Extending Radia into a Service Delivery Controller

Sven Graupner, Tilo Nitzsche
Internet Systems and Storage Laboratory
HP Laboratories Palo Alto
HPL-2005-52
March 11, 2005*

E-mail: sven.graupner@hp.com, tilo.nitzsche@hp.com

Adaptive Enterprise, system management, software deployment, model-based automation

This paper shows how a Service Delivery Controller (SDC) [1,11] can be created for Radia such that Radia can participate and operate as part of a broader management context that is highly automated and driven by external policy. The concept of a SDC has been introduced as building block of the Conceptual Architecture for Management of the Adaptive Enterprise [2] to provide uniform abstractions and the integration framework for different management systems participating in a solution.

Like other management systems, Radia today primarily relies on an administrator role for configuring its policy. With SDC, "programmability" can be introduced for an entire management platform such as Radia, enabling substantially more automation driven from outside systems rather than administrators.

The Radia SDC follows the SDC conceptual architecture. It includes an abstracted, externally exposed "Shared Model" upon which an interface (based on web services management standards) provides operations, which in turn drive changes in the underlying system utilizing the existing (unchanged) Radia infrastructure. Two alternatives for implementing the shared model are discussed: Web-Services Distributed Management (WSDM) and the Resource Description Framework (RDF).

Extending Radia into a Service Delivery Controller

Sven Graupner, Tilo Nitzsche

OST/HP Labs and SGBU/MSO

sven.graupner@hp.com, tilo.nitzsche@hp.com

Abstract

This paper shows how a Service Delivery Controller (SDC) [1,11] can be created for Radia[1] such that Radia can participate and operate as part of a broader management context that is highly automated and driven by external policy. The concept of a SDC has been introduced as building block of the Conceptual Architecture for Management of the Adaptive Enterprise [2] to provide uniform abstractions and the integration framework for different management systems participating in a solution.

Like other management systems, Radia today primarily relies on an administrator role for configuring its policy. With SDC, "programmability" can be introduced for an entire management platform such as Radia, enabling substantially more automation driven from outside systems rather than administrators.

The Radia SDC follws the SDC conceptual architecture. It includes an abstracted, externally exposed "Shared Model" upon which an interface (based on web services management standards) provides operations, which in turn drive changes in the underlying system utilizing the existing (unchanged) Radia infrastructure. Two alternatives for implementing the shared model are discussed: Web-Services Distributed Management (WSDM) and the Resource Description Framework (RDF).

1. Overview

Problem. The reality in IT systems today is that the different aspects of management are performed by dedicated systems that are part of a solution stack. For example, a typical IT solution may be instrumented and monitored by components from OpenView, software configurations may be managed by Novadigm/Radia, virtual machines, if present, may be managed by the virtual machine management system, etc. Integrating participating management systems into a solution today requires substantial manual effort and continuous maintenance. In case of change, a number of affected systems and management systems must be reconfigured, often manually. Automation is achieved through scripts, which are hard to maintain and become increasingly brittle over time..

Approach. In [2], a Conceptual Architecture for Management of the Adaptive Enterprise introduces a set of concepts and abstractions that provide better integration among management systems in a solution stack. We apply the SDC concept, following the SDC Specification [11], to Radia and propose a design for a Radia SDC.

SDC. A *Service Delivery Controller* is a management endpoint in a Service-Oriented Architecture through wich management tasks can be performed on managed services. Core characteristics of SDC are that they provide unifom interfaces through which changes to models are passed which lead to changes in the underlying managed services. Models and changes to models drive the changes made to managed services. SDC themselves can be composed. SDCs enable modularization and functional separation between different management aspects through the use of explicit models and uniform interfaces.

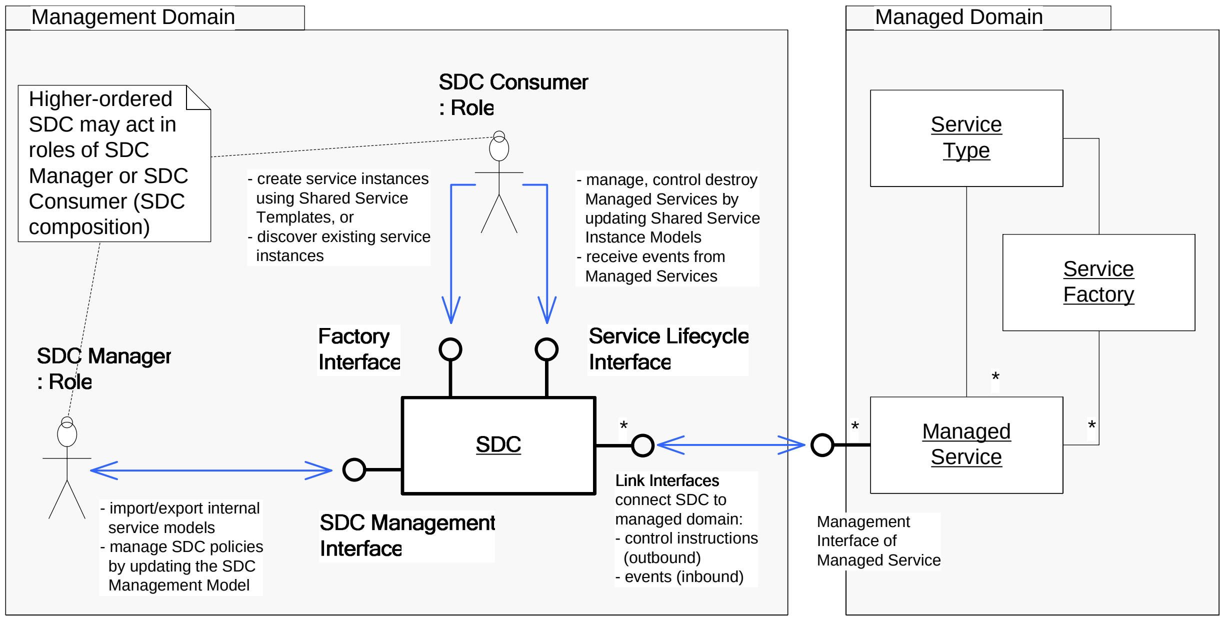
Radia. Radia[1] is a model-based software deployment and configuration system. Radia maintains its own internal model about an environment in regard to its software configurations (applications, packages, images, etc.) and entitlements (to users, systems, etc.). Radia today relies on (human) roles of administrator and publisher for defining policy.

Radia+SDC. The notion of a SDC encompasses entire (large and complex) management systems such as Radia and allows them to be viewed and integrated with other management products. In a management environment, a Radia SDC would provide software deployment and configuration services on sets of machines (the aspect primarily managed by Radia) via a small set of uniform interfaces. Radia would expose a *Shared Service Model Template* through which changes can be made on its internal model, which in turn would result in execution of changes in the environment managed through Radia. The Shared Service Model Template enables abstraction from *how* Radia operates intenally and the models it uses internally to the description and delivery of a function, its *service*, for managing software configurations and entitlements.

[1] Radia is the software deployment and configuration management system from Novadigm (acquired by HP in 2004).

2. Service Delivery Controller

A Service Delivery Controller is a management endpoint in a Service-Oriented Architecture through wich management tasks can be performed on managed services. Figure 1 shows the external relationships of an SDC.



Management Domain
Managed Domain
Higher-ordered SDC may act in roles of SDC Manager or SDC Consumer (SDC composition)
SDC Consumer : Role
- create service instances using Shared Service Templates, or
- discover existing service instances
- manage, control destroy Managed Services by updating Shared Service Instance Models
- receive events from Managed Services
Factory Interface
Service Lifecycle Interface
SDC Manager : Role
SDC
SDC Management Interface
- import/export internal service models
- manage SDC policies by updating the SDC Management Model
Link Interfaces connect SDC to managed domain:
- control instructions (outbound)
- events (inbound)
Service Type
Service Factory
Managed Service
Management Interface of Managed Service
*
*
*
*

Figure 1: External relationships of SDC [11].

Models play a central role in SDC for representing management information for Managed Services in SDC. Models are classified in two dimensions. The first categorization of models maintained in SDC for Managed Services is:

- Service Template - for specifying Policy that applies to a Managed Service defining new "desired state". Templates exist at design stages (Managed Service to be created does not exist yet) and operational stages (during run-time of the Managed Service). Only SDC Consumer roles can read and write Service Model Instances.
- Service Model Instance- for representing the "current state" of the associated Managed Service. Service Model Instances exist only when the Managed Service exists (in a sense that is has state associated in the Managed Domain). Changes to Service Model Instances can only occur in effect of state changes in the Managed Service (e.g. reported by events from the Managed Service). SDC Consumer roles can only read the Service Model Instance.

The second categorization of models maintained in SDC for Managed Services is:

- Shared Model - defines the information that is externally shared with other SDC in SDC Consumer roles for the Service Template and the Service Model Instance of a Managed Service. The Shared Model defines the "view", SDC Consumer roles have on Managed Services through the SDC.
- Internal Model - defines the information that is internally maintained in the SDC for the Service Template and the Service Model Instance. The Internal Models are not exposed

Shared Models have been introduced mainly for the following to reasons, to abstract from detail and amount of information a SDC may need to maintain internally for performing management tasks and to encapsulate proprietary internal models used by legacy management systems allowing them to integrate at model-level in SDC management environments.

It may occur that Shared Models and Internal Models are the same. In the case of Radia, the Internal Model of Radia exists in form of the (proprietary) model maintained in Radia's Confirgration Service (RCS). The following sections will show how a Shared Model can be defined for Radia along with the representation of the Shared Model in two frameworks: WSDM and RDF.

The combination leads to the four models that are maintained in a SDC for a Managed Service:

- Shared Service Model Template – reflects the desired state of the Managed Service.
- Shared Service Model Instance – reflects the current state of the Managed Service.
- Internal Service Model Template – reflects the internal representation of desired state for the Managed Service.
- Internal Service Model Instance – reflects the current state of the Managed Service in the internal model.

The model of interest that is exposed from the Radia SDC is the Shared Service Model Template.

3. Extending Radia into a SDC

Figure 2 shows the overall system with managed machines at the bottom. Machines have software configurations (shown as "State") managed by Radia. Radia components are shown in the center of the figure. Above, the Radia SDC contains the Shared Model and provides the API to an external management system.

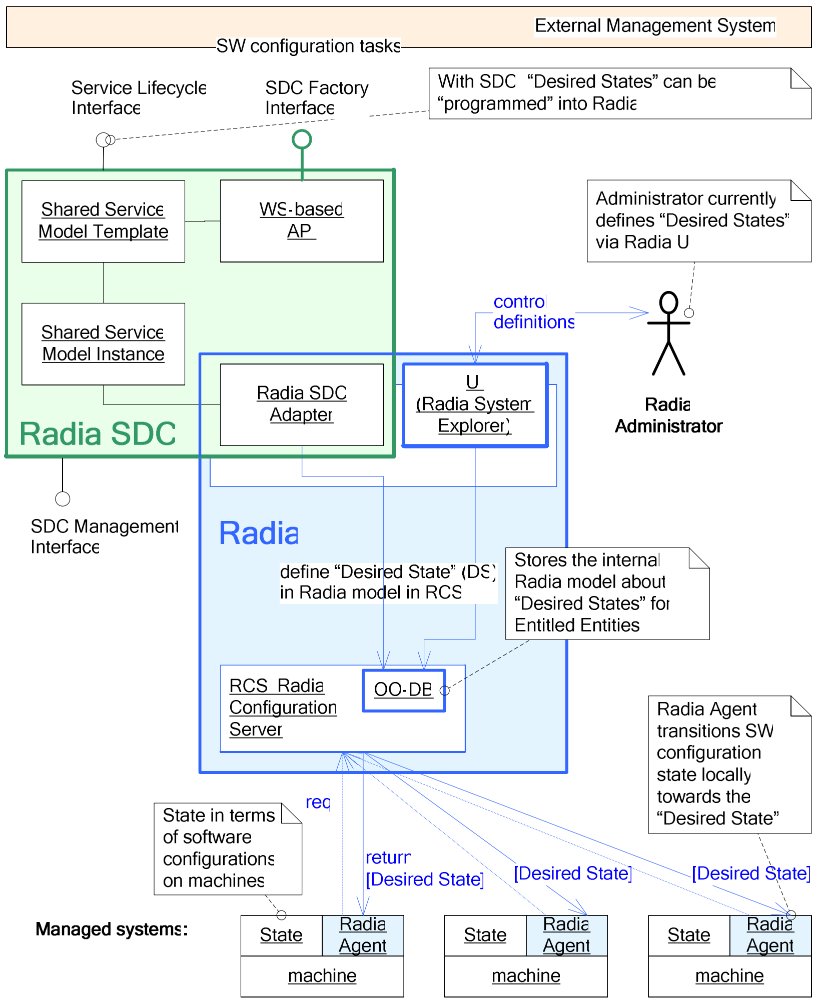

External Management System
SW configuration tasks
Service Lifecycle Interface
SDC Factory Interface
With SDC "Desired States" can be "programmed" into Radia
Shared Service Model Template
WS-based API
Administrator currently defines "Desired States" via Radia UI
control definitions
Shared Service Model Instance
Radia SDC Adapter
UI (Radia System Explorer)
Radia SDC
Radia Administrator
SDC Management Interface
Radia
define "Desired State" (DS) in Radia model in RCS
Stores the internal Radia model about "Desired States" for Entitled Entities
RCS Radia Configuration Server
OO-DB
Radia Agent transitions SW configuration state locally towards the "Desired State"
req
State in terms of software configurations on machines
return [Desired State]
[Desired State]
[Desired State]
Managed systems:
State
Radia Agent
machine
State
Radia Agent
machine
State
Radia Agent
machine

Figure 2: Radia-managed environment with Radia SDC.

The Radia agent on each system requests information about the "Desired State" of that system and compares it to its current state. In case of difference, the agent per-forms actions to transition this state toward the Desired State by installing, upgrading or removing packages. Desired State definitions for classes of systems or users, which are subsequently referred to as "Entitled Entities", are defined in the model in the Radia Configuration Server (RCS).

This model is maintained in an object-oriented database (OO-DB). The Radia administrator uses a UI tool, the Radia System Explorer, to set Desired State definitions in RCS.

The Radia SDC also manipulates the Radia model in RCS via the Radia SDC Adapter. The Radia SDC includes a "Shared Model" which is externally exposed through the WS-based API. Section 2 defines this model. Operations upon the Shared Model drive changes in the underlying Radia model, which in turn drive changes on managed systems. The Shared Model maintains run-time state in the Shared Model Repository. A WS-based API provides access to the Shared Model state. This is the external API of the Radia SDC is described in section 3.

SDC differentiate between *Internal Models*, which refer to the detailed information each underlying management system maintains for its own operation, and *Shared Models*, which contain only the information needed for the external interaction with the SDC reflecting its functions (or services). SDC hide internal models. Shared models are exposed via the API. Operation of a SDC is controlled by operations over the Shared Model.

4. Shared Service Model Template for Radia SDC

Radia's operation is based on defining Desired State in terms of software configurations for software services (comprised of Applications, Packages, and Resources). Those software services can be assigned (entitled) to "Entitled Entities" (users, systems or groupings of them). The entitlement association is also referred to as policy in Radia. The key task of a Radia administrator is to define, manipulate and dissolve Entitled Entities, Software, and entitlements among them.

The Radia SDC exposes these operations through an interface. Internally, the SDC transforms operations into manipulations over the Shared Service Model Template that are propagated into Radia's internal model in RCS.

Desired State. The Shared Service Model Template contains Desired State in the following form: Desired State for an Entitled Entity $ee \in EE$ is defined as the subset of services $S_{SD} \subseteq S$ for which an entitlement association $ea \in EA$ exists.

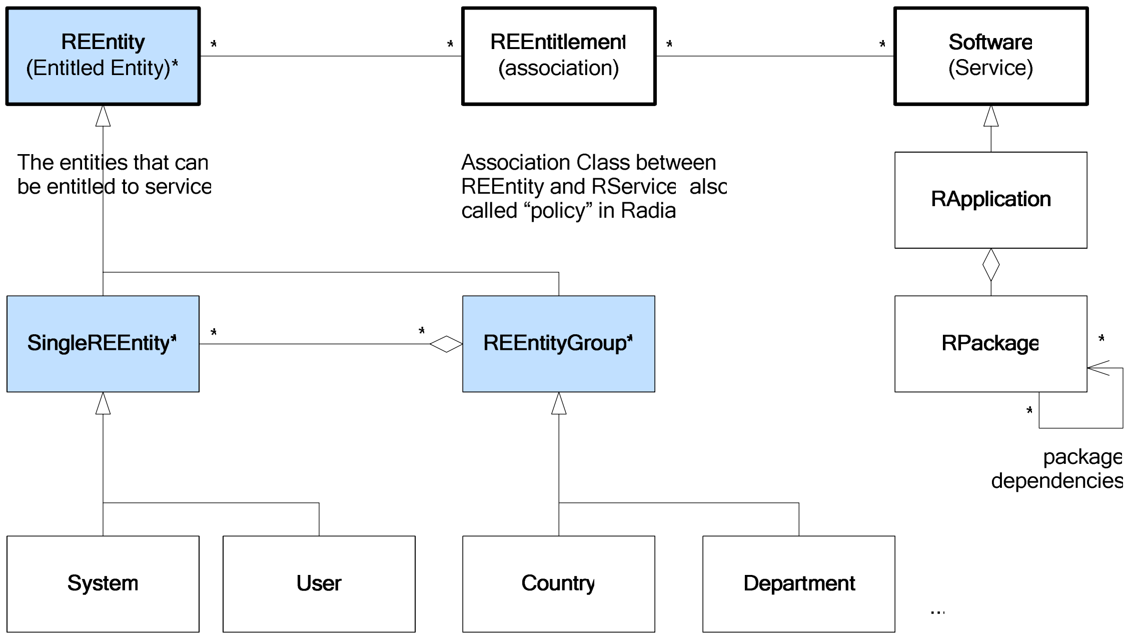


Figure 3: UML Representation of the Shared Service Model Template for Radia SDC.

The following sections will present two techniques for presenting the Shared Service Model Template: Web Services Distributed Management (WSDM) [6] and the Resource Description Framework (RDF) [9].

5. SDC Model Representations

5.1 Web Services Binding for the SDC

The SDC interface is data-driven, thus a Web Services standard like WS-RF (WS-ResourceFramework) which is focused on explicitly exposing the state of a web service is an excellent fit for a model interface. This WS-RF web service that explicitly exposes state is also called a Web Service Resource.

The OASIS WSDM (Web Services Distributed Management) MUWS (Management Using Web Services) standard extends WS-RF, most importantly providing a way to express relationships between the Web Services exposing the model.

Standard web services are generally uniquely identified by the URL that is used to access them. In WS-RF, typically multiple web services will share the same access URL (web services endpoint). The service (WS Resource) is uniquely identified by an EndpointReference.

The following Web Service interfaces are provided:

- Model Container service that provides access to the whole model (a WSRF Endpoint Reference to this service is returned by the SDC Management interface operation getModel());
- Services for each class in the model, in the most abstract case for 'REEntity', 'REntitlement' and 'Software'; these services represent the classes themselves
- Services for each instance in the model, in the most abstract case for each instance of 'REEntity', 'REntitlement' and 'Software'

Linkages between these services are established via WSDM MUWS relationships. The Model Container has a 'contains' relationship to each of the model classes. Each of the classes has 'instances' relationships to each instance of the respective class. Relationships between REntitlement instances and REEntity/Software represent the associations shown in the UML model (association REntitlement <-> REEntity; REntitlement <-> Software).

Figure 4 shows a picture of the various services and the WSDM relationships.

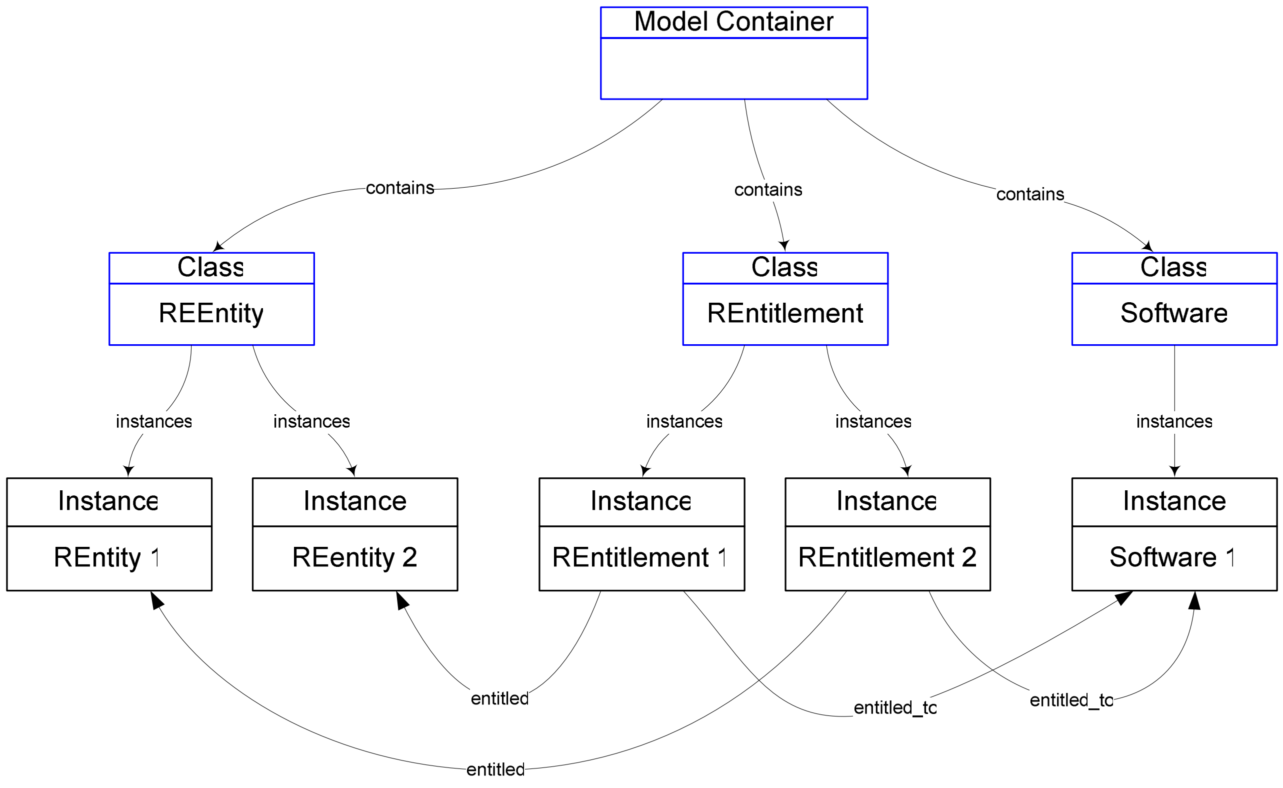


Figure 4: WSDM Web Services for Radia Model.

These web services do not provide any specific operations. They expose the model data via WS-RF ResourceProperties. These ResourceProperties can have an arbitrary XML schema type, with the only restriction that the top element must be a GED (Global Element Declaration). A Resource Property can have an arbitrary cardinality. The most common use case will be that each property of a UML class will be mapped to one ResourceProperty.

The collection of resource properties that a web service exposes is called property document. This property document is a virtual XML document that can be accessed via a set of standard methods:

- GetResourceProperty: return value of a single resource property
- GetMultipleResourceProperties: return multiple resource properties (each specified by name)
- QueryResourceProperties: run query against property document, return query result; one possible query language is XPath allowing extremely powerful filtering of the property document
- SetResourceProperties: change the value of one/multiple, add or delete a ResourceProperty

WSDM MUWS exposes relationships as ResourceProperties. The standard resource property operations can be used to access them.

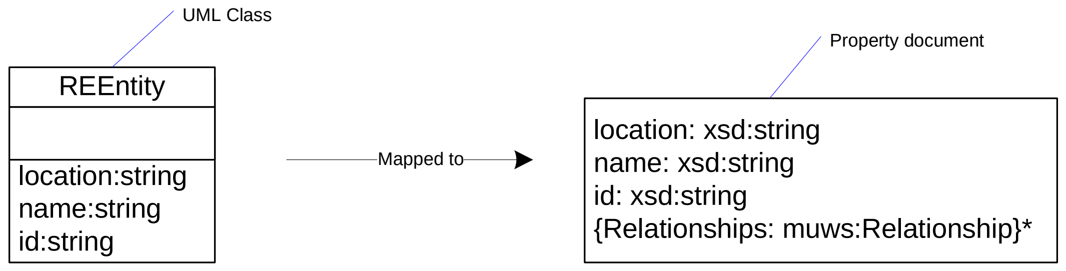


Figure 5: WS-RF Property Document.

No SDC-specific operations are exposed. The SDC is controlled by modifying the model data by updating the property document (using standard WS-RF/WSDM operations). The lifetime of a WS Resource is controlled as defined in the WS-ResourceLifetime standard (it provides a destroy method and timed destruction). New class instances can be created via a generic ‘createInstance()’ operation (factory method) on the Class WS Resources (e.g. the REEntity class web service will provide the ‘createInstance()’ method that can be invoked to create a new REEntity instance).

The WS-RF suite of specifications provides very sophisticated event mechanism with WS-Notification. A standard notification called ResourcePropertyValueChangeNotification is provided so that event consumers can subscribe for changes on any particular resource property. Event consumers can provide a filter expression (e.g. based on XPath), so that they only receive the notifications they are really interested in. Thus, model consumers can selectively subscribe for change notifications for any part of the model they are interested in.

5.2 RDF Binding for the SDC

RDF can be used as an externalized model representation of the SDC model. The SDC Management Interface operations getModelSchema() and getModel() will return a reference to where the RDF model schema and RDF model can be obtained.

The RDF model schema is expressed in RDF/S (RDF Schema), a schema language for RDF.

Figure 6 shows the RDF/S schema for the most abstract subset of the Radia SDC. Classes are similar to the classes in the UML model. ObjectProperties and DataTypeProperties represent attributes on the classes pointed at with a ‘domain’ arrow.

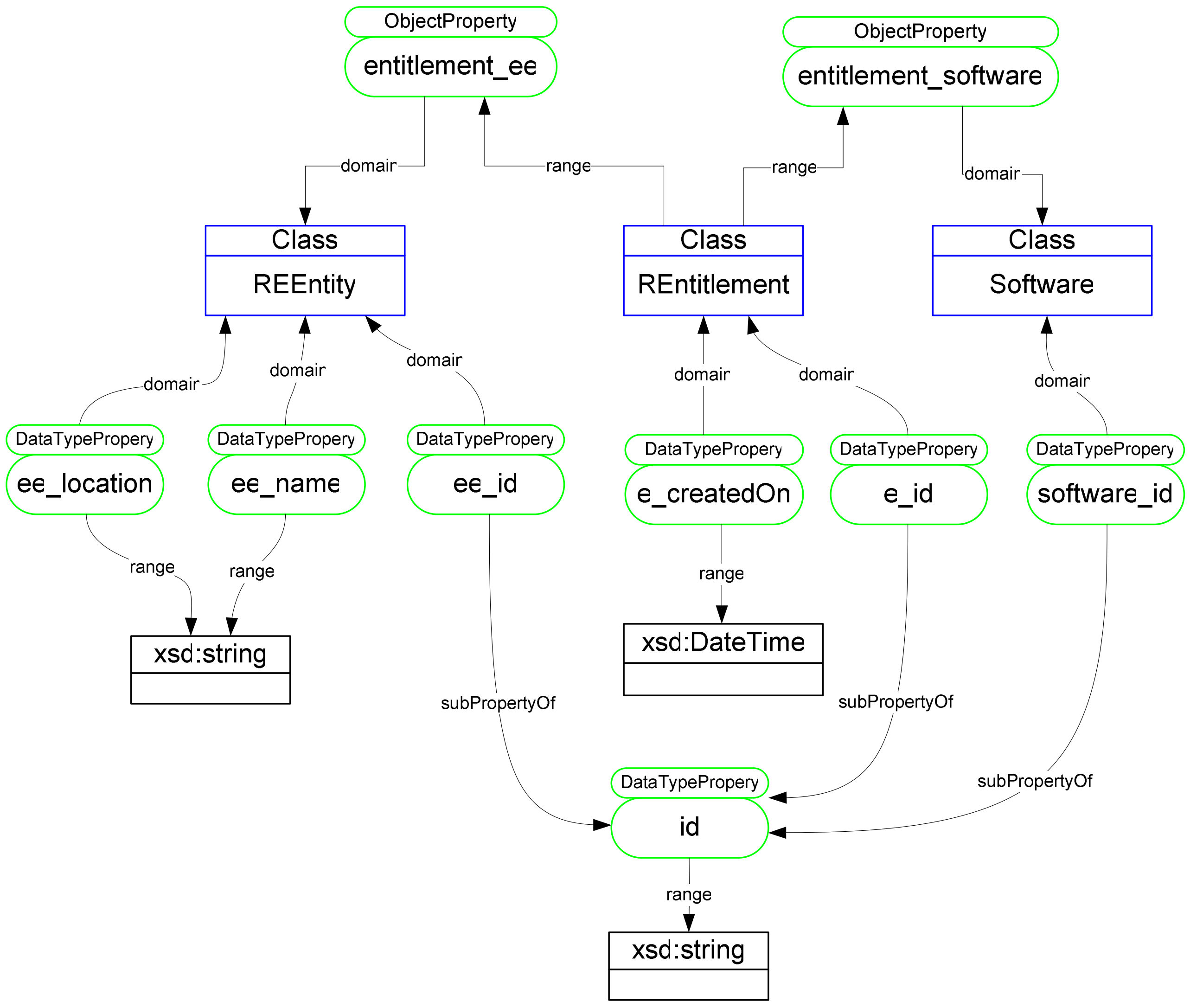


Figure 6: RDF ontology for Radia SDC.

Properties are first class citizens in RDF/S and thus form an own type hierarchy. ‘ee_id’, ‘e_id’ and ‘software_id’ are all sub properties of ‘id’.

Note that property names are not local to a class as in most other model representation forms. The ‘domain’ arrow means that anything that has a property of the type the arrow is originating from is of the class type it points to. Thus, unique property names must be used, unless multiple inheritance is implied.

The ‘range’ arrow defines the type of a property. The name ‘range’ can be confusing; it is the equivalent of the attribute type in UML.

RDF/S does not allow defining the cardinality of a property. Each property can have any cardinality. OWL (Web Ontology Language), which is an extension of RDF/S allows the definition of cardinality restrictions. Figure 7 shows an example, where the ‘ee_id’ property of ‘REEntity’ is defined to have exactly one value.

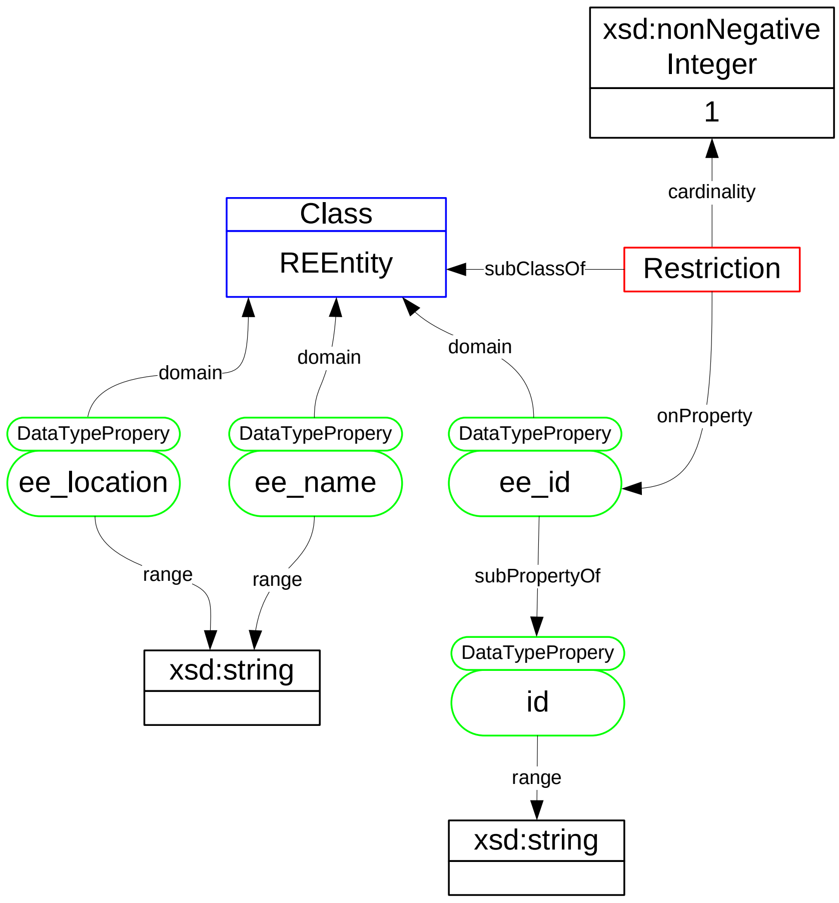


Figure 7: OWL cardinality restriction

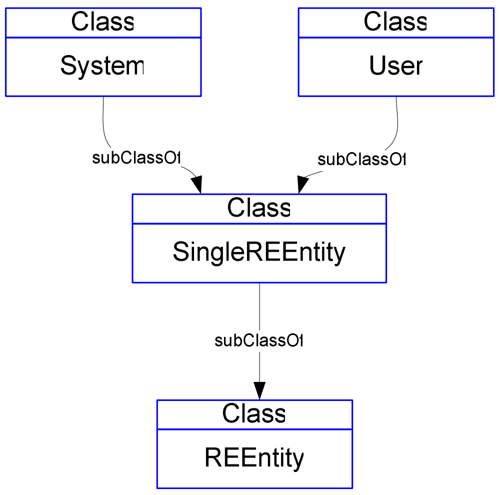


Figure 8: RDF ontology, inheritance

Figure 8 shows how class inheritance is represented in the RDF model.

The 'normal' model schema (as is used for the model returned by 'getModel()') can be used for the 'update()' operation of the SDC management interface.

Update operations will take an RDF change set as parameter. RDF documents are additive, thus if two RDF documents are simply merged, the new model will contain everything that was in the two documents. If for example the 'ee_id' of a 'REEntity' is supposed to be changed and the RDF document with the new ID is merged, the combined RDF model would now have two IDs.

Two RDF documents, which together comprise the RDF change set, are used to perform a model update. The first one will contain the information that is supposed to be removed from the model, the second one information that is added. In the case of the ID change example, the first document would contain the old ID (so that it can be removed); the second one would contain the new value.

Alternately, the 'update()' operation could use a more complex schema that will mark which parts of the model are supposed to be additions, changes or removals from the model.

The XML representation of the RDF schema is shown in Figure 9. It contains the RDF schema as shown in Figure 6 and Figure 10.

```
<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE rdf:RDF [
 <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-
syntax-ns#'>
 <!ENTITY kb 'http://radia.hp.com/kb#'>

 <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-
schema#'>
]>

<rdf:RDF xmlns:rdf="&rdf;"
         xmlns:kb="&kb;"
         xmlns:rdfs="&rdfs;">

<rdfs:Class rdf:about="&kb;REEntity"
            rdfs:label="REEntity">
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>

<rdfs:Class rdf:about="&kb;REntitlement"
            rdfs:label="REntitlement">
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>

<rdfs:Class rdf:about="&kb;Software"
            rdfs:label="Software">
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>

<rdfs:Class rdf:about="&kb;WSDMEndpoint"
            rdfs:label="WSDMEndpoint">
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>

<rdf:Property rdf:about="&kb;createdOn"
              rdfs:label="createdOn">
  <rdfs:domain rdf:resource="&kb;REntitlement"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&kb;e_epr"
 rdfs:label="e_epr">
  <rdfs:subPropertyOf rdf:resource="&kb;epr"/>
  <rdfs:domain rdf:resource="&kb;REntitlement"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&kb;e_id"
              rdfs:label="e_id">
  <rdfs:domain rdf:resource="&kb;REntitlement"/>
  <rdfs:subPropertyOf rdf:resource="&kb;id"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&kb;ee_epr"
 rdfs:label="ee_epr">
  <rdfs:subPropertyOf rdf:resource="&kb;epr"/>
  <rdfs:domain rdf:resource="&kb;REEntity"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&kb;ee_id"
              rdfs:label="ee_id">
  <rdfs:domain rdf:resource="&kb;REEntity"/>
  <rdfs:subPropertyOf rdf:resource="&kb;id"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&kb;ee_location"
 rdfs:label="ee_location">
  <rdfs:domain rdf:resource="&kb;REEntity"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&kb;ee_name"
              rdfs:label="ee_name">
  <rdfs:domain rdf:resource="&kb;REEntity"/>
  <rdfs:domain rdf:resource="&kb;Software"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&kb;entitlement_ee"
              rdfs:label="entitlement_ee">
  <rdfs:range rdf:resource="&kb;REEntity"/>
  <rdfs:domain rdf:resource="&kb;REntitlement"/>
</rdf:Property>

<rdf:Property rdf:about="&kb;entitlement_software"
              rdfs:label="entitlement_software">
  <rdfs:domain rdf:resource="&kb;REntitlement"/>
  <rdfs:range rdf:resource="&kb;Software"/>
</rdf:Property>

<rdf:Property rdf:about="&kb;epr"
              rdfs:label="epr">
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&kb;id"
              rdfs:label="id">
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&kb;software_epr"
              rdfs:label="software_epr">
  <rdfs:domain rdf:resource="&kb;Software"/>
  <rdfs:range rdf:resource="&kb;WSDMEndpoint"/>
  <rdfs:subPropertyOf rdf:resource="&kb;epr"/>
</rdf:Property>

<rdf:Property rdf:about="&kb;software_id"
              rdfs:label="software_id">
  <rdfs:domain rdf:resource="&kb;Software"/>
  <rdfs:subPropertyOf rdf:resource="&kb;id"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

</rdf:RDF>
```

Figure 9: XML representation of the RDF schema.

5.3 Combined RDF and WSDM Binding for the SDC

The two approaches of using RDF and WSDM can be combined. The WSDM container resource will provide access to the RDF model, so that the model can be downloaded and queried (to access a subset of the model). A semi-standard query language called RDQL or SPARQL (which is currently under development by W3C) could be used.

The same WSDM managed objects as before are implemented. The RDF model will contain references to the MOs as shown in Figure 10. The ‘REEntity’, ‘REntitlement’ and ‘Software’ classes each have an attribute that can contain a WSDM endpoint reference.

