Configuring Private Data Management as Access Restrictions: From Design to Enforcement

Aurélien Faravelon[1], Stéphanie Chollet[2], Christine Verdier[1], and Agnès Front[1]

[1] Laboratoire d' Informatique de Grenoble,
220, rue de la chimie, BP 53 F-38041 Grenoble Cedex 9
{aurelien.faravelon,christine.verdier,agnes.front@imag.fr}@imag.fr
[2] Laboratoire de Conception et d'Intégration des Systèmes
F-26902, Valence cedex 9, France
stephanie.chollet@lcis.grenoble-inp.fr

Abstract. Service-Oriented Computing (SOC) is a major trend in designing and implementing distributed computer-based applications. Dynamic late biding makes SOC a very promising way to realize pervasive computing, which promotes the integration of computerized artifacts into the fabric of our daily lives. However, pervasive computing raises new challenges which SOC has not addressed yet. Pervasive application relies on highly dynamic and heterogeneous entities. They also necessitate an important data collection to compute the context of users and process sensitive data. Such data collection and processing raise well-known concerns about data disclosure and use. They are a brake to the development of widely accepted pervasive applications. SOC already permits to impose constraints on the bindings of services. We propose to add a new range of constraints to allow data privatization, *i.e.* the restriction of their disclosure. We extend the traditional design and binding phases of a Service-Oriented Architecture with the expression and the enforcement of privatization constraints. We express and enforce these constraints according to a two phases model-driven approach. Our work is validated on real-world services.

Keywords: Access restriction, SOA, workflow, private data.

1 Introduction

Service-Oriented Computing (SOC) is a major trend in designing and implementing distributed computer-based applications. Applications are implemented by composing already existing functionalities called services which exposed over networks such as the Internet. Services are loosely coupled and SOC thus promotes the distinction between the design of the composition and its execution. Indeed, the application is designed without knowing which services will actually be available. The application is then executed by invoking and binding the necessary services among the set of available services.

Dynamic late biding makes SOC a very promising way to realize pervasive computing, a new paradigm which promotes the integration of computerized artifacts into the fabric of our daily lives. Pervasive computing relies on distributed and highly heterogeneous and dynamic entities. Sensors, softwares or devices are such entities. Their composition is crucial to build efficient and innovative applications. Pervasive applications have to be flexible and adaptive. Exposing functionalities as services, distinguishing the application's design from its execution and realizing the application by binding the actual services meet these requirements.

However, pervasive computing raises new challenges which SOC has not addressed yet. Pervasive computing necessitates an important data collection. Collecting the location of users who interact with the application, for instance, is necessary to identify their contexts of use. Using pervasive computing thus means sharing data which flow in the application - such as medical files or credit application - and disclosing data about the users of the composition. As sensitive data can be derived from seemingly inoffensive pieces of data, these two groups of people can be modeled in details. Such a possibility raises well-known concerns about data disclosure and use. They are a brake to the development of widely accepted pervasive applications. As a result, there must exist a mechanism to constraint data disclosure.

SOC already permits to impose constraints on the bindings of services. We propose to add a new range of constraints to allow data privatization, *i.e.* the restriction of their disclosure. Privatization constraints bear on what an observer external to the composition can deduce from the binding of services, on what a client can ask a service provider about and on what a service provider can ask a client about.

We extend the traditional design and binding phases by allowing the expression and the enforcement of privatization constraints. We express and enforce these constraints according to a two phases model-driven approach:

- At design level, we extend a composition language with a platform independent privatization language to express privacy constraints which must be enforced at binding time. Privacy designers can use this language to express privatization constraints.
- At binding time, automatic model-to-text transformations inject the appropriate code to enforce privatization constraints in a Service-Oriented Architecture (SOA). We modify the SOA to store the information necessary to enforce access restriction.

The paper is structured as it follows. In Section 2, we introduce a global overview of our approach. We detail our design level in Section 3 and the execution level in Section 4. Eventually, we validate our approach in Section 5 before discussing the works related to ours in Section 6 and concluding in Section 7.

2 Global Approach

Our goal is to ease the design and implementation of privacy-aware pervasive applications realized as compositions of heterogeneous and dynamic services.

We propose a model-based approach where security properties related to private data management as access restriction are specified at design time. These properties are realized through code generation at binding time depending on the available services.

Because of the inherent heterogeneity and dynamism of services, access restriction is a problem throughout the entire application's life-cycle. Services that are actually used at runtime are generally not known at design time. Their access restrictions capabilities cannot be predicted or relied on. Furthermore, design time involves non-technical stakeholders who cannot make sense of technical minutes.

We have designed and connected two platform-independent views, a service composition one and an access restriction one. From these views and their relationships, we automatically generate an executable privacy-aware service composition as shown on Figure 1.

