WS-Governance: a policy language for SOA Governance*

José Antonio Parejo, Pablo Fernandez, and Antonio Ruiz-Cortés

Universidad de Sevilla, Spain

Abstract. The widespread use of Service Oriented Architectures (SOA) is beginning to create problems derived from the governance of said structures. To date there is not a single effective solution to solve all existing challenges to govern this type of infrastructure. This paper describes the problems encountered when designing a SOA governance solution in a real e-Government scenario. More specifically, we focus on problems related to specification and automated analysis of government policies. We propose a novel SOA governance specification model as a solution to these problems. We have named this model WS-Governance. In order to ease its adoption by SOA practitioners it: i) shares WS-Policy guidelines and is compatible with it, ii) has XML serialization as well as a plain-text one and iii) has a CSP based semantics that provides a precise description as well as facilitating the automation of some editing and WS-Governance related activities such as consistency checking.

1 Introduction

SOA adoption brings an increase on the number of elements of the IT architecture, where proper management and control become capital issues. In this context, SOA Governance is defined as the management process aimed at delivering the SOA promise of reuse, business goals support and responsiveness [1,2]. According to [3] SOA Governance Lifecycle can be divided into six stages, from more abstract business levels to more concrete operational levels: Create a SOA strategy, Align Organization, Manage Service Portfolio, Control Service Lifecycle, Policy Definition and Enforcement and Service Level Management. In this paper we focus on the policy definition as the key stage that requires a deeper analysis in order to support an agile governance.

Effective governance requires policy management, including : (i) the definition of policies that encode governance rules and (ii) the establishment of appropriate conformance testing and enforcement mechanisms. Moreover, we have identified the need to incorporate the structure of the organization as essential information to take into account when the governance policies are designed. In our case study, the structure, size and departmental autonomy of the organization implies that multiple administrators could specify policies in a distributed and independent way, boosting the possibility of specifying inconsistent policies. In this context, the capability of automatic consistency checking of policies is highly valuable, and governance tools should support it. However, the current governance tools market is vendor-driven and turbulent, where tools are based on proprietary technology and their features are guided by the specific aspects where their vendors have expertise [4,5].

The contribution of this paper is twofold: (i) First, a language for governance policies definition is presented. This language defines governance documents; which make policies unambiguous by providing a rich context for governance policies and their meta-data. This is done whilst maintaining their definition independently of the SOA elements to govern; their internal organization and the underlying infrastructure. (ii) Secondly, a formal definition of governance document is proposed, describing the elements

* This work has been partially supported by the European Commission (FEDER) project SETI (TIN2009-07366), and project ISABEL P07-TIC-2533 funded by the Andalusian local Government.

to govern, their properties and the policies that govern them. This formal definition allows the automation of policies' consistency checking.To the best of our knowledge, this proposal provides a novel approach, paving the way for building more powerful and automated governance tools. This proposal has been developed and tested by creating two applications: a consistency analyzer and an on-line editor.

The rest of the paper is structured as follows: In sec. 2, the case study that motivates the research presented on this paper is described. Sec. 3 depicts the limitations of WS-Policy for governance policy specification and motivates the need of a governance document. Sec. 4 presents WS-Governance, our XML-based language proposal for governance policies specification, and its plain text equivalent WS-Gov4People. Sec. 5 presents a mapping of governance documents to Constraint Satisfaction Problems that allow the automated checking of consistency. Finally, in sections 6 and 7 related work is described and conclusions are drawn.

2 A motivating use case

The motivation of our approach is derived from a case-study based on a real scenario we addressed on a research project, involving a regional-wide governmental organization. This organization has a complex structure divided into 16 governmental departments and thousands of end users using a shared IT infrastructure. This infrastructure is distributed in the different departments both logically and physically and is usually managed autonomously in each location. In recent years there has been a shift toward SOA in the organization, and currently there is an important number of core services replicated in the infrastructure with different QoS capabilities.

From an architectural point of view, the infrastructure is designed as a federated bus of services; in this context, each department represents a node with two main elements: an Enterprise Service Bus and a Management System that provide different horizontal functionalities (such as monitoring, transactions or security). All the different nodes are integrated conforming the global infrastructure. The different services are deployed in the bus and the consumer applications ask the bus for the appropriate provider.

Due to the structure of the organization, each department has developed a high autonomy in its IT infrastructure management. Consequently, the integration of applications and services amongst different departments has raised an important issue: the need to specify a consistent normative framework on the whole organization for meeting business needs without breaching autonomy. Such a framework can be created by specifying a set of Governance Policies. A Governance Policy represents a capability, requirement or behavior that allows the SOA to achieve its goals, and whose meeting is quantifiable and monitorable through time [6,1,7,8]. Governance policies are as heterogeneous as the said governed elements, addressing the distributed, flexible, and heterogeneous nature of current SOAs. Moreover, governance policies originate from disparate sources, from legal regulations and their derived compliance issues to strictly technical details.

There is a real and urgent need of a language to define governance policies unambiguously with precise semantics; as a first step toward governance policy definition, enforcement and automatic consistency checking.

Figure 1 shows three excerpts of different real governance documents found in our case study -conveniently modified in order to preserve privacy and meet confidentiality clauses-. Currently, these fragments correspond to human-oriented policies that should be enforced by administrators by means of configurations of the IT-infrastructure. Each document is enacted by a different organization: the first document by the main authority so it should be enforced by all sub-organizations (departments); the second document represents an integration agreement amongst two departments (1 and 2) and finally, the last document is an internal governance document of department 1.

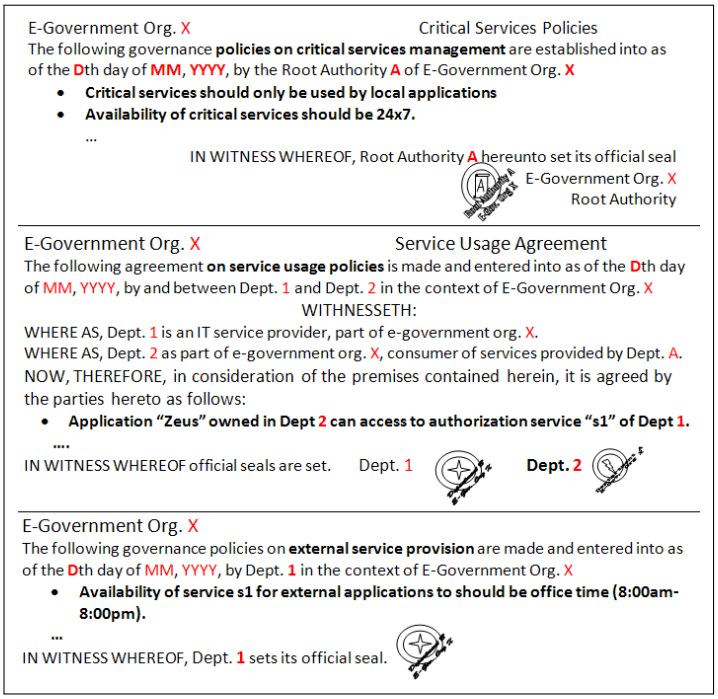


Fig. 1. Governance Documents

3 Using WS-Policy for SOA Governance

In working for a public administration, we were concerned with developing mainstream policies that would avoid ad-hoc solutions. For this reason, we decided to use the W3C recommendation WS-Policy for defining policies [9]. As can be seen in Fig. 3 where the UML metamodel of WS-Policy is shown[1], the building blocks of policies are assertions (*PolicyAssertion*) that are composed using *and*, *or* and *xor*–like compositors (*All*, *ExactlyOne*, and the top level compositor *PoliciAlternative*). Policy nesting is supported by meaning a logical AND operation of the assertions of the global policy and those of the nested one.

