SOA Policies

Enterprise Services Implementation Policies

Ver. 1.2

Official Documentation

SOA Policies - Ver. 1.2

Enterprise Services Implementation Policies

19. Feb. 2013

Official Documentation

Author:

Enterprise Architecture Group

Describes what has to be guaranteed when implementing services within the IT system of the Autonomous Province of Bolzano.

Content

1. Abstract

This chapter describes the policies to obey when building and consuming enterprise services within the IT system of the Autonomous Province of Bolzano. As all architecture policies do, they aim at guaranteeing implementation of architectural principles. In addition to the well known SOA benefits, their main goals are

- Guaranteed interoperability by limiting the adopted standards to a reasonable, but required subset of available competing industry standards;
- Automation of service and consumer stub building for speeding up solution development and leveraging software industrialization;
- Avoiding of proliferating technologies and standards for consolidating specialized know how within the operations staff;
- SSO as a required architecture principle;
- and highly dynamic discovery capabilities of service consumers in distributed environments for drastically reducing maintenance costs.

All policies defined by this document relate to architecture principles:

- 4 (Service Orientation),
- 19 (Control Technical Diversity),
- and 20 (Interoperability).

If a policy supports additional principles you can find them in the policy description.

2. Componentization Policies

The following policies help with optimizing service implementation and consumption, and forming the basement of platform interoperability.

Policy ID	Policy description
1.1	**Services are bundled up to Business Components** Multiple services having functional affinity MUST be bundled up to Business Components. **Rationale** Services pertaining to one and the same functional area and having a strong functional affinity typically share the same business objects and offer a coherent set of functionality, often relying on each other. Bundling up such services not only from a conceptual point of view but also at an implementation level allows ▪ To build component wide and precompiled interfaces for consuming the functionality ▪ To share the business objects instead of heaving to redefine them for each single service ▪ To enhance performance by avoiding object translation and by making direct method calls within a component when relying on sibling services ▪ To build compact (atomic) and functionally coherent deployment deliverables **Refers to principles**: 8, 9, 10, **11**, 12, **14**
1.2	**Well formed component namespace** Every Business Component MUST define a root namespace for its services and data objects which follows the following composition rule: `<org-ns>.<layer>.<fa>.<bc>` where `<>` denote placeholders and are not part of the namespace `org-ns` is the namespace of the organization. For the Autonomous Province of Bolzano it is `it.bz.prov`. `layer` is either one of the official component layers of the Autonomous Province of Bolzano except *Core* (so *Base* or *Foundation*), or a macro functional area for the case the component belongs to the *Core* layer. `fa` is the name of a functional area

	`bc` is the name of the Business Component **Rationale** The root namespace of a Business Component allows to identify services and business objects in an enterprise wide unique way: the organization is subdivided into disjoint functional areas, and every functional area can have multiple Business Components whose names are however disjoint within the functional area, even though not necessarily when crossing functional area boundaries. The same can be said of names of services and data objects belonging to a single Business Component. The root namespace of a Business Component is used to associate services and objects with the component: ▪ All "official" references to services and data objects of a Business Component (such as when querying or browsing service registries) will allow to identify their objects directly in the root namespace ▪ Service provider and service consumer implementations, hidden from the rest of the enterprise, may freely expand the root namespace for their needs (for tiering purposes for example, for readability, for performance issues, …) **Refers to principles**: -
1.3	**Well formed service names** A service name MUST NOT end with the `Service` suffix. **Rationale** Apart from unnecessary information redundancy, some platforms autonomously append the `Service` suffix to service names when describing them for example in a WSDL document, while others don't. If such documents – which don't reveal by which platform they have been created - are used as a modeling and a generation source for Business Components, the service names must be recognized in a reliant matter. If one can't know if the `Service` suffix made or did non make part of the original service name, registry querying and other tasks which rely on service names could be compromised. **Refers to principles**: -

3. Service Description Policies

The following policies aim at ensuring that a single service (not a Business Component) is described in a way suitable for both being imported into a Business Component (acting as a modeling and generation source) and acting as a service proxy generation source at design- and runtime (for dynamic service binding scenarios).