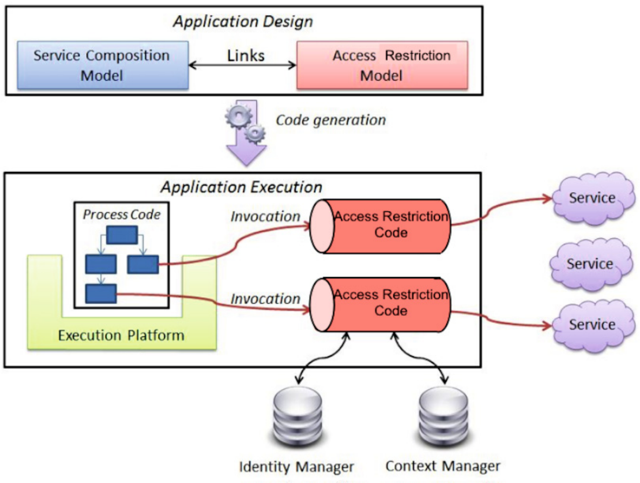


Fig. 1. Background of our proposition

Composition and access restriction are specified at design time. We are driven by two principles: abstraction and separation of concerns. Modeling allows us to abstract away technical details and separation of concerns permits to accommodate different viewpoints of the same application. Privacy experts can design the privacy policy.

At binding time, the application is realized as a service orchestration. Access restriction features are injected through the generation of a proxy for each service instance. We have also developed an identity manager which deals with users privileges and a context manager which handles contextual information retrieval and processing.

3 Design Level

Figure 2 displays the service composition and the access restriction metamodels and their relationships.

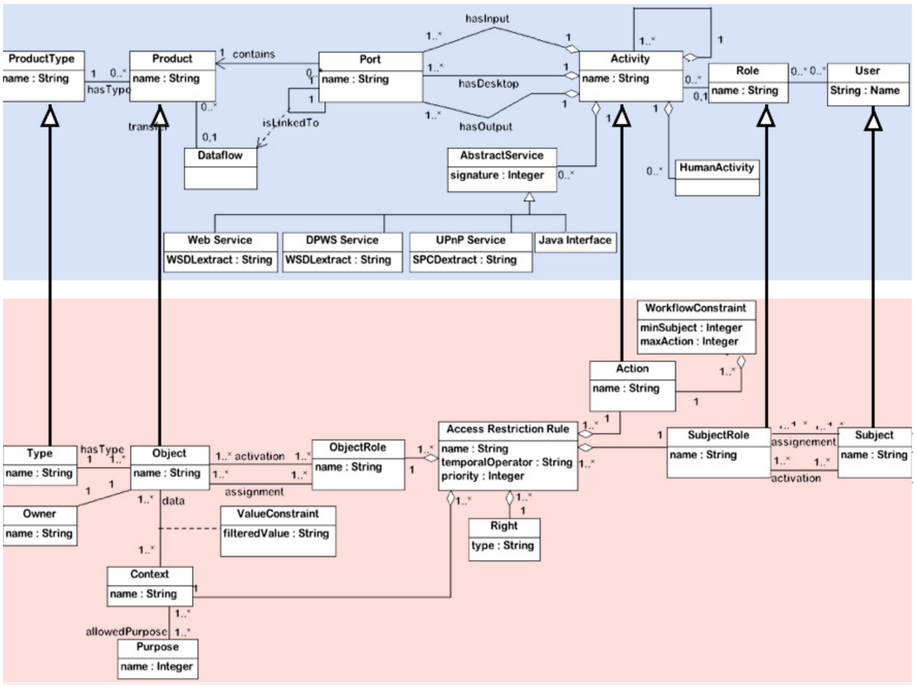


Fig. 2. Relations between Service Composition and Access Control Metamodels

3.1 Service Composition Metamodel

Our approach regarding service composition builds on the Abstract Process Engine Language (APEL) [5]. As visible at the top of Figure 2, APEL is a high level process definition language. We chose APEL because it natively supports any type of service when other process languages, such as WS-BPEL, are dedicated to a specific technology. It contains a minimal set of concepts sufficient to specify a process:

– An *Activity* is a step in the process that results in an action, realized by a human or a computer. Activities can be made of sub-activities; they are then said to be composite. An activity has *Ports* representing communication interfaces. An *Activity* must be realized by a *User*.

- A *Product* is an abstract object that flows between activities. *Dataflows* connect output ports to input ports, specifying which product variables are being transferred between activities.
- An *Abstract Service* can be attached to an activity. It represents a type of service to be called in order to achieve the activity.