Assertions represent domain-specific capabilities, constraints or requirements, where their grammar is left open by WS-Policy, thus allowing the use of XML-based Domain Specific Languages (DSLs) for that purpose (see the "VariationPoint" stereotype of PolicyExpression in Fig. 3). WS-Policy has been mainly focused on the definition of policies related to specific service capabilities such as security, reliability, etc. In fact, there are a number of DSLs for those purposes. Unfortunately, describing assertions for SOA governance policies is more complex and as far as we know there is currently no DSL to describe this kind of policies.

Two mechanisms are available in WS-Policy to associate policies with the subjects to which they apply, *i.e.* for defining their scope. The first one associates one or more policy definitions as a part of the subject definition. For example, if we want to apply a policy to a web service described in WSDL, the policy specified in WS-Policy has to be inserted into the WSDL code. We call this mechanism *endogenous attachment*, since the definition of the policy is internal to the element one. Left column in table 1 shows an example of this kind of attachment. In this case, two polices on the web ser-

[1] The UML class diagram in 3 represents our interpretation of the metamodel described by the XML schema specified in [9] and [10].

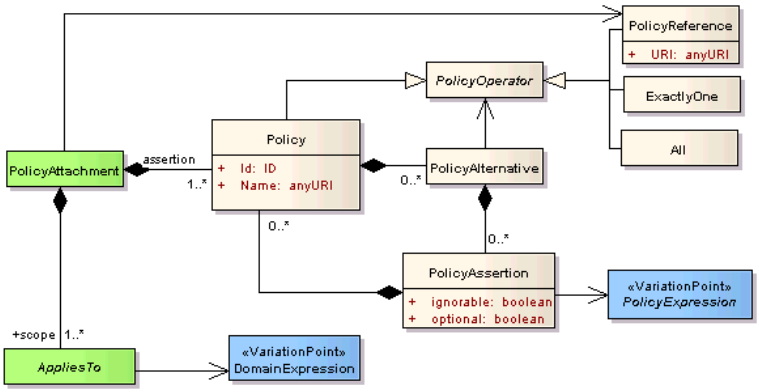


Fig. 2. UML metamodel of WS-Policy

vice "StockQuote" are defined: one to ensure a reliable message and another to specify security mechanisms on web service binding. Notice that with endogenous attachment: i) attaching a set of policies to a set of subjects at the same time is not possible and ii) changing a policy requires modifying the definition of the subject, *i.e.* it is an *intrusive* mechanism.

The second mechanism associates one or more policy definitions to one or more subjects. The WS-PolicyAttachment recommendation [10] proposes the use of *PolicyAttachemt* and *AppliesTo* (shaded classes in Fig. 2). In this case, the WS-Policy is not encoded in the same file that the specification of the element, thus we call this mechanism *exogenous attachment*. Right column in table 1 shows an example of this kind of attachment. In this case, secure binding mechanisms are asserted on the "StockQuote" and "MortgageRisk" services using its endpoint reference address. As shown in Fig. 2, WS-PolicyAttachment also leaves open the language to specify the scope, but provides a basic language to specify it based on URIs.

Note that with exogenous attachment: i) it is possible to attach a policy to a set of elements at the same time and ii) changing the policy attachment does not require modifying the definition of an element, *i.e.* it is *non–intrusive* mechanism.

WS-Policy 1		WS-Policy 2	
<wsdl:definitions name='StockQuote' xmlns:wsp='...' ...>	Element attached	<wsp:PolicyAttachment>	
<wsp:Policy wsu:Id='RmPolicy' >		<wsp:AppliesTo>	
<rmp:RMAssertion>		<wsa:EndpointReference>	Policy Scope
<wsp:Policy/>		<wsa:Address>.../MortageRisk.wsdl</...>	Scope
</rmp:RMAssertion>		<wsa:Address>.../StockQuote.wsld</...>	def.
</wsp:Policy>		</wsa:EndpointReference>	
<wsp:Policy wsu:Id='X509Policy'	Policy	</wsp:AppliesTo>	
<sp:AsymmetricBinding>	assertions	<wsp:Policy wsu:Id='X509Policy'	
...		<sp:AsymmetricBinding>	Policy
</sp:AsymmetricBinding>		...	assertions
</wsp:Policy>		</sp:AsymmetricBinding>	
...		</wsp:Policy>	
</wsdl:definitions>		</wsp:PolicyAttachment>	

Table 1. Endogeous vs Exogeous Attachment

WS-Policy defines a mechanism to test the compatibility of two policies, called *policy intersection*. According to the WS-Policy specification [9] "policy intersection is optional but a useful tool when two or more parties express policies and want to limit the policy alternatives to those that are mutually compatible". The intersection consists of two parts: a domain-independent policy intersection and domain-specific processing. The former takes into account the assertion type equality, *i.e.* the XML element type equality and its nested elements, but not its parameters. The latter is not defined

in WS-Policy, thus assertions authors have to define specific mechanisms for incorporating the intended semantics of the assertions, and the specification does not provide any mechanism to integrate or define it. For instance, the first policy specified in the left column and the policy in the right column of table 1 are incompatible, since their element types *<rmp:RMAssertion>* and *<sp:AsymmetricBinding>* are different. The notion of policy intersection is thus basically syntactical and structural, and it is not valid for complex domain-specific processing or semantic reasoning about policies, as shown in [11] and [12].

When using WS-Policy to specify the policies the previously described documents we encountered the following limitations:

Lack of Context and meta-data (LCD): Governance policies need a rich context to ensure their validity, specifying who enacts the policies and providing additional meta-data, in order to ensure authorization for policy enactment and the integrity of the policies as enacted, thus avoiding tampering. This is the role of seals and signatures of the real documents shown in figure 1. In using WS-Policy, there is not a single point where we can insert the required information that assures the validity of the policy, such as official seals, a declaration of validity by the enacting authority or a preamble. We call this problem Lack of Context Data (LCD).

Scope Definition Limitations (SDL): Defining a policy P1 as simple as "*All services provide an Availability greater than 99%*" may become a nightmare since WS-Policy has not been designed keeping in mind that the scope of a policy could be defined by intension; *i.e.* we need specify the properties of services belonging the scope, not enumerate them. If an exogenous attachment with the proposal described in the WS-Policy specification is used, then all the services references will be inserted in the *AppliesTo* section of the policy. Thus, if there was a change in the policy scope, like *All services except S23, S45, . . .*) this would entail a re-working of all services as well as modifying the content of the scope section. This is not an adequate solution when dealing with a SOA comprising of hundreds of applications and services. Summarizing, governance policies' scope should be defined by intension through scope predicates in most cases.