Policy ID	Policy description
2.1	**A service is described by WSDL** Every service MUST be described by a WSDL document. **Rationale** WSDL describes a service in a platform independent and vendor neutral manner and in a self-contained, sufficiently granular way for guaranteeing, by its schema description, structural integrity of exchanged data. These are prerequisites for cross platform integration and secure method invocation. **Refers to principles**: -
2.2	**Service Description is freely accessible** WSDL documents MUST NOT require authentication for being read. **Rationale** Infrastructural components may have the need to access WSDL documents at runtime for dynamic proxy creation or endpoint discovery. Authentication, however unnecessary for a mere service description which is never worth being protected, makes this cumbersome and expensive. **Refers to principles**: -
2.3	**Well formed WSDL target namespace** The WSDL target namespace MUST follow the following composition rule: `http://[<spec-ns>.]<bc>.<fa>.<layer>.<org-domain>/` where `<>` denote placeholders and are not part of the namespace `[]` are optional elements `spec-ns` specifies an arbitrary specializing namespace `bc` is the name of a Business Component `fa` is the name of a functional area `layer` is either one of the official component layers of the Autonomous Province of Bolzano except *Core* (so *Base* or *Foundation*), or a macro functional area for the case the

	component belongs to the *Core* layer. `org-domain` is the domain of the organization. For the Autonomous Province of Bolzano it is `prov.bz.it`. **Rationale** The WSDL may act as source description for component interface generation. It must be guaranteed that the interface members, especially the namespace, can later be found in a service registry. So there must be a correlation between the namespace described by the WSDL as a generation source and the namespace found in the service registry, queried by using the compiled component interface members. **Refers to principles**: -
2.4	**Well formed service name** The service name found in the WSDL MUST follow the following composition rule: `<service-name>[Service]` where `<>` denote placeholders and are not part of the name `[]` are optional elements `service-name` specifies the name of the service (see policy 1.3). **Rationale** The WSDL may act as a source description for component interface generation. It must be guaranteed that the interface members, especially the service itself, can later be found in a service registry. So there must be a correlation between the service name described by the WSDL as a generation source and the service name found in the service registry, queried by using the compiled component interface members. **Refers to principles**: -

10

4. Service Publishing and Lookup Policies

The following policies aim at dynamic and platform independent service endpoint lookup at runtime[1].

Policy ID	Policy description
3.1	**Services' WSDL locations are published in a service registry** Every service MUST be on the occasion of deployment dynamically published by the application server to an enterprise wide service registry without the need of manual configuration tasks. Every service instance is ▪ identified by its fully qualified name and ▪ associated with at least one WSDL location (exactly one per deployment instance) in form of a valid URL. Every WSDL location MUST pertain to a exactly one logical cluster. **Rationale** Deployment and running of service instances must be completely decoupled from service development. For unburdening the operations staff from proliferating configuration tasks in changing deployment scenarios, every service consumer must be able to discover a service's endpoint dynamically. **Refers to principles**: 15
3.2	**Well formed service fully qualified name** The service name found in the service registry MUST follow the following composition rule: `<org-ns>.<layer>.<fa>.<bc>.<service-name>` where `<>` denote placeholders and are not part of the service name `org-ns` is the namespace of the organization. For the Autonomous Province of Bolzano it is `it.bz.prov`. `layer` is either one of the official component layers of the Autonomous Province of Bolzano except *Core* (so *Base* or *Foundation*), or a macro functional area for the case the component belongs to the *Core* layer. `fa` is the name of a functional area

[1] Please refer to the *prov.bz Service Registry.pdf* document for implementing service publishing and service lookup.

	`bc` is the name of a Business Component `service-name` specifies the name of the service (see policy 1.3). **Rationale** Clients will work with compiled service proxies or business component interfaces. It must be guaranteed that the proxy or interface members, especially the service itself, can be used for service endpoint lookup without the need for further client side configuration effort. So there must be a correlation between the service name described by the WSDL as a generation source and the service name found in the service registry, queried by using the compiled component interface members. **Refers to principles**: 15
3.3	**Service consumers find service endpoints dynamically at runtime** Every service consumer MUST at runtime – when instantiating the service proxy - query the service registries of the Autonomous Province of Bolzano for WSDL locations for a given service. If multiple service registry instances exist, the service consumer MUST be able to dynamically connect to the most reasonable one in terms of a ratio of connectivity to costs. If multiple WSDL locations (in form of URLs) are returned by a registry instance, the service consumer MUST be able to dynamically pick the most reasonable one in terms of a ratio of connectivity to costs. **Rationale** Deployment and running of service instances as well as service consumers must be completely decoupled from service and client development. For unburdening the operations staff from proliferating configuration tasks in changing deployment scenarios (for example as a consequence of evolving connectivity and consolidation initiatives), every service consumer must be able to discover a service's endpoint dynamically. **Refers to principles**: 15

5. Service Endpoint Policies

15 The following policies aim at reducing the configuration effort at transport level on the one hand and reducing implementation costs by avoiding repeated implementation of message interpreters on the other hand.