Specifically, an *Abstract Service* is a service specification retaining high level information and ignoring as many implementation details as possible. Our model defines an abstract service in the following terms:

- A signature defining the identifying name of the service, its inputs and outputs in terms of products.
- Possibly, technology-specific information. This part corresponds, for instance, to WSDL extracts for Web Services or SCPD extracts for UPnP services. Extracts only contain implementation-independent information. For instance, no address is provided, contrarily to complete WSDL descriptions.

Providing technology-specific information means that the service technology is chosen before hand by designers, which is actually frequently the case. This information is useful since it permits to generate a better and leaner code.

3.2 Privacy Metamodel

We hold that private data management can be efficiently modeled as access restriction. As visible on the bottom of Figure 2, users design access restriction rules.

Specifically, a rule gathers:

- A *Subject i.e.* a user or a software agent that acts in the application. *Subject* are categorized in *SubjectRoles* according to their position in an organization.
- An *Object i.e.* any entity a *Subject* can affect. *Objects* can be categorized according to their functionalities for instance according to *ObjectRoles*. Each *Object* has an owner and a type. The owner is responsible for managing the access to their data.
- An *Action i.e.* an access mode to an *Object*.
- A *Right i.e.* the modality of a *Subject*'s relation to an *Action*. *Rights* are divided into permissions, obligations and prohibitions, they apply to *SubjectRoles* and *ObjectRoles*.

Several *Access Control Rules* may apply to the same set of *Actions*, *ObjectRoles* and *SubjectRoles*. *Access Control Rules* are conditioned by:

- *Context i.e.* a situation defined by a constraint over the values of a set of *Objects*. The *Context* entity captures the specificity of pervasive access restriction. A *Context* indicates for which purpose the access restriction rule is satisfied. A purpose is the reason why an *Action* is performed.

– The satisfaction of a workflow security patterns. We accomodate two of them, separation and binding of duties. Both of them restrict the number of *Subjects* that can intervene in a group of *Action* and the number of *Action* each *Subject* can perform. We gather these constraints under the name *Workflow Constraints.* They are defined by the maximum amount of *Actions* a set of *Subjects* can perform in a group of *Actions.*

Managing rules conflict is important to ensure the consistency of the privacy policy. Each rule is associated with a priority level. The rules with the highest priority win over the ones with lowest levels. When several *Access Control Rules* apply to the same set of *Actions*, *ObjectRoles* and *SubjectRoles*, these rules cannot share the same conditions. Eventually, when a permission or an obligation and a prohibitions are conflicting, the prohibition always takes the precedence.

3.3 Logical Semantics

Having described the basis of our access restriction model, we now integrate them in order to allow the computation of privatization policies.

We have focused on `Compositions` as a temporally ordered flow of `Activities`, *i.e.* the activation of `Roles` by `Subjects`. This can be seen as a model of Computational Tree Logic CTL, [6]. CTL relies on a tree-like structure suitable for workflows where a moment can lead to several others. For example, from an `Activity`, another `Activity`, or an error can follow, leading to two different ends of the process.

Quantifiers to express that an access restriction rule bears on a single moment or on all of them are thus needed. We note **A** if all the moments are involved and **E** if only one of them is. CTL relies on temporal operators to indicate that a clause must be *always* (noted $\Box$), or *sometime* (noted $\Diamond$) true or that it must be true *until the next moment* (noted $\bigcirc$) or *until a moment in general* (noted u). Past CTL, PCTL [9], adds S, *since*, X^{-1}, *previous* and N, *from now on.*

We have previously emphasised three modalities of `Rights`. We note the `Permission` of doing something **P** and the Obligation **O**.

The syntax of our logical language in BNF, with ϕ and ψ, two access restriction policies is as follows:

$$\phi, \psi := \neg\phi | p | \phi \wedge \psi | (\mathbf{A}|\mathbf{E})\Box\phi | (\mathbf{A}|\mathbf{E})\Diamond\phi | (\mathbf{A}|\mathbf{E}) \bigcirc \phi | (\mathbf{A}|\mathbf{E})\phi u \psi | S\phi | X^{-1}\phi | N\phi | \mathbf{P}\phi | \mathbf{O}\phi$$

Computability. At binding time, evaluating the access restriction policy to determine a user's right, can be seen as a model-checking problem. We specify access restriction rules and `Flow Constraints` with logical propositions that are built as Kripke structures [11]. We define a satisfaction relation $\models$ between the policies ϕ and ψ and a `Composition` C. A `Composition` can be run, *i.e.* a set of `Activities` can be performed by a set of `Subjects` on a set of `Resources` under a certain `Context`, if and only if $C \models \phi, \psi$.

