XACML as a Security and Dependability Pattern for Access Control in AmI environments

Antonio Muñoz[1], Francisco Sanchez[1], Paul El Khoury[2], Luca Compagna[2]

[1]University of Malaga, {amunoz,cid}@lcc.uma.es
[2]SAP Labs France {paul.el.khoury, luca.compagna}@sap.com

Abstract. One of the most interesting paradigms of Ambient Intelligence is that networks of pervasive intelligent interfaces recognize our presence and mould our environment to our immediate needs. In this paper, we present an example of how an access control model such as XACML adapts its functionality at runtime to new and unforeseen requirements. In previous work, we have proposed a three levels hierarchy of artefacts to semantically represent Security and Dependability solutions so that they can be automatically applied and adapted to new context requirements. Here we apply those artefacts throughout two case studies covering (i) the representation of the XACML model and (ii) a Policy Enforcement Point. The use of these artefacts provides the interoperability, run-time reaction to changes in the application context, and the possibility to monitor the applied solutions.

1 Introduction

Most of current techniques to address Security and Dependability (S&D) issues are designed for static architectures, with well-defined pieces of hardware, software, communication links, limits and owners. Thus, they fail when confronting new and challenging computing paradigms for highly dynamic environments such as Ambient Intelligence, where networks of pervasive intelligent interfaces recognize our presence and mould our environment to our immediate needs.

Several approaches have faced the security management of multiple devices with a large number of small interconnected applications [1, 2], paving the way to further advance in the three building blocks of AmI technologies [3]: *Ubiquitous Computing*, entailing the apparition of numerous heterogeneous computing nodes, which must cooperate despite their heterogeneity, and lack of a central control; *Ubiquitous Communication* implying the intercommunication of these objects, probably in an unpredicted way, introducing two important challenges: (i) the development of an adequate interface to permit secure communication among diverse components and (ii) the ability to negotiate the different parameters of the communication; and finally, *Intelligent User Interfaces* that will enable people to control and interact with the environment in a natural (voice, gestures) and personalized way (dependence on the context) without saturating users with technical decisions.

AmI considerations lead us to argue that it is essential for S&D mechanisms to be able to adapt themselves to renewable context conditions in order to be applied to the at development time, to S&D Patterns and S&D Implementations, devoted to deployment and runtime use. In order for these artefacts to be exploited at runtime, a whole architecture is introduced and described as the SERENITY Runtime Framework (SRF). The SFR, already introduced in [4, 6], represents an integral approach for the automated selection, adaptation and monitoring of the S&D Artefacts. This paper provides a guide on how to use the S&D Artefact, creating a set of them from the analysis of two well-known security solutions: the XACML access control model and a Policy Enforcement Point.

In what follows, section 2 covers a necessary introduction to the concepts of S&D Pattern, Classes and Implementations, along with a brief description of the architecture where the SRF and S&D Artefacts converge. Section 3 presents the analysis and representation of a security solution as an S&D Pattern. Section 4 follows the results of section 3 and presents the concept of Integration Scheme as set of S&D Patterns whose composition allow us to characterize a complex access control model. The previous work on analysis and representation of security models and the different approaches for adaptive dependability is given in section 5. We close with conclusion and further work.

2 S&D Artefacts for Runtime use

2.1 A hierarchy of solutions: Classes, Patterns, and Implementations

As outlined in the introduction, one of our goals is the development of artefacts for the representation of S&D Solutions in the form of semantic descriptions, in such a way that these solutions can be selected, adapted, used and monitored at runtime by automated means. In the interest of enabling this automation, three are the artefacts that integrate the hierarchy proposed to represent the S&D Solutions: S&D Classes, S&D Patterns, and S&D Implementations. Although this paper emphasized the use of S&D Pattern artefact, [5] presents an intuitive and extensive description of them all.

