoice   Role is allowed
Approve_job_completion           Role is not allowed
Revoke_access_rights             Role is not allowed
Complete_work_order              Role is allowed
Close_maintenance_job            Role is allowed
```

Figure 5.9: Results of testing the SPCC in YAWL.

For example, SPCC is telling the process modeler that, according to the selected policy, the role assigned to perform the task *Soft_Reset* is not allowed to perform such a task. Therefore, the modeler can choose another role to assign to this task and then check again. In another example, we assigned the role *coordinator* for the task *Hard_reset*, which is actually in compliance with the authorisation policy. Therefore, as can be seen in the figure, the result is *Role is allowed*. To test the result of *no match found*, we assigned a new role to perform the task *Send_work_order*; this role has no policy related to it to say whether it is allowed or not. The result actually came as *no match found*, and because of the current design of CC, it translated the decision to *Role is allow (no match found)*. To be more secure, CC code can be easily modified to translate the results of *no match found* to be 'Role is *Not* allowed (no match found)'.

SPCC, as an addition to YAWL, helps in making sure that the modeler assignments are in compliance with the policy before run time. The current implementation of SPCC uses a specific structure that is designed solely for the purpose of testing the SPCC components. SPCC originally is meant to check assignment requests against the organisation's actual policy. In real life the policy is an organisation-wide policy, therefore the language used for such policies should have the ability to represent the policies and constraints for different kinds of systems. There is a need for a standardised, structured language, such as XACML [62]. The current version of the SPCC application needs to be improved to be able to deal with such a structured policy language. However, a first step is to identify the language requirements. XACML for example, is a structured, standardised language. It is also machine-readable, which is important for enforcing the policies by the system. The next chapter will investigate the issue of using XACML as an authorisation policy language.

5.5 Conclusion

SPCC can be used to overcome the lack of policy compliance checking in YAWL. It helps the process modeler, who is not a security expert, to make sure that all assignments of roles to tasks are in compliance with the authorisation policy. Therefore, it helps in making sure that during run time the role-to-task assignment does not breach of the organisation's authorisation policy.

As stated in the beginning of this chapter, SPCC is a subset of BP-TRBAC. BP-TRBAC is designed to check authorisation requests during run time, and can satisfy more conditions. SPCC is only a design time checking tool. The successful implementation of SPCC demonstrates the feasibility of some of the key concepts we introduced in this thesis.

This implementation showed us the need for a policy language, and that having the model by itself is not enough. When we tried to test the implementation we were faced with the need for a structured policy language that the machine can understand and use. We used the opportunity proposed by this implementation to identify the elements of such a language. As shown in Section 5.2.3, the policy files need to have certain elements. The analysis from Chapter 3 identified the characteristics of such a language. The following chapter will use these findings to introduce a policy language that can be used with BP-TRBAC as a policy model. The language will be designed to be generic and can be used with other authorisation models besides BP-TRBAC.

Chapter 6

BP-XACML

The BP-TRBAC authorisation model introduced in Chapter 4 extends RBAC to support business process systems. The implementation in Chapter 5 showed that with such models comes the need for an authorisation policy language that will work with BPM authorisation models. An authorisation policy language provides the means by which access control policies are expressed in a manner that can be enforced in an information system. One authorisation policy language that has become widely used and accepted is the eXtensible Access Control Markup Language (XACML) [52], an XML defined standard language for authorisation policies. In [7] it was shown that XACML by itself is not enough to support all types of authorisation models. For example, it does not support RBAC policies, as it does not have a way to represent the notion of 'role'. 'XACML-RBAC Profile' was proposed to extend the initial version of XACML to support the notion of roles and be able to support role-based access control policies [7].

In a similar way, BPM authorisation models include specific characteristics that neither XACML nor the XACML-RBAC profile can support. Both profiles do not provide support for business process aspects. For example, they do not provide representation of the notion of tasks or task instances. Moreover, they do not support instance-level restrictions. Therefore, there is a need to extend XACML to support BPM. To the best of the author's knowledge, currently there is no published work that aims to extend the XACML language to support authorisation policies for business processes. A complete description of the characteristics needed for a business process authorisation policy language can be found in Chapter 3.

This chapter will introduce BP-XACML, an extension to XACML, to express authorisation policies for business processes. The proposed extension builds on the RBAC profile to support the notion of tasks and task instances, to support instance level restrictions and separation of duties (SoD) constraints. BP-XACML is designed to support the requirements described in Chapter 3. This chapter describes the new XACML profile, BP-XACML, which introduces a new function called 'performers list' to support the history-based instance-level restrictions. It also proposes a new policy set to support tasks, and a new attribute to recognise the task instances. It introduces new conditions and functions to support SoD in BP-XACML. Figure 6.1 shows the complete framework of BP-XACML with all 'policy-sets' and authorities. Policy-sets are distinguished from authorities by having a fold in the corner. Elements shaded in white are from the XACML standard; those with dotted background are added in the RBAC-XACML profile; while the ones shaded with dark background are newly introduced in this thesis (BP-XACML).

Context handler is responsible for translating received requests into the XACML context, and translating results back to the native language of the other system. It is also responsible for communicating between the other components. In XACML the PDP (Policy Decision Point) is responsible for making decisions on the authorisation requests based on the policy sets. In RBAC-XACML there is a new type of request that deals with role activation. In RBAC-XACML it was decided that role activation should be out of the scope of PDP. For this reason the Role Enablement Authority (REA) was introduced. REA is a specialised PDP that deals only with role activation decisions. BP-XACML has a new type of request, to perform a workflow task. In order to deal with such a request a new specialised PDP called Task Authority (TA) is introduced, to be responsible for workflow task performance decisions.

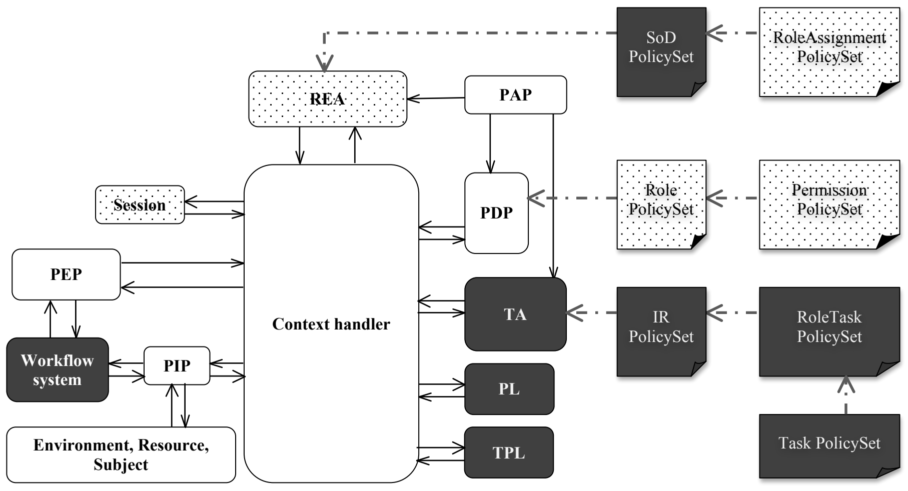


Figure 6.1: BP-XACML authorities and policy sets

The rest of the chapter is organised as follows. The structure of the policy language is discussed in Section 6.1. Section 6.2 explains the policy model. The language semantics are described in Section 6.3. Section 6.4 provides example policies using the BP-XACML language. Section 6.5 discusses BP-XACML pointing out strengths and weaknesses of the new policy model. Section 6.6 provides concluding remarks.

6.1 BP-XACML: Policy Structure

BP-XACML is based on XACML, which implements rule-based access control [76]. An authorisation policy may contain multiple authorisation rules (AR), which are the basic building blocks for stating authorisation restrictions. Each AR consists of four elements: Subject, Object, Action, and Condition, the evaluation of which results in a Allow or Deny decision.

$$AR = \{S, O, A, C\} \rightarrow \{Allow, Deny\}. \tag{6.1.1}$$

Action (A) is implementation specific. Condition (C) is a boolean expression that is evaluated, based on the value of variables determined at run time, as either true or false. Conditions can be used to represent complex constraints. The rule has its specified effect (allow or deny) if the condition evaluates as true.

In BP-XACML the interpretation of the concepts of subject and object is 'policy-set' specific. It differs from one policy set to another depending on what the policy set is regulating access to. For example, in the `Task<PolicySet>` the subject is a role, and the object is a task, while in the `RoleAssignment<PolicySet>` the subject is a user, and the object is a role. Section 6.1.2 will explain the interpretation of subject and object for each policy set in more detail.

Rules are grouped together in 'policies', which may contain a target that limits the applicability of rules to requests matching the target's subject, object and action [62]. Policies also specify a rule-combining algorithm, which resolves potential conflicts when more than one rule is applicable [62]. Policies can be grouped together in a 'PolicySet' that also contains a target, and a policy-combining algorithm. PolicySets may also contain other policy sets included by reference [62].

6.1.1 Request and Decision

An XACML request message is sent to the PDP when a user tries to access a controlled resource. The PDP identifies matching policies based on their target and evaluates the request against them to arrive at an authorisation decision. The Request (RQ) is in the form of {S,O,A}. In BP-XACML there are three types of resources whose related policies are defined in three different policy sets (see Section 6.1.2). In the context of the request, the interpretation of S, O and A are different for each type. Because of this, each type of request is processed by a different authority.

In the case of a user requesting to perform a workflow task, the subject (S), will be the identifier for the specific user making the request. The object (O) is the task that the user wants to perform. Since a task explicitly defines its associated permissions (*i.e.* object-action pairs), they are not separately identified in the request. The Action (A) is simply the request to 'perform'. In the second case a user requests access to a resource object that is not a workflow 'task', for example (based on the scenario from

Chapter 3) to access the ‘pump room’. In this case, O will be the ‘pump room’, and A will be ‘access’. The third type of request is to activate a role, for example, ‘Adam’ wants to activate the role ‘coordinator’. In such requests, S is ‘Adam’, O is the role ‘coordinator’, and A is ‘activate’. The decision (DS) will be either {Allow}, {Deny}, or {Not applicable} if no matching policies are found.

6.1.2 Policy Sets

As explained earlier, ‘Policy sets’ are used to group related policies, which group related access control rules. The RBAC profile of XACML predefines some policy sets and makes use of them to determine the access control decision. For example, a `RoleAssignment<PolicySet>` will include all policies and rules related to role assignment. In this extended profile we make use of these policy sets and introduce new policy sets.

BP-XACML includes seven types of access control policy sets. The PDP will use two policy sets, the `Role<PolicySet>` and the `Permission <PolicySet>`, to make decisions on the requests directed to the PDP. The `Task<PolicySet>` and the `RoleTask<PolicySet>` are used to state the tasks that a role is allowed to perform. `IR<PolicySet>` is used to state instance-level restrictions. The `SoD<PolicySet>` and the `RoleAssignment <PolicySet>` are used for stating and activating roles of each user. The `Role<PolicySet>`, `Permission<PolicySet>`, and `RoleAssignment<PolicySet>` are adopted from the XACML RBAC profile [62]. The rest of the `PolicySets` are newly introduced in BP-XACML. This section will explain the policy sets and their relationship. The mechanism and application of these policy sets will be discussed in more detail in Section 6.3.

Policy Sets for Standard RBAC Requests

Figure 6.2 shows the relation between the policy sets involved in the authorisation decision for a request to access a resource that is not a workflow task (`Role<PolicySet>` and `Permission<PolicySet>`). It also gives a summary of the structure of each of these policy sets, which are described in detail in the following paragraphs.

`Role<PolicySet>` (RPS) is a `<PolicySet>` that links a role with its authorised permissions as specified in the referenced `Permission<PolicySet>`. The `<Target>` element of the `Role<PolicySet>` is used to restrict the applicability of the `<PolicySet>` to subjects holding the corresponding role attribute. Each `Role<PolicySet>` references a single corresponding `Permission<PolicySet>`. It does not contain or reference any other `<Policy>` or `<PolicySet>` elements. In the Role`<PolicySet>`, Subject (S) refers to the user’s role and Object (O) refers to the non-workflow resource (*e.g.* File2).

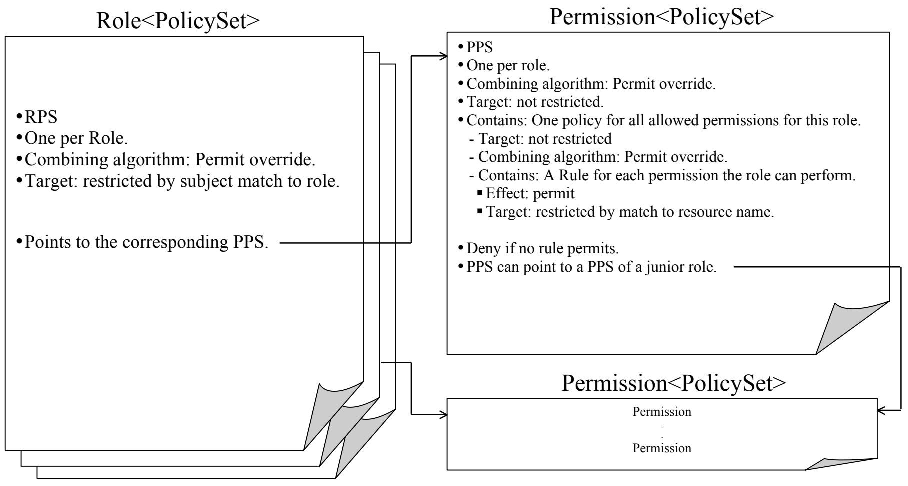

Role<PolicySet>
•RPS
•One per Role.
•Combining algorithm: Permit override.
•Target: restricted by subject match to role.
•Points to the corresponding PPS.
Permission<PolicySet>
• PPS
• One per role.
• Combining algorithm: Permit override.
• Target: not restricted.
• Contains: One policy for all allowed permissions for this role.
- Target: not restricted
- Combining algorithm: Permit override.
- Contains: A Rule for each permission the role can perform.
▪ Effect: permit
▪ Target: restricted by match to resource name.
• Deny if no rule permits.
• PPS can point to a PPS of a junior role.
Permission<PolicySet>
Permission
Permission

Figure 6.2: Policy sets for a standard RBAC requests authorisation

The `Permission<PolicySet>` (PPS) is a `<PolicySet>` that contains the actual permissions associated with a given role. A `Permission<Policy-Set>` contains 'policy' and 'Rules' elements describing the actions and resources that a given subject is permitted to perform. It should also express any conditions related to that access. The `<Target>` element of a `Permission<PolicySet>` should not limit the applicability of the `<PolicySet>`. To achieve role hierarchy, the `Permission<PolicySet>` associated with a senior role may also contain references to `Permission <PolicySet>`s associated with junior roles, thereby allowing the senior role to inherit all access to Permissions associated with the junior roles. We adopted the `Permission<PolicySet>` from the RBAC profile for XACML. In the `Permission<PolicySet>` the subject is the role and the object is a non-workflow resource (*e.g.* File2).

Policy Sets for Workflow Requests

Figure 6.3 shows the relation between the policy sets involved in the authorisation decision on task performance request (IRPS, RTPS, and TPS). It also gives a summary of the structure of each of these policy sets, which are described in detail in the following paragraphs.

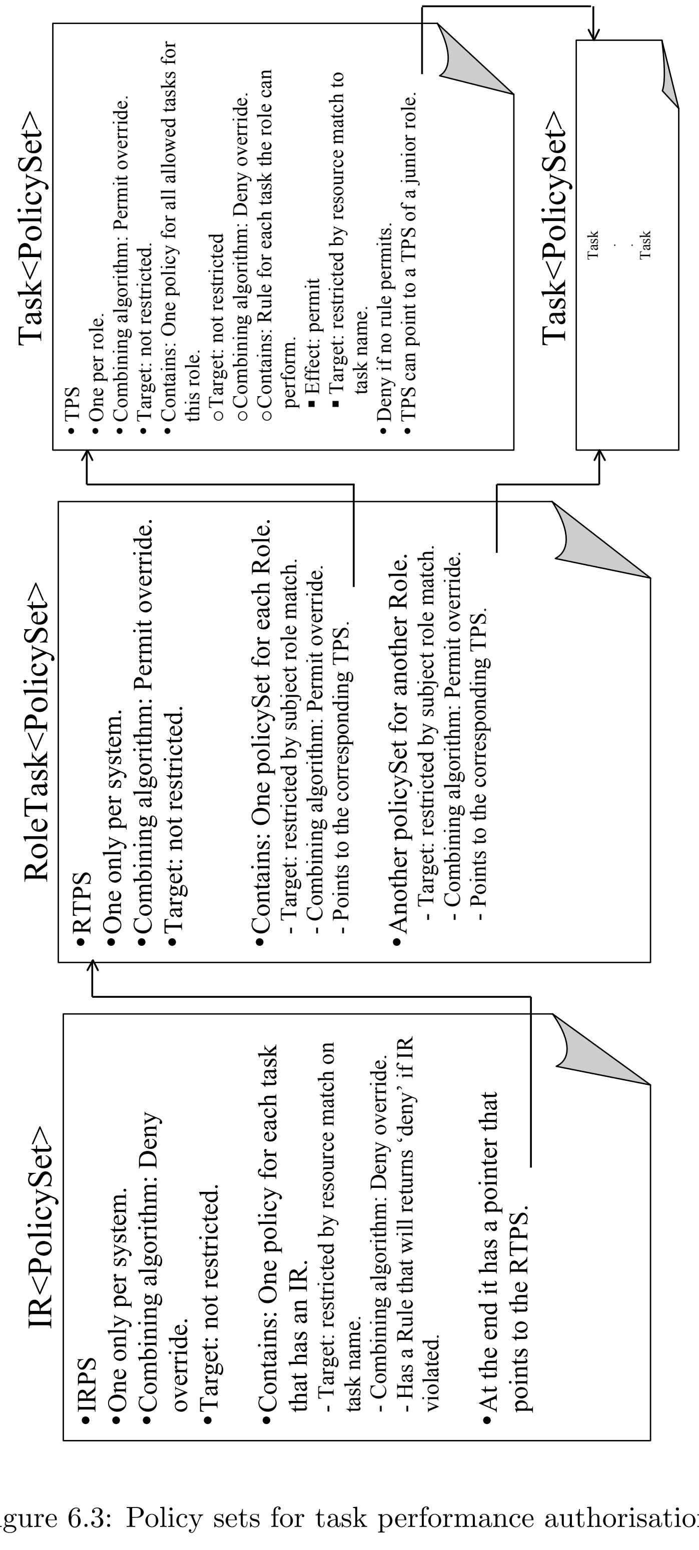

IR<PolicySet>
• IRPS
• One only per system.
• Combining algorithm: Deny override.
• Target: not restricted.
• Contains: One policy for each task that has an IR.
- Target: restricted by resource match on task name.
- Combining algorithm: Deny override.
- Has a Rule that will returns 'deny' if IR violated.
• At the end it has a pointer that points to the RTPS.
RoleTask<PolicySet>
• RTPS
• One only per system.
• Combining algorithm: Permit override.
• Target: not restricted.
• Contains: One policySet for each Role.
- Target: restricted by subject role match.
- Combining algorithm: Permit override.
- Points to the corresponding TPS.
• Another policySet for another Role.
- Target: restricted by subject role match.
- Combining algorithm: Permit override.
- Points to the corresponding TPS.
Task<PolicySet>
• TPS
• One per role.
• Combining algorithm: Permit override.
• Target: not restricted.
• Contains: One policy for all allowed tasks for this role.
○ Target: not restricted
○ Combining algorithm: Deny override.
○ Contains: Rule for each task the role can perform.
▪ Effect: permit
▪ Target: restricted by resource match to task name.
• Deny if no rule permits.
• TPS can point to a TPS of a junior role.
Task<PolicySet>
Task
.
.
Task

Figure 6.3: Policy sets for task performance authorisation

`Task<PolicySet>` (TPS) is a `<PolicySet>` that contains the actual tasks authorized for a given role. The `<Target>` element of a TPS should not limit the applicability of the `<PolicySet>`, as the `IR<PolicySet>` and the `RoleTask<PolicySet>` restrict access (see Figure 6.3). To achieve role hierarchy, a TPS associated with a senior role may also contain references to TPSs associated with junior roles, thereby allowing the senior role to inherit all access to tasks associated with the junior roles. In a TPS, (S) refers to user's role, and (O) refers to task. The `Task<PolicySet>` is newly introduced in BP-XACML.

`RoleTask<PolicySet>` (RTPS) is a `<PolicySet>` that contains the Roles. For each role it points to the corresponding `Task<PolicySet>` (*i.e.* the TPS is included in the RTPS by reference). The `<Target>` element of a RTPS should not restrict the applicability of the `<PolicySet>`, but the `<PolicySet>s` for each role (included within the RTPS) have a target restricting applicability for the specified role only. The RTPS is used to achieve role hierarchy. In BP-XACML users can have more than one role active at the same time. For this reason, the `RoleTask<PolicySet>` includes a `<PolicySet>` for each system role. To evaluate a request to perform a task, each role is checked to see if the user has it active and if so, the corresponding `Task<PolicySet>` is evaluated to see if it grants permission to perform the requested task. In the RTPS, Subject (S) refers to the user's role and Object (O) refers to the task. The `RoleTask<PolicySet>` is newly introduced in BP-XACML.

`IR<PolicySet>` (IRPS) is a `<PolicySet>` that describes instance-level restrictions. The RTPS and TPS can be reached only through the IRPS, where they are included by reference. The Task Authority will first access this policy set to check that there is no violation of an IR constraint; then it will be pointed via the `RoleTask<PolicySet>` to the related `Task <PolicySet>`. Section 6.3.4 explains this in more detail. In the IRPS, the subject (S) is not restricted because IR constraints deal with task instances regardless of the subject. The object (O) is the task, and the action (A) is 'perform'. For example, no user is allowed to perform both 'issue work order' and 'approve work order'. In this case the IRPS will have both tasks in one policy, making sure that the user does not perform both for the same instance. `IR<PolicySet>` is newly introduced in BP-XACML.

Policy Sets for Role Activation Requests

Figure 6.4 shows the relation between the policy sets involved in the authorisation decision on role activation request (SoDPS, and RAPS). It also gives a summary of the structure of each of these policy sets, which are described in detail in the following paragraphs.

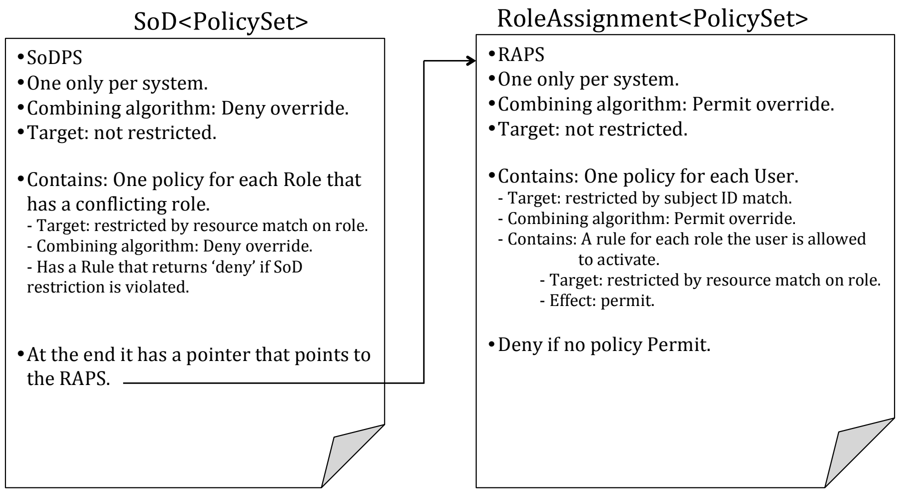

SoD<PolicySet>
•SoDPS
•One only per system.
•Combining algorithm: Deny override.
•Target: not restricted.
•Contains: One policy for each Role that has a conflicting role.
- Target: restricted by resource match on role.
- Combining algorithm: Deny override.
- Has a Rule that returns 'deny' if SoD restriction is violated.
•At the end it has a pointer that points to the RAPS.
RoleAssignment<PolicySet>
•RAPS
•One only per system.
•Combining algorithm: Permit override.
•Target: not restricted.
•Contains: One policy for each User.
- Target: restricted by subject ID match.
- Combining algorithm: Permit override.
- Contains: A rule for each role the user is allowed to activate.
- Target: restricted by resource match on role.
- Effect: permit.
•Deny if no policy Permit.

Figure 6.4: Policy sets for role activation authorisation

`RoleAssignment<PolicySet>` (RAPS) is a `<PolicySet>` that describes which roles can be enabled by which users. This type of policy is used by a Role Enablement Authority, which will be explained in 6.2.1. In the `RoleAssignment<PolicySet>`, the subject (S) is the user, the object (O) is the role, and the action (A) is 'request for activation'. RoleAssignment `<PolicySet>` was adopted from the RBAC profile for XACML.

`SoD<PolicySet>` (SoDPS) is a `<PolicySet>` that describes separation of duties constraints. It restricts access to the RoleAssignment `<PolicySet>`. The Role Enablement Authority will first access this policy set to check that there is no violation of any SoD constraint, and then will be pointed to the `RoleAssignment<PolicySet>`. In a SoDPS, the subject (S) is not restricted because SoD constraints deal with roles regardless of the subject. The object (O) is a role. Each policy in the `SoD<PolicySet>` includes a pair of conflicting roles. An early draft of the RBAC-XACML [6] had proposed the idea of the `SoD<PolicySet>`. This policy set was then removed in the final version [7], which states that "the policies specified in this profile do not deal with static or dynamic separation of duties" [7]. In this chapter we have adopted the idea of the policy set, but we implemented the `SoD<PolicySet>` in a different way. In that earlier RBAC-XACML draft [6], all the roles assigned to a user are assumed to be simultaneously active, which effectively means there is no way to prevent the activation of pairs of roles which are subject to a dynamic separation of duty constraint. Instead they use the `SoD<PolicySet>` to prevent "a user who possesses conflicting role attributes from gaining any access to resources" [6]. While in the BP-XACML architecture the `SoD<PolicySet>` is used to avoid activating two conflicting roles from the beginning. The architecture of BP-XACML implements the `SoD<PolicySet>` in a way to make use of the 'session' function, which reports a user's active roles.

6.1.3 Conditions