Let $C = (a_0, a_1, ..., a_n)$ where each a_i is an action, *i.e.* a tuple of the form $<context, right, subject, object>$. Then, $C \models \phi$ if and only if $(C, |C|) \models \phi, \psi$. C is defined by structural induction on ϕ and ψ as:

$$\begin{array}{l}
(C,i) \models p \text{ iff } p \in a_i \\
(C,i) \models \neg\phi \text{ iff } (C,i) \not\models \phi \\
(C,i) \models \phi \wedge \psi \text{ iff } (C,i) \models \phi \text{ and } (C,i) \models \psi \\
(C,i) \models \mathbf{E} \bigcirc \phi \text{ iff } (C,i+1) \models \phi \\
(C,i) \models \mathbf{E}\phi u \psi \text{ iff there exists } k \geq 0 \text{ s.t. } (C,i+k) \models \phi \\
\qquad\qquad \text{and } (C,i+j) \models \psi \text{ for all } k{>}i \geq 0 \\
(C,i) \models \mathbf{A}\phi u \psi \text{ iff for all } a_n \text{ there exists } k \geq 0 \text{ s.t. } (C,i+k) \models \phi \\
\qquad\qquad \text{and } (C,i+j) \models \psi \text{ for all } k{>}i \geq 0 \\
(C,i) \models X^{-1}\phi \text{ iff } n{>}0 \text{ and } (C,i-1) \models \phi \\
(C,i) \models \phi S \psi \text{ iff there exists } k \geq n \text{ s.t. } (C,k) \models \phi \\
\qquad\qquad \text{and } (C,i) \models \psi \text{ for all } k{<}i \leq 0 \ (C,i) \models N\phi \text{ iff } a_n \models \phi
\end{array}$$

3.4 Linking Service Composition and Access Control Views

When designing an application from multiple points of view, three problems must be addressed [16]. First, the metamodels must be related in order to build complete specifications. Then, views must be synchronized *i.e.* a mechanism must be provided to preserve coherency between views at execution. Relationships between the service composition and the access restriction metamodels are displayed on Figure 2.

Two points are of foremost interest: the classes to link in each metamodel, and the cardinalities of their relations. In the access restriction view, we define an *Action* as an access mode to an *Object*. In the process view, we define an *Activity* as an operation on a *Product*. In order to compose the two views, we thus express that an *Action* is a specific type of *Activity* constrained by access restriction rules. The *Action* class thus inherits from *Activity*. The same stands for *Objects* in the process view, that are specific *Products* to which access is restricted.

Views are designed in conformity with their metamodel. Views are then composed according to the inheritance defined between the metamodels: each activity in the process specification is refined into several possible actions constrained with access restriction rules defined in the access restriction view.

4 Execution Level

At runtime, available services cannot be trusted because they may not enforce access restriction. We secure an heterogeneous and dynamic composition in two steps:

– Before execution, orchestration code and access restriction insertion code are generated from each view's specifications. To synchronize the view, insertion points of access restriction code in the orchestration are identified.

– At execution time, the access restriction code is inserted between the orchestrator and the available services.

Figure 3 displays the execution of a pervasive orchestration secured by access restriction. When a new service is discovered by the execution machine, a secured proxy is generated and registered in the registry. Thus, the registry only contains secured Web-Services and the orchestrator cannot directly access unsecured services. Consequently, the composition cannot be executed without access restriction enforcement.

Access restriction enforcement relies on three components. The *Decision Point* evaluates the access restriction policy for a user and a given context. The *Context Manager* stores the path to contextual information sources such as users' smartphones or the composition's log file. The *Identity Manager* stores the roles of users and their identity.

When a secured proxy is invoked, it calls the access restriction *Decision Point.* The proxy provides the *Decision Point* with the current user's name and the current *Activity*'s name. The *Decision Point* retrieves the user's privileges from the *Identity Manager* and the necessary contextual information from the *Context Manager.* It then checks the access restriction policy according to the retrieved information and provides the secured proxy with a decision. The access restriction policy is composed of the global access restriction policy defined at the level of the composition and the restriction imposed by the concerned data owners. If the user is allowed to access the current activity, the secured proxy invokes the available service it protects. Otherwise, it rejects the invocation. Each communication between the proxy and the other components is secured with authentication.