S&D Solutions refer to widespread and well-known S&D mechanisms, going from a secure key exchange protocol or a VPN application, to a data encryption algorithm. *S&D Patterns* are detailed descriptions of abstract S&D Solutions that contain all the information necessary for the selection, instantiation and adaptation, and dynamic application of the solution represented in the S&D Pattern. One important aspect of the solutions represented as S&D Patterns is that they can contain a description of the results of any static analysis performed on them. Such descriptions provide a precise foundation for the informed use of the solution and enhance the trust in the model. Despite of that, the limitations of the current static analysis tools introduce the need to support the dynamic validation of the behaviour of the described solutions by means of monitoring mechanisms. However, we will skip the details of the monitoring mechanisms to concentrate in the functionality of the S&D Solutions presented here.

While each S&D Pattern describes the behaviour of one solution, the complexity of the S&D requirements to address in a system might require the combination of S&D Patterns. *Integration Schemes* (IS for short) capture this compositionality means, melting the functionality of two or more S&D Patterns, which –if not properly analysed, can lead to interferences among them causing an unpredicted functionality.

If two S&D Patterns provide the same S&D Properties (e.g. confidentiality or non-repudiation) and comply with a common interface, then we group them in an S&D Class. *S&D Classes* represent abstractions of a set of S&D Patterns characterized for providing the same S&D Properties and complying with a common interface [16]. (Notice that S&D Patterns that belong to an S&D Class can have different interfaces, but they must describe how these specific interfaces map into the S&D Class interface.) With this approach it is possible for developers (at development time) to create an application bound to a specific S&D Class. Given that this artefact defines the high-level interface, all S&D Patterns belonging to this S&D Class and matching the context requirements will be selectable by the framework at runtime, thus providing a high degree of interoperability.

Finally, to close the gap between the real executable components (HW or SW) and the S&D Patterns, one last artefact is introduced: the *S&D Implementation* is an artefact that describes the specific context conditions to meet before deploying the executable component. It conforms directly to the interface, monitoring capabilities, and any other characteristic described in the S&D Pattern implemented.

2.2 Underlying Architecture: the Serenity Runtime Framework

This section describes how the SERENITY Runtime Framework (SRF) navigates throughout the S&D Artefacts' hierarchy in order to discover, select and deploy an S&D Solution. Figure 1 shows a simplified structure of the SRF.

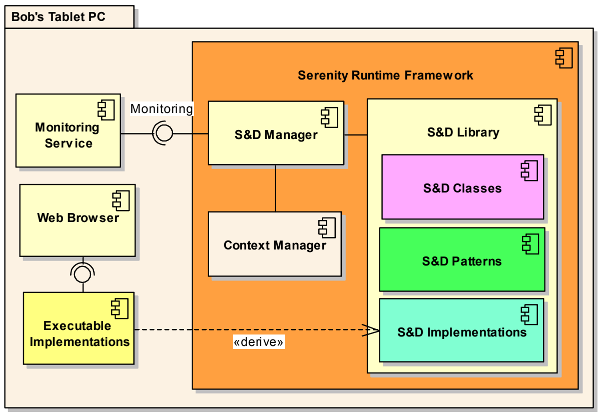

Figure 1 Simplified perspective of SRF

Our approach assumes that instances of SRF can be embedded in any type of device with a minimum computational power. Every SRF instance acts as a dynamic provider of S&D solutions to applications also allowing the monitoring of the behaviour of the solutions. Each SRF instance has an *S&D Library* containing a selected set of the artefacts described in section 2.1, including all S&D Classes, Patterns and Implementations corresponding to the solutions already deployed, along with additional solutions for future use. When a change in the context is detected via the *Context Manager,* the library is accessed by the SRF to look for the best pattern to meet the new requirements (detailed in section 3.3). Eventually, if a solution is selected, the SRF uses the information provided by the S&D Implementations and dynamically deploys the corresponding *Executable Component*.

After that, the run-time *monitoring* mechanism starts monitoring the workflow of the system. This process is indispensable in case of change of context (e.g. swapping the connection from a trusted to an untrusted network), when the Framework has to adapt on-time the current solution in order to face the new requirements.

3 A worked-out example: XACML captured via S&D Patterns and Integration Scheme

A brief example will help us to understand the overall functionality. Bob uses the internal forum of his company to manage his team. The internal forum is the company's essential communicating platform, where news ragings from confidential to top secret are posted. Alice, the network and forum administrator have to keep the confidentiality of the data maintained, however provide these data only to legitimate employees. This example clearly asks for an Access Control (AC) mechanism to be in place at the company that Bob is working for. An AC mechanism is the means by which a system grants or denies the right for a subject to perform some actions on some resources according to the policy defined by the system's security officer.

