Policy expression and checking in XACML, WS-Policies, and the jABC

Martin Karusseit
Chair of Programming Systems
TU Dortmund, Germany
martin.karusseit@cs.uni-dortmund.de

Tiziana Margaria
Chair of Service and Software Engineering
Universität Potsdam
Germany
margaria@cs.uni-potsdam.de

Holger Willebrandt
Chair of Programming Systems
TU Dortmund, Germany
holger.willebrandt@cs.uni-dortmund.de

ABSTRACT

Web-based access to sensitive and confidential data is realized today via different approaches, using a variety of methods to specify and combine access control policies. In an optic of change management and evolution, a structured and flexible model is needed to handle dynamicity, particularly when handling rights in systems with many users which hold different roles. Furthermore the validation of security constraints is an important key to warrant the reliability of control mechanisms.

This paper compares the temporal logic-based approach for modeling access control used by the jABC framework with two popular XML-based description languages (XACML and WS-Policy), which are quasi-standards for policy expression in Web applications. Its usage is illustrated here on the example of the web-based Online Conference Service (OCS). The respective functionalities are described and examined in consideration of their ability to validate and enforce the needed policies.

Keywords

access control, policies, validation, model checking

1. INTRODUCTION

Services that provide sensitive and confidential data and resources to different users with dissimilar roles and access permissions need to care about a substantiated framework to control access to such items. Especially with regard to web-based services which need to deal with multiple parallel requests from different users or to aggregate data from other services, the requirements for access control frameworks rise in complexity. For users utilizing these services, privacy is not only a must but taken for granted. This privacy implies compartmenting the users' sessions and data in the service to safely isolate and protect data and resources.

The key to the ability of sophisticated access control is the capability of defining and expressing access control policies.

TAV-WEB – Workshop on Testing, Analysis and Verification of Web Software, July 21, 2008

With the complexity of a service which needs access control, the complexity of the policies that imply the rules increase alike. Administrators and service developers need powerful but yet manageable frameworks that help in creating access control policies. Policies must be refinable as well as composable to deal with recurrent changes to workflows and to reflect the compositional character of today's web services.

Reliability of access control mechanisms is the crucial factor in web service security. Therefore, the specification of a policy can only be one step towards securing a service. Additionally, it must be assured that the policies are actually met by the real service implementation. Therefore, a methodology founded on formal methods that ensures the coherence of model and implementation is essential.

We compare here the popular XML-based policy definition languages XACML and WS-Policy (Sect. 2, with the temporal logic-based approach for modeling access control supported by the jABC [**?**], an application development framework for model-driven, service-oriented design. We illustrate jABC's facilities for policy expression and checking (Sect. 3.1) on the example of the *Online Conference Service (OCS)*[6, 7]. In particular we investigate how to specify, apply, and verify policies in a dynamic and process-oriented way and compare them with the previous two (Sect. 4).

2. XACML AND WS-POLICY

2.1 XACML

The *eXtensible Access Control Markup Language (XACML)* [1] is an OASIS[14] standard approved in 2003. It was designed to specify access control policies and assertions in an XML-based style. It is used to define constraints that must be matched in order to gain access to a service resource. These constraints are expressed as combinations of attributes and data types which are to be checked by the help of generic functions. Furthermore, *Policy Sets* can be formed by using combinational statements (all, at least one, ...) that wrap simple policies.

XACML targets web-based systems that need to control access to service resources for incoming HTTP requests. Figure 1 (from [2]) shows how the components that build the XACML access control model interact when a request arrives. An incoming request is first transformed to an XACML request that encapsulates information about the requester, its credentials and the requested resource. This transformation is done by the *Policy Enforcement Point*(PEP), the entity that ensures that only requests that respect de-

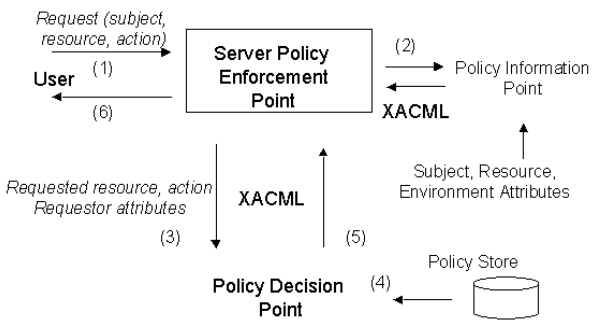

Figure 1: Request processing in XACML (from [2])

fined security constraints are passed to the system. The PEP communicates with the *Policy Information Point* (PIP), which holds information about mapping HTTP request attributes and sender information to resource and environment identifiers conform to the XACML schema.