The expression of these scope predicates require a rich predicate DSL and the specification of the elements and data sources needed to feed the predicate, in order to effectively evaluate policy scope. This problem is particularly acute for governance policies, since governance relevant information is stored in disparate sources, such as UDDI registries, LDAP directories, *ad hoc* databases, etc. For instance, in our sample GDs (fig. 1) there are properties such as "*critical services*" and "*local/external apps & services*" whose values must be obtained from different information sources.

However, WS-Policy extension mechanisms allow the definition of DSL for specifying policy scope (see fig. 3), thus we propose such a DSL as part of our proposal.

Inadequate consistency checking: In our use case, the most valuable property was consistency checking. The only analysis operation WS-Policy envisioned for this was the intersection of policies. The open and flexible nature of WS-Policy makes it difficult to provide homogeneous semantics to policies, since each domain specific DSL would have its own semantics. This problem motivates the merely structural-syntactical nature policy intersection operation as defined in WS-Policy, avoiding its usage in diverse scenarios [11,12]. For example, two identical-meaning policies may prove inconsistent by using the intersection operation (see [11]). We name this drawback as **Syntax/Structure Driven Semantics (SDS)**. This limitation can be addressed by defining a domain specific intersection processor, and consequently, our proposal for providing semantic consistency checking can be integrated in such a way. However, authors consider that defining an entirely new operation called consistency and provide an explicit semantic for the DSLs defined is a better approach. Otherwise, the intersection operation, could process some assertions based on their structure, while others will be treated semantically, leading to to incoherent results (as shown in [11]). Thus, we leave intersection as an essentially sintactically/structural operation and define a new operation named consistency that supports semantic reasoning based on CSPs.

4 From WS-Policy to Ws-Governance

WS-Governance Documents (GDs) address limitations described previosly by incorporating an extensible context to the contained policies (addresing LCD), and defining a global document structure that contains a set of policies under an umbrella governance scope (where the data sources that feed predicates for defining policies scope can be included, addressing SDL). They also describe relevant governance properties of services, applications and organizational structures, and provide mechanisms for incorporating disparate governance-relevant information sources. Moreover, LCD and SDS motivate the creation of two general purpose XML-based DSLs for specifying governance policy assertions and SOA modelling, that are described in detail below. Based on those DSLs and the authors‘ experience providing formal semantics for SLAs specified in WS-Agreement by using CSPs [13,14,15], a CSP based semantics for WS-Governance documents using those DSLs is proposed in Sec. 5 addressing SDS. Based on those semantics a consistency property for policies and governance documents is defined. The structure of a GD in WS-Governance comprises of:

- **Governance Document Context:** It currently defines the governing organization but its grammar is left open in order to support the expressions of authorizations to enact policies on this GD, and the data needed to ensure GD authenticity and integrity.
- **Governance Scope:** It provides information about the SOA where policies are established. Different information sources could be used in this section, from UDDI registries to ad hoc databases, since any SOA element could be a governance policy subject; such as projects, developers, organizations, messages, XML-schemas or applications servers.
- **Governance Properties:** It defines all properties that are relevant for governance policies. Following the philosophy of WS-Policy, its grammar is left open, allowing the use of XML-based DSLs for specifying those properties.
- **Governance Policies:** It defines the policies that conform the governance. Those polices are fully WS-Policy compliant, where the exogenous policy attachment mechanism is mandatory. Assertion and scope definition grammar is left open, allowing the use of XML-based DSLs.

It is noticeable, that neither the definition of the new concept of Governance Document, nor the use of the DSLs for Governance Policies, break WS-Policy compatibility. Although authors consider that the document-oriented treatment of policies is more natural in governance contexts (as shown in the example of fig. 1) and better supports the life-cycle of policy creation [16], from authoring by humans to deployment into servers; any governance document can be transformed into a unique policy fully compliant with WS-Policy[2]. Moreover, the DSLs described below ares used in WS-Policy extension mechanisms and variation points as described in fig. 3 for providing suitable languages for governance policy definition. The UML class diagram shown in Fig. 3 represents our proposal of metamodel for WS-Governance documents.

Some elements of WS-Governance are intentionally left open for extension in order to allow a high degree of flexibility. This flexibility is based on the use of XML-based DSLs in some variability points, allowing the creation of a whole family of governance languages. In Fig. 3 variability points are decorated with a *VariabilityPoint* UML stereotype. A brief description of these variability points is provided as follows:

- **Context DSL:** the GD metadata in the context element can be extended with any information needed by means of the nesting of new XML elements and attributes.
- **SOA Specification:** The architecture and elements to govern must be described in order to define unambiguous policies.
- **Governance Property Specification:** A description of the properties of the governed elements of the SOA is needed in order to define expressive policies. Those properties must be expressed using a XML-Based DSL.

[2] In `http://labs.isa.us.es/gda/WS-Policy-transformation.xslt` is available an XSLT tranformation that allows to perform this conversion automatically.

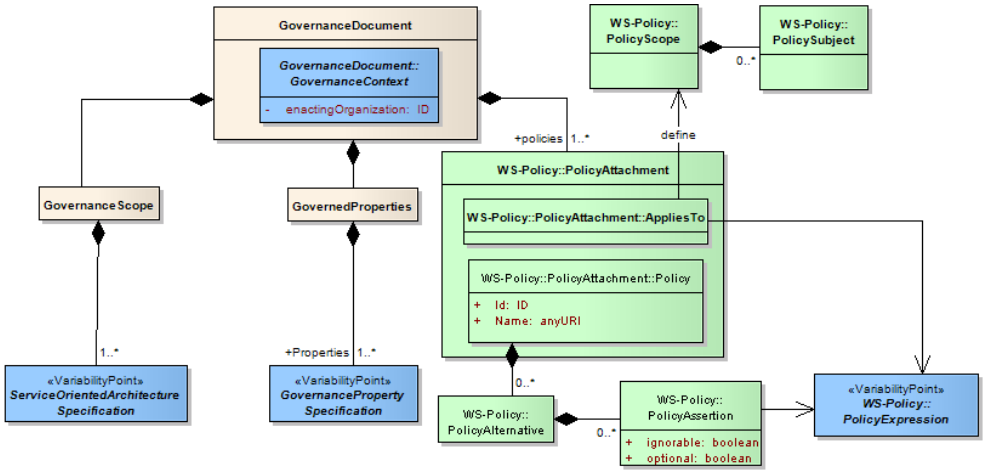


Fig. 3. UML Metamodel of WS-Governance

- **Policy Expression Specification:** Policy scope and assertions can be expressed using any predicate-oriented DSL.

In order to define effective governance documents, those DSLs must be set. In our proposal we provide two DSLs that allow the creation of service-focused governance policies, *i.e.* policies that specify assertions defined on service properties and their directly related elements such as consumers, providers, and governance relevant information such as organizational structure. Those DSLs are *Service oriented Architecture Description Language (SADL)*, addressing the SOA Specificaton variation point, and *Governance Assertion Language (GAL)*, addressing the Governance Property Specification and Policy Expression Specification variation points. The UML Class Diagrams in Figs. 4 and 5 depicts the metamodel of SADL and GAL respectively.