Generating an Executable Access Control Policy. The *Decision Point* checks an executable access restriction policy derived from a process specification and its associated access restriction requirements. We generate the access restriction rules that apply to each *Action* and their temporal ordering from the designer's specifications. We gather all these information into an executable access restriction policy represented as an XML file. The access restriction policy is expressed according to the following grammar represented in Backus-Normal form, where S is a *Subject*, SR a *Subject Role* defined by a set of *Constraints* Cs. OR an *Object Role* defined by a set of *Constraints* Co and O an *Object.* SRA refers to the activation of a *Subject Role* by S, and ORA, the activation of an *Object Role* by O.

$$SR := Cs^+$$
$$OR := Co^+$$
$$SRA := (SasSR)^+ \; ORA := (OasOR)^+$$

An *Action* A, performed by a *Subject* S, playing the *Role* SR, on the *Object* O, playing the *Role* OR, under the *Context* Ctx with the *Right* R is represented in BNF as:

$$A := Ctx\,(R\,(SRAORA))$$

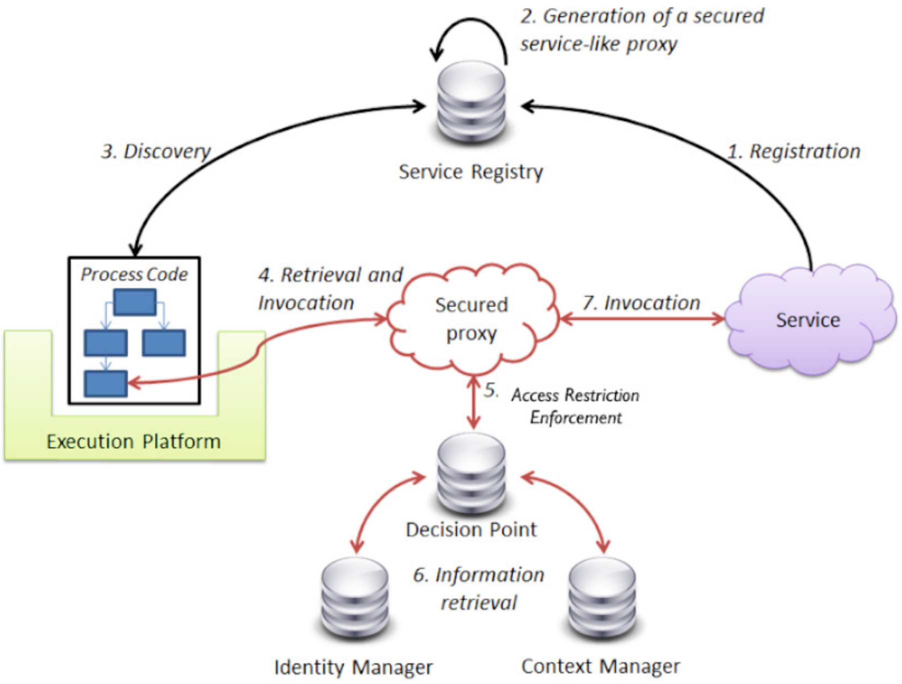


Fig. 3. Execution of a Secured Composition of Services

A *Process* consists in a temporally ordered flow of *Activities*. A *Process* P is represented in BNF as:

$$P := A^+$$

One or several *Workflow Constraint* P, *i.e. Separation of Duties* and *Binding of Duties* can be added to the composition. Such a constraint is represented by the following boolean constraint where $MaxS$ is the maximal amount of *Subjects* allowed to perform $MinA$, a minimal number of Activities:

$$P \rightarrow MaxS \text{ and } MinA$$

We see a *Process* as a temporally ordered flow of *Activities*, *i.e.* the activation of *Roles* by *Principals*. This can be seen as a model of Computational Tree Logic CTL, [6], an executable logic which holds a tree-like structure of time. Each node of the three is an action and we can specify the lifetime of its associated access restriction rules according to five temporal operators, *until the next activity*, *until an activity in general*, *since an activity*, *since the previous activity* and *from now on*.

Identifying Insertion Points and Enforcing Access Control at Execution. At execution, each *Activity* is realized by a service. Each available service is secured as it registers to the service registry by a proxy. This step is comparable to the compilation of the access restriction and the composition model for a specific platform. The proxy is built at runtime according to the target service through code generation from a template. Each template is parametrized by a set of variables such as the endpoint to call or the service's implementation. Each variable is set with values from the actual service to protect. We rely on Java Emitter Templates (JET) to perform code generation. Figure 4 shows a snippet of a JET for a secured proxy implemented as a Web Service.

```
...
 DecisionPointService dp = new DecisionPointServiceLocator();
 try {
        DecisionPoint portToDecisionPoint = dp.getDecisionPoint();
        //Test: Is the user allowed to access the current activity?
        right = portToDecisionPoint.makeDecision(userName, activity, sessionID);
        if (!right.equalsIgnoreCase("DENIED")) { // Calls the protected service if the user is allowed to do so
               <%=serviceName%> port = portToService.<%=getServicePortMethod%>();
               <%if (!returnType.equals("void")){%><%=returnType%> resultat=<%}%>port.<
               %=methode.getName()%>(<% numParam=0; for (Class c:
               parametresMethode) {numParam++;%><%=nomParam+numParam%><%if (numParam<
               (parametresMethode.length)){%><%=", "%><%}%><%}%>);
               <%if (!returnType.equals("void")){%>return resultat;<%}%>
 } catch (ServiceException e) {
        e.printStackTrace();
 }
 } catch (RemoteException e) {
               e.printStackTrace();
 }
...
```