The information returned by the PIP is incorporated into the XACML request, then passed to the policy evaluating entity, the *Policy Decision Point*(PDP). This decides whether a request should be accepted or denied. To evaluate the XACML request the PDP needs to select applicable policies from a policy store. If appropriate policies are found, they are evaluated and the PEP delivers the authorization decision.

As it is possible that multiple policies can be applicable to one request, the system uses combining algorithms to deduce a single decision. The combining algorithms also will be applied when a Policy Set needs to be evaluated.

The root element of a XACML document is one of the elements `<Rule>`, `<Policy>` and `<PolicySet>`. The rule element "*consists of a boolean expression that can be evaluated in isolation*"[1]. A rule by itself does not form a valid policy definition and is not meant to be accessed directly by a PDP: it has to be part of a Policy. A Policy combines a set of rules, together with a named procedure that depicts how the rules should be combined. Policies themselves can be accommodated into a Policy Set, consisting of multiple `<Policy>` or `<PolicySet>` elements. The combining algorithms can also be specified for the contents of the `<PolicySet>` element.

Valid combining algorithms for rules and policies are

- **Deny-overrides** If any rule or policy evaluates to `Deny` the whole Policy / Policy Set evaluates to `Deny`
- **Permit-overrides** If any rule or policy evaluates to `Permit` the whole Policy / Policy Set yields `Permit`
- **First applicable** The first applicable rule or policy encountered determines the evaluation result.
- **Only-one-applicable** This combining algorithm can be used to force the evaluation to be based exactly one policy of a policy set. If exactly one policy is applicable to the request, it is evaluated, yielding the decision result. If multiple policies are applicable, the result will be `Indeterminate`. If no policy is applicable, it returns `NotApplicable`. This combining algorithm can only be used on policy sets and is not applicable to force the use of one rule.

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
03   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:
schema:os
05   http://docs.oasis-open.org/xacml/access_control-xacml-2.0-
policy-schema-os.xsd"
06   PolicyId="urn:oasis:names:tc:example:SimplePolicy1"
07   RuleCombiningAlgId="identifier:rule-combining-algorithm:
deny-overrides">
08   <Description>
09     Medi Corp access control policy
10   </Description>
11   <Target/>
12   <Rule RuleId= "urn:oasis:names:tc:xacml:2.0:example:Simple
Rule1" Effect="Permit">
13     <Description>
14       Any subject with an e-mail name in the med.example.com
 domain
15       can perform any action on any resource.
16     </Description>
17     <Target>
18       <Subjects>
19         <Subject>
20           <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.
0:function:rfc822Name-match">
21             <AttributeValue DataType="http://www.w3.org/2001
/XMLSchema#string">
22                 med.example.com
23             </AttributeValue>
24             <SubjectAttributeDesignator
25              AttributeId="urn:oasis:names:tc:xacml:1.0:
subject:subject-id"
26              DataType="urn:oasis:names:tc:xacml:1.0:
data-type:rfc822Name"/>
27           </SubjectMatch>
28         </Subject>
29       </Subjects>
30     </Target>
31   </Rule>
32 </Policy>
```

Figure 2: Example XACML policy (from [1])

Applicability of a rule or a policy depends on a `<Target>` element: it specifies (as a predicate) to which subjects, resources, or environments the rule or policy applies. The target is the first item evaluated when a policy is processed. If the target matches the request, the evaluation of that policy is continued. Otherwise, the next policy's target is checked.

Fig. 2 (taken from [1]) shows an example policy. It has a single rule (lines 12–31), whose meaning (semantics) is described verbally inside the `<Description>` Element (lines 13–16): "*Any subject with an e-mail name in the med.example.com domain can perform any action on any resource*". The empty `<Target>` element in line 11 says that this policy applies to any resource.

2.2 WS-Policy

The Web Services Policy Framework (WS-Policy) defines the general assembly of policies based on an XML syntax for specifying communicating web services' requirements and capabilities. These are described as conjunctive and disjunctive combinations of assertions on either the web service's own communication abilities or on the requirements a web service's communication partner has to meet in order to interact with it. They are therefore *propositional* in the sense of propositional logics. The general purpose of the WS-Policy language is to express domain independent service policies which are to be connected to domain specific contexts via extensions of the WS-Policy framework (e.g. WS-SecurityPolicy [15] for assertions on secure communication, WS-PolicyAssertions [16] for assertions on web services' text