A condition is specified as a Boolean expression that is evaluated at runtime. There are two main types of policy conditions of interest in specifying access restrictions in this policy model. The first one is dynamic Separation of Duty (SoD) conditions on role level. The other is instance-level restrictions (IR).

A dynamic SoD condition is an expression that can be evaluated for user-role relation by testing the current active roles for this user. It is used to prevent a user from activating two conflicting roles at the same time. They are defined as policies in the `SoD<PolicySet>`.

$$SoD : (Role1, Role2). \tag{6.1.2}$$

Dynamic SoD constraints are defined in the `SoD<PolicySet>`. The role enablement authority (REA) will be able to know the SoD restriction before enabling a role. It will use the 'session' component to check the current status of role enablement for a user requesting role activation. The session maintains the list of active roles for each user.

An 'instance-level restriction' (IR) condition is an expression that can be evaluated to check the relation between two objects within the same instance. They are defined as policies within the `IR<PolicySet>`.

$$IR : (\{Task1, Task2\}, type). \tag{6.1.3}$$

IR is a type of SoD (or BoD) restriction on a task level that applies only within the same instance. It makes sure that the restriction is met within the same instance. For example, the task of 'closing work order' should have a restriction that it can be done only by the same user who performed 'issue work order' for this same instance.

6.2 BP-XACML: Policy Model

BP-XACML is designed to support backward-compatibility with the RBAC-XACML policy structure. This has an important benefit. It means that role-based authorisation policies can be defined and managed independently of the workflow authorisation system. These policies will still be applied when a user requires access to a controlled resource to execute an instance of a business process. This design approach introduces some complexity, most notably in the inclusion of the Task Authority as a separate PDP to authorise task activation. But it is necessary because the role-based authorisation policies that control access to an organisation's valuable resources, (*e.g.* customer records, financial records) are typically created and maintained independently of the business processes. These policies will often exist before a workflow is created that uses the controlled resources. These policies still need to be applied in the context of the workflow, but we argue that this should not be done by a parallel and duplicated workflow authorisation system, since this would be inefficient and difficult to maintain. Accordingly, we have designed the BP-XACML policy structure to work with an existing RBAC-XACML policy set. This results in an integrated system which can handle both workflow and non-workflow requests from a single (and therefore consistent) set of policies.

After explaining the structure of the BP-XACML policy language, this section describes the BP-XACML policy model, showing how access decisions are made using the defined 'policy sets', describes the needed authorities and repositories, and explains the policy model framework.

6.2.1 Authorities and Repositories

In this policy model we introduce a new authority and two repositories that are needed to fulfill the requirements. They are the Task Authority (TA), Performers List (PL), and Task-Permissions List (TPL). We also include the 'Role Enablement Authority (REA)' and the 'session' concept from the XACML RBAC profile, and we elaborate on how to use them, as the RBAC profile does not provide these details.

It might appear unnecessarily complex to add the REA and TA, which are essentially specialised PDPs. One might argue that one PDP should be enough. However, the RBAC-XACML profile [7], which is proposed by OASIS, adopts this approach in introducing the REA. The RBAC profile shows that role enablement should be out of the scope of the PDP; that is why REA was introduced to be responsible for role assignment and enablement [7]. The justification for having a specialised PDP for role enablement can be understood by looking at the basic request concept of XACML, where each request contains a subject and an object. The PDP is designed to deal with one interpretation of each aspect of the request (subject, object, action). For example, in RBAC-XACML if the PDP receives a request, it will look at the subject as the user's role, and the object as the resource that the user wants to perform an action on. For example, Adam, who is a manager, wants to read file2. The PDP will use the permission policy to determine if managers are allowed to read file2. A request to activate a role has the subject as the user ID, and the object as the role that needs to be activated. That is why it was necessary to have a specialised PDP called REA. This REA is designed to look at the subject as the user's ID, with the object as the role that the user wants to activate. Therefore it will be able to deal with activation requests.

BP-XACML deals with three different type of requests, where each type of request has a different interpretation of subject and object. The change in subject and object interpretation makes it necessary to have a different authority to deal with each different type of request. The request to perform a task has the user's role as the subject and the task ID as the object. Therefore, the authorisation of task performance is out of the scope of the PDP. TA is introduced in this model to be the specialised PDP responsible for making task performance decisions. It permits compatibility with the role and permission policy sets defined in RBAC-XACML. TA deals with requests on the basis that the subject is the user's role and the object is the task ID. Therefore, it is able to deal with requests to perform a task.

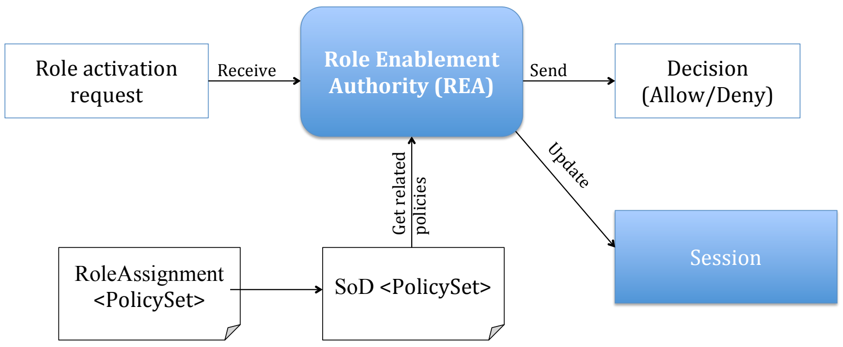


Figure 6.5: Role enablement authority

Role Enablement Authority (REA): uses the `SoD<PolicySet>`, and `RoleAssignment<PolicySet>` to either allow or deny activation of a specific role for a specific user.

Session: provides a queryable service, which maintains and continuously refreshes the state of user-role enablement relations.

Figure 6.5 shows how REA uses the `RoleAssignment<PolicySet>` to know if the user is allowed to enable a role or not. Before reaching the `RoleAssignment<PolicySet>`, REA checks the `SoD<PolicySet>` for an SoD policy for this role. If such a policy exists, REA needs to know the status of the user's activated roles. This information can be retrieved from the user's session. The information allows REA to evaluate if the condition is met or not. Based on that, the REA will send the final decision on the role enablement request.

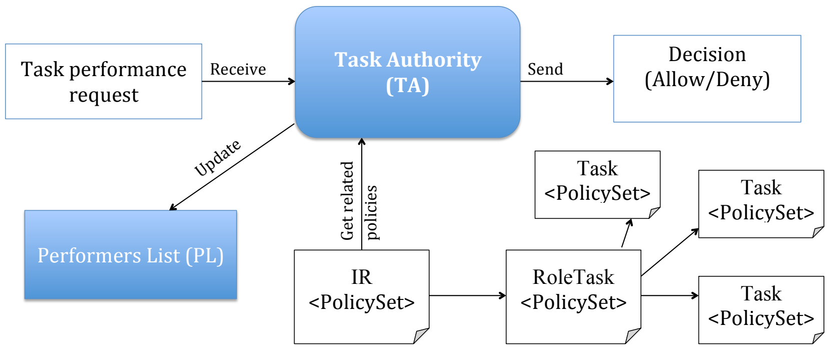


Figure 6.6: Task authority

Task Authority (TA) uses the `IR<PolicySet>`, and `Task<PolicySet>` to either allow or deny a user's request to perform a specific task.

Performers List (PL) is introduced to provide a queryable service to report the user that performed a completed task instance. It maintains the state of user 'task instance' performance relations, and continuously refreshes the state.

As can be seen in Figure 6.6, TA uses the `Task<PolicySet>` to check if the role is allowed to perform the task or not. Before checking the `Task<PolicySet>`, TA first checks the `IR<PolicySet>` to determine if there are any instance-level restrictions on the requested task. If a restriction is found, TA needs to retrieve extra information to assess the restriction. This information can be found through the 'Performers List' (PL). In order for an IR condition to be evaluated, it is necessary to know the performer of a given task instance.

Task-Permissions List (TPL) is a new proposal. It maintains the state of task-permissions relations. As can be seen in Figure 6.7, TPL is used by the context handler (CH) to determine the set of permissions associated with each task. TPL provides a list of permissions for each task, where a permission is an action on a resource.

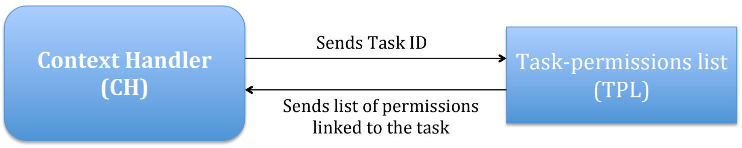

Context Handler (CH)
Sends Task ID
Sends list of permissions linked to the task
Task-permissions list (TPL)

Figure 6.7: Task-Permissions List (TPL)

6.2.2 Access Control

BP-XACML policy framework controls three types of access control requests: activating a role (controlled by the REA), performing a workflow task (controlled by the TA), and performing an action on a resource; we call this 'standard resource' request (controlled by the PDP). A standard resource request can arise in two ways. Firstly, a standard resource request can arise in the context of a non-workflow request where a user requests, for example, to read a file. Secondly, a standard resource request can arise in the context of a workflow request: after the TA has authorised the user to perform the task, the individual permissions required to perform a task are authorised by the PDP individually.

The context handler is responsible for forwarding the request to the corresponding authority depending on its type. Access control is implemented using the seven defined types of `<PolicySet>s`: `Role<PolicySet>`, `Permission<PolicySet>`, `IR <PolicySet>`, `RoleTask<PolicySet>`, `Task <PolicySet>`, `SoD<PolicySet>`, and `RoleAssignment<PolicySet>`.

One `Role<PolicySet>` is defined for each role. To make sure that the 'PolicySet' is applicable only to subjects with the given role attribute, the 'PolicySet' contains a `<Target>` element to limit the applicability of the 'PolicySet'. The `Role<PolicySet>` also contains a single `<PolicySetId Reference>` element, which refers to a unique `Permission<PolicySet>` associated with this role. No other 'Policy', 'PolicySet', or 'PolicyIdReference' elements are included in the `Role<PolicySet>`. These relationships are shown in Figure 6.2.

`Permission<PolicySet>` is a set of all permissions that can be performed by a given role. For each role, one `Permission<PolicySet>` is defined. It is applicable for a specific role, but should not be limited to this role. As explained previously, the `Permission<PolicySet>` may contain a `<PolicySetIdReference>` element that references another `Permission<P-olicySet>` of a junior role to achieve permission inheritance for hierarchal roles.

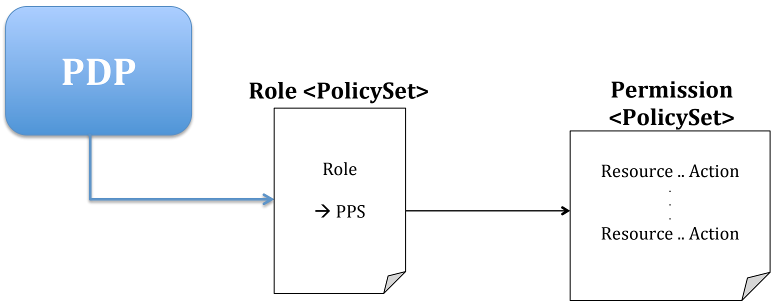


Figure 6.8: PDP can only access role policy set

As shown in Figure 6.8, `Permission<PolicySet>` instances must be stored in a policy repository in such a way that they can never be reached directly by the PDP; `Permission<PolicySet>` instances must be reachable only through the corresponding `Role<PolicySet>`. This is because, in order to support 'hierarchical roles', the `Permission<PolicySet>` depends on its corresponding `Role<PolicySet>` to ensure that only subjects holding the corresponding role attribute will gain access to the permissions in the given `Permission<PolicySet>`.

For the PDP to make a decision on an access request it accesses the `Role<PolicySet>s` for the specific role(s) that the requester has activated. These 'PolicySets' will point to their related `permission<PolicySet>` that contains the access control rules related to the role.

A single `SoD<PolicySet>` is defined in the system, which contains all SoD restrictions. The policy set itself is not limited (*i.e.* the target is empty and therefore does not restrict the applicability of the included policies), but each policy is limited to a specific role. The policy set contains a single `<PolicySetIdReference>` element, which refers to the `RoleAssignment <PolicySet>` (RAPS). There is a single RAPS in a system, which contains the information on whether to allow or deny the role activation for a specific user.

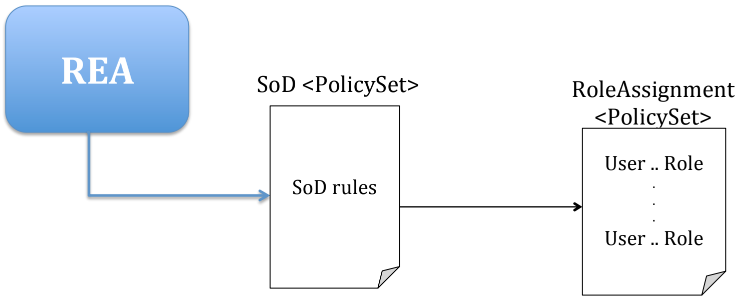


Figure 6.9: REA can only access SoD policy set

As shown in Figure 6.9, the RAPS must be stored in a policy repository in such a way that it can never be reached directly by the REA; RAPS must be reachable only through the SoDPS. This is because, in order to support separation of duties, it is important that the SoD policies are checked before reaching the RAPS. For REA to achieve a decision on a role activation request, it will first check any related SoD restrictions through SoDPS, then it will be pointed to RAPS to check if the user is allowed to activate the role.

A single `IR<PolicySet>` is defined in the system, which contains all IR restrictions. The policy set itself is not limited, as the policy set target is empty, but each policy is limited to a specific task. The policy set contains a single `<PolicySetIdReference>` element, which refers to the `RoleTask<PolicySet>`. For the system there is a single `RoleTask<Policy Set>`, with `<PolicySet>s` for each role, pointing to their corresponding `Task<PolicySet>`. A user will be authorised to perform a task if there is a permit rule for the task in the TPS for a role that the user has active.

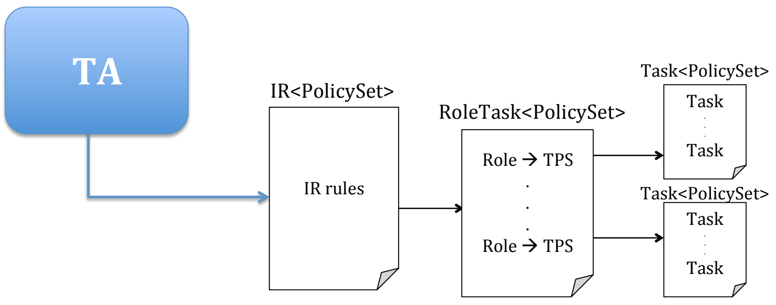


Figure 6.10: TA can only access the IR policy set

As shown in Figure 6.10, TPS instances and the RTPS must be stored in a policy repository in such a way that they can never be reached directly by the TA. RTPS must be reachable only through the IRPS. This is because, in order to support 'role hierarchy', the TPS depends on the RTPS to ensure that only subjects holding the corresponding role attribute (or senior role) will gain access to perform tasks in the given TPS. For TA to make a decision on a request to 'perform a task', it first must access the IRPS and check if there are any related IR policies. IRPS will then point to RTPS. Using the user's role, RTPS points to the corresponding TPS, which contains rules stating whether this role is allowed to perform a task or not. These `<PolicySet>` relationships and constraints are summarised in Figure 6.3.

6.2.3 Policy Framework

Figure 6.11 shows the complete BP-XACML framework without the 'policy sets'. It includes all the authorities, components, and repositories. As explained earlier, the policy model should be unified and should deal with all authorisation requests, regardless of whether or not they arise in the context of a workflow. For this reason, the BP-XACML policy model is designed to deal with several types of requests. It could be a request to activate a role for a user, a request to perform a task, or a standard resource request to perform an action on a resource. This section discusses each type of request separately, showing how it is handled within the framework.

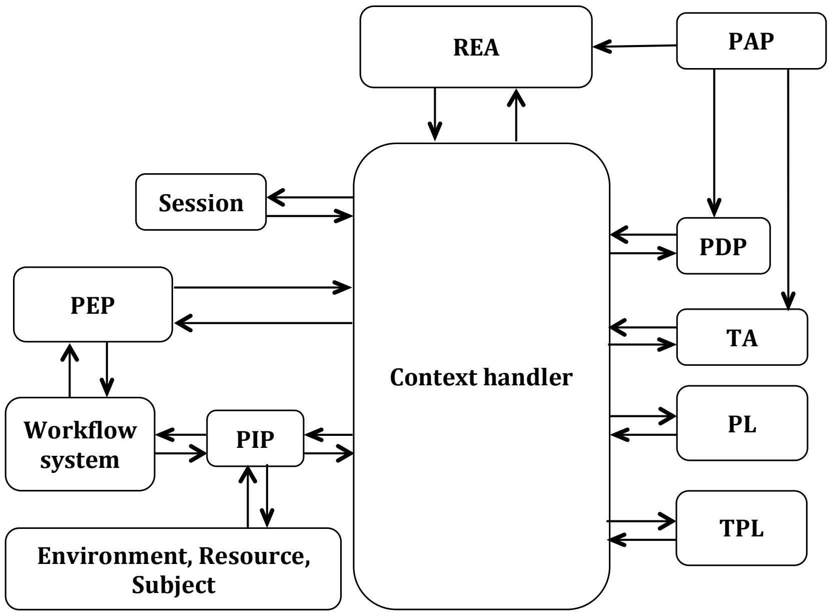


Figure 6.11: BP-XACML framework

The role activation requests are directed to the 'role enablement authority (REA)' by the context handler. The REA will use the `SoD<PolicySet>` to check for any SoD restrictions on this role. It will then point to the RAPS to decide if this user is allowed to activate this role or not. If there was an SoD constraint on the role, REA needs to make sure that activating this role will not breach the SoD condition by requesting information about the activated roles of this user from the 'session' through the CH.

Figure 6.12 shows the steps related to a role activation request. As can be seen, the first step is when the PAP loads the `SoD<PolicySet>` to the 'role enablement authority (REA)'. When CH receives a request for role activation from the PEP (Step 2), it will forward the request to the REA (Step 3). If REA finds any SoD restriction related to this request it then asks the CH for extra information regarding this user's session (Step 4). CH will query the Session to get the user's active roles (Step 5). After getting the session information (Step 6), CH will forward the information to the REA (Step 7). After that the REA will make a decision based on evaluating the `SoD<PolicySet>` and the `RoleAssignment<PolicySet>`. The decision will be sent to the CH (Step 8), and then forwarded to PEP (Step 9). CH will also update the user's session to add the new role if it was activated (Step 10).

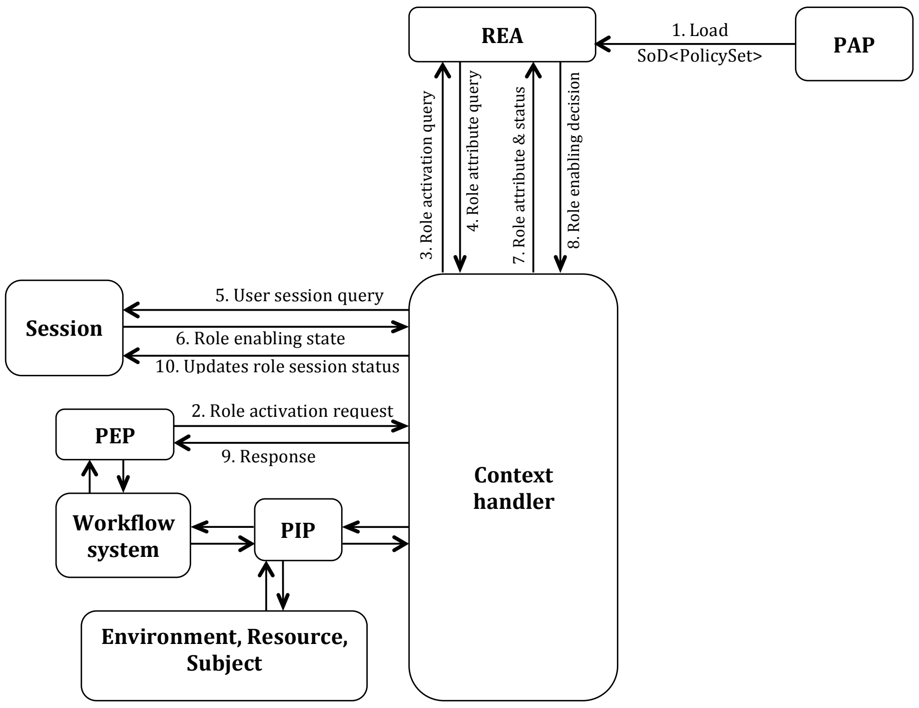


Figure 6.12: Flowchart of an activating a role request

The standard resource request is a request to perform an action on a resource, where the resource is not a role or a task (*e.g.* read a file). It is the type of request that was described in Chapter 4 as a non-workflow request. This type of request is directed to the PDP, and will be handled in the same way access requests are handled in the RBAC-XACML profile [62] using the `Role<PolicySet>` and `Permission<PolicySet>`. The context handler will forward the request to the PDP. The PDP will use the user's active roles to identify the proper `Role<PolicySet>s`, which contain the permissions that this user is allowed to perform based on the roles activated at the request time. The `Role<PolicySet>s` will point to the related `Permission<PolicySet>s`, where each is a collection of permissions that are assigned to a specific role. A permission is an action on a resource. To achieve role hierarchy, a `Permission<PolicySet>` can include a `<PolicySetIdReference>` to refer to `Permission<PolicySet>s` of junior roles. In BP-XACML, a request to perform a task will produce a set of one or more requests of this type (standard resource request), generated by the context handler and submitted individually to the PDP.

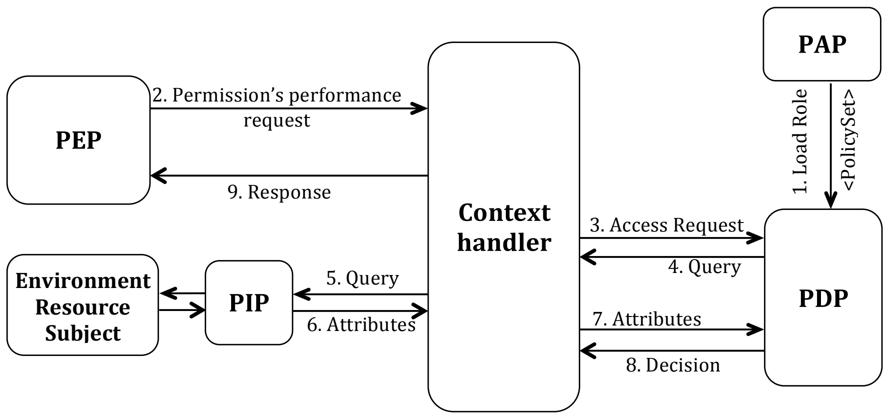


Figure 6.13: Flowchart of a standard resource request

Figure 6.13 shows the flowchart of a standard resource request. First, PAP loads `Role<PolicySet>s` to the PDP (Step 1). When the CH receives the request from PEP (Step 2), it forwards the request to PDP (Step 3). Sometimes the PDP requires extra information to be able to evaluate a condition. PDP sends a query to the CH (Step 4), which will be gathered via the PIP (Step 5). When CH receives the needed information (Step 6), it forwards it to the PDP (Step 7). After that PDP makes a decision on the request based on the `Role<PolicySet>` and the `Permission<PolicySet>`, and sends the decision to the CH (Step 8). CH then forwards the decision to PEP (Step 9).

If the request was to perform a specific task, the context handler will foreword the request to the TA. The TA will use the IRPS to check if there are any IR restrictions on this task. Violation of an IR results in a deny decision as the combining algorithm for the `IR<PolicySet>` is deny overrides. Then it will use the user's active roles to identify the applicable TPS through the RTPS. TPS identifies the tasks that this user is allowed to perform, based on the roles activated at the request time. If an IR is restricting the assigning of a task, the TA will obtain extra information from the 'performers list (PL)' through the context handler to evaluate the IR condition. If the condition is violated, the effect of the IR rule will be 'deny' and the task request will be denied since the combining algorithm for the `IR<PolicySet>` is 'deny overrides'.

If the TA allows the user to perform the task, CH will use the Task Permissions List (TPL) to retrieve all permissions associated with the task. Each permission is a pair of an action and a resource. CH will create a standard resource request for each permission, with requests containing the subject-ID, which can be used to retrieve the active roles, action, and resource. These requests will be sent to the PDP. The PDP will send back each decision individually. CH will combine the decisions, where deny overrides. So, if one request was denied, the whole request to perform the task will be denied, since the task cannot be completed. If all requests were allowed, the CH will then send back to the PEP that this user is allowed to perform the task.

Figure 6.14 shows the flowchart for task performance requests. Firstly, PAP will load the `Role<PolicySet>s` to the PDP (Step 1), and the `IR <PolicySet>` to the TA (Step 1.b). When the PEP sends a task performance request to the CH (Step 2), the CH will forward the request to the TA (Step 3). TA will check IRPS for any related instance-level restrictions. If more information is needed to be able to evaluate the IR policy, TA will query the CH for this information (Step 4). CH will use the PL to get the information about the users who performed certain completed task instance (Step 5). After retrieving the information (Step 6), CH will forward the information to the TA (Step 7). After that TA will make a decision on the request based on the IRPS, RTPS, and TPS. The decision will then be sent to the CH (Step 8). If the decision was ‘deny’ then CH will send ‘deny’ to PEP. If it was allow then CH will query TPL to get all the permissions related to this task (Step 9). After receiving the set of permissions related to the task (Step 10), CH will create a standard resource request for each permission and send it to the PDP (Step 11). Similar to Figure 6.13, PDP might require extra information that will be obtained through the CH and provided to PDP (Steps 12, 13, 14, and 15). A decision on the permission request will be sent to the CH (Step 16). CH will repeat these steps for each permission related to the task in the request. It makes a final decision based on the collective decisions from the PDP using combining algorithm ‘deny-override’. Then it sends the final decision to PEP (Step 17).

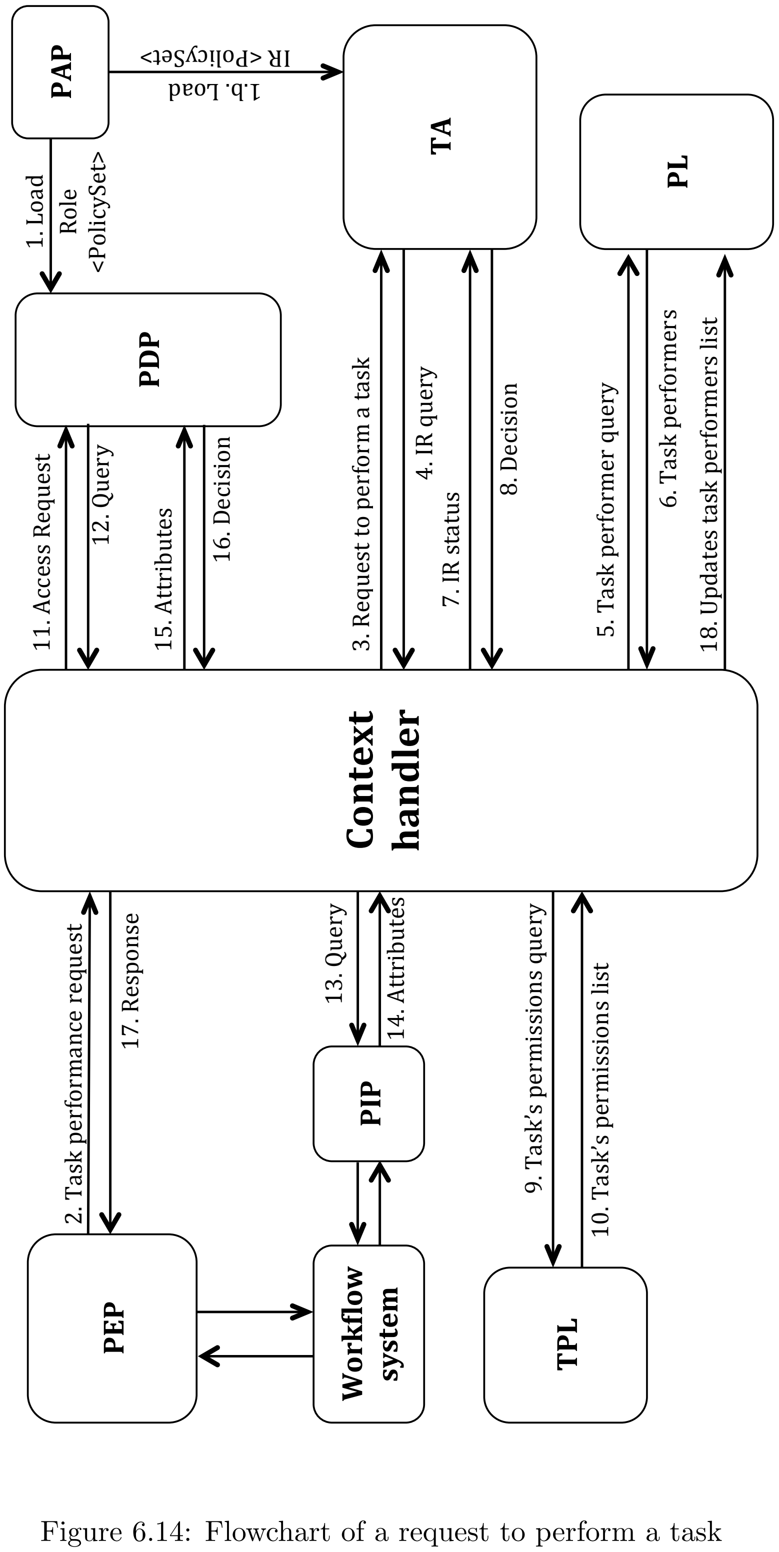


Figure 6.14: Flowchart of a request to perform a task

6.3 BP-XACML: Policy Semantics

In this section, we will refine the previously described policy structure with specific data and language representations.

6.3.1 Users, Roles, Operations, and Permissions

Users, roles, operations and permissions are all part of the RBAC profile of XACML. In this chapter we adopt these entities and the way they are expressed from RBAC-XACML [7]. So, reflecting how they are expressed in RBAC-XACML, in BP-XACML users are expressed as subjects, and Roles are expressed as attributes of subjects. Operations are expressed using XACML Actions, and Permissions are expressed using the `Permission<PolicySet>`. In the listings we use simple examples such as 'coordinator', 'perform', and 'work-order' to improve readability. Note that in an actual implementation an appropriate naming convention should be adopted to ensure the uniqueness of these labels.

Listing 1 is an example permission policy set that includes a permission policy. The example permission policy includes two rules. One rule (lines 13 to 29) allows the action 'write' on the object 'work order'. The other rule (lines 33 to 49) allows the action 'read' on the object 'work order'.