Fig. 4. Extract of the proxy.javajet file

The proxy is itself a Web-Service and is thus transparent for the composition. As a consequence, our approach is independent from a specific platform or specific service type. The proxy acts as an access restriction enforcement point. To do so, Figure 4 shows that the proxy intercepts the invocation and asks the *Decision Point* to check if the current user is allowed to access the current activity. If and if only so, it invokes the service it protects.

We have adopted a centralized approach: the orchestrator is a centralized entity. However, for scalability reasons, the identify manager, the context manager and the decision point can be replicated.

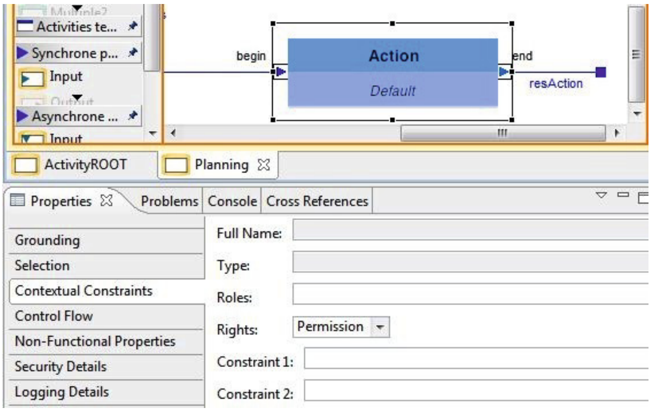
5 Validation

In order to validate our approach, we have developed an environment to model and execute a privacy-aware service composition secured by access restriction. This tool is a significative extension of the FOCAS orchestrator [5]. In this Section, we present its use and the results of our approach.

5.1 Design Level: Modeling Environment

The first part of the tool is dedicated to modeling compositions from multiple points of views. Functional experts can outline composition as processes by drawing activities, the links between them and the products that flow from one activity to another. The tool can also be used by privacy designers to visualize the data flow in the composition and the context-sensitivity of each activity in order to restrict data disclosure.

For each activity, access restriction rules can be defined. Figure 5 shows a snapshot of our tool for the alert management process.



Full Name:

Type:

Roles:

Rights: Permission

Constraint 1:

Constraint 2:

Fig. 5. Snapshot of our modeling environment

Each activity is associated to a set of property tabs which permit to edit its functional properties and the access restriction properties. Our tool permits the synthesis and the abstraction of process and access restriction views. Security experts and data owners can thus restrict object flows. We represent the execution of access restriction as a composition of dedicated services. Data owners can thus restrict the access to contextual data necessary to compute and access restriction decision.

Our tool allows several stakeholders to work together at various points of the composition's lifecycle. Moreover, it has two major advantages. First, as all models instantiate our domain specific modeling language and their links, specifications are *de facto* valid and coherent. Then, the tool provides a global view on the composition while allowing to define access restriction rules at service level. Temporal logic is hard to handle, especially when users are not familiar with such languages. Our tool presents time ordering of activity as a process, an intuitive representation. Temporal operators are derived from the process structure.

5.2 Binding Time: Execution Environment

At runtime, we add computation time dedicated to proxy generation and access restriction enforcement. We analyze this extra cost for four services in our service composition.We have constrained four activities with privatization constraints and we have secured a service for each activity. Services 1, 2 and 3 can only be accessed if the user is in a specific location and possesses a specific role. Service 4 is constrained with the same properties to which we had a constraint on the hours shifts it can be accessed. The client requesting the access to Service 4 must be on duty. The client's work schedule must thus be checked. Figure 6 displays, for each service, the duration of the service call, of the proxy generation and of access restriction enforcement.