4.1 SOA Modeling with SADL

SADL has been designed to model the SOA state and structure, making our proposal independent of the specific governance information sources available on each SOA, such as UDDI Registries, LDAP directories, *ad hoc* databases, etc. SADL describes both the SOA structure as elements, and its state as the corresponding governance properties values for those elements. In this paper we focus on service-related governance policies, so SADL mainly contains elements related with services; however SADL is extensible, supporting the use of any XML-based construct as sub-elements of its basic structural elements. Specifically, structural elements in SADL are described as follows:

- Service Oriented Architectures are networks of participants providing and consuming services to fulfill a purpose. In SADL these participants are specified as organizations and applications.
- Organizations are participants with governance relevant identity and properties, tracing an organizational boundary on their owned applications and services. Organizations are arranged hierarchically, where an organization can contain various sub-organizations (*e.g.* departments) and have a unique parent.
- Applications represent business processes, related capabilities and software packages. They allow the arrangement of software artifacts and capabilities independently of the organizational hierarchy in a governance-meaningful way. Applications are owned by a unique organization. Applications have a set of provided and consumed services.
- Services represent capabilities that participants provide and consume.

Regarding SOA state description, SADL allows the specification of property values for all the aforementioned elements based on GAL.

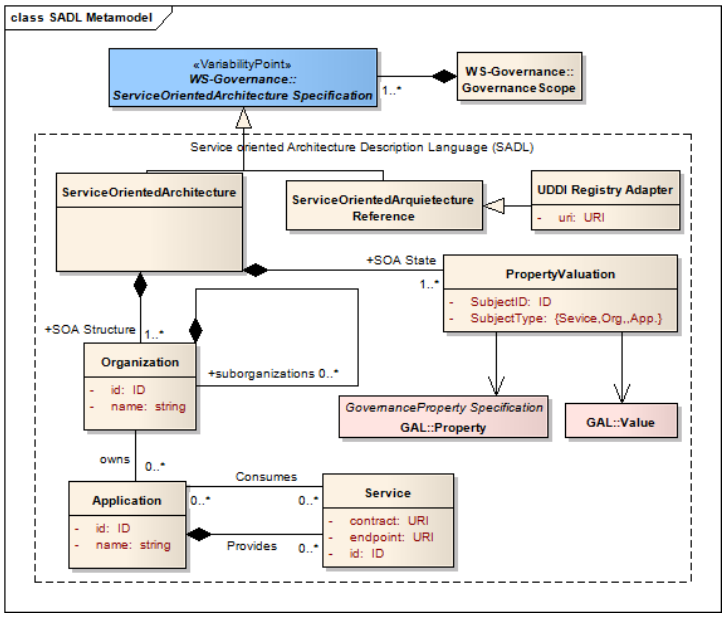


Fig. 4. UML Metamodel of SADL

Finally, SADL provides a generic element for the specification of the concrete governance data sources as references; such as UDDI registries, that should be queried to obtain the governance-relevant SOA structure and state in order to check properties and test policies adherence. By creating adapters that query those data sources and create a SADL compliant SOA model, our proposal becomes independent of those specific data-sources, thus semantics of GDs are based on explicit SADL models.

4.2 Specifying Governance properties, and policy assertions with GAL

Governance Assertion Language GAL is a generic and expressive language designed to declare governance properties and assertions. Property definitions in GAL have a name and an identifier as attributes, comprising of: (i) type definition, where basic XML-Schema [17] types are supported, (ii) an optional domain definition that restricts the space of valid values of the property; where it could be described as a GAL assertion (by intension) or as a set of values (by extension); and (iii) an optional SADL governance subject declaration, that defines the type of SOA element that can present the property (service, organization, policy, all, etc.). Through GAL constraints we provide a suitable language to specify policy assertions on governance properties. Assertions can be composed using WS-Policy composition operators: All, ExactlyOne and PolicyAlternative.

In order to allow consistency checking, in this paper we use a subset of WS-Governance a bit less expressive, called WS-Governance*. A WS-Governance* document must use SADL to describe the SOA to govern and GAL to define governance properties, policy scopes and policy assertions. A WS-Governance* document (ρ) comprises of:

- Governance Scope defines the set of organizations O, applications A and services S to govern, and their relationships, namely: consumption of services by applications and organizations, provision of services by applications and organizations, ownership of applications by organizations and hierarchy of organizations. Only those elements and relationship functions are used to define policies.
- Governance Vocabulary must define the set of all properties V used in the guarantee terms.

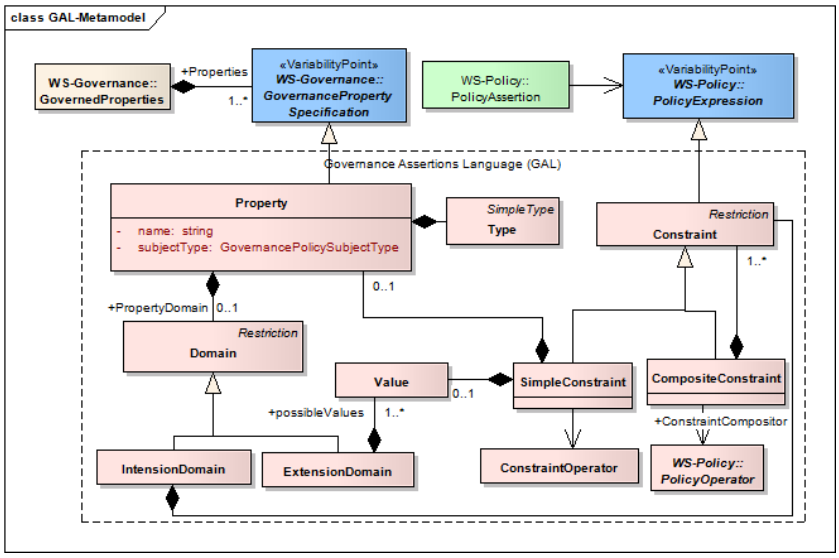


Fig. 5. UML Metamodel of GAL

- For each governance policy both scope (s) and assertion (a) must be defined as GAL assertions on the properties defined in the governance vocabulary section, and only to those applied to the sets and relationship functions defined in Governance Scope.
- Policy assertions can be composed using the compositors defined in WS-Policy.

XML-Schemas that model WS-Governance* documents conforming the previously described syntax are available at `http://www.isa.us.es/gda/schemas`. Although readable for humans, XML is not as understandable as plain text. In Table 2 the structure of a WS-Governance* document is described in a plain text language, named WS-Gov4People, that is equivalent to the WS-Governance*. The mapping of XML elements onto its corresponding WS-Gov4People sentences is shown in Table 2. A WS-Gov4People document describing the policies specified in figure 1 and a simple SOA structure is shown in the first column of table 4.