```
1  <PolicySet ... PolicySetId="PPS:coordinator:role"
       PolicyCombiningAlgId="&policy-combine;permit-overrides">
2   <Target>
3    <Subjects><AnySubject/></Subjects>
4    <Resources><AnyResource/></Resources>
5    <Actions><AnyAction/></Actions>
6   </Target>
7  <Policy PolicyId="Permissions" RuleCombiningAlgId="&rule-combine;
       permit-overrides">
8   <Target>
9    <Subjects><AnySubject/></Subjects>
10   <Resources><AnyResource/></Resources>
11   <Actions><AnyAction/></Actions>
12  </Target>
13 <Rule RuleId="Permission:to:write:in:work:order" Effect="Permit">
14  <Target>
15   <Subjects><AnySubject/></Subjects>
16   <Resource>
17    <ResourceMatch MatchId="&function;string-match">
18     <AttributeValue DataType="&xml;string">work order</
           AttributeValue>
19     <ResourceAttributeDesignator AttributeId="&resource;resource-id
           " DataType="&xml;string"/>
20    </ResourceMatch>
21   </Resource>
22   <Actions>
23    <ActionMatch MatchId="&function;string-match">
24     <AttributeValue DataType="&xml;string">write</AttributeValue>
```

```
25     <ActionAttributeDesignator AttributeId="&action;action-id"
           DataType="&xml;string"/>
26    </ActionMatch>
27   </Actions>
28   </Target>
29   </Rule>
30   .
31   .
32   .
33 <Rule RuleId="Permission:to:read:work:order" Effect="Permit">
34  <Target>
35   <Subjects><AnySubject/></Subjects>
36   <Resource>
37    <ResourceMatch MatchId="&function;string-match">
38     <AttributeValue DataType="&xml;string">work order</
           AttributeValue>
39     <ResourceAttributeDesignator AttributeId="&resource;resource-id
           " DataType="&xml;string"/>
40    </ResourceMatch>
41   </Resource>
42   <Actions>
43    <ActionMatch MatchId="&function;string-match">
44     <AttributeValue DataType="&xml;string">read</AttributeValue>
45     <ActionAttributeDesignator AttributeId="&action;action-id"
           DataType="&xml;string"/>
46    </ActionMatch>
47   </Actions>
48   </Target>
49   </Rule>
50   </Policy>
51   </PolicySet>
```

Listing 1: Permission policy-set example

The policy set includes a permission policy (lines 7-50) that includes rules that allow the action *write* on the object *work order* and the action *read* on the object *work order*. As can be seen, the subject is not limited to the role 'coordinator' (line 3). This is because, as explained in section 6.1.2, `Permission<PolicySet>s` should not be limited. As can be seen in Listing 2, the `Role<PolicySet>` for the role 'coordinator' limits the applicability to the 'coordinator' role (lines 4-10). Then it points to the corresponding permission policy set (line 15). It can be also seen in Listing 2 that 'coordinator' is seen as a value of the attribute 'role' in the subject (lines 6-7).