```
01 <wsp:Policy xmlns:sp= "http://schemas.xmlsoap.org/ws/2005/07/
securitypolicy"
02   xmlns:wsp= "http://schemas.xmlsoap.org/ws/2004/09/policy" >
03   <sp:TransportBinding>
04     <wsp:Policy>
05       <sp:AlgorithmSuite>
06         <wsp:Policy>
07           <wsp:ExactlyOne>
08             <sp:Basic256Rsa15 />
09             <sp:TripleDesRsa15 />
10           </wsp:ExactlyOne>
11         </wsp:Policy>
12       </sp:AlgorithmSuite>
13       <sp:TransportToken>
14         <wsp:Policy>
15           <sp:HttpsToken RequireClientCertificate= "true " />
16         </wsp:Policy>
17       </sp:TransportToken>
18       <!-- Details omitted for readability -->
19     </wsp:Policy>
20   </sp:TransportBinding>
21 </wsp:Policy>
```

Figure 3: Example WS-Policy policy (from [3])

encoding, language support, etc. or WS-MetadataExchange [17] to incorporate web service metadata). An example for the use of WS-Policy could concern two web services that need to negotiate which encryption mechanism should be applied to their communication. Here, one of them might want to force encryption with a minimum strength and deny any non-encrypted service calls, whereas the other might only offer a limited range of encryption algorithms. With the use of WS-Policy these requirements and capabilities can be expressed and used to infer a (possiblx empty) set of applicable communication parameters.

The individual properties of the behaviour (capabilities and requirements) of a web service are defined by the Policy Assertions. They "*indicate domain-specific [...] semantics which are expected to be defined in separate [...] specifications*" [3]. The assertions are therefore subject of the specific extensions to the WS-Policy framework used. Multiple policy assertions are grouped into Policy Alternatives, formed by surrounding the assertions with the `<wsp:All>` or `<wsp:ExactlyOne>` Element. `<wsp:All>` expresses that all assertions included in the policy alternative must be satisfied, whereas `<wsp:ExactlyOne>` requires only one assertion to be satisfied. The policy alternatives themselves are mutually exclusive and can be expressed in a disjunctive normal form, called *Normal Form Policy Expression*. The *Compact Policy Expression* is an equivalent way to express policies, it may be converted for evaluation to the corresponding normal form. Furthermore, policy expressions are associated via *Policy Attachments* to *Policy Scopes*, which are basically a collection of *Policy Subjects*. The subjects are entities affected by the policy, like message endpoints or resources.

Fig. 3 (adapted from [3]) illustrates a policy in compact form which uses the WS-SecurityPolicy extension.

2.3 Comparing XACML and WS-Policy

Both XACML and WS-Policy define languages to express policies. While XACML concentrates on access control policies for service resources, WS-Policy provides a more general approach which allows one to define policies by the use of separate extension modules that add relevant meaning. Both allow composing complex policies from simpler ones, to build up a comprehensive rule set.

The two languages need a domain-specific vocabulary to form the semantics and represent the predicates of a specific use case. But how the expression of policies depends on the application domain differs significantly. While XACML allows the policies to be in a standard form and does not require domain-specific extensions to express constraints, WS-Policy depends on the use of specialized modules to build policies for a concrete domain. In XACML, the policies are connected to a given domain by generic functions that operate on the attributes of a request, and therefore depend only on the semantics of the involved data types. In contrast, WS-Policy uses specialized assertions for every domain, that are defined in separate specifications. Therefore it is much more difficult for an implementation of WS-Policy to decide whether two policies are comparable, equivalent, contradictory, or whether one proposition implies another.

In terms of *expressive power*, which is the central issue in this paper, both languages support *local* policies: they refer to what holds or not at the given point in a process where they are invoked or checked.

The *global character* of temporal logics, that have as primitives temporal operators (like next, until, before, after) ranging over sequences of states, is here not part of the language definition, which in both cases is close to a first order logic. Recent work on static verification of XACML (as reported e.g. in [5], containing also a survey) makes it explicit: the policies considered there are of local nature, thus they are well suited to being treated by boolean verifiers like e.g. SAT solvers. Any temporal requirement, however, would have to be expressed and dealt with outside XACML and WS-Policy.