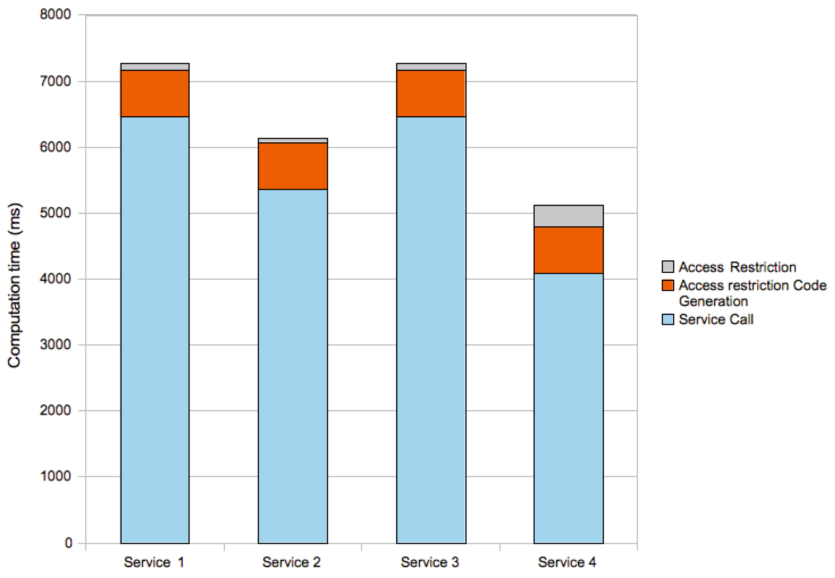


Fig. 6. Overhead Entailed by our approach

The proxy generation time is stable. It is caused by the parsing of the description of the unsecured service (such as a WSDL file) and the generation of the proxy with the JET template. The generation only happens once when the service registers to the service registry.

This analysis shows that access restriction enforcement takes at least 1% and at most 8% of the execution time of a service secured by our method. This time encompasses the retrieval of contextual data and, the processing of an access restriction decision and its enforcement. In formal terms, verifying a rule entails a small cost of $O(|C| * |\phi|)$ where $|C|$ is the size of the current achieved composition and $|\phi|$ the access restriction rule's size, *i.e.* the number of literals and operators in a rule. As a result, we can expect access restriction enforcement time to remain small throughout the execution of a secured composition.

6 Related Works

In [4] [14], the authors propose to specify security properties such as audit or encryption at design time and enforce them at runtime. In comparison, we focus on modeling and enforcing private data management as access restriction.

Privacy-oriented languages, such as the Enterprise Privacy Authorization Language (EPAL) and the Platform for Privacy Preferences (P3P) use access control concepts. Most access control models rely on the notions of *Principals*, *Categories*, *Actions* and *Permissions* [1]. *Principals* gather users, software agents and resources. They may belong to several *Categories*, that can be composed to refine them, and the *Permissions* are granted to these *Categories* for the performance of a set of *Actions*. The Role-Based Access Control (RBAC) model [7], for instance, categorizes *Users* according to *Roles* which express jobs or positions in an organisation. *Permissions* are attributed to *Roles*, which are stable *i.e.* organizational categories. The Attribute-Based Access Control model (ABAC) is another promising way of modeling access control. In order to obtain Rights, a User must exhibit a set of attributes with the correct values. This approach is suitable for pervasive access control.

However, the ABAC policies are not as readable as the RBAC ones as roles clearly architecture the policies. Moreover, EPAL is only a proposition and is not widely supported. P3P has been abandoned due to a lack of support and accessibility by non-technical users. A usable privacy-oriented language is thus yet to come.

Languages, such as the eXtensible Access Control Markup Language (XACML) or the WS-Policy permit to express access control. However, they are specific to a type of service implementation, Web Services for application integration. Several works use these languages to model access control at process level. [17] annotate process specifications with access control constraints from which they generate access control code. However, [17] proposes to generate XACML access code without addressing its execution when access control is known to influence the architecture of an application [13]: XACML, for instance, relies on a dedicated architecture. In contrast, we implement specific components to maintain the data related to access restriction enforcement.

Many works which focus on securing an executable application or service composition [3] [15] [10] [2] are dedicated to a service implementation or make strong assumptions on the access control capabilities or the availability of services. Heterogeneity is then still a brake to the development of service compositions secured by access control. The UPnP standard, for instance, defines no access control mechanism for UPnP aware devices. The dynamism of services is another challenge. Several works extend the Business Process Execution Language (BPEL) with access control features [12] [8]. Thus, they suppose they already know the service to invoke. It is not necessarily the case in a pervasive environment.

In contrast, we promote a platform-independent specification of access control and service composition. High level concepts can be easily grasped by non technical people. We automatically transform this specification into an executable secured process at runtime, according to the available service. We also investigate in depth the impact of access control on the composition's architecture by building components dedicated to access control enforcement.

7 Conclusion

In this paper, we have addressed the issue of designing and executing privacy-aware service compositions for pervasive applications. We have introduced a model-driven approach to the production of such compositions. We have understood privacy-awareness as private data management throughout the composition. We have provided a high-level language to privatize data, *i.e.* to express this management as access restriction. The validation on real world services shows that access restriction can be captured at design time in an abstract way. At runtime, the extra computation time entailed by access restriction enforcement remains reasonable.