```
1 <PolicySet ... PolicySetId="RPS:coordinator:role"
      PolicyCombiningAlgId="&policy-combine;permit-overrides">
2  <Target>
3   <Subjects>
4    <SubjectMatch MatchId="&function;any-of">
5    <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:string-
         equal">
```

```
6      <AttributeValue DataType="&xml;string">coordinator</
           AttributeValue>
7      <SubjectAttributeDesignator AttributeId="urn:someapp:attributes
           :role" DataType="&xml;string"/>
8      </Apply>
9     </SubjectMatch>
10   </Subjects>
11   <Resources><AnyResource/></Resources>
12   <Actions><AnyAction/></Actions>
13  </Target>
14  <!-- Use permission associated with the "coordinator" role -->
15  <PolicySetIdReference>PPS:coordinator:role</PolicySetIdReference>
16 </PolicySet>
```

Listing 2: An example role policy-set for the role ‘coordinator’

The policy set is limited to a specific role. For example, the policy set in Listing 2 is limited to subjects with role attribute equal to ‘coordinator’ (lines 5-7), and points to the permission policy set called PPS:coordinator:role (line 15).

6.3.2 Task and Task instances

Task and task instances are new features that are not supported in RBAC-XACML. In BP-XACML tasks are expressed as an XACML Resource. Listing 3 shows an example `Task<PolicySet>` showing the task as a resource (lines 16-21).

```
1  <PolicySet ... PolicySetId="TPS:coordinator:role"
       PolicyCombiningAlgId="&policy-combine;permit-overrides">
2   <Target>
3     <Subjects><AnySubject/></Subjects>
4     <Resources><AnyResource/></Resources>
5     <Actions><AnyAction/></Actions>
6   </Target>
7  <Policy PolicyId="Allowed tasks" RuleCombiningAlgId="&policy-
       combine;permit-overrides">
8   <Target>
9     <Subjects><AnySubject/></Subjects>
10    <Resources><AnyResource/></Resources>
11    <Actions><AnyAction/></Actions>
12  </Target>
13  <Rule RuleId="issue:work:order:task Effect="Permit">
14 <Target>
15   <Subjects><AnySubject/></Subjects>
16   <Resources>
17    <ResourceMatch MatchId="&function;string-match">
18     <AttributeValue DataType="&xml;string"> issue work order</
           AttributeValue>
19     <ResourceAttributeDesignator AttributeId="&resource;resource-id
           " DataType="&xml;string"/>
20    </ResourceMatch>
```

```
21    </Resources>
22    <Actions>
23      <ActionMatch MatchId="&function;string-match">
24      <AttributeValue DataType="&xml;string">perform</AttributeValue>
25      <ActionAttributeDesignator AttributeId="&action;action-id"
          DataType="&xml;string"/>
26     </ActionMatch>
27    </Actions>
28   </Target>
29  </Rule>
30  </Policy>
31  </PolicySet>
```

Listing 3: Task policy set

In Listing 3, task was represented as a resource (lines 16-21) in the policy because TPSs are linking the user's role to the task, so the task is the object. To be able to represent 'Task instance', a new object attribute called 'instance' is introduced. It is similar to the 'role' attribute from the RBAC-XACML profile. In section 6.3.4 an example listing showing an instance-level restriction will show how to make use of the new attribute 'instance'.

The `Task<PolicySet>` includes a policy containing rules for each task the role is allowed to perform (lines 7-30). The example includes a rule for the task 'issue work order' (lines 13-29) as a part of the policy set. The rule says if someone wants to perform the action 'perform' (lines 22-27) on the resource 'task:issue work order' (lines 16-21) they will be permitted (line 13). If there exists another rule that has the effect of denying the user from performing the task, the end result will still be allowed, because the combining algorithm is 'permit overrides' (line 7), which means if one rule says 'permit' then the result will be permit regardless of the other rules. As can be seen in the listing, the policy set target (lines 2-6) is not limiting the applicability of the policy set. `RoleTask<PolicySet>` will limit the applicability to users with the role 'coordinator' and then point to this policySet. Listing 4 is an example `RoleTask<PolicySet>`. As can be seen the `RoleTask<PolicySet>` contains a `<PolicySet>` for each role. The example in Listing 4 shows an example of two roles: coordinator (lines 7-21), and contractor (lines 25-39). Each `<PolicySet>` applicability is limited to the role itself, via subject match on the role attribute, and then points to the corresponding `Task<PolicySet>`. For example, it can be seen that `<PolicySet>` for the role 'coordinator' is limited to subject role = coordinator (lines 9-16), and is pointing to the TPS for the role coordinator (line 20).