5 Automatic consistency checking through CSPs

As shown in [18], the definition of the semantics of a language can be accomplished through the definition of a mapping between the language itself and another language with well-defined semantics such as Abstract State Machines, Petri Nets, rewriting logic or CSPs. These semantic mappings between semantic domains are very useful not only to provide precise semantics to DSLs, but also to be able to simulate, analyze or reason about them. In this section we define the mappings that tranform GDs* onto *Constraint Satisfaction Problems* (CSPs) that provide their precise semantics, allowing the usage of CSP solvers to reason about policies and complete GDs*. A CSP $\psi = (V, D, C)$ is defined as a set of variables V, a set of domains D (one for each variable), and a set of constraints C specifying which combinations of variables and values are acceptable. A solution σ to a CSP ψ consists of an assignment in which each variable gets a value from its corresponding domain, as long as it satisfies each constraint. The solution space of a CSP ψ, denoted as $sol(\psi)$, is composed of all its possible solutions, if the CSP has at least one solution it is satisfiable; *i.e.* $sat(\psi) \Leftrightarrow sol(\psi) \neq \oslash$.

Mapping a GD* into CSPs. The mapping ($\mu^{GD} : \rho \rightarrow \psi$) of a WS-Governance* GD (ρ) to a CSP (ψ) is performed in two steps as follows:

WS-Governance/SADL/GAL XML Element	WS-Gov4People Document Structure
<wsg:GovernanceDocument Name='*name?*' Id='*Id?*'> <wsg:Governor id='*Org. Id?*' name='*Org. Name?*'/> <wsg:GovernanceScope> {<saml:ServiceOrientedArchitectureReference>...</...> — <saml:ServiceOrientedArchitecture>...</...>}+ </wsg:GovernanceScope> <wsg:GovernanceVocabulary> <gal:Property> ...</gal:Property>+ </wsg:Vocabulary> <wsg:GovernancePolicies> {<wsp:PolicyAttachment> <wsp:AppliesTo> <gal:QuantifiedAssertion> {<gal:Quantifier type='$Exists\|ForAll$' varname='*VarName?*' in='$Service\|Org\|App$'>}+ <gal:Assertion>*Scope expr*?</gal:Assertion> </gal:QuantifiedAssertion> </wsp:AppliesTo> <wsp:Policy name='*name?*' id='*Id?*'>*Assertion expr?*</.../> </wsp:PolicyAttachment>}+ </wsg:GovernancePolicies> <wsg:GovernanceDocument>	***Governance Document*** *- Name* (*Id*) **Governor:** *Org. Name?*(*Org. Id?*) **Scope:** {SOA Registry Reference— SOA Governance Model}+ **Vocabulary:** { Property: ... }+ **Policies**: {**Policy** *name?* (*Id?*) {$forall\|Exists$ *VarName?* in $(Servs\|Orgs\|Apps)$}+ **Scope**: *Scope expr*? **Assertion**: *Assertion expr?* }+
<saml:ServiceOrientedArchitecture> {<saml:Organization name='*Org. Name?*' > id='*Org. Id?*'> <saml:SubOrganizations> {<saml:Organization ...> ... </saml:Organization>}* </saml:SubOrganizations> <saml:Applications> {<saml:Application name='*App. Name?* id='*App. Id?*'> <saml:ProvidedServices> {<saml:Serivce name='*Serv. Name?* id='*Serv. Id?*'/>}* </saml:ProvidedServices> <saml:ConsumedServices> {<saml:Serivce name='*Serv. Name?* id='*Serv. Id?*'/>}* </saml:ConsumedServices> </saml:Applications>}* }+ {<gal:PropertyValue property='IdP' subject='IdS' value='*Val. Expr?*' / > </saml:ServiceOrientedArchitecture>	**SOA Governance Model:** STRUCTURE: {**Organization:** *Org. Name?* (*Org. Id?*) **SubOrganizations:** *Org. Id1?,. . . ,Org. IdN?* **Applications:** {Application: *App. Name?* (*App. Id?*) Provides: {Service: *Serv. Name?* (*Serv. Id?*)}* Consumes: {Service: *Serv. Name?* (*Serv. Id?*)}* }* STATE: *Val. Expr?** }+
<gal:Property name='*Prop. Name?*' id='*Prop. Id?*' subjectType='$Serv\|Org\|App$'> <gal:Type>*Type. Expr.?*</gal:Type> <gal:Domain><gcl:Constraint>*Domain Expr.?* </gal:Constraint></gal:Domain> </gal:Property>	**Property:** *Prop. Name?* (*Prop. Id?*) **for** $\{Servs\|Orgs\|Apps\}$ Type: *Type. Expr.?* Domain: *Domain Expr.?*

Table 2. Mapping from WS-Governance, GAL and SADL to WS-Gov4People

First, for each policy $p_i = (s, a)$ in the GD*, p_i is mapped for the concrete governance scope (SOA model in terms of services, applications, organizations and its relationships) into a constraint p_i^C that contains only variables and literals composed using logical and algebraic operators. This constraint is constructed so that it tells exactly when the policy holds in the given governance scope. This transformation is performed by the explicit enumeration of the sets and relationships (ownership, provision, consumption, and organizational hierarchy) on the governance scope for each quantifier, combining the resulting constraints using logical AND ($\wedge$) operators for universal quantifiers and logical OR ($\vee$) operators for existential quantifiers.

Next, the set of constraints $p^C = \{p_i^C\}_{i=1}^n$ and original GD* ρ are mapped into a CSP $\psi = \{V, D, C\}$ by creating:

- a variable $v^x_{s_i|o_i|a_i}$ in V for each property x and corresponding element in the SOA (service, organization and application).

- variables $v_{o_i}^{supOrg}$,$v_{s_i}^{prov}$,$v_{s_i}^{cons}$ and $v_{a_i}^{own}$ in V for the relation functions $supOrg$ (hierarchical relationship among organizations), $provider$ and $consumer$ (relationship among services and applications) and $owner$. Additionally, their domain of organizations, applications, and services are created.
- constraints $\{v_{o_i}^{supOrg} = o_j\}$, $\{v_{s_i}^{prov} = a_k\}$, $\{v_{s_i}^{cons} = a_k\}$, and $\{v_{a_i}^{own} = o_l\}$ in C for each variable created in the previous step specifying the values of the relationships$supOrg$, $provider$,$consumer$ and $owner$. Those constraints and variables express the SADL SOA structural model of the governance scope.
- a constraint $\{v_{s_i|o_i|a_i}^{x} = \{\textit{Value Expr?}\}\}$ in C for each property valuation specified in the state section of the governance scope in ρ.

Finally, for each constraint in p^C property invocation functions $X(s_i|o_j|a_j)$ are exchanged by their corresponding variables $v_{s_i|o_i|a_i}^{x}$, and the resulting constraint is added to C. The mapping of different elements of a GD* is shown in Table 3.