More precisely, a *subject* is the initiator of a request for access, e.g., Bob. A *resource* is the valuable data to be protected, e.g., the information stored in the working group forum of Bob's company. *Actions* are the set of operations that subjects can request to interact with the resources, e.g., post a message in the working group forum. Last but not least, a *policy* is the set of rules prescribing whether a subject can perform an action on a resource or not, e.g., the security officer Alice has defined the following policy rule: a message can be post in the working group forum if, and only if, *(i)* the sender is subscribed to the forum, *(ii)* the message is sent from within the intranet network that contains the server running the working group forum. As a matter of fact, not all the available AC mechanisms allow for the specification and enforcement of environmental conditions such as *(ii)* strongly required by our (simple) example and by AmI scenarios in general.

In the sequel of this section we will present XACML (eXtensible Access Control Markup Language [17]), an AC mechanism offering that level of granularity and adaptability necessary to express and enforce general environmental conditions as required by AmI scenarios. Then we will show how our S&D artefacts hierarchy allows for capturing this AC mechanism through an Integration Scheme and S&D patterns. With respect to our worked-out example and at development time, the SERENITY framework would suggest to the system designer the XACML Integration Scheme as security solution and AC mechanism for the system to develop.

Last but not least we will discuss how the SRF exploits the S&D artefacts hierarchy to deal at runtime with a context change in our example.

3.1 XACML

As depicted in Figure 1, XACML is built on top of six basic entities: the Policy Enforcement Point (PEP), the Policy Decision Point (PDP), the Policy Administration Point (PAP), the Context Handler (CH), the Obligation Service (OS), and the Policy Information Point (PIP). The PEP is the XACML's front-end that receives a subject's request, initializes its evaluation process, and sends back the answer. The PDP selects the applicable policies and computes the authorization response by evaluating the requests with respect to these policies. The PAP stores the policy rules required for the PDP. The CH acts as a bridge translating the received requests into a proper XACML format and vice versa for the responses. Finally, the OS and PIP are used to retrieve obligations resulting from evaluating the policies and to retrieve the attributes for the subjects or resources, respectively.

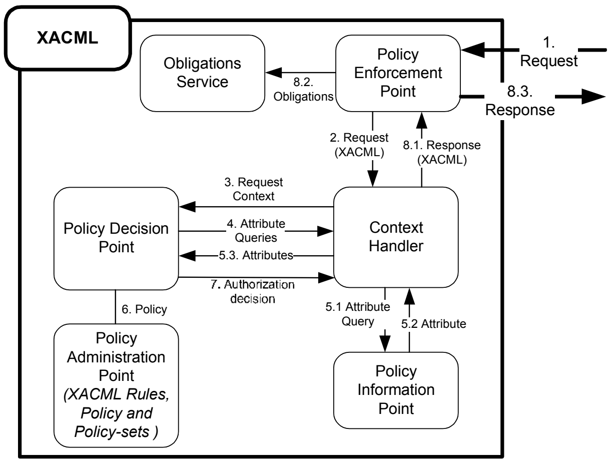

Figure 2. XACML process overview

The AC decision process is hereafter described in more details (see again Figure 2):
1. The Subject requests authorization from the PEP to access the resources.
2. The PEP sends the request for access to the CH in its native request format, including attributes describing the subjects, resource, etc.
3. The CH takes the received data, constructs a corresponding XACML request context, and sends it to the PDP.
4. The PDP analyses the XACML request and if necessary, ask the CH to retrieve additional attributes.
5. The CH requests the necessary attributes from the PIP and sends them back to the PDP.

6. The PDP gets the applicable policies, stored in the PAP, and check if the subject's request satisfies the policies. Notice that since several policies can be in place, the PDP uses algorithms such Deny-Override, Permit-Override or Only-Once Applicable to decide over them.
7. The PDP returns the authorization decision to the CH. Notice that the positive authorization decisions can comprise some additional obligations the subject has to satisfies to get access to the required resource.
8. The CH translates the XACML response received from the PDP to the native response format and returns the authorization decision to the PEP. Using this information, the PEP checks the obligations (if any) and either grant or deny access to the subject.