```
<PolicySet ... PolicySetId="RTPS" PolicyCombiningAlgId="&policy-combine;permit-overrides">
 <Target>
   <Subjects><AnySubject/></Subjects>
   <Resources><AnyResource/></Resources>
   <Actions><AnyAction/></Actions>
 </Target>
<PolicySet ... PolicySetId="RTPS:coordinator:role" PolicyCombiningAlgId="&policy-combine;permit-overrides">
 <Target>
   <Subjects>
   <SubjectMatch MatchId="&function;any-of">
   <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:string-equal">
    <AttributeValue DataType="&xml;string">coordinator</AttributeValue>
    <SubjectAttributeDesignator AttributeId="urn:someapp:attributes:role" DataType="&xml;string"/>
    </Apply>
   </SubjectMatch>
  </Subjects>
  <Resources><AnyResource/></Resources>
  <Actions><AnyAction/></Actions> </Target>
 <!-- Use tasks associated with the "coordinator" role -->
 <PolicySetIdReference> TPS:coordinator:role</PolicySetIdReference>
</PolicySet>
  .
  .
  .
  <PolicySet ... PolicySetId="RTPS:contractor:role" PolicyCombiningAlgId="&policy-combine;permit-overrides">
 <Target>
   <Subjects>
   <SubjectMatch MatchId="&function;any-of">
   <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:string-equal">
    <AttributeValue DataType="&xml;string">contractor</AttributeValue>
    <SubjectAttributeDesignator AttributeId="urn:someapp:attributes:role" DataType="&xml;string"/>
    </Apply>
   </SubjectMatch>
  </Subjects>
  <Resources><AnyResource/></Resources>
  <Actions><AnyAction/></Actions> </Target>
 <!-- Use tasks associated with the "contractor" role -->
 <PolicySetIdReference> TPS:contractor:role</PolicySetIdReference>
</PolicySet>
</PolicySet>
```

Listing 4: An example role task policy set

6.3.3 SoD on Role Level

In BP-XACML, SoD is expressed as policies in the `SoD<PolicySet>`. SoD refers to the dynamic role level separation of duty, which is used to make sure that no one user activates two conflicting roles at the same time. Listing 5 shows an example SoD policy set. The policy set includes policies stating conflicting roles. For example, the policy set in Listing 5 includes a policy (lines 7-44) stating that in order to activate the role ‘coordinator’, the role ‘manager’ should not be in the activated roles of the same user (lines 18-30). Also the role ‘contractor’ should not be in the activated roles of the same user (lines 31-43).

```
 1 <PolicySet ... PolicySetId="SoD" PolicyCombiningAlgId="&policy-
       combine;First-applicable">
 2  <Target>
 3    <Subjects><AnySubject/></Subjects>
 4    <Resources><AnyResource/></Resources>
 5    <Actions><AnyAction/></Actions>
 6  </Target>
 7 <Policy PolicyId="Coordinator:Role" RuleCombiningAlgId="&rule-
       combine;deny-overrides">
 8  <Target>
 9    <Subjects><AnySubject/></Subjects>
10   <Resources>
11       <ResourceMatch MatchId="&function;string-match">
12     <AttributeValue DataType="&xml;string"> Coordinator </
           AttributeValue>
13     <ResourceAttributeDesignator AttributeId="&resource;resource-id
           " DataType="&xml;string"/>
14    </ResourceMatch>
15   </Resources>
16   <Actions> activate </Actions>
17  </Target>
18 <Rule RuleId="role:manager:active" Effect="Deny">
19  <Target>
20    <Subjects><AnySubject/></Subjects>
21    <Resources><AnyResource/></Resources>
22    <Actions><AnyAction/></Actions>
23   </Target>
24   <Condition FunctionId="urn:oasis:names:tc:xacml:3.0:function:any-
         of">
25     <AttributeValue DataType=&xml;string">manager</AttributeValue>
26    <Apply FunctionId="http://localhost/BPXACML/function#function;
          Session">
27    <SubjectAttributeDesignator AttributeId="&subject;subject-id"
          DataType="&xml;string"/>
28    </Apply>
29   </Condition>
30  </Rule>
31 <Rule RuleId="role:contractor:active" Effect="Deny">
32  <Target>
```

```
   <Subjects><AnySubject/></Subjects>
   <Resources><AnyResource/></Resources>
   <Actions><AnyAction/></Actions>
  </Target>
  <Condition FunctionId="urn:oasis:names:tc:xacml:3.0:function:any-of">
    <AttributeValue DataType=&xml;string">contractor</AttributeValue>
   <Apply FunctionId="http://localhost/BPXACML/function#function;Session">
   <SubjectAttributeDesignator AttributeId="&subject;subject-id" DataType="&xml;string"/>
   </Apply>
  </Condition>
 </Rule>
</Policy>
<Policy PolicyId="manager:Role" RuleCombiningAlgId="&rule-combine;deny-overrides">
 <Target>
   <Subjects><AnySubject/></Subjects>
  <Resources>
      <ResourceMatch MatchId="&function;string-match">
    <AttributeValue DataType="&xml;string"> manager </AttributeValue>
    <ResourceAttributeDesignator AttributeId="&resource;resource-id" DataType="&xml;string"/>
   </ResourceMatch>
  </Resources>
  <Actions> activate </Actions>
 </Target>
<Rule RuleId="role: Coordinator:active" Effect="Deny">
 <Target>
   <Subjects><AnySubject/></Subjects>
   <Resources><AnyResource/></Resources>
   <Actions><AnyAction/></Actions>
  </Target>
  <Condition FunctionId="urn:oasis:names:tc:xacml:3.0:function:any-of">
    <AttributeValue DataType=&xml;string"> Coordinator </AttributeValue>
   <Apply FunctionId="http://localhost/BPXACML/function#function;Session">
   <SubjectAttributeDesignator AttributeId="&subject;subject-id" DataType="&xml;string"/>
   </Apply>
  </Condition>
 </Rule>
</Policy>
<PolicySetIdReference> Role:Assignment </PolicySetIdReference>
</PolicySet>
```

Listing 5: SoD policy set example

The function 'Session' (line 64) is a new function that verifies that the given role is not available in the activated roles of the given user. This function takes one argument of data-type '..#string', which is the user's ID (line 65). It returns a list of all roles currently activated for this user. Then the predefined function *any-of* (line 62) will compare the given string (line 63) with the list retrieved by the session function. If the role was found the function will return the result *true*, and if it was not found, it will return *false*. If the condition was true the rule (line 56) will return *deny* and the request will be denied. If the condition returns *false*, the rule will not do anything and continue to the `RoleAssignment<PolicySet>` (line 70). The combining algorithm (line 1) of the `SoD<PolicySet>` is *'First applicable'* this algorithm will cause the REA to halt and not continue through the `<PolicySet>` file as soon as a match is found. The reason for using this combining algorithm is that we have designed the `<PolicySet>` file to have one policy for each role, which will contain a rule for each role that is in dynamic SoD relation with this role. So, as soon as the REA hits the correct policy for the requested role there is no reason to look into the rest of the `<PolicySet>`. It needs only to check all rules within the matched policy. For example, Listing 5 shows a policy for the role 'coordinator' (line 7). It shows that to be able to activate this role, the user should not have the role 'manager' active, and also should not have the role 'contractor' active. To simplify readability we included only one reverse policy, the one for the role 'manager' (lines 45-69), and did not include a policy for the role 'contractor'.

Listing 6 shows an example `RoleAssignment<PolicySet>`. The `<PolicySet>` contains a policy for each user, which contains rules for each role the user can activate. The policy set in Listing 6 includes an example policy for the user Adam (lines 7-30), which includes an example rule for activating the role 'coordinator' (lines 18-29).

```
 1  <PolicySet ... PolicySetId="Role:Assignment" PolicyCombiningAlgId
        ="&policy-combine;deny-overrides">
 2   <Target>
 3     <Subjects><AnySubject/></Subjects>
 4     <Resources><AnyResource/></Resources>
 5     <Actions><AnyAction/></Actions>
 6   </Target>
 7  <Policy PolicyId="Roles:For:user:Adam" RuleCombiningAlgId="&rule-
        combine;deny-overrides">
 8   <Target>
 9    <Subjects>
10     <SubjectMatch MatchId="&function;string-match">
11      <AttributeValue DataType="&xml;string"> Adam </AttributeValue>
12      <SubjectAttributeDesignator AttributeId="&subject;subject-id"
            DataType="&xml;string"/>
13     </SubjectMatch>
14    </Subjects>
15    <Resources><AnyResource/></Resources>
```

```
  <Actions><AnyAction/></Actions>
 </Target>
<Rule RuleId="Permission:to:activate:coordinator:role" Effect="Permit">
 <Target>
  <Subjects><AnySubject/></Subjects>
  <Resources>
   <ResourceMatch MatchId="&function;string-equal">
    <AttributeValue DataType="&xml;string"> Coordinator </AttributeValue>
    <ResourcesAttributeDesignator AttributeId="&resource;resource-id" DataType="&xml;string"/>
   </ResourcetMatch>
  </Resources>
  <Actions> Activate </Actions>
  </Target>
 </Rule>
</Policy>
</PolicySet>
```

Listing 6: Role-assignment policy set example

6.3.4 Instance-level Restrictions (IR)

Instance-level restrictions (IR) are used to fulfill the need to apply history-based restrictions within the same instance. For example, the scenario in Section 3.1 states that the user who closes the 'work order' should be the same user who issued it. So, for the same *work order* (same instance), the user to perform *close work order* must be the same user who performed *issue work order*. IR restrictions are written as policies in the IR policy set.

Listing 7 is an example `IR<PolicySet>` that includes instance-level restriction using the BP-XACML language. The IR policy set has a policy for the task 'close work order' (lines 7-34). The policy has a rule (lines 18-33) stating that the user must be the same user who issued the work order for the same instance.

```
 1 <PolicySet ... PolicySetId="IRPS" PolicyCombiningAlgId="&policy-
       combine;Deny-overrides">
 2  <Target>
 3    <Subjects><AnySubject/></Subjects>
 4    <Resources><AnyResource/></Resources>
 5    <Actions><AnyAction/></Actions>
 6  </Target>
 7 <Policy PolicyId="close:work:order:task"
 8  <Target>
 9    <Subjects><AnySubject/></Subjects>
10      <Resources>
11       <ResourceMatch MatchId="&function;string-match">
```

```
12         <AttributeValue DataType="&xml;string"> close work order</
               AttributeValue>
13         <ResourceAttributeDesignator AttributeId="&resource;resource
               -id" DataType="&xml;string"/>
14        </ResourceMatch>
15       </Resources>
16       <Actions><AnyAction/></Actions>
17   </Target>
18   <Rule RuleId="Must:be:who:issued:work:order" Effect="Deny">
19   <Target>
20     <Subjects><AnySubject/></Subjects>
21     <Resources><AnyResource/></Resources>
22     <Actions><AnyAction/></Actions>
23    </Target>
24      <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:
            not">
25       <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:any-
             of">
26         <SubjectAttributeDesignator AttributeId="&subject;subject-id
               " DataType="&xml;string"/>
27       <Apply FunctionId="http://localhost/BPXACML/function#function;
             PL">
28        <AttributeValue DataType=&xml;string"> issue work order </
              AttributeValue>
29        <ResourceAttributeDesignator AttributeId="urn:someapp:
              attributes:instance" DataType="&xml;string"/>
30       </Apply>
31       </Apply>
32     </Condition>
33  </Rule>
34  </Policy>
35   <!-- Point to the RoleTask policy set -->
36   <PolicySetIdReference> RTPS </PolicySetIdReference>
37  </PolicySet>
```

Listing 7: Example IR policy set

The function 'PL' (line 27) is a new function that retrieves the performers list of a specific task for a specific instance. This function takes two arguments of data-type '..#string', a task name and an instance number (lines 28-29). It returns a list of all users who performed the task for this instance. Then the predefined function *any-of* (line 25) will compare the given string, which is the username of the user who requests to perform the task, with the list retrieved by the PL function. The function *any-of* will return *true* if the user was in the performers list, and it will return 'false' if it was not found in the performers list. If it was a SoD-IR then this condition will be satisfied and the user will be denied if found to be part of the list. Because it is a binding of duties constraint in this case, we want the rule to deny only if the user was not found in the list (*i.e.* the function *any-of* returned *false*), and permit it if found in the list (*i.e.* the function ‘any-of’ returned true). For this reason the function *not* (line 24) has been added to reverse the output of the function.

6.4 Example Access Control policies

This section shows example requests in BP-XACML and explains how to assess these requests. It details the policy sets and rules needed as examples. This section will use the example from Chapter 3 to show how BP-XACML can be used to manage authorisation.

Let us consider the example scenario from Chapter 3. After receiving a *‘pump malfunction notification’*, the user *Adam* wants to activate the role *‘coordinator’* to deal with the pump malfunction. Listing 8 shows an example of a BP-XACML request where *Adam* wants to activate the role *‘coordinator’*.

```
<Request>
 <Subject>
  <Attribute AttributeId="urn:someapp:subject:subject-id" DataType
      ="&xml;string">
   <AttributeValue> Adam </AttributeValue>
  </Attribute>
 </Subject>
 <Resource>
  <Attribute AttributeId="urn:someapp:resource:resource-id"
      DataType="&xml;string">
   <AttributeValue> coordinator </AttributeValue>
  </Attribute>
 </Resource>
 <Action> activate </Action>
</Request>
```

Listing 8: Example role-activation request in BP-XACML

To be able to assess the request, the context handler will forward the request to the Role Enablement Authority (REA), since it is a request to activate a role. REA can only access the SoD policy set, so, it will search for an SoD policy with a matching object *coordinator*. Listing 5 in Section 6.3.3 shows an example `SoD<policySet>`. It can be seen that there is a policy for the role *‘coordinator’* (lines 7-44) with a target match on the resource to *coordinator* (lines 8-17). The policy states that in order to activate the role coordinator the user should not have the role *manager* active (lines 18-30). In this example we assume that *Adam* does not have the role *manager* active. Therefore the rule will not apply. The `SoD<policySet>` will then point to the `Role-assignment<policySet>`. In this example it is named *(Role:Assignment)* (line 70). Listing 6 from Section 6.3.3 shows an example `RoleAssignment<policySet>`. It can be seen that there is a policy for the user *Adam* (lines 7-30). Within this policy there is a rule stating that the user *Adam* can activate the role *coordinator* (lines 18-29). Based on the policy sets, REA will send the decision on this request, which is in this case *permit*. Listing 9 is an example response on the role activation request.

```
<Response>
 <Request>
  <Subject>
   <Attribute AttributeId="urn:someapp:subject:subject-id" DataType
       ="&xml;string">
    <AttributeValue> Adam </AttributeValue>
   </Attribute>
  </Subject>
  <Resource>
   <Attribute AttributeId="urn:someapp:resource:resource-id"
       DataType="&xml;string">
    <AttributeValue> coordinator </AttributeValue>
   </Attribute>
  </Resource>
  <Action> activate </Action>
 </Request>
 <Result>
  <Decision>Permit</Decision>
  <Status>
  <StatusCode Value="urn:someapp:status:ok"/> </Status>
 </Result>
</Response>
```

Listing 9: Example response for role-activation request in BP-XACML

After having the pump fixed, Adam wants to close the work order he issued. Listing 10 is an example request in BP-XACML. It is a request by the user *Adam* to perform the task *'close work order'*.