WS-Governance* Element	CSP Mapping
Governance Document *- Name (Id)*	GD Name, Id and Governor Org.
Governor: *Org. Name?(Org. Id?)*	are not mapped to CSP
SOA Governance Model:	For each organization o_i a variable $v_{o_i}^{suporg}$ is created
STRUCTURE:	denoting the parent org. of o_i and a domain
{**Organization:** *Org. Name?* (*Org. Id?*)	$d_i^{suporg} = \{o_1, \ldots \emptyset_n\}$ is added to D
SubOrganizations: *Org. Id1?,…,Org. IdN?*	a constraint $\{C_{o_i}^{prov} = o_j\}$ is created encoding the orgs. hierarchy
Applications:	For each application a_j a variable $v_{a_j}^{own}$ is created
Provides:	denoting the owner org. of a_j, a domain
{Service: *Serv. Name?* (*Serv. Id?*)}*	$d_j^{own} = \{o_1, \ldots \emptyset_n\}$ is added to D
Consumes:	and a constraint $\{C_{a_i}^{own} = o_j\}$ is created encoding the app. ownership
{Service: *Serv. Name?* (*Serv. Id?*)}*	For each service s_k a variable $v_{s_k}^{prov}$ is created
}*	, a domain $d_k^{prov} = \{a_1, \ldots, a_m\}$ is added to D
}+	and a constraint $\{v_{s_k}^{prov} = a_l\}$ is created encoding the provisioning
STATE:	
{*Property val. Expr?*}*	Add Constraint: $v_{s_i|o_i|a_i}^{x}$[*Proverty val. Expr.?*]
Vocabulary:	Add variables & domains:
Property: *X* **for** *Services*	for each property x we create a variable
for $\{Servs|Orgs|Apps\}$	$v_{s_i|o_i|a_i}^{x}$ for each element in
Type: *boolean*	its corresponding set $S|O|A$
Domain: *Domain Expr.?*	Add Constraint: $v_{s_i|o_i|a_i}^{x}$[*Domain Expr.?*]
Policies:	Add contraint:
Policy *P1*	
{**forall** *y*	$\bigwedge_{v_{s_i|o_i|a_i}}^{S|O|A}$ (*Scope expr*)$[y/v_{s_i|o_i|a_i}^{y}]$
in $(Servs|Orgs|Apps)$}+	$\Rightarrow$ (*Assertion expr*)$[y/v_{s_i|o_i|a_i}^{y}]$
{**exists** *y*	$\bigvee_{v_{s_i|o_i|a_i}}^{S|O|A}$ ((*Scope expr*)$[y/v_{s_i|o_i|a_i}^{y}]$
in $(Servs|Orgs|Apps)$}+	$\Rightarrow$ (*Assertion expr*)$[y/v_{s_i|o_i|a_i}^{y}]$
Scope: *Scope expr?* ,	For each constraint $c \in C$ do:
Assertion: *Assertion expr* —	$(c)[v_{s_i|o_i|a_i}^{y}/v_{s_i|o_i|a_i}^{x}]$
where E[x/y] means: ' the expression E, but with occurrences of x replaced by y'	

Table 3. Mapping of WS-Governance* elements onto CSPs

In order to exemplify the use of GD*, in the left column of table 4 we can see the expression of the policies contained in the governance documents found in the case study (Figure 1). In the table we can see the four different parts of a GD* document: Context, SOA Model, Vocabulary and Policies. In this context, the SOA Model part has been enriched with the structure of the organization: on the one hand, Department 1 has one A1 application that provides s1 service and on the other hand, Department 2 has two applications (Zeus and A3). The former consumes s1 and s2 and provides service s3; the latter only provides s2. It is important to highlight that the policy found in the second document fragment of the example is not translated into a policy in the GD* document but it represents structural information expressed in the SOA Model section. Following the mapping described in this section, in the right side of Table 3 the CSP transformation of the GD* is presented in table 4.

Sample SGDL4People Document	Corresponding CSP
Governance Document *- Critial Services Management (GDI)* **Governor:** (o_x)	$\phi = \{V, D, C\}$ where $C = \{C^D \cup C^{SOA} \cup C^{\rho}\}$ $V = \{V^{Prop} \cup V^{W} \cup V^{\rho}\}, D = \{D^{Prop} \cup D^{SOA}\}$
Scope: STRUCTURE: **Organization:** *E-gov. Org. X* (o_x) **Suborganizations:** o_1,o_2 **Organization:** *Department 1* (o_1) **Applications:** Application:*A1 App* (a_1) Provides: *s1* (s_1) **Organization:** *Department 2* (o_2) **Applications:** Application:*Zeus* (a_2) Consumes: *s1* (s_1) *s2* (s_2) Provides: *s3* (s_3) Application:*A3 App* (a_3) Provides: *s2* (s_2) STATE: Critical(s_1)=true	V^{SOA}=$\{V^{prov} \cup V^{cons} \cup V^{own} \cup V^{supOrg}\}$ $V^{prov} = \{v^{prov}_{s1}, v^{prov}_{s2}, v^{prov}_{s3}, v^{prvO}_{s1}, v^{prvO}_{s2}, v^{prvO}_{s3}\}$ $V^{cons} = \{v^{cons}_{s1}, v^{cons}_{s2}, v^{cons}_{s3}, v^{cnsO}_{s1}, v^{cnsO}_{s2}, v^{cnsO}_{s3}\}$ $V^{own} = \{v^{own}_{a1}, v^{own}_{a2}, v^{own}_{a3}\}$ $V^{suporg} = \{v^{suporg}_{o1}, v^{suporg}_{o2}\}$ $D^{SOA} = \{D^{prov} \cup D^{cons} \cup D^{own} \cup D^{suporg}$ $, A = \{a_1, a_2, a_3\}, O = \{o_1, o_2\}$ $C^{soa} = \{C^{Struct} \cup C^{Stat}\}$ $C^{Struct} = \{\{v^{prov}_{s1} = a_1\}, \{v^{prov}_{s2} = a_3\}, \{v^{prvO}_{s1} = o_1\}, \{v^{prov}_{s3} = a_2\}, \{v^{prvO}_{s2} = o_3\}\}$ $\{v^{prov}_{s3} = a_2\}, \{v^{prvO}_{s3} = o_2\}, \{v^{cons}_{s1} = a_2\}, \{v^{cons}_{s2} = a_2\}, \{v^{cnsO}_{s1} = o_2\},$ $\{\{v^{cnsO}_{s2} = o_2\}, v^{cons}_{s3} = \oslash\}, \{v^{own}_{a1} = o_1\}, \{v^{own}_{a2} = o_2\},$ $\{v^{own}_{a3} = o_2\}, \{v^{supOrg}_{o1} = o_x\}, \{v^{supOrg}_{o2} = o_x\}\}$ $C^{Stat} = \{\{v^{Crit}_{s1} = true\}\}$ $D^{own} = \{O, O, O\}, D^{supOrg} = \{O, O\}$ $D^{prov} = D^{cons} = \{A, A, A, O, O, O\}$
Vocabulary: **Property:** *Critical* **for** *Services* Type: *boolean* **Property:** *Availability* **for** *Services* Type: *enum* Domain: *'24x7','office'*	$V^{Prop} = \{v^{Crit}_{s1}, v^{Crit}_{s2}, v^{Crit}_{s3} v^{Avail}_{s1}, v^{Avail}_{s2}, v^{Avail}_{s3}\}$ $D^{Prop} = \{\{true, false\}, \{true, false\}, \{true, false\},$ $\{'24x7', office'\}, \{'24x7', office'\}, \{'24x7', office'\}\}$ $C^D = \emptyset$ since there is no domain constraints
Policies: **Policy** *P1* (ρ_1) **forall** *s* in Services **Scope**: *Critical(s)=true* **Assertion**: *ProviderO(s)=ConsumerO(s)* **Policy** *P2* (ρ_2) **forall** *s* in Services **Scope**: *Critical(s)=true* **Assertion**: *Availability(s)=' 24x7'* **Policy** *P3* (ρ_3) **forall** *a* in Applications **Scope**: $a \in Consumer(s_1) \wedge Owner(a) \neq Owner(Provider(s_1))$ **Assertion**: *Availability(s_1)='office'*	$C^{\rho} = \{C^{\rho_1}, C^{\rho_2}, C^{\rho_3}\}$ $C^{\rho_1} = \{\wedge^3_{i=1}((v^{crit}_{s_i} = true) \Rightarrow (v^{prvO}_{s_i} = v^{consO}_{s_i}))\} = \{((v^{crit}_{s1} = true) \Rightarrow (v^{prvO}_{s1} = v^{cnsO}_{s1})) \wedge$ $((v^{crit}_{s2} = true) \Rightarrow (v^{prvO}_{s2} = v^{cnsO}_{s2})) \wedge ((v^{crit}_{s3} = true) \Rightarrow (v^{prvO}_{s3} = v^{cnsO}_{s3}))\}$ $C^{\rho_2} = \{\wedge^3_{i=1}((v^{crit}_{s_i} = true) \Rightarrow (v^{Avail}_{s_i} =' 24x7'))\} = \{((v^{crit}_{s1} = true) \Rightarrow (v^{Avail}_{s1} =' 24x7')) \wedge$ $((v^{crit}_{s2} = true) \Rightarrow (v^{Avail}_{s2} =' 24x7')) \wedge ((v^{crit}_{s2} = true) \Rightarrow (v^{Avail}_{s2} =' 24x7'))\}$ $C^{\rho_3} = \{\wedge^3_{i=1}((s^{cons}_{s1} = a_i \wedge v^{prov}_{s1} \neq v^{own}_{a_i}) \Rightarrow (v^{Avail}_{s1} =' office'))\} =$ $= \{((s^{cons}_{s1} = a_i \wedge v^{prov}_{s1} \neq v^{own}_{a1}) \Rightarrow (v^{Avail}_{s1} =' office')) \wedge ((s^{cons}_{s1} = a_2 \wedge v^{prov}_{s1} \neq v^{own}_{a2}) \Rightarrow$ $\Rightarrow (v^{Avail}_{s1} =' office')) \wedge ((s^{cons}_{s1} = a_3 \wedge v^{prov}_{s1} \neq v^{own}_{a3}) \Rightarrow (v^{Avail}_{s1} =' office'))\}$