3.2 XACML as an Integration Scheme and S&D Patterns

S&D Integration Schemes (IS) are particularly suited to capture composed object like the XACML model. The basic idea is to capture each component of the XACML model as an S&D Pattern and to capture their composition and interaction through an IS. Table 2 **S&D Class for Policy Enforcement Point**shows an excerpt of the IS capturing the XACML model for automated application by the SRF (see the Appendix for a complete commented version of the IS). Components, e.g., the PEP, are specified within *row 6*. The name of the component is used to distinguish calls to one S&D Pattern or another (e.g., a call to the PEP is specified through `PEP.operationName`). *Row 4* describes the IS interface, where several calls to operations fulfilled by external components are defined. *Row 5* includes the explicit mapping from the S&D Class grouping AC patterns and IS to the XACML's IS. The *PatternClass* consists of the *S&D Class Reference* providing the name of the *S&D Class* parent for this IS (`Access_Control.security.uma.es`) and the *Interface Adaptor* providing the sequence of operations applied by the IS. In this case, the pseudo-code points to the `accessRequest` operation that constructs the sequence of actions in the IS. The full sequence presented in the Appendix depicts the XACML overview in getting the AC decision as described previously. For instance, the `PEP.getDataFromSubject()` describes the operation where the PEP gets the access information from the Subject, including the resource to be accessed, the action to perform, etc. The `authorizationDecision` holds the response decision that the IS returns to the requester when the process is finalized.

S&D IntegrationScheme: `XACMLAccessControl`	
…	**…**
4	**Interface**

	Calls:
	`PEP.getDataFromSubject(in:resourceAttributes,in:environmentAttributes,`
	`in:contentResources, in: subjectAttributes,out: requestForAccess)`
	`PEP.getObligationFromResponse(in: responseForAccess, out: obligationID)`
	`CH.translateRequest(in: requestForAccessData, out: XACMLRequest)`
	`....`

5	**PatternClass**
	S&DClass Reference: `Access_Control.security.uma.es`

	Interface Adaptor:
	`accessRequest (in: resourceAttributes, in: environmentAttributes, in: contentResources,`
	`in: subjectAttributes, out: authorizationResult)`
	`{ requestForAccess = PEP.getDataFromSubject(resourceAttributes, environmentAttributes,`
	`contentResources, subjectAttributes)`
	`XACMLRequest = CH.translateRequest(requestForAccess)`
	`requiredAttributesID = PDP.getListOfRequiredAttr(XACMLRequest)`
	`...`
	`return authorizationResult }`

6	**Components**
	Component `PEP (S&D Pattern for Policy Enforcement Point)` **….**
…	…

Table 1 S&D Integration Scheme XACML

The XACML IS depends on the PEP, PAP and other entities to provide the S&D solution for Alice's requirement. For sake of simplicity we only discuss how the Serenity tool captures in our artefact the PEP patterns' instances i.e., the two S&D Patterns Fedora-PEP and the QoS-PEP through their shared S&D Class. The S&D Class represents the Policy Enforcement Point requirement at high level. The system receives (from a subject) a request for access and returns an enforcement of the access response. The process in between is transparent from the subject's side, but includes a complex procedure that provides different types of PEP S&D Properties to the system. Table Table 1 **S&D Integration Scheme XACML** is a visual representation of the structure for the PEP S&D Class. All *provided properties* are specified in the S&D Class, making it possible for the Serenity tool to decide whether this class matches with the system requirements. The *ID* field of the property describes a universal identifier (e.g. *Fedora-PEP*, identifies the PEP property as the one formally defined by *Fedora.com*). The S&D Class also includes an interface that defines the operations that conforms to its functionality. In the example, a unique *call* is specified and all S&D Patterns belonging to this S&D Class have to consent with this interface.