Policy ID	Policy description
4.1	**Service messages are transported by HTTP or HTTPS over TCP** Service messages MUST be transported either by HTTP or HTTPS over TCP. **Rationale** These protocols are the most promising when it comes to overcome network boundaries without the need to reconfigure firewalls to an unduly extent. **Refers to principles**: 15
4.2	**Service messages use the SOAP protocol (any version)** For calling a service's functionality, the message MUST be structured according to the SOAP protocol (any version). **Rationale** Since in every case a service is called by sending a message, the client and the service must reach an agreement on how to build this message (including functional and non-functional aspects: method calls and data on the one hand, infrastructural things such as security or routing and addressing on the other hand). So the most important thing for reducing complexity is to define a single enterprise wide message format for service calls. SOAP (in all versions) as on open industry standard, implemented by most vendors and Open Source Projects, and having all the necessary capabilities, is just the right thing for this purpose. Such a standard unburdens the architects from redefining a proprietary message format, and the developers from re-implementing message interpreters for every single service. Among the required capabilities of the message protocol is the possibility to specify *what* with the data *has to be done*. Such an information cannot be implicit to a mere data document: one cannot presume that a service when receiving a data object can do only one or few, however universally predefined things with it (expect CRUD operations which are not worth being a service as not representing any business logic). So, either one reinvents a message protocol with these descriptive capabilities, and so he reinvents SOAP, or one directly uses SOAP. **Refers to principles**: -

6. Security Policies

20 The following policies ensure authenticated service access without the need for the end user to interactively provide credentials, neither for domain internal nor external service access.

These policies are described in greater detail in the *SOA Infrastructure Security Essentials.pdf* document.

Policy ID	Policy description
5.1	**Authentication is proved at message level according to the WS-Security standard** If services require authentication, they MUST verify user authentication at message level, according to the WS-Security standard. **Rationale** Since web servers may offer on the same HTTP(S) listener not only a web service, but, for example, also the service description in form of a WSDL document, different authentication needs may arise for web server access on the same port: as policy 2.2 specifies, the service description must be accessible without authentication for dynamic proxy creation scenarios, while the service (reachable over the same transport), may require authentication. Authentication at message level solves the problem without burdening the deployment staff with complicated and cumbersome separation of service description and service endpoints. **Refers to principles**: 12, 15

5.2	**Domain internal authentication is exclusively proved by Kerberos V.5 tokens**
	
	Kerberos V.5 tokens are the only allowed tokens, integrated into the message's SOAP header, for proving authentication within the domain of the Autonomous Province of Bolzano. The following rules are in place:
	
	▪ The tokens MUST be forwardable and must be suitable for Kerberos delegation.
	
	▪ The Service Principal Name (SPN) used for requesting the Kerberos token MUST be built dynamically at runtime and without relying on static configuration files, according to the following rule:
	
	`HTTP/<servicehost>.<realm>[:<port>]@<realm>`
	or
	`cee/<servicehost>.<realm>@<realm>`
	
	where
	
	<> denote placeholders and are not part of the SPN
	
	[ ]are optional elements

	`servicehost` is the plain name of the host found in the service endpoint address (hostname without domain name) `realm` is the Kerberos realm of the service consumer `port` is a TCP port. SPN resolution MUST try a fallback on `HTTP/<servicehost>.<realm>@<realm>` when `HTTP/<servicehost>.<realm>:<port>@<realm>` did not return a Kerberos token. **Rationale** Kerberos tokens are perfect for being transported without the need of further encryption and integrity assurance. They don't require that client credentials be presented and are therefore an excellent means for realizing SSO scenarios. Dynamic SPN building is necessary for completely delegating service account handling to the operations staff: dynamic SPN building allows to run different service instances with different service accounts. **Refers to principles**: 12, 15, 16
5.3	**Domain authentication from the outside is done by plain WSS Username tokens over HTTPS** Domain services accessed from outside the domain MUST accept Username tokens, integrated into the message's SOAP header according to the WSS Standard. Transport for such services MUST be secured (HTTPS). **Rationale** Kerberos tokens cannot be issued by systems that are not members of an un-trusted destination Kerberos realm. Username tokens are the most simple and therefore the most inexpensive way to authenticate users from the outside. **Refers to principles**: 12, 15, 16
5.4	**Automatic domain boundaries discovery** Service consumers MUST be able to automatically discover at runtime and without relying on static configuration files – starting from the destination hosts endpoint address - if a service is part of the own security domain (which corresponds to a Kerberos realm). This includes automatic, dynamic discovery of the current Kerberos realm. **Rationale** The name of the current Kerberos realm cannot be hard coded for the fact that for the Autonomous Province of Bolzano different Kerberos realms will be in place for preproduction and production systems. This policy saves the operation staff from unduly configuration effort when deploying service consumers. **Refers to principles**: 12, 15

25