```
<Request>
 <Subject>
  <Attribute AttributeId="urn:someapp:subject:subject-id" DataType
      ="&xml;string">
   <AttributeValue> Adam </AttributeValue>
  </Attribute>
 </Subject>
 <Resource>
  <Attribute AttributeId="urn:someapp:resource:resource-id"
      DataType="&xml;string">
   <AttributeValue> close work order </AttributeValue>
  </Attribute>
 </Resource>
 <Action> Perform </Action>
</Request>
```

Listing 10: Example task performance request in BP-XACML

To be able to assess the request, the context handler will forward the request to the Task Authority, since it is a request to perform a task. As TA can only access the IR policy set, it will search for an IR policy with a matching object (close work order). Listing 7 in Section 6.3.4 shows an example IR policy set that applies to this situation. As can be seen (lines 7-34) there is a policy stating that in order to allow a person to perform *'close work order'*, it must be the same user who performed 'issue work order' for this instance. The PL function (line 27), takes in the task name of the other task *'issue work order'* (line 28), and the instance of the requested task (line 29) and sends back a list of users who performed this task for this instance. After applying the IR-BoD function, the result was that *Adam* is in the performers list of the task *'issue work order'* for this instance. So, the condition will return *true* and the rule result will be permit. The IR policy set then will point to the RoleTask policy set. Listing 4 from Section 6.3.2 shows an example `RoleTask<PolicySet>` that is applicable for this situation. It is a `RoleTask<PolicySet>` for the role 'coordinator', which states that if the role of the subject is 'coordinator' (lines 9-16) then go to the task policy set TPS: coordinator:role (line 38).

Listing 3 from Section 6.3.2 shows an example TPS:coordinator:role policy set. As can be seen, the task *'close work order'* is included in the task policy set of this role (lines 13-29), which means the role coordinator is allowed to perform this task. So, TA will send back to the context handler that Adam is allowed to perform the task *'close work order'*. Listing 11 is an example response to the request from Listing 10, based on the policy sets in Listings 3, 4 and 7.

```
<Response>
 <Request>
  <Subject>
   <Attribute AttributeId="urn:someapp:subject:subject-id" DataType
       ="&xml;string">
    <AttributeValue> Adam </AttributeValue>
   </Attribute> </Subject>
  <Resource>
   <Attribute AttributeId="urn:someapp:resource:resource-id"
       DataType="&xml;string">
    <AttributeValue>close work order</AttributeValue> </Attribute>
  </Resource>
  <Action> Perform </Action>
 </Request>
 <Result>
   <Decision>Permit</Decision>
   <Status>
   <StatusCode Value="urn:someapp:status:ok"/>
   </Status>
 </Result>
</Response>
```

Listing 11: Example response on task performance request in BP-XACML

The CH will then retrieve the permissions associated with this task using the TPL. As explained each permission is a pair of action and a resource. In this case the TPL will show the following as permissions associated with this task: (access, assets management system), (read, work order), (read, invoice), and (edit, work order). Then the CH will create a single standard RBAC request for each one of these permissions using the roles activated by the user and forward these requests to the PDP. An example request will be (coordinator, read, work order). Listing 12 shows how such a request is presented using BP-XACML. Such a request will be repeated using other roles activated by the user.

```
<Request>
 <Subject>
  <Attribute AttributeId="urn:someapp:subject:subject-id" DataType
      ="&xml;string">
   <AttributeValue> coordinator </AttributeValue>
  </Attribute>
 </Subject>
 <Resource>
  <Attribute AttributeId="urn:someapp:resource:resource-id"
      DataType="&xml;string">
   <AttributeValue> work order </AttributeValue>
  </Attribute>
 </Resource>
 <Action> read </Action>
</Request>
```

Listing 12: Example standard RBAC request in BP-XACML

To be able to assess this type of request it will be forwarded to the PDP. This type of request is handled by the PDP in the same way it is handled in the RBAC-XACML profile. PDP will be able to access the `Role<PolicySet>` for the roles activated by Adam, which is in this case *coordinator*. Listing 2 from Section 6.3.1 is an example `Role<PolicySet>` for the role *coordinator*. The policy set target is set to match *coordinator* with any of the roles active for the subject (lines 3-10). The policy set then points to the corresponding `Permission<PolicySet>` (line 15). Listing 1 from Section 6.3.1 shows an example `Role<PolicySet>` for the role *coordinator*. The policy set has a policy for all the permissions for this role. This policy includes a rule (lines 33-49) stating that this role can perform the action *read* on the resource *work order*. So, the PDP decision will be permit this request. Listing 13 shows an example response to such request.

```
<Response>
 <Request>
  <Subject>
   <Attribute AttributeId="urn:someapp:subject:subject-id" DataType
       ="&xml;string">
    <AttributeValue> Adam </AttributeValue>
   </Attribute>
  </Subject>
  <Resource>
   <Attribute AttributeId="urn:someapp:resource:resource-id"
       DataType="&xml;string">
    <AttributeValue> work order </AttributeValue>
   </Attribute>
  </Resource>
  <Action> read </Action>
 </Request>
 <Result>
   <Decision>Permit</Decision>
   <Status>
   <StatusCode Value="urn:someapp:status:ok"/> </Status>
  </Result>
</Response>
```

Listing 13: Example response for standard RBAC request in BP-XACML

The CH will collect responses from the PDP for each request. If all request decisions were permitted then the CH will send back the result as *permit*. If one of the requests was denied then the CH will send back *deny*, and the request of the user to perform the task will be denied.

To demonstrate an example were the request is denied, let us assume that after having the pump fixed, another user (Smith) wants to close the work order that Adam has issued. Listing 14 is an example request in BP-XACML. It is a request by the user *Smith* to perform the task *'close work order'*.

```
<Request>
 <Subject>
  <Attribute AttributeId="urn:someapp:subject:subject-id" DataType
     ="&xml;string">
   <AttributeValue> Smith </AttributeValue>
  </Attribute>
 </Subject>
 <Resource>
  <Attribute AttributeId="urn:someapp:resource:resource-id"
     DataType="&xml;string">
   <AttributeValue> close work order </AttributeValue>
  </Attribute>
 </Resource>
 <Action> Perform </Action>
</Request>
```

Listing 14: Example task performance request in BP-XACML

To be able to assess the request, the context handler will forward the request to the Task Authority, since it is a request to perform a task. As TA can only access the IR policy set, it will search for an IR policy with a matching object (close work order). Listing 7 in Section 6.3.4 shows an example IR policy set that applies to this situation. As can be seen (lines 7-34) there is a policy stating that in order to allow a person to perform *'close work order'*, it must be the same user who performed 'issue work order' for this instance. The PL function (line 27), takes in the task name of the other task *'issue work order'* (line 28), and the instance of the requested task (line 29) and sends back a list of users who performed this task for this instance. After applying the IR-BoD function, the result was that *Smith* is not in the performers list of the task *'issue work order'* for this instance. So, the condition will return *false* and the rule result will be deny. Since the combining algorithm for this IR policy set is "Deny-overrides" then the final result of this request will be deny, because Smith is not allowed to close the work order since he is not the one who issued it.

So, TA will send back to the context handler that Smith is not allowed to perform the task *'close work order'*. Listing 15 is an example response to the request from Listing 14, based on the policy set in Listing 7.