S&D Class: `PolicyEnforcementPoint`	
… **3**	**…** **Provided Properties:**
	Property: `PolicyEnforcementPoint:` **ID**: `PEP.Fedora.com` **Property:** `PolicyEnforcementPoint`**:** **ID:** `PEP.QoS.Fedora.com`
4	**Interface Definition:**
	Calls: `getAccess ( in: subjects::text, in: objects::text, in: ACL::text)`

Table 2 S&D Class for Policy Enforcement Point

Table 3 S&D Pattern Fedora-PEPshows the Fedora-Policy Enforcement Point representation in our S&D pattern structure, while Table 4 S&D Pattern QoS-PEP shows the one for the QoS Policy Enforcement Point. The basic difference between these two Patterns remains in the implementation applied at the lower level. The Fedora-PEP pattern can not retrieve the data from the requester if the requester is using a SmartPhone while the QoS-PEP pattern enables it with the

`getDataFromSubjectSupportingQoS`. Further details are omitted on the basic difference as it falls out of the paper's scope.

S&D Pattern: Fedora-`PEP`	
…	…
5	**PatternClass**

	S&DClass Reference: `Policy_Management.security.uma.es`
	Interface Adaptor
	`getAccess (subjects [], objects [], ACL []){`
	`    getDataFromSubject(in: subjects, in: objects, in: ACL, out: requestForAccess)`
	`    getObligationFromResponse(in: responseForAccess, out: obligationID)`
	`    checkObligation(obligationToFulfil, responseForAccess)}`

…	…

Table 3 S&D Pattern Fedora-PEP

S&D Pattern: QoS-`PEP`	
…	…
5	**PatternClass**

	S&DClass Reference: `Policy_Management.security.uma.es`
	Interface Adaptor
	`getAccess (subjects [], objects [], ACL []){`
	` getDataFromSubjectSupportingQoS(in:subjects, in:objects, in:ACL,`
	`                                  out:requestForAccess )  ... }`

…	…

Table 4 S&D Pattern QoS-PEP

3.3 SRF at work

Bob's job requires posting messages in the working group forum very frequently. He usually works from office using his company laptop under Windows XP. Due to Bob's outstanding results, he got promoted and awarded with a brand new SmartPhone that gives Bob a 24h access to the Internet. The SmartPhone has some QoS requirements that the actual PEP does not provide. Bob uses his new device to access the forum, however he access was denied. As a matter of fact, the *Context Manager* realizes that the browser is trying to connect to the intranet from an unsupported system. As the SRF was configured to work over systems supporting the Fedora-PEP, the active S&D Pattern providing the PEP is no longer valid, and the system must be reconfigured (needs to support the QoS requirement). The *S&D Manager* analyses the context information coming from the *Context Manager* along with the current S&D Requirements and triggers a query to find the best solution available in the S&D Library. Since the new requirement is to extend the PEP pattern with the QoS option, the SRF queries the S&D Library and finds the S&D pattern QoS-PEP. The new S&D pattern QoS-PEP is activated and its implementation is selected and replaced in PEP part of the XACML IS. On the next day at the restaurant, using his SmartPhone, Bob succeeds in posting some messages on the internal forum.

4 Related Work

Our work proposes a complete framework for the rigorous treatment of S&D solutions that covers the automated selection, deployment and monitoring of the artefacts. Several approaches going from component-based to multi agent systems have been proposed for this problem in the literature.

Current work on component-based software development is mainly focused on the dynamic analysis of component compatibility, usually from a functional point of view, with the objective of adapting components and synthesizing suitable software architectures [7–10]. Several component-based security models have been proposed in the literature. Unfortunately, these proposals have been based on oversimplified views of security, like those based on security levels [11], not applicable in AmI.

The most related approach to our work is the one presented in [12], where security patterns are used to construct secure and efficient inter-company coordination systems. This approach addresses the selection of the security pattern issue by dealing with the messaging models and the security models in the design phase. As a result, they were able to give the developers guidelines that consist in modelling the performance of data associated with each pattern for model selection in which security is considered. However, these guidelines are in natural language and thus do not allow security patterns to be expressed and supported for their automatic selection and posterior deployment.