Late code generation addresses the heterogeneity and the dynamism of actual services. Our experience shows that metamodeling is a demanding task. Identifying the necessary concepts to specify access restriction, for instance, is long and cumbersome. The same can be said of the creation of templates for each target service technology. However, the benefits of our approach overstep these difficulties. Metamodeling and building up templates allow knowledge capitalization. It also permits to obtain generic specifications, what is important in a highly heterogeneous envrionment. Focusing on non-technical concepts permits to integrate a wide range of stakeholders to an application's lifecycle and to build up early a coherent and extensive access restriction policy.

Finally, our approach calls for several future works. First, in term of access restriction, we have posited that a single designer designed the entire policy. For legal reasons, we may need data subjects to express their privacy preferences. As a result, we are currently exploring the distributed administration of the privacy policy. Second, our work is going to be validated in the frame of the INNOSERV project by the French research agency. Then, the spirit of our approach can be applied to other non-functional properties. When the metamodels of non-functional properties do not overlap with the composition metamodel, the adequacy of our proposition remains. Our future works will be dedicated to adding extra non-functional properties to service compositions.

References

1. Barker, S.: The next 700 access control models or a unifying meta-model? In: Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, SACMAT 2009, pp. 187–196. ACM, New York (2009)

2. Basin, D., Doser, J., Lodderstedt, T.: Model Driven Security: from UML Models to Access Control Infrastructures. ACM Transactions on Software Engineering and Methodology 15, 39–91 (2006)
3. Carminati, B., Ferrari, E., Hung, P.: Security Conscious Web Service Composition. In: International Conference on Web Services (ICWS), pp. 489–496. IEEE Computer Society, Los Alamitos (2006)
4. Chollet, S., Lalanda, P.: Security specifcation at process level. In: SCC 2008: Proceedings of the 2008 IEEE International Conference on Services Computing, pp. 165–172. IEEE Computer Society, Washington, DC (2008)
5. Dami, S., Estublier, J., Amiour, M.: APEL: A Graphical Yet Executable Formalism for Process Modeling. Automated Software Engg. 5(1), 61–96 (1998)
6. Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 995–1072. MIT Press (1990)
7. Ferraiolo, D.F., Kuhn, D.R.: Role-based access controls. In: Proceedings of the 15th National Computer Security Conference, pp. 554–563 (1992)
8. Garcia, D.Z.G., de Toledo, M.B.F.: Ontology-based security policies for supporting the management of web service business processes. In: ICSC, pp. 331–338 (2008)
9. Laroussinie, F., Schnoebelen, P.: Specification in ctl + past for verification in ctl. Inf. Comput. 156, 236–263 (2000)
10. Orriëns, B., Yang, J., Papazoglou, M.P.: Model Driven Service Composition. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 75–90. Springer, Heidelberg (2003)
11. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, pp. 46–57. IEEE Computer Society, Washington, DC (1977)
12. Rodríguez, A., Fernández-Medina, E., Piattini, M.: A BPMN Extension for the Modeling of Security Requirements in Business Processes. IEICE - Transactions on Information and Systems E90-D(4), 745–752 (2007)
13. Samarati, P., de Capitani di Vimercati, S.: Access Control: Policies, Models, and Mechanisms. In: Focardi, R., Gorrieri, R. (eds.) FOSAD 2000. LNCS, vol. 2171, pp. 137–196. Springer, Heidelberg (2001)
14. Souza, A.R.R., Silva, B.L.B., Lins, F.A.A., Damasceno, J.C., Rosa, N.S., Maciel, P.R.M., Medeiros, R.W.A., Stephenson, B., Motahari-Nezhad, H.R., Li, J., Northfleet, C.: Incorporating Security Requirements into Service Composition: From Modelling to Execution. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 373–388. Springer, Heidelberg (2009)
15. Srivatsa, M., Iyengar, A., Mikalsen, T.A., Rouvellou, I., Yin, J.: An Access Control System for Web Service Compositions. In: International Conference on Web Services (ICWS), pp. 1–8. IEEE Computer Society, Los Alamitos (2007)
16. Vallecillo, A.: On the Combination of Domain Specific Modeling Languages. In: Kühne, T., Selic, B., Gervais, M.-P., Terrier, F. (eds.) ECMFA 2010. LNCS, vol. 6138, pp. 305–320. Springer, Heidelberg (2010)
17. Wolter, C., Schaad, A., Meinel, C.: Deriving XACML Policies from Business Process Models. In: Weske, M., Hacid, M.-S., Godart, C. (eds.) WISE Workshops 2007. LNCS, vol. 4832, pp. 142–153. Springer, Heidelberg (2007)