Table 4. Sample WS-Governance4People onto CSP mapping

5.1 Checking for Consistency

Checking a GD ρ written in WS-Governance* for consistency lets us know whether it has internal contradictions or not. The root of the inconsistencies can be: (i) that a policy in ρ is intrinsically inconsistent; (ii) that the set of policies in ρ are inconsistent; or (iii) even when the set of policies in ρ, $P^{\rho} = \{\rho_1, \ldots, \rho_n\}$, are initially consistent, then ρ can be inconsistent due to the additional information added by the SADL SOA state and structure specified. For instance, the GD* shown in the first column of table 4 is inconsistent. The cause of the inconsistency is that policies ρ_2='*Critical services availability should be "'24x7' "* and ρ_3='*Service* s_1 *availability is "window" if it is consumed by external applications*' are inconsistent, since s_1 is both consumed by application a_2 'Zeus' of department 2 and critical. This information, service consumption and criticality of services is specified by the SOA Structure and State section of the GDs*, thus the corresponding constraint $Availability(s_1) =' 24x7' \wedge Availability(s_1) =' window'$ is obviously unsatisfiable due to the SOA state and structure, not because of an incosistency of policies *per se*, and consequently ρ_2 and ρ_3 are inconsistent in ρ.

Internal Consistency: A GD* ρ is said to be consistent iff its corresponding equivalent CSP is satisfiable; *i.e.* $consistent(\rho) \Leftrightarrow sat(\psi^{\rho})$

Consistency: A non empty set of GDs in WS-Governance* $P = \{\rho_1, \ldots, \rho_n\}$ is said to be consistent iff its corresponding equivalent CSPs are simultaneously satisfiable; *i.e.* $consistent(P) \Leftrightarrow sat(\bigwedge_{i=1}^{n} {\psi^{\rho}}_i)$

As an example, our approach found an additional inconsistence: in the mapping of the GD* shown in Table 4, the corresponding CSP ψ contains among others the following constraints: $\{v_{s1}^{crit} = true \Rightarrow v_{s_1}^{provO} = v_{s_1}^{consO}\}$, $\{v_{s2}^{crit} = true\}$, $\{v_{s_1}^{consO} = o_2\}$, and $\{v_{s1}^{provO} = o1\}$. This set of constraints in unsatisfiable, since $v_{s_1}^{prov} = o_2$ and $v_{s_2}^{prov} = o_1$ are unsatisfiable, and consequently the GD* is inconsistent.

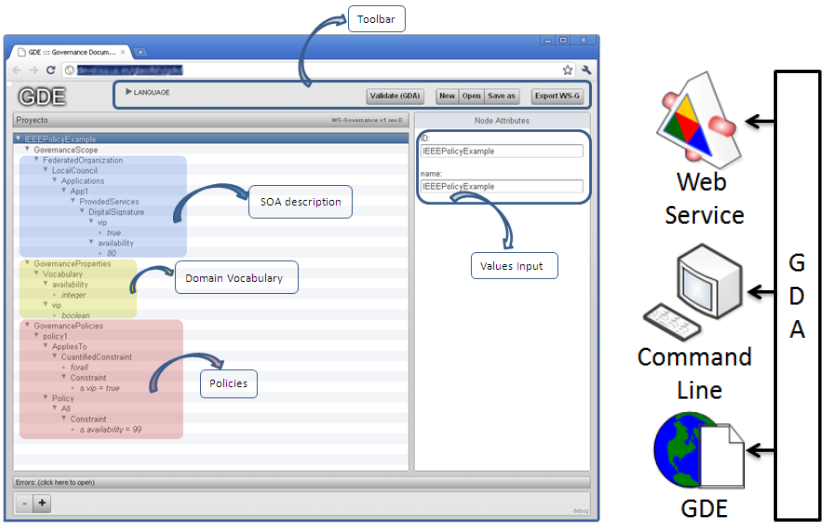
5.2 WS-Governance tooling: GDA and GDE

Governance Document Analyzer (GDA) is an automatic analyzer for GDs. It performs consistency analysis in three levels: i) individual policy consistency, ii) consistency of the set of policies in the document, and iii) consistency of the whole GD. This analysis starts with the transformation of GD statements to a CSP, that is solved with a CSP solver [19]. The tool is available as a web service so it can be easily integrated into other tools and interoperate. Furthermore, a command line client is provided for direct use. A set of sample GDs has been created, in order to ensure that the analyzer works properly. Both the analyzer and the samples are available at `http://www.isa.us.es/gda`.