The agent paradigm is especially well-suited for highly distributed environments where independent components from different owners coexist and interact. In [13] authors present a methodology that considering the organizational structures of agent systems, designs the abstract models of agents in a top-down manner. This method can cope with simple access controls and interaction patterns, but when modelling security aspects agent paradigms are quite limited, since an agent is an independent entity by definition and many security solutions like XACML, can not be represented.

A security architecture that system administrators, users, and application developers can use to compose secure systems is presented in [14]. This architecture is designed to support the dynamic composition of systems and applications from individual components, but it lacks of flexibility and is restricted to access control models. The CORBA RAD service is an example of this type of security model for access control to resources in component systems [15].

5 Conclusions and future work

This work presents a bottom-up approach that includes a Framework to provide S&D Solutions to applications by means of three S&D artefacts: (i) S&D Implementations to describe SW/HW components; (ii) S&D Patterns, which describe S&D Solutions and groups the S&D Implementations that realize them, and (iii) S&D Classes, which groups the S&D Patterns that provide the same S&D Properties and share a common interface. We have used them to analyse and describe XACML model as an Integration Scheme. In addition, we provided the full deployment of Policy Enforcement Point pattern, one of the components of that Integration Scheme.

Next steps include the provision of a framework to assist security experts in the creation of S&D Solutions, currently in progress, along with the definition of (i) a language for S&D Properties and (ii) the formalization of the monitoring rules.

References

1. Khan, K. M., Han, J., and Zheng, Y. 2000. Security Characterization of Software Components and Their Composition. In *Proceedings of the 36th international Conference on Technology of Object-Oriented Languages and Systems (Tools-Asia'00)* (October 30 - November 04, 2000). TOOLS. IEEE Computer Society, Washington, DC, 240.
2. Sewell, P. and Vitek, J. 1999. Secure Composition of Insecure Components. In *Proceedings of the 1999 IEEE Computer Security Foundations Workshop* (June 28 - 30, 1999). CSFW. IEEE Computer Society, Washington, DC, 136.
3. Kurt Bauknecht. 2002. LV-Nummer: 400376. Ambient Intelligence: The Vision of Information Society. BWZ der Universität Wien.
4. Francisco Sanchez-Cid, Antonio Muñoz, Daniel Serrano, M.C. Gago. Software Engineering Techniques Applied to AmI: Security Patterns**.** *In Proceedings of the First International Conference on Ambient Intelligence Developments* (September, 2006). Developing Ambient Intelligence, Springer. Pages 108-124. ISBN**:** 2-287-47469-2
5 Francisco Sanchez-Cid, Antonio Maña. Patterns for Automated Management of Security and Dependability Solutions. *1stInternational Workshop on Secure systems methodologies using patterns* (SPattern'07)*,* Regensburg (Germany), September 03-07, 2007.
6. Antonio Maña, Francisco Sanchez-Cid, Daniel Serrano, Antonio Muñoz. Building Secure Ambient Intelligence Scenarios*.* *Eighteenth International Conference on Software Engineering and Knowledge Engineering* (SEKE'06), San Francisco (USA), 2006.
7. Becker, S.; Canal, C.; Diakov, N.; Murillo, J.M.; Poizat, P.; Tivoli, M. 2006. Coordination and Adaptation Techniques: Bridging the Gap between Design and Implementation. Report on the ECOOP'2006 Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'06). ECOOP 2006 Workshop Reader, LNCS, Springer.
8. Khan, K.; Han, J. Composing Security-aware Software. 2002. IEEE Software, Vol. 19, Issue 1, pp 34—41. IEEE.
9. Brogi, A.; Cámara, J.; Canal, C.; Cubo, J.; Pimentel E. 2006. Dynamic Contextual Adaptation CONCUR'2006 Workshop on the Foundations of Coordination Languages and Software Architectures (FOCLASA'06). Electronic Notes in Theoretical Computer Science, Elsevier, ISSN 1571-0661.
10. Khan, K.; Han, J.; Zheng, Z.; Security properties of software Components. 1999. Proceedings of Information Security: Second International Workshop, ISW'99, Lecture Notes in Computer Science, Volume 1729.
11. McDermid, J.A; Shi, Q. 1992. Secure composition of systems. Proceedings of Eighth Annual Computer Security Applications Conference. Pp. 112 – 122.
12. Nobukazu Y., Shinichi H., Anthony F. Security Patterns: A Method for Constructing Secure and Efficient Inter-Company Coordination Systems, Enterprise Distributed Object Computing Conference, 2004. Eighth IEEE International Volume , Issue , 20-24 Sept. 2004 Page(s): 84 – 97
13. M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems, 3(3),pp. 285–312, 2000.
14. Jaeger, T; Liedtke, J; Pantellenko, V; Park, Y; Islam, N. 1998. Security Architecture for component-based Operating System . In ACM Special Interest Group in Operating Systems (SIGOPS) European Workshop, 1998. 118.
15. López, J; Maña, A; Ortega, J.J; Troya, J.; Yagüe, M.I. 2003. Integrating PMI Services in CORBA Applications. Computer Standards & Interfaces, 25, 4, pp. 391—409, Elsevier.
16. C. Canal, L. Fuentes, E. Pimentel, J.M. Troya, A. Vallecillo. "Adding Roles to CORBA Objects". IEEE Transactions on Software Engineering 29(3):242-260, Mar. 2003.
17. XAMCL and OASIS Security Services Technical Committee, "eXtendible Access Control Markup Language (xacml) committee specification 2.0," Feb 2005.