```
<Response>
 <Request>
  <Subject>
   <Attribute AttributeId="urn:someapp:subject:subject-id" DataType
       ="&xml;string">
    <AttributeValue> Smith </AttributeValue>
   </Attribute> </Subject>
  <Resource>
   <Attribute AttributeId="urn:someapp:resource:resource-id"
       DataType="&xml;string">
    <AttributeValue>close work order</AttributeValue> </Attribute>
  </Resource>
  <Action> Perform </Action>
 </Request>
 <Result>
   <Decision> Deny </Decision>
   <Status>
   <StatusCode Value="urn:someapp:status:ok"/>
   </Status>
 </Result>
</Response>
```

Listing 15: Example response on task performance request in BP-XACML

After that the CH will send back *deny*, and the request of the user to perform the task will be denied.

6.5 Discussion

BP-XACML is designed based on the characteristics described in Chapter 3. BP-XACML is designed to extend RBAC-XACML to be able to support the notions of task, task instance, instance-level restrictions, and role-level SoD. An important feature, that is part of core RBAC, is the idea of user-selected role activation and sessions, which implies having multiple roles active at the same time for the same user. The current version of the RBAC-XACML profile does not require support for sessions. It makes session support an implementation issue. However, NIST-RBAC [30] defines sessions as a required feature of core RBAC. For this reason, BP-XACML was designed to support sessions and the idea of having multiple roles active at the same time for the same user. So if a user requests to perform a task, BP-XACML is designed to look into all the active roles of this user to make a decision. More specifically, to evaluate a request to perform a task, the TA must do a subject match for each `Role<PolicySet>` for every role in the system to determine if the user has this role active. If they do, it must check the associated `Task<PolicySet>` to see if it grants the permission to perform the task. If an organisation has hundreds of roles, the PDP still needs to look into all these roles before making a decision, and this may have a negative impact on performance.

Given the design approach, it is necessary to consider the performance of PDPs when they must deal with large numbers of policies or rules. The time a PDP needs to load and evaluate policies is implementation dependent [89]. PDP efficiency in general differs from one XACML implementation to another [53]. For example, Sun's XACML PDP [61], which is one of the most widely used engines [89], requires less than 40 milliseconds to evaluate 1000 policies, and less than 14 seconds to load the 1000 policies when the system is initialised [89]. Turkmen and Crispo in [89] studied the performance of three different implementations (Sun's XACML [61], XACML Enterprise [31], and XACMLight [35]). According to their study, the time needed to load the policies grows linearly with the number of policies [89]. They also state that the time for evaluation became proportional to the number of rules in the policy after 100 rules, while before reaching 100 rules the increase in the number of rules did not show any obvious effect on evaluation time [89]. Liu *et al.* in [53] proposed XEngine, a PDP that is designed to deal specifically with large numbers of policies and rules. The paper shows that in some cases XEngine was 3000 times faster than Sun's PDP [53]. The performance differs from one experiment to another, but in general the experiments demonstrate that "XEngine is highly scalable and efficient in comparison with the Sun PDP" [53]. PDP efficiency with large number of policies and rules is implementation dependent. Most of the popular implementations are sufficiently fast that they take milliseconds to evaluate thousands of policies. Moreover, some implementations (*i.e.* XEngine) are designed to deal with very large numbers of policies and rules. We believe this provides a basis for a reasonable degree of confidence that BP-XACML is scalable, and that having a large number of policies or rules will not have a noticeable effect on the performance, particularly if the PDPs were implemented using a high performance implementation such as XEngine.

BP-XACML is designed to be a generic authorisation policy language that can be used with any workflow authorisation model. By supporting backward compatibility, BP-XACML can also to be used with NIST-RBAC authorisation models. We have used the characteristics from Chapter 3 to design the policy language. These characteristics were achieved through an independent analysis that focused on what such a language should support, regardless of which model is being used. Therefore, the language produced based on that, BP-XACML, is designed to be generic and can be used to support the features commonly found in workflow authorisation models or any NIST-RBAC authorisation model. On the other hand, BP-XACML is bound by the defined characteristics. So, if a model was designed with specific requirements that are not generic, BP-XACML might not be able to support all the authorisation policies for such a specific model. For example, BP-XACML can be used along with the BP-TRBAC authorisation model introduced in Chapter 4. The only feature of BP-TRBAC that was not supported in BP-XACML is the ‘active access control’ feature. The concept of ‘activation condition’, which was used in BP-TRBAC to achieve active access control, requires positive verification that the preceding tasks in a workflow have been completed before authorising a task. Control over the sequenced execution of tasks in a workflow is a primary function of a workflow engine. Therefore, we argue that it is reasonable to rely on the workflow engine to assure that tasks are performed at the right time. Since the context handler is designed to have the ability to distinguish a workflow request, the CH can be implemented to do the following: when a task performing request is received the CH can send the workflow engine a query to confirm that it is the right time for the task to be performed. Since workflow engines are designed to be aware of the task sequence and also be aware of the status of each process instance, the workflow engine will be able to answer the question whether it is time for this task to be performed or not. Positive verification by the CH ensures that the request to perform the task did not originate from another system where the action would be out of sequence, or from a user trying to perform an unauthorised action.

At the time of designing BP-XACML we kept in mind the idea of backward compatibility and supporting legacy systems. We designed BP-XACML so that the final decision is based on the current organisation policies for permissions. Therefore, even if a TA said that the user is allowed to perform a task, BP-XACML would still look into all permissions associated with such task and check if the user is allowed to perform these permissions or not. To achieve this property it was important to identify a link between the task and the associated permissions. Because PDPs are designed to deal with one type of subject and one type of object, and because the link between role-task-permission is a three-way link, it is not applicable to have one request to one PDP that gives the permissions associated with a task and then decide whether the role is allowed to perform these permissions. For this reason BP-XACML was designed to let the CH deal with retrieving the permissions associated with each task. We introduced the Task-Permissions List (TPL), a function that takes in a task ID and sends back a list of permissions associated with the task. In a way it is similar to the idea of 'session'. A session takes in a user ID and sends back a list of active roles, while TPL takes in a task and sends back a list of permissions. The only difference is that TPL does not need to be updated as frequently as the session.

The first draft of the RBAC-XACML profile proposed the idea of the `SoD<PolicySet>`, which was removed in the final version. Although in BP-XACML we have adopted the idea of the `SoD<PolicySet>`, there is a significant difference in the design architecture and the way this `<PolicySet>` was used. The RBAC-XACML draft used the `SoD<PolicySet>` to avoid giving a user with conflicting roles access to the resources. So, the model would allow the user to have conflicting roles but then will not allow him to access the resources. They have designed the model in a way that PDP will access `SoD<PolicySet>` and if a user has conflicting roles active the decision will be deny. In any case, support for SoD was optional. In BP-XACML we designed the model in a way that the REA controls role activation and enforces dynamic SoD constraints. It will not allow the user to activate conflicting roles in the first place. So, instead of checking at the time of requesting access, BP-XACML checks at the time of activating the role. The `SoD<PolicySet>` in BP-XACML is designed to have a single policy for each role. This policy will contain rules for all the conflicting roles. Setting the combining algorithm for the `SoD<PolicySet>` to be 'First applicable' will mean that the REA will need to look into only one policy and will not need to check the rest of the `<PolicySet>` because 'first applicable' means that the PDP should halt as soon as a match was found. As soon as an SoD violation is found the REA responds with deny. On the other hand this means for each pair of conflicting roles two policies are required, one for each role, to ensure that the first role cannot be activated if the second is already active and that the second role cannot be activated if the first role is active.

6.6 Conclusion

This chapter introduced BP-XACML, a new profile that extends RBAC-XACML and enables the specification of business process authorisation policies. In addition to supporting the XACML RBAC profile, the extended language also supports the representation of tasks and tasks instances. It proposes new a policy set called `Task<PolicySet>` for the incorporation of business process tasks. BP-XACML also supports separation of duties and binding of duties constraints at the level of process instances. The new `IR<PolicySet>` supports the representation of instance-level restrictions in a way that can be linked to the related tasks and can be evaluated. The new function ‘performers list’ helps in evaluating the instance-level restrictions. The new repository TPL links tasks to permissions. Finally, it supports the ‘separation of duties’ on the role level, making use of the ‘REA’ and the ‘sessions’ to find and evaluate the SoD restrictions from the `SoD<PolicySet>`.

Although BP-XACML is designed to work with the model introduced in Chapter 4, BP-TRBAC, it is also designed to be generic. BP-XACML is designed using the characteristics from Chapter 3 to be able to support authorisation policies for any workflow authorisation model.

Chapter 7

Conclusion and Future Directions

Authorisation management for business process environments requires specific capabilities. The reliance on expert employees to make sure that authorisation policies are complied with is not always practical. This approach will keep the organisation relying on particular experts who are always needed to do the integration. It will also be potentially costly, inaccurate and inconsistent, as a human might forget a policy or change their behavior. Moreover, the organization will be exposed to risks if the authorisation policy is not enforced. To automate the enforcement of the authorisation policies in business process environments first, there is a need for an authorisation model to enforce the authorisation policies. Secondly, there is a need for an authorisation policy language to express these policies in a machine-enforceable way. As this thesis showed, in the current literature there is no authorisation model that supports all the required characteristics, and there is no authorisation policy language that satisfies the required characteristics.

The main argument of this thesis is that integrating BPM and authorisation management can improve the security of these environments by making sure that the authorisation policies are enforced and that all task performances are in compliance with the authorisation policies. To be able to do so, there is a need for a model for enforcing the policies, and a machine-enforceable language to write the policies.

We have interviewed experts and performed an analysis using the literature to identify the required characteristics. Using these characteristics we proposed BP-TRBAC, an authorisation model that supports the required characteristics. A use case implementation of a subset of the model was done with YAWL as the workflow system. This implementation showed that there is a need for an authorisation policy language to work in concert with the BP-TRBAC. BP-XACML was then introduced as the authorisation policy language for business process environments.

It was shown in Chapter 3 that the current models and languages can not support the necessary characteristics to be used as an organisation-wide authorisation management system and support business process authorisation requests. This thesis introduces an authorisation model and an authorisation policy language that can be used together to allow the organisation to manage and control authorisation requests from workflow systems and other legacy systems. To do so, a range of challenges in authorisation management in business process environments are addressed, and a number of contributions are made, as summarised in the following section. A future research direction is presented in Section 7.2. Concluding remarks for both the chapter and the thesis provided in Section 7.3.

7.1 Summary of Outcomes and Objectives

This section lists the main outcomes of this thesis and maps them to the research objectives identified in Chapter 1. It will list the three objectives from Chapter 1, and for each objective it will state the related outcomes that were achieved.

Objective 1: Conduct a preliminarily analysis identifying the need the characteristics of the authorisation model and the authorisation policy language. The following outcomes are related to this objective:

- A case scenario from a real-life security-sensitive environment showed the need for authorisation management in such complex environments. After collecting data and interviewing with stakeholders we were able to provide the scenario of performing a specific job in this complex organisation through a business process.

- The characteristics of an authorisation model for business process environments, and the characteristics that a business process authorisation language should satisfy were presented.

- A literature review of the BPM authorisation models, and the BPM authorisation policy languages, demonstrated the need for a new model, and a new policy language.

Objective 2: Provide an authorisation model for business process environments. The model should be used to control the authorisation requests by enforcing the authorisation policies. The following outcomes are related to this objective:

- Introduced BP-TRBAC, a business process authorisation model. It extends RBAC to support business process requirements. BP-TRBAC is a unified organisation-wide authorisation model that can support workflow authorisation requests and non-workflow authorisation requests. BP-TRBAC is designed to support all the needed characteristics identified in the first objective.

- A use-case implementation is provided. The use-case represents a subset of BP-TRBAC. The subset is called SPCC, and is intended to check design time assignments. The implementation showed that this subset of BP-TRBAC can be used with a workflow system. It showed that SPCC was able to communicate with YAWL, obtain the required information, and respond with an informed decision on whether this role is allowed to perform this task or not. It was able to deal with tasks rather than permissions, and to use the policy file to make the decision.

Objective 3: Provide a structured machine-readable language that has the ability to represent authorisation policies in business processes. The following outcomes are related to this objective:

- Introduced BP-XACML an authorisation policy language for business processes. BP-XACML extends XACML to provide a new profile that supports specific business process requirements. BP-XACML provides backward compatibility with the RBAC-XACML profile. Therefore it can support standard RBAC policies. It can be used to write authorisation policies for business process systems and other legacy systems. BP-XACML is designed to support all the required characteristics identified in Chapter 3. The policy language can be used with the authorisation model introduced in this thesis, BP-TRBAC, but it was also designed to be generic and can be used with other workflow authorisation models.

- A complete policy model for BP-XACML is provided. The policy model showed how the language can be used. It showed how, using this language, a system can handle and evaluate authorisation requests. The policy model extends the XACML policy model with the needed authorities, attributes, and policy sets to support the BP-XACML language. The policy model is in compliance with the authorisation model BP-TRBAC, which was introduced in the second objective.

7.2 Future Research Direction

This section provides a discussion on potential opportunities for future research on the proposals presented in this thesis. It also summarises various limitations of the previous chapters.

7.2.1 BP-TRBAC

The main objective of the implementation of SPCC presented in Chapter 5 was to show the technical feasibility of implementing part of the BP-TRBAC and integrating it with a workflow management system, which demonstrates the feasibility of key concepts proposed in this thesis. A decision was made to implement a subset of BP-TRBAC, and leave out certain components of BP-TRBAC, due to resource constraints. Even though the current version of our implementation is not intended to demonstrate the full authorisation model that can be used in an operational environment, it serves as practical evidence to support our claim that there is a need for better authorisation models for business processes. We demonstrated the shortcoming in the workflow system YAWL, and showed how this BP-TRBAC subset can be used to address this shortcoming.

An important future direction for this research is to implement the proposed authorisation model (BP-TRBAC) as a whole in an operational environment. The environment should be one that uses business processes and has complex authorisation requirements, such as an airport, a chemical plant, or other security-sensitive environment. This would enable testing and evaluating our proposal, and the idea that integrating BPM and authorisation management would provide better security for business process environments. Another associated step would be to explore the integration of BP-TRBAC with various workflow systems, showing that it can work with different workflow systems.

Besides testing the technical capabilities of the proposal, it is also necessary to test the feasibility of implementing the whole model, and the usability of such a model with end users. Such testings of non-technical aspects is desirable, but this would require integration with live commercial systems, and that was not feasible within the scope of this thesis.

A possible next step is to perform a field trial to evaluate and support the claims by integrating a number of operational systems, including a workflow system, in a security-critical environment with a BP-TRBAC system. This would require building a system with appropriate user interfaces and testing it among staff that are normally tasked with similar tasks of authorisation administration. In addition to the required systems development and integration, it is also necessary to find appropriate skilled people to participate in such a study. For these reasons such a field trial was beyond the scope of the thesis.

7.2.2 SPCC

As explained in Chapter 5 SPCC is implemented using our own policy file format. In order for SPCC to be a useful plug-in that can be used with YAWL it should be implemented to deal with a structured, standardised policy language. BP-XACML as shown in Chapter 6 is a standardised, machine-readable language that is able to represent authorisation policies for different systems including the business process authorisation policies. An interesting future research direction is to develop an implementation of SPCC using BP-XACML as the policy language. This will allow for using SPCC in real life scenarios, which can be also used to test the usability of SPCC.

7.2.3 BP-XACML

In Chapter 6, we discussed how BP-XACML as a policy model deals with authorisation requests. The chapter provides details on how the model works including all the new authorities, policy sets, and attributes. Although the chapter provides details on how these new components are integrated with the XACML model, and provides the details of how each type of request is handled, due to resource constraints the BP-XACML policy model was not implemented. This is one of the interesting future research directions that could significantly influence the implementation of the authorisation model as a whole.

7.2.4 Building Information Models

A PhD thesis by Nimalaprakasan Skandhakumar [83] used building information models to control physical access in smart buildings. The thesis integrated authorisation management with building information models. It also used XACML as the policy language after extending it to represent buildings models aspects. An exciting future research direction is the integration of the work of this thesis with the work from our thesis. The integration may produce an authorisation model that can control organisation-wide authorisation requests, and control physical access using the building information models for such requests.

7.3 Concluding Remarks

Business process management is an important domain that requires the attention of information security researchers, specifically in the authorisation management field. Business process environments require specific characteristics to be supported by the authorisation management system. This thesis addresses important challenges in research associated with authorisation management for business process environments by investigating and developing an authorisation model and an authorisation policy language that are designed specifically to meet the identified characteristics.

Increasing adoption of business processes in critical infrastructures, such as airports, makes it important to find effective measures to integrate authorisation management with business processes. The work presented in this thesis contributes to this effort. The work provided in this thesis would be enhanced by extending the evaluation of this work with practical testing and deployment of these ideas in real operational environments.

Bibliography

[1] S. Abiteboul and S. Grumbach. A rule-based language with functions and sets. In *ACM Transaction on Database Systems*, volume 16, pages 1–30, 1991. 42

[2] M. Adams and A. ter Hofstede. *YAWL - User Manual*. The YAWL Foundation, version 3.0 edition, 2014. 65, 66, 67, 74

[3] C. Alm, R. Wolf, and J. Posegga. The opl access control policy language. In S. Fischer-Hübner, C. Lambrinoudakis, and G. Pernul, editors, *TrustBus*, volume 5695 of *Lecture Notes in Computer Science*, pages 138–148. Springer, 2009. 42

[4] S. Alter. *Information Systems : A Management Perspective*. Addison Wesley, 4th edition, 2002. 13

[5] A. Anderson. Xacml profile for role based access control (rbac). *OASIS Access Control TC committee draft*, 1:13, 2004. 21

[6] A. Anderson. Xacml profile for role based access control (rbac), committee draft 01. Standard, OASIS, February 2004. 88

[7] A. Anderson. Core and hierarchical role based access control (rbac) profile of xacml version 2.0, oasis standard. Standard, OASIS Open, February 2005. 21, 81, 88, 90, 101

[8] C. Anderson. What's the difference between policies and procedures? Electronic: www.Bizmanualz.com, April 2005. 17

[9] V. Atluri and W. kuang Huang. An authorization model for workflows. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *European Symposium on Research in Computer Security*, volume 1146 of *Lecture Notes in Computer Science*, pages 44–64. Springer, 1996. 32, 33, 37

[10] M. Y. Becker, C. Fournet, and A. D. Gordon. Design and semantics of a decentralized authorization language. In *20TH IEEE COMPUTER SECURITY FOUNDATIONS SYMPOSIUM*, pages 3–15, 2007. 42

[11] E. Bertino, E. Ferrari, and V. Atluri. A flexible model supporting the specification and enforcement of role-based authorization in workflow management systems. In *ACM Workshop on Role-Based Access Control*, pages 1–12, 1997. 38, 60

[12] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Transactions on Information and System Security (TISSEC)*, 2(1):65–104, 1999. 31

[13] B. Blobel. Authorisation and access control for electronic health record systems. In *International journal of medical informatics*, volume 73, pages 251–257, 2004. 36