3. POLICIES AND ACCESS CONTROL NEEDS IN OCS

The Online Conference Service (OCS) is a web-based collaborative decision support service that proactively helps authors, program committee chairs, program committee members, and reviewers to cooperate efficiently during their collaborative handling of the composition of a conference program. The OCS has been successfully used since 2000 [8]. It is now hosted by Springer Verlag on their premises in Dordrecht (NL) to support events with LNCS proceedings and it is currently being tightly integrated with Springer's volume production process. A description of the service and of its method of development is available in [6, 7].

Two central characteristics of this application determine our extended requirements to a suitable policy description and enforcement system:

- OCS capabilities are not just static, but depend on the process, i.e. on the single execution trace and on context information.
- the OCS is customizable and flexibly reconfigurable online at any time for each role, for each conference, and for each user. This means that we need to be able to dynamically change the role and right definition of a running installation.

The central need that arises here is therefore the capability to express and check policies with a *global character*. This must be ensured by the development environment used to design and implement the OCS.

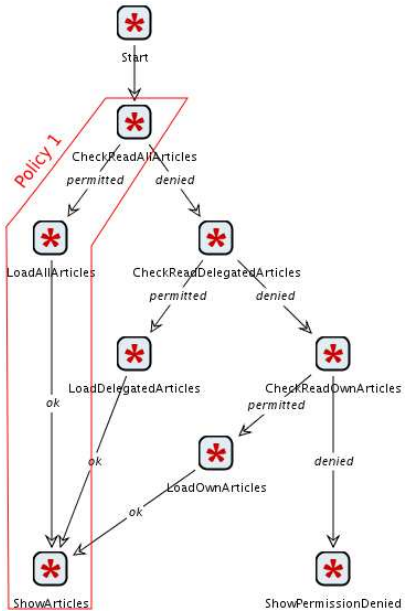

Figure 4: `ReadArticleList` Feature - Simplified SLG

Due to the needs of flexibility to support a rich variety of processes and of evolution, we develop the OCS in a model driven and service oriented way. To this aim we use the jABC [9, 10], our framework for modelling, analyzing, executing, deploying, and testing applications built in a service-oriented way [11, 12]. As for any jABC product, the OCS processes are defined in terms of hierarchical Service Logic Graphs (SLGs) (see Figure 4): SLGs express in a control flow-like style the coordination (orchestration) of functionalities that are themselves basic reusable services called Service Independent Building Blocks (SIBs). In jABC we design and develop complex applications like the OCS as follows:

- Model the service's business logic according to its functional and non functional requirements. This is done in terms of SLGs that orchestrate reusable services expressed as SIBs. Complex services are built hierarchically, a heavily used feature in the OCS.
- Model the (mostly process-dependent) policies and the interoperability and interference constraints between several (sub)services in terms of *temporal logic formulas*. Here, we use mainly CTL [4], for which the jABC offers a user-friendly pattern-based graphical editor (the FormulaBuilder [13]).
- Model check the designed SLGs wrt. the policies using these libraries of constraints: we can at any time ensure the compliance of the service logic to the policies.

In the following we illustrate how this is realized in jABC for the OCS.

3.1 Global OCS Policies

Policies in the OCS define the cooperation of SIBs/Services and their dependencies. For example the capability of reading submitted articles depends on the user's role, on conference-dependent customs, and on his/her personal conflict situation with single submissions. We consider here a simplified model of the business logic underlying the `ReadArticleList` feature, a subservice of the OCS and purely role-related policies [1] Simple role-specific access policies for the article list are as follows:

- **Policy 1**: Only users logged in as PC Chair can read all articles.
- **Policy 2**: Users logged in as PC Member are only permitted to read articles delegated to them.
- **Policy 3**: Users logged in as Author are only permitted to read their own articles.

Figure 4 shows the simplified workflow for accessing the article list in the OCS. It distinguishes three ways to access the SIB `ShowArticles`, which dynamically displays the articles loaded either by the SIB `LoadAllArticles`, or `LoadDelegatedArticles`, or `LoadOwnArticles` (which in reality include more complex subprocesses). Once the user has requested in the online service the `ReadArticleList` service, the decision which articles must be loaded and shown in the next webpage is delegated to the three Check SIBs. They examine the permissions of this user, who may be associated in general to several roles, and evaluate the context information, which tells for instance that this user, who is both a reviewer and an author, is currently logged in as reviewer. A role is defined as a collections of rights, whereby a right specifies the permission to access an object or a service.