Appendix

Table below includes a short version of the Integration Scheme structure. Where new fields like *Creator* and *TrustMechanisms* are included and the Interface adaptor is fully deployed.

S&D IntegrationScheme: XACMLAccessControl	
1	**Creator:** University of Malaga, 2007-05-05
2	**Trust mechanisms** ^*{[{TE€#¬€RT$)(=?)?=/UHYG5€€\|#EW·$%&$}ç+
3	**Provided Properties**
	Property: AccessControl **ID:** accessControl.security.uma.es **Timestamp:** 1083753687
4	**Interface**

	Calls:
	PEP.getDataFromSubject(in:resourceAttributes,in:environmentAttributes,in:contentResource , in: subjectAttributes, out: authorizationResult out: requestForAccess)
	PEP.getObligationFromResponse(in: responseForAccess, out: obligationID)
	CH.translateRequest(in: requestForAccessData, out: XACMLRequest)
	....
	SO.getObligation(in: obligationID, out: obligationToFulfil)

5	**PatternClass**

	S&DClass Reference: Access_Control.security.uma.es
	Interface Adaptor:
	accessRequest (in: resourceAttributes, in: environmentAttributes, in: contentResources, in: subjectAttributes, out: authorizationResult)
	{ requestForAccess = PEP.getDataFromSubject(resourceAttributes, environmentAttributes, contentResources, subjectAttributes)
	XACMLRequest = CH.translateRequest(requestForAccess)
	requiredAttributesID = PDP.getListOfRequiredAttr(XACMLRequest)
	for attributeID in {requiredAttributesID} do
	  if (attributeID = null) then
	    break
	  endif
	  requiredAttributes + = PIP.getAttributeFromName(attributeID)
	endfor
	CH.sendAttributesToPDP(requiredAttributes)
	requiredPolicy = PDP.analiseRequestForPolicy(XACMLRequest)
	policy = PAP.getPolicy(requiredPolicy)
	responseContext = PDP.analiseRequest(requiredAttributes, policy)
	responseForAccess = CH.translateResponse(responseContext)
	obligationID = PEP.getObligationFromResponse(responseForAccess)
	obligationToFulfil = SO.getObligation(obligationID)
	authorizationDecision = PEP.checkObligation(obligationToFulfil, responseForAccess)
	return authorizationResult = authorizationDecision
	}

6	**Components**
	Component PEP (S&D Pattern for Policy Enforcement Point) **….** **Component** PIP (S&D Pattern for Policy Information Point)
7	**Pre-Conditions**
	Supports labelling XACML PAP, PDP, PIP, ContextHandler and PEP are available
8	**Static Tests Performed**
	Test: Tested for Confidentiality satisfied for objects **Conditions:** of test: APA from SIT **Attack models considered:** Unauthorized disclosure, Message replay, Message insertion, Message deletion, Message modification, NotApplicable results, Negative rules