Governance Document Editor (GDE) is a web application to edit government documents in an assisted way. The editor window is divided in three areas as in Figure 1 (a). On the top section, there are different options to open, save and analyze GDs. On the left side of the main area, every section in the GD is organized in a tree view. Every node on the tree is attributed on the right-hand side. The main tree has separate sections for each subsection in the GD. WS-Governance elements are automatically generated from the tree, so edition focuses on the relevant government contents avoiding errors. It has been integrated with GDA, showing the consistency analysis reports graphically. The editor is available on-line at `http://labs.isa.us.es/apps/gde/`.

6 Related Work

Concerning policy definition, Ponder is the pioneer and probably most widely used language. Ponder [20] is a declarative, object oriented language for the specification of management polices in distributed object systems. Additionally, Ponder provides structuring techniques for policy administration in large scenarios and systems. WS-Governance incorporates similar concepts by the explicit declaration of governor and



(a) GDE User Interface (b) GDA Depl. Opt.

Fig. 6. GDE User Interface and GDA Deployment Options

document context, and uses SADL and GAL assertions to model scope. The usage of WS-Policy as the base policy expression construct, and explicit declaration of governance relevant information sources, makes WS-Governance better suited for SOA governance policies declaration. However, an interesting capability supported by Ponder that we plan to add to WS-Governance in the future is the declaration of policy types as templates. Rei and KAoS [21], both based on semantic web concepts are proposals oriented to the definition of policies for expressing web services capabilities policies. However, is WS-Policy the proposal that has more successful in industrial scenarios, thus we have chosen it for extension in order to define SOA governance policies.

Our proposal provides a richer consistency notion allowing the detection of semantical inconsistences in policies with complex interaction as shown in this paper. General policy conflict analysis is not a novel problem, but its application in the context of SOA governance policies in this paper is original. Several approaches have been proposed for policy expression and conflict analysis in the context of network management [22], and security [23], but they are based on Binary Decision Diagrams (BDD), forcing the reasoning with less expressive policies than our CSP based proposal.

7 Conclusions and Future Work

The results obtained during the development of this work provide three important conclusions: i) there exists the need of a language for governance policies specification that integrates governance data sources; enables effective governance through a formal semantics, and supports automated consistency checking; ii) this need is not fulfilled by WS-Policy, that has limitations in this context; and iii) WS-Governance allows the specification of expressive Governance Documents independently of the underlying infrastructure. It is based on the WS-Policy framework, and overcomes its previously identified limitations to achieve an effective governance. The GD semantics and consistency operation paves the way for the creation of a new generation of governance tools. In this scenario governance policies are created in a distributed, independent yet collaborative way, maintaining global consistency. This collaborative process helps governance boards on the creation of the essential normative framework needed for the achievement of the SOA promise.

In this context we identify the following ideas as promising future work to be addressed: i) enacting organizations need extensions of the GD Context to effectively ensure authentication and authorization, in order to avoid tampering when defining policy (currently this extension is supported by the language through the variability point but those mechanisms are not specified); (ii) extend WS-Governance to support the definition of policy templates improving reuse and usability; iii) carry out a performance analysis of our implementation in order to study the influences of the SOA model, governance properties and number and complexity of governance policies.

References

1. Marks, E.A.: Service-Oriented Architecture Governance for the Services Driven Enterprise. John Wiley & Sons (2008)
2. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: State of the art and research challenges. IEEE Computer **40**(11) (2007) 38–45
3. Schepers, T.G.J., Iacob, M.E., Van Eck, P.A.T.: A lifecycle approach to soa governance. In: SAC '08: ACM Symposium on Applied computingk. (2008) 1055–1061
4. Kenney, L.F., Plummer, D.C.: Magic quadrant for integrated soa governance technology sets. Technical report, Gartner (2009) http://mediaproducts.gartner.com/reprints/oracle/article65/article65.html.
5. Kontogiannis, K., Lewis, G.A., Smith, D.B.: A research agenda for service-oriented architecture. In: SDSOA '08: 2nd Int. workshop on Sys. devel. in SOA env. (2008) 1–6
6. Bernhardt, J., Seese, D.: A conceptual framework for the governance of service-oriented architectures. In: ICSOC 2008 Workshops. Springer (2009) 327–338
7. Derler, P., Weinreich, R.: Models and tools for soa governance. In: Trends in Enterprise Application Architecture. Springer (2007) 112–126
8. Parejo, J.A., Fernández, P., Ruiz-Cortés, A.: Towards automated sla-based governance policy enforcement. In: Int. Joint Conference on Service Oriented Computing (ICSOC). (2009)
9. Vedamuthu, A.S., Orchard, D., Hirsch, F., Hondo, M., Yendluri, P., Boubez, T., Ümit Yalçinalp: Web services policy 1.5 framework. W3C Recommendation (2007)
10. Vedamuthu, A.S., Orchard, D., Hirsch, F., Hondo, M., Yendluri, P., Boubez, T., Ümit Yalçinalp: Web services policy 1.5 - attachment. W3C Recommendation (2007)
11. Hollunder, B.: Domain-specific processing of policies or: Ws-policy intersection revisited. In: ICWS. (2009) 246–253
12. Anderson, A.H.: Domain-independent, composable web services policy assertions. In: POLICY. (2006) 149–152
13. Ruiz-Cortés, A., Martín-Díaz, O., Durán, A., Toro, M.: Improving the automatic procurement of web services using constraint programming. International Journal of Cooperative Information Systems **14**(4) (2005) 439–467
14. Müller, C., Ruiz-Cortés, A., Resinas, M.: An initial approach to explaining sla inconsistencies. In: 6th. Int. Conf. on Service-Oriented Computing (ICSOC). Volume 5364. (2008) 394–406
15. Müller, C., Resinas, M., Ruiz-Cortés, A.: Explaining the non-compliance between templates and agreement offers in ws-agreement. In: 7th Int. Conf. on Serv. Oriented Comp. (ICSOC). Volume 5900. (2009) 237–252
16. Zhang, Y., Liu, X., Wang, W.: Policy lifecycle model for systems management. IT Professional **7** (2005) 50–54
17. Peterson, D., Gao, S.S., Malhotra, A., Sperberg-McQuee, C.M., Thompson, H.S.: W3c xml schema definition language (xsd) 1.1 part 2: Datatypes. W3C Working Draft (2009)
18. Vallecillo, A.: A journey through the secret life of models, Schloss Dagstuhl - Leibniz-Zentrum fÃ¼r Informatik (2008)
19. Laburthe, F., Jussien, N., Rochart, G., Cambazard, H., Prud'homme, C., Malapert, A., Menana, J.: Choco, java library for constraint satisfaction problems (csp). (Open Source, http://www.emn.fr/z-info/choco-solver/)
20. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The ponder policy specification language. In: POLICY '01: Int. Workshop on Policies for Dist. Systems and Networks. (2001) 18–38
21. Uszok, A., Bradshaw, J., Johnson, M., Jeffers, R., Tate, A., Dalton, J., Aitken, S.: Kaos policy management for semantic web services. Intelligent Systems, IEEE **19**(4) (2004) 32–41
22. Samak, T., Al-Shaer, E., Li, H.: Qos policy modeling and conflict analysis. In: POLICY. (2008) 19–26
23. Hamed, H.H., Al-Shaer, E.S., Marrero, W.: Modeling and verification of ipsec and vpn security policies. In: ICNP. (2005) 259–278