If the user has the right to see all the articles, the SIB `CheckReadAllArticles` evaluates positively, thus the outgoing branch labelled `permitted` is selected and the service logic leads to the execution of `LoadAllArticles` as next SIB. Otherwise, the branch `denied` is followed, leading to the next Check SIB in the workflow. These Check SIBs implement in a reusable and modular way the role access control mechanism used in the OCS and will be explained later on.

3.2 Model Checking the Policies

An advantage of model driven development is the ability of checking policy compliance already at design time, on the models. In this case, since we deal with intellectual property-sensitive data (unpublished scientific papers), this is of particular importance, for trust and non-disclosure issues. In jABC, we express graphically the policies as collections of (temporal) logic constraints using the FormulaBuilder plugin [13] and automatically check them wrt. the SLG by means of the GEAR [18] model checker.
In the following we will use Computation Tree Logic (CTL) [4] to express desired system properties.

Definition 1. **Syntax of CTL**
For $p \in \mathsf{AP}$, the set of CTL formulas is defined by:

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \mathsf{EX}[\phi] \mid \mathsf{E}[\phi\ \mathsf{U}\ \phi] \mid \mathsf{A}[\phi\ \mathsf{U}\ \phi]$$

The semantics of CTL is defined using paths in a Kripke Transition System (KTS), a transition system with action labels on the edges and atomic propositions on the nodes. These paths represent the possible execution traces of the system.

[1] More elaborate policies that consider context and personal overrides are also defined and enforced similarly, but require a detailed model of the business logic that we cannot disclose here. We will show corresponding examples at the workshop.

Definition 2. **Path**
A path is a sequence of states $s_0, s_1, s_2, \cdots$, such that $(s_i, a, s_{i+1}) \in \rightarrow$ for some $a \in \mathsf{Act}$. S^ω denotes the set of all paths. We refer to the $(i+1)$-th state of a path $\pi \in S^\omega$ as $\pi[i]$.

$P_K(s) = \{\pi \in S^\omega | \pi[0] = s\}$ denotes the set of paths starting from state s in the Kripke Transition System K.

Definition 3. **Semantics of CTL**
Let $K = (S, \mathsf{Act}, \rightarrow, I)$ be a Kripke Transition System over atomic propositions AP, $p \in \mathsf{AP}$ be an atomic proposition, $s \in S$ be a state, and ϕ, ψ be CTL formulas. The satisfaction relation $\models$ is defined by:

$s \models p$	iff $p \in I(s)$
$s \models \neg\phi$	iff $s \models \phi$ does not hold
$s \models \phi \vee \psi$	iff $s \models \phi$ or $s \models \psi$
$s \models \mathsf{EX}[\phi]$	iff $\exists\ \pi \in P_K(s).\pi[1] \models \phi$
$s \models \mathsf{E}[\phi\ \mathsf{U}\ \psi]$	iff $\exists\ \pi \in P_K(s).\exists\ j \geq 0.\pi[j] \models \psi \wedge$ $\forall\ 0 \leq k < j.\pi[k] \models \phi$
$s \models \mathsf{A}[\phi\ \mathsf{U}\ \psi]$	iff $\forall\ \pi \in P_K(s).\exists\ j \geq 0.\pi[j] \models \psi \wedge$ $\forall\ 0 \leq k < j.\pi[k] \models \phi$

As usual, we can derive the following (dual) operators.

- $\phi \Rightarrow \psi \equiv \neg\phi \vee \psi$ and $\phi \wedge \psi \equiv \neg(\neg\phi \vee \neg\psi)$
- $\mathsf{AX}[\phi] \equiv \neg\mathsf{EX}[\neg\phi]$
- $\mathsf{EF}[\phi] \equiv \mathsf{E}[\text{true}\ \mathsf{U}\ \phi]$ and $\mathsf{AF}[\phi] \equiv \mathsf{A}[\text{true}\ \mathsf{U}\ \phi]$
- $\mathsf{EG}[\phi] \equiv \neg\mathsf{AF}[\neg\phi]$ and $\mathsf{AG}[\phi] \equiv \neg\mathsf{EF}[\neg\phi]$

Structurally, CTL operators consist of a *path quantifier* that specifies whether we are interested in at least one possible execution trace (E) or every possible execution trace (A), and a *path formula* describing the behavior along paths.

- X (next-time) talks about the next state
- F (finally) talks about a state that is reached eventually
- G (generally) talks about the entire path
- $\phi\ \mathsf{U}\ \psi$ (until) requires ϕ to hold until ψ holds at some future state, or ψ may also hold immediately

We interpret SLGs as Kripke Transition Systems where a SLG's SIBs are the nodes and its branches are the edges of the KTS. The atomic propositions in the formulas can reflect the existence of a SIB itself, the values of SIB parameters, and the accessibility of SIBs/Services via branches.

Policy 1 defines abstractly that only users logged in with role PC Chair have read access to all articles. On the basis of the current service collection, the SIB `CheckReadAllArticles` that checks if the current user has the corresponding right should always precede the execution of the SIB `LoadAllArticles`. We model this graphically with the FormulaBuilder, yielding the formula graph shown in Figure 5.

The DIAB temporal operator (read *diamond backward*) acts like the EX operator with the transition arrow reversed. From now we name this behavior EXB, specifying that at least one predecessor SIB exists with that name, that is reachable backwards over a branch with that label.

The shown graph is translated in CTL syntax:

```
(('ShowArticles => EXB[ok] 'LoadAllArticles)
   /\ ('LoadAllArticles => EXB[permitted]
   'CheckReadAllArticles))
```

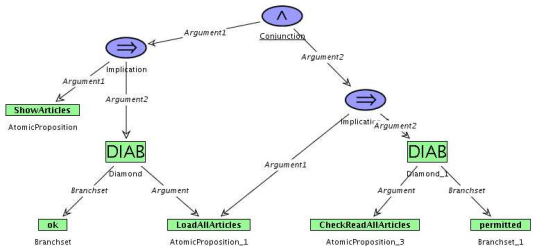

Figure 5: Policy 1 (PC Chair access) in jABC

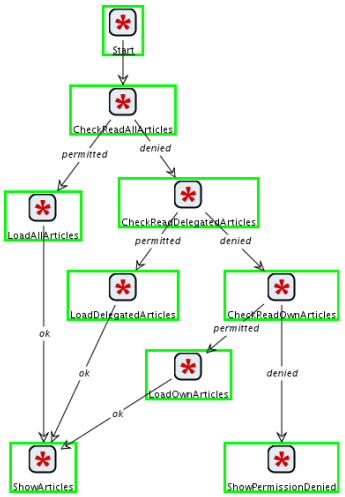

Figure 6: The `ReadArticleList` Feature is Policy 1 compliant

This constraint is translated internally in μ-calculus for the GEAR model checker.

Ensuring that the SLG complies to this policy amounts now to model checking the corresponding SLG wrt. this property. We see in Figure 6 that the property is satisfied by the entire SLG. All SIBs satisfying the property are highlighted with a (green) rectangle by our model checker, thus the SLG is compliant to **Policy 1**.

In our experience, SLGs and policies evolve separately. It is quite frequent that concrete features modify their SLGs, mostly due to the introduction of additional cases or to refinement of the workflow. Policies, on the contrary, have proven to be much more stable. In Figure 7 we show a new version of the SLG where the branch `ok` of the SIB `LoadAllArticles` has been redirected to SIB `LoadDelegatedArticles`. This could be an intermediate step in an evolution of the workflow. Checking this version of the SLG we see that a violating node is detected and highlighted with a (red) rectangle by the model checker. Additionally we have emphasized this node using a circle. This means that at least a path originating in that node now violates **Policy 1**.

3.3 Basic Access Control in the OCS

In the OCS we use an own concept of role-based management of user rights, as described in [6]. A role is a collection of rights, which gives users the permission to access an object, feature, or service. Features are seen as a collection of

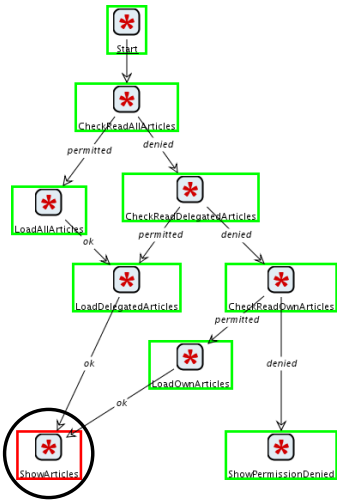

Figure 7: Violation of Policy 1 in the modified `ReadArticleList` Feature

functionalities of a service. In the OCS, service functionalities have unique names, with naming scheme

`F-<FeatureCategory>-<SubfeatureID>.<Filter>`

- The `FeatureCategory` is the name of a service, implemented as an own SLG in the main service.
- The `SubfeatureID` specifies a subservice of a service. Subservices can either have an own SLG within the service, or be SIBs.
- The `Filter` suffix is optional and allows steering the fine granular right management: it restricts the runtime access to capabilities of the business objects.

As an example, the OCS FeatureCategory `ART` contains all the specific rights to access and work with articles. Sect. 3.1 already presented some policies for reading articles. The concrete permission for these policies is `F-ART-05`. This permission says that the subservice that provides access to the content of a submission can be executed, but it does not specify on which articles. This is realized through filters, which define a finer granular access to the articles. The permission `F-ART-05.all` gives read access for all articles and is equivalent to the scope of **Policy 1**, `F-ART-05.delegated` specifies the permission to read only articles delegated to that user, as defined in **Policy 2**. `F-ART.05.own` is equivalent to **Policy 3** and allows reading of own articles.

The set of permissions compactly describes the space of possible actions on objects, features, or services. At runtime the access to service functionalities depends on

- the user's role, which defines the allowed permissions.
- the user's specific permissions and restrictions. It is possible to assign additional permissions to a user, which are not included in his role. Also the specification of restrictions is supported.
- the object's permissions, which specify the needed access rights depending on the object's state.
- the conference's restrictions, which depend on the conference deadlines. Once a deadline is expired, e.g. article submission deadline, the corresponding right to submit articles is revoked.

All the elements just listed are highly dynamic, because they can be modified automatically (by the service, due to a expired deadline), or manually (by the conference leader) at any time. Because of this dynamics, authorization is computed at every request in the OCS as follows:

1. Calculate the current user's permissions $Perm$, which include role, object, and user's permissions and restrictions, and the conference's restrictions:
$$Perm = ((role_{perm} \cup user_{perm}) \cap Object_{perm}) \setminus user_{restr} \setminus conference_{restr}$$
2. Check if this set contains the necessary access permissions for the object:
$$Access_{perm} \subseteq Perm$$
3. If this is true, the user is allowed to perform the action on the object.

This computation is done by our access control engine [6] during runtime. The engine takes into account all specific permissions and restrictions, and is responsible for active control of varying user's rights.

As one sees, in order to be efficient at runtime we must have permission sets of static nature: any process-oriented requirement, like the precedence constraint we saw in **Policy 1** must be ensured once and forever on the SLG. Maintaining permission sets that include histories would be computationally awkward and extremely difficult to evolve and maintain. Therefore, for the OCS and similar applications the capability of expressing and checking policies that include *temporal constraints* is of central importance.

4. COMPARISON WITH OUR APPROACH

As described in Sect. 2.3, both XML-based languages allow combining simple policies to complex policy expressions by means of simple operands. However, these complex expressions are of propositional (conjunctive or disjunctive) nature. Temporal operands are not foreseen, therefore pure XACML or WS-Policy-like approaches are insufficient for our needs, since they cover only the description of the permission sets.

Conversely, policies specified in XACML or WS-Policy can easily be modeled as a (predicate logic) graph with our FormulaBuilder. Because of the hierarchical structure of the formula graphs (they are themselves just specialized SLGs) it would be simple to reuse and combine several policies to one complex propositional constraint. The tool based editing enables users in fact to maintain and understand even very large rule sets.

Another central point for the successful usage of policies is the close association between the specified policies and the application they were designed for. The WS-Policy specification does not state how the policy should be applied to the service, leaving this to a different aspect of the global design. In contrast XACML's Policy Enforcement Point ensures at runtime that policies are used to secure the requests. As described in section 3.3 we have an own role access control

mechanism and engine, which supports the link to the policy compliance validation. Here we miss the validation aspect in the original XACML approach.

We can also specify the communication between services in WS-Policy style. Our advantage is that we can graphically describe the dependencies between services and their service parameters. Thus permissions can be made dependent on the state of a service and this relationship can be automatically validated by the model checker.

Neither XACML nor WS-Policy support or provide state dependent policies or mechanisms for the validation of policies. We know by experience that the complexity of policies comes along with the increasing size of the service's functionalities. It is very difficult to manually maintain very large rule sets, keeping them consistent and aligned. Especially the editing of huge XML files represents an enormous challenge for humans. An XML-based approach to express access policies does not imply that the execution of these policies automatically secures a service: there is no guarantee that the policies themselves are correct or that they cover all application aspects. Of course it is possible to test a service manually, but this is a time consuming and error prone practice, especially when policies overlap or depend on dynamic and context conditions.

Ensuring that the whole application/service is covered by all policies requires an automatic validation mechanism at design time. From our point of view this mechanism should be based on formal methods, since services like the OCS require high compliance standards. XACML and WS-Policy have broad specifications and guidelines on how to define and use local policies. Approaches as in [5] however require first complex translations and encodings of the policies, to make them amenable to model checking.

From our point of view it is essential to have an *integrated solution* for application development which provides the possibility to continuously check service policies natively, from the very beginning, so that the policies can be incrementally refined and curated along the development of the service, this way adequately supporting change management and evolution. The current standard proposals are therefore still insufficient for our needs.

5. CONCLUSIONS

Access to sensible data via complex online services requires an adequate mechanism to define, verify, and enact policy compliance, that ensures the trustability of a service.

We described and compared the XML-based description languages XACML and WS-Policy with the constraint and graph-based approach for modeling access control as used by the web-based Online Conference Service (OCS). In an optic of change management and evolution, a structured and flexible policy model is needed to handle dynamicity, particularly when handling rights in systems with many users which hold different roles.

While XACML and WS-Policy can be sufficient to describe local role/rights models, adequate to express static policies, for the OCS we need to express process-oriented policies: such policies embed local policies in a temporal component, in the sense of linear time temporal logics or as in our case branching time temporal logics. This is elegantly covered by the jABC based capabilities, but not covered by XACML nor WS-Policy, which are purely propositional.

Furthermore, the model based validation of policy constraints via model checking is an important key to warrant the reliability of service control mechanisms. This enables the continuous compliance checking along the incremental refinement and evolution along the entire lifetime of the service, this way adequately supporting he alignment between a service and its policy set. This is for the moment provided in the jABC, but not by XACML and WS- Policy, which are therefore still insufficient for our needs.

6. REFERENCES

[1] T. Moses (Ed.): *eXtensible Access Control Markup Language (XACML) Version 2.0*, Feb. 2005 http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf,

[2] P. Griffin: *Introduction to XACML*, February 2004 http://dev2dev.bea.com/pub/a/2004/02/xacml.html

[3] B. Siddharth et al.: *Web Services Policy 1.2 - Framework (WS-Policy)*, April 2006, http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425/

[4] E. A. Emerson, C. S. Jutla, A. P. Sistla. On model-checking for fragments of μ-calculus. 1993.

[5] G. Hughes, T. Bultan: Automated Verification of XACML Policies Using a SAT Solver. WQVV 2007, Worksh. on Web Quality, Verification and Validation, at 7th ICWE, Como (I), July 2007, Worksh. Proc. pp. 378-392.

[6] M. Karusseit, T. Margaria: *A Web-based Runtime-Reconfigurable Role Management Service* Proc. WWV'06, 2nd Int. Worksh. on Automated Specification and Verification of Web Sites, Cyprus, 2006, IEEE Press, pp.53-60.

[7] M. Karusseit, T. Margaria: *Feature-based Modelling of a Complex, Online-Reconfigurable Decision Support Service* Proc. WWV'05, 1st Int. Worksh. on Automated Specification and Verification of Web Sites, Valencia (E), March 2005, ENTCS N. 1132.

[8] T. Margaria, M. Karusseit: *Community Usage of the Online Conference Service: an Experience Report from three CS Conferences*, 2nd IFIP Conf. on "e-commerce, e-business, e-government" (I3E 2002), Lisboa (P), Oct. 2002, Kluwer, pp.497-511.

[9] *jABC: JavaABC Framework* `http://www.jABC.de`

[10] B. Steffen, T. Margaria, R. Nagel, S. Jörges, C. Kubczak: *Model-Driven Development with the jABC*, Proc. HVC'06, IBM Haifa Verification Conf., Haifa (IL), LNCS 4383, Springer Verlag, 2006.

[11] T. Margaria, B. Steffen: *Service Engineering: Linking Business and IT*, Computer, IEEE Computer Society, October 2006, pp.45-55

[12] B. Steffen, P. Narayan: *Full Life-Cycle Support for End-to-End Processes*, Computer, IEEE Computer Society, November 2007, pp.64-73

[13] S. Jörges, T. Margaria, B. Steffen: *FormulaBuilder: A Tool for Graph-based Modelling and Generation of Formulae*, ICSE 2006, Shanghai, China, ACM Press, pp. 815-818.

[14] *OASIS* http://www.oasis-open.org/home/index.php

[15] Della-Libera et al.: *Web Services Security Policy Language (WS-SecurityPolicy) Version 1.1*, July 2005 http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf

[16] D. Box et al.: *Web Services Policy Assertions Language (WS-PolicyAssertions) Version 1.1*, May 2003 http://xml.coverpages.org/ws-policyassertionsV11.pdf

[17] K.Ballinger et al.: *Web Services Metadata Exchange (WS-MetadataExchange) Version 1.1* August 2006 http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf,

[18] M. Bakera, T. Margaria, C. Renner, B. Steffen: *Game-based Model Checking for Reliable Autonomy*, SMC-IT'06, 2nd IEEE Int. Conf. on Space Mission Challenges for Information Technology, Workshop on Autonomous and Autonomic Systems, July 2006.