[14] R. A. Botha and J. H. P. Eloff. Separation of duties for access control enforcement in workflow environments. *IBM Systems Journal*, 40(3):666–682, 2001. 32, 38, 60

[15] A. D. Brucker, I. Hang, G. Lückemeyer, and R. Ruparel. Securebpmn: Modeling and enforcing access control requirements in business processes. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, SACMAT '12, pages 123–126, New York, NY, USA, 2012. ACM. 43

[16] J. Bryans. Reasoning about xacml policies using csp. In *the 2005 workshop on Secure web services*, page 35. ACM, 2005. 19, 20

[17] I. Celino, A. K. A. de Medeiros, G. Zeissler, M. Oppitz, F. M. Facca, and S. Zoeller. Semantic business process analysis. In *SBPM*. Citeseer, 2007. 12

[18] D. W. Chadwick, W. Xu, S. Otenko, R. Laborde, and B. Nasser. Multi-session separation of duties (msod) for rbac. In *ICDE Workshops*, pages 744–753, 2007. 38, 60, 62

[19] F. J. G. Clemente, G. M. Pérez, J. A. B. Blaya, and A. F. G. Skarmeta. Representing security policies in web information systems. In *Proceedings of WWW 2005*, May 2005. 42

[20] J. Crampton and H. Khambhammettu. Xacml and role-based access control. In *Presentation at DIMACS Workshop on Security of Web Services and e-Commerce*, page 174. Springer, 2005. 20, 42

[21] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Workshop on Policies for Distributed Systems and Networks (Policy2001)*, pages 29–31, Bristol, UK, Jan 2001. Springer-Verlag. 42

[22] N. C. Damianou, A. K. Bandara, M. S. Sloman, and E. C. Lupu. A survey of policy specification approaches. Technical report, Department of Computing, Imperial College, 2002. 2, 18, 19

[23] F. D'Aubeterre, R. Singh, and L. Iyer. Secure activity resource coordination: empirical evidence of enhanced security awareness in designing secure business processes. *European Journal of Information Systems*, 17(5):528 – 542, 2008. 1, 4, 24

[24] V. Dhankhar, S. Kaushik, and D. Wijesekera. Securing workflows with xacml, rdf and bpel. In V. Atluri, editor, *Data and Applications*

Security XXII, volume 5094 of *Lecture Notes in Computer Science*, pages 330–345. Springer Berlin Heidelberg, 2008. 43

[25] G. Dhillon. *Principles of Information Systems Security: text and cases*. John Wiley and Sons, New York, 2007. 17, 21

[26] M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede, editors. *Process-aware Information systems: Bridging people and software through process technology*. John Wiley and Sons, 2005. 3, 13, 45

[27] E. D. Falkenberg. Concepts for modelling information. In G. M. Nijssen and Freudenstadt, editors, *IFIP Working Confrence on Modelling in Data Base Management Systems*, pages 95–109, Germany, 1976. North-Holland Publishing. 48

[28] C. Feltus. Preliminary literature review of policy engineering methods - toward responsibility concept. In *3rd international conference on information and communication technologies : from theory to applications (ICTTA 08)*, Damascus, Syria, 2008. 19

[29] D. Ferraiolo and D. Kuhn. Role-Based Access Control. In *15th National Computer Security Conference*, pages 554–563, October 1992. 35

[30] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactoin of Information System Security*, 4(3):224–274, Aug. 2001. 22, 34, 45, 46, 47, 51, 55, 117

[31] T. A. S. Foundation. Apache xerces2 java parser. Technical report, The Apache Software Foundation, http://xerces.apache.org/xerces2-j/, 2010. 117

[32] Gartner. Leading in times of transition: The 2010 CIO agenda. In *Gartner EXP CIO report*, 2010. 2

[33] I. Ghalimi. BPM, ECM, ESB, and security. in Enterprise Irregulars, Retrieved: 15 December 2011, From: http://www.enterpriseirregulars.com/ei/20794, 2007. 36

[34] G. Governatori and S. Sadiq. The journey to business process compliance. In *Hand- book of Research on BPM*, pages 426–454. IGI Global, 2009. 2, 19

[35] O. Gryb. Xacmlight. Technical report, XACMLight, http://xacmllight.sourceforge.net/, 2013. 117

[36] T. Halpin. Object-role modeling: an overview, Retrieved: 6 March 2011, From: http://www.orm.net/pdf/ormwhitepaper.pdf. unpublished, 2001. 48

[37] P. Harmon. *Business process change: a manager's guide to improving, redesigning, and automating processes*. Morgan Kaufman Publishers, San Francisco, 1st edition, 2003. 10

[38] G. Herrmann and G. Pernul. Viewing business process security from different perspectives. In *11th International Bled Electronic Commerce Conference*, Slovenia, 1998. 1, 24

[39] P. C. K. Hung and K. Karlapalem. A secure workflow model. In *AISC on ACSW frontiers 2003*, volume 21, pages 33–41. Australian Computer Society Inc., 2003. 32

[40] ISACA. Cisa review manual 2006. In *Information Systems Audit and Control Association*, 2006. 2

[41] ISO. Iso/iec 27002: Information technology – security techniques – code of practice for information security controls. Standard, International Organisation for Standardisation, 2013. 17, 18

[42] K. Jensen. Coloured petri nets: A high level language for system design and analysis. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 342–416, Berlin, 1990. Springer-Verlag. 14

[43] J. Jeston and J. Nelis. *Business process management.* Routledge, 2014. 1

[44] L. Kagal, T. W. Finin, and A. Joshi. A policy language for a pervasive computing environment. In *POLICY*, pages 63–. IEEE Computer Society, 2003. 42

[45] S. Kandala and R. S. Sandhu. Secure role-based workflow models. In M. S. Olivier and D. L. Spooner, editors, *DBSec*, volume 215 of *IFIP Conference Proceedings*, pages 45–58. Kluwer, 2001. 23, 33, 37

[46] R. K. L. Ko. A computer scientist's introductory guide to business process management (bpm). In *ACM Crossroads*, volume 15, New York, NY, USA, 2009. ACM Press. 10

[47] M. Kohlbacher. The effects of process orientation on customer satisfaction, product quality and time-based performance. In *the 29th International Conference of the Strategic Management Society*, pages 11–14, Washington DC, October 2009. 10

[48] A. Lanz, B. Weber, and M. Reichert. Workflow time patterns for process-aware information systems. In *Enterprise, Business-Process and Information Systems Modeling*, pages 94–107. Springer, 2010. 13

[49] T. P. Layton. *Information Security: Design, Implementation, Measurement, and Compliance.* Auerbach publications, Boca Raton, FL, 2007. 2

[50] M. Leitner, S. Rinderle-Ma, and J. Mangler. Aw-rbac: access control in adaptive workflow systems. In *Sixth International Conference on Availability, Reliability and Security (ARES)*, pages 27–34. IEEE, 2011. 35, 40

[51] M. Leitner, S. Rinderle-Ma, and J. Mangler. Responsibility-driven design and development of process-aware security policies. In *Sixth*

International Conference on Availability, Reliability and Security, 2011. 12, 14

[52] A. X. Liu, F. Chen, J. Hwang, and T. Xie. Xengine: A fast and scalable xacml policy evaluation engine. In *Proceedings of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '08, pages 265–276, New York, NY, USA, 2008. ACM. 21, 81

[53] A. X. Liu, F. Chen, J. Hwang, and T. Xie. Xengine: a fast and scalable xacml policy evaluation engine. In *ACM SIGMETRICS Performance Evaluation Review*, volume 36, pages 265–276. ACM, 2008. 117

[54] J. Lobo, R. Bhatia, and S. A. Naqvi. A policy description language. In *AAAI/IAAI*, pages 291–298, 1999. 42

[55] G. B. M. Ajmone Marsan and G. Conte. Modelling with generalized stochastic petri nets. In *Wiley series in parallel computing*, New York, 1995. Wiley. 14

[56] J. R. M. Owen. Bpmn and business process management introduction to the new business process modeling standard. Internet, Popkin Software, 2003. 11

[57] W. Mallouli, F. Bessayah, A. R. Cavalli, and A. Benameur. Security rules specification and analysis based on passive testing. In *Proc. of the Global Communications Conference on Exhibition and Industry Forum Co-located with WTC (GLOBECOM'08)*, pages 2078–2083, New Orleans, LA, USA, November-December 2008. Institute of Electrical and Electronics Engineers (IEEE). 19

[58] D. McCoy. BPM and Security: Not feeling so good, Retrieved: 12 December 2010, From: http://blogs.gartner.com/dave_mccoy/2008/10/16/bpm-and-security-not-feeling-so-good, 2008. 2, 24

[59] Merriam-Webster. Merriam-webster online, Retrieved: 17 March 2011, From: http://www.m-w.com. 17

[60] A. t. H. Michael Adams and S. Clemens. *YAWL - Technical Manual*. The YAWL Foundation, 2.1 edition, 2010. 66, 67, 68

[61] S. Microsystems. Sun's xacml implementation. Technical report, Sun Microsystems, http://sunxacml.sourceforge.net, 2005. 117

[62] T. Moses. Extensible access control markup language (xacml) version 2.0. oasis standard. Technical report, OASIS Open, 2005. 18, 19, 42, 80, 83, 84, 97

[63] J. Mülle, S. v. Stackelberg, and K. Böhm. A security language for bpmn process models. In *Karlsruhe Reports in Informatics*. Karlsruhe, 2011. 43

[64] T. Murata. Petri nets: Properties, analysis and applk a tions. In *IEEE*, volume 77, pages 541–580, APRIL 1989. 14

[65] N. Nagaratnam, A. Nadalin, M. Hondo, M. McIntosh, and P. Austel. Business-driven application security: From modeling to managing secure applications. In *IBM Systems Journal*, volume 44, pages 847–867, Riverton, NJ, USA, October 2005. IBM Corp. 34

[66] T. Neubauer, M. D. Klemen, and S. Biffl. Secure business process management: A roadmap. In *The International Conference on Availability, Reliability and Security (ARES)*, pages 457–464, 2006. 1, 24

[67] T. Norman. *Integrated Security Systems Design: Concepts, Specifications, and Implementation*. Butterworth-Heinemann, Burlington, MA, USA, January 2007. 21

[68] OASIS. XACML language proposal, version 0.8. Technical report, OASIS, 2001. 20

[69] A. O'Connor and R. Loomis. Economic analysis of role-based access control. Technical report, National Institute of Standards and Technology, December 2010. 21, 45

[70] S. Oh and S. Park. Task-role based access control (t-rbac): An improved access control model for enterprise environment. In M. T. Ibrahim, J. Küng, and N. Revell, editors, *DEXA*, volume 1873 of *Lecture Notes in Computer Science*, pages 264–273. Springer, 2000. 23, 31, 32, 33, 35, 39, 45, 53, 60

[71] M. Pesic, H. Schonenberg, and W. van der Aalst. Declarative workflow. In *Modern Business Process Automation*, pages 175–201. Springer, 2010. 16

[72] C. A. Petri. *Communication with automata*. PhD thesis, University of Hamburg, 1966. 14

[73] T. Phan, J. Han, J.-G. Schneider, T. Ebringer, and T. Rogers. A survey of policy-based management approaches for service oriented systems. In *Australian Software Engineering Conference*, pages 392–401. IEEE Computer Society, 2008. 42

[74] C. Ribeiro, A. Zúquete, P. Ferreira, and P. Guedes. SPL: An access control language for security policies with complex constraints. In *In Proceedings of the Network and Distributed System Security Symposium*, pages 89–107, 1999. 1, 2

[75] A. Rodríguez, E. Fern'andez-Medina, and M. Piattini. A BPMN extension for the modeling of security requirements in business processes. *The Institute of Electronics, Information and communication Engineers (IEICE) TRANSACTIONS on Information and Systems*, 90:745–752, 2007. 24

[76] P. Samarati and S. C. Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, pages 137–196, Berlin Heidelberg, 2001. Springer. 36, 42, 83

[77] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role - based access control models. In *IEEE Computer*, volume 29, pages 38–47, 1996. 2, 37, 51

[78] R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST model for role-based access control: towards a unified standard. In *Proceedings of the fifth ACM workshop on Role-based access control*, RBAC '00, pages 47–63, New York, NY, USA, 2000. ACM. 22

[79] R. S. Sandhu. Separation of duties in computerized information systems. In *DBSec*, pages 179–190, 1990. 22

[80] A. Schaad, V. Lotz, and K. Sohr. A model-checking approach to analyzing organizational controls in a loan origination process. In *ACM Symposium on Access Control Models and Technologies;*, volume 6 of *Symposium on Access Control Models and Technologies (SACMAT)*, pages 139–149, New York, USA, 2006. ACM. 45

[81] J. Shi, Y. Li, H. R. Deng, W. He, and E. W. Lee. A secure platform for information sharing in epcglobal network. *International Journal of RFID Security and Cryptography*, 2(1-4):107–118, 2013. 36

[82] S. Sinha, S. Sinha, and B. Purkayastha. Synchronization of authorization flow with work object flow in a document production workflow using xacml and bpel. In V. Das and R. Vijaykumar, editors, *Information and Communication Technologies*, volume 101 of *Communications in Computer and Information Science*, pages 365–370. Springer Berlin Heidelberg, 2010. 43

[83] N. Skandhakumar. *Integrated access control for smart buildings using building information models.* PhD thesis, Queensland University of Technology, 2014. 125

[84] M. Strembeck and J. Mendling. Modeling process-related rbac models with extended uml activity models. *Information & Software Technology*, 53:456–483, 2011. 34, 40, 62

[85] A. M. ter Hofstede, W. van der Aalst, M. Adams, and N. Russell, editors. *Modern Business Process Automation: YAWL and its Support Environment.* Springer, 2010. 15, 16

[86] R. K. Thomas and R. S. Sandhu. Task-based authorization controls (tbac): A family of models for active and enterprise-oriented autorization management. In T. Y. Lin and S. Qian, editors, *DBSec*, volume 113 of *IFIP Conference Proceedings*, pages 166–181. Chapman & Hall, 1997. 2, 23, 31, 32, 33, 37, 45, 46

[87] S. Tjoa, S. Jakoubi, and G. Quirchmayr. Enhancing business impact analysis and risk assessment applying a risk-aware business process modeling and simulation methodology. In *International Conference on Availability, Reliability and Security (ARES)*, pages 179–186. IEEE Computer Society, 2008. 24

[88] S. Torjman. What is policy? White Paper 1-55382-142-4, The Caledon Institute of Social Policy, Ontario, Canada, September 2005. 17

[89] F. Turkmen and B. Crispo. Performance evaluation of xacml pdp implementations. In *Proceedings of the 2008 ACM workshop on Secure web services*, pages 37–44. ACM, 2008. 117

[90] W. M. Van der Aalst and A. H. Ter Hofstede. Yawl: yet another workflow language. *Information systems*, 30(4):245–275, 2005. 7, 11, 14, 66

[91] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003. 53

[92] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske. Business process management: A survey. In W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, editors, *Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2003. 10, 11, 12

[93] J. Vom Brocke and M. Rosemann. *Handbook on business process management*. Springer, 2010. 1, 3, 9

[94] J. vom Brocke and M. Rosemann, editors. *Handbook on Business Process Management: Strategic Alignment, Governance, People and Culture*, volume 1 of *International Handbooks on Information Systems*. Springer, Berlin, 2010. 10

[95] J. Wainer, A. Kumar, and P. Barthelmess. WRBAC a work-flow security model incorporating controlled overriding of constraints. In *International Journal of Cooperative Information Systems (IJCIS)*, volume 4, pages 455–486, 2003. 2, 32, 35, 38, 46, 62

[96] J. Wainer, A. Kumar, and P. Barthelmess. DW-RBAC: A formal security model of delegation and revocation in workflow systems. *Inf. Syst.*, 32(3):365–384, 2007. 35

[97] B. Weber, M. Reichert, W. Wild, and S. Rinderle. Balancing flexibility and security in adaptive process management systems. In *CoopIS*, volume 3761 of *LNCS*, pages 59–76, 2005. 40

[98] B. Weber, S. Rinderle-Ma, and M. Reichert. Change support in process-aware information systems-a pattern-based analysis. Technical report, University of Twente, Enschede, The Netherlands, 2007. 12, 14

[99] M. Weske. *Business Process Management: concept, Languages, Architectures*. Springer, Berlin, 2007. 9, 10, 11, 12, 19

[100] C. Wolter, M. Menzel, and C. Meinel. Modelling security goals in business processes. In T. Kühne, W. Reisig, and F. Steimann, editors, *Modellierung*, volume 127 of *LNI*, pages 197–212. GI, 2008. 24, 36

[101] C. Wolter and A. Schaad. Modeling of task-based authorization constraints in bpmn. In G. Alonso, P. Dadam, and M. Rosemann, editors, *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 64–79. Springer, 2007. 33

[102] C. Wolter, A. Schaad, and C. Meinel. Deriving xacml policies from business process models. In M. Weske, M.-S. Hacid, and C. Godart, editors, *Web Information Systems Engineering – WISE 2007 Workshops*, volume 4832 of *Lecture Notes in Computer Science*, pages 142–153. Springer Berlin Heidelberg, 2007. 43

[103] C. Wolter, C. Weiss, and C. Meinel. An xacml extension for business process-centric access control policies. In *Policies for Distributed Systems and Networks, 2009. POLICY 2009. IEEE International Symposium on*, pages 166–169, July 2009. 43

[104] W. Xu, J. Wei, Y. Liu, and J. Li. Sowac: A service-oriented workflow access control model. In *COMPSAC*, pages 128–134. IEEE Computer Society, 2004. 37

[105] L. Yang, Y. Choi, M. Choi, and X. Zhao. Fwam: A flexible workflow authorization model using extended rbac. In *CSCWD*, pages 625–629. IEEE, 2008. 37

[106] L. Zhang, G.-J. Ahn, and B.-T. Chu. A rule-based framework for role-based delegation and revocation. *ACM Trans. Inf. Syst. Secur.*, 6:404–441, August 2003. 42

[107] M. zur Muehlen and M. Indulska. Modeling languages for business processes and business rules: A representational analysis. *Information Systems*, 35(4):379–390, Elsevier, 2010. 19

[108] M. zur Muehlen and M. Rosemann. Multi-paradigm process management. In J. Grundspenkis and M. Kirikova, editors, *International Conference on Advanced Information Systems Engineering (CAiSE) Workshops (2)*, pages 169–175. Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia, 2004. 10

Appendix A

Case study

This appendix provides a brief outline of the case study and the interviews conducted with the experts and stakeholders from the security sensitive environment.

Introduction and Justification

Business Process Management (BPM) is a widely accepted and used in industrial organisations. For instance, modern airports adopt BPM in order to efficiently manage and allocate tasks to employees. Different types of information systems are becoming a Process-Aware Information Systems (PAIS). Organisations in all sectors are becoming business process oriented, where they rely and heavily depend on their process-aware systems. These systems are now the backbone to control, administer and enact all core business activities.

However, existing BPM methodologies barely consider information security requirements and policies. Business processes and information security requirements are dealt with separately and often do not follow a coherent strategy, e.g. information security considerations are typically overlooked in business process models. It is however crucial to ensure the security of corporate business processes for the success of an organisation, as failure to follow security policies is no longer an option.

Moreover, business processes in large organisations such as airports usually run across multiple systems, where each system has its own policy and its own access control restrictions. In managing complex organisations such as airports a lot of the integration of these policies happens in the mind of expert employees, which makes the organisation heavily dependent on such employees and subject to human error. Therefore, it is important to provide a method of integrating security restrictions and access control into the business processes.

Aim

The aim of this case study is to collect relevant information by interviewing experts and stakeholders to be able to build a real-life case scenario, which can be used to understand how business processes are executed in real-life security sensitive organisations. It will also show how business processes run across multiple systems and deal with different authorisation management policies.

Methodology

After recognising the problem, an organisation with a complex, and security-sensitive environment (which is a partner of the Airports of the Future Project) was approached to build a case study from their environment, demonstrating the existence of the problem. The organisation was chosen, because it exists in a complex environment, where business processes runs across multiple systems, besides being a partner to the project. Moreover, it is a perfect example where security is an important aspect as well as the use of business processes.

Specific processes were chosen to investigate, where they show the complexity of the environment, and run across multiple systems. Processes were diverse, where some require more physical access and others are more logical access. Some are processes that happen everyday, where others are maintenance.

Several interviews were done with personnel who are directly related to the processes in this case study. Processes were chosen and specific questions were developed before having the meetings. After building the processes and stating the access control policies from the gathered information, the same people were asked to validate the output and notify of any errors or inconsistencies.

Choosing the process

Based on the criteria identified earlier on choosing the business process (show the complexity of the environment, run across multiple systems, and includes sensitive access control requirements), and after long discussions with the representative of this organisation, three operations were chosen to be investigated and designed as business processes. They are: ‘fixing a pump malfunction’, ‘baggage handling’, and ‘getting physical access clearance’.

Targeted interviewees

As the three chosen processes are operational and deal with the assets and systems of the organisation, and because access control is our focus the following were the target to conduct the interviews with them:

- Building facilities assets manager
- Operational manager
- Security compliance manager

Interviews

Interviews were conducted separately with few days between each interview and the other. They were conducted in the offices of each interviewee to be close to the systems they deal with everyday.

A set of questions were developed to be asked during the interview. The following are the questions that were asked during each interview.

Pump malfunction process:

1. How is a pump malfunction detected ?
2. If a pump malfunction is detected what are the steps to fix the malfunction ?
3. Who are the people involved in the process ?
4. What are the roles of the people involved ?
5. What is the role hierarchy of these roles (chain of command) ?
6. What are the systems involved in this process and who is responsible of these systems ?
7. What kind of information security concerns do you have in relation to this process ?
8. Are there any access control restrictions related to systems involved in this process ?

Baggage handling process:

1. What are the steps for a normal bag to travel from customer to the airplane ?
2. Who are the people involved in the process ?
3. What are the roles of the people involved ?
4. What is the role hierarchy of these roles (chain of command) ?

5. What are the systems involved in this process and who is responsible of these systems ?
6. What kind of information security concerns do you have in relation to this process ?
7. Are there any access control restrictions related to systems involved in this process ?

Getting physical access clearance:

1. Who are the people involved in the previous processes, and how do they get clearance ?
2. What is the process of getting ID ?
3. What is the process of getting authorisation ?
4. Is authorisation temporal or permanent ?
5. If a user is authorised to perform a task, is he allowed to do it anytime? or are there some extra time-based restrictions ?

Conclusion

After conducting the interviews enough data was gathered to design the three business processes. After that the process models were presented to the same stakeholders and experts to validate that the process models are describing the exact process. The feedback was taken, and the final exact process for each operation was presented in the corresponding process model.

The results of the case study showed that business processes do not consider access control although they are closely related. It showed three different processes from a complex environment and how each process deals with different types of access controls along the execution of the process. Each example showed how the business process is represented, and that there are some access control policies that need to be followed which are not visible nor known to people executing the business process.

The results of the the case study work included the three business process models and the access control requirements associated with each process. However in this thesis only the process of 'fixing a pump malfunction' was used. This process was chosen after discussing the three processes and their details with the research team. The process of fixing the pump malfunction was found to be the most relevant and the one with more complexity and more access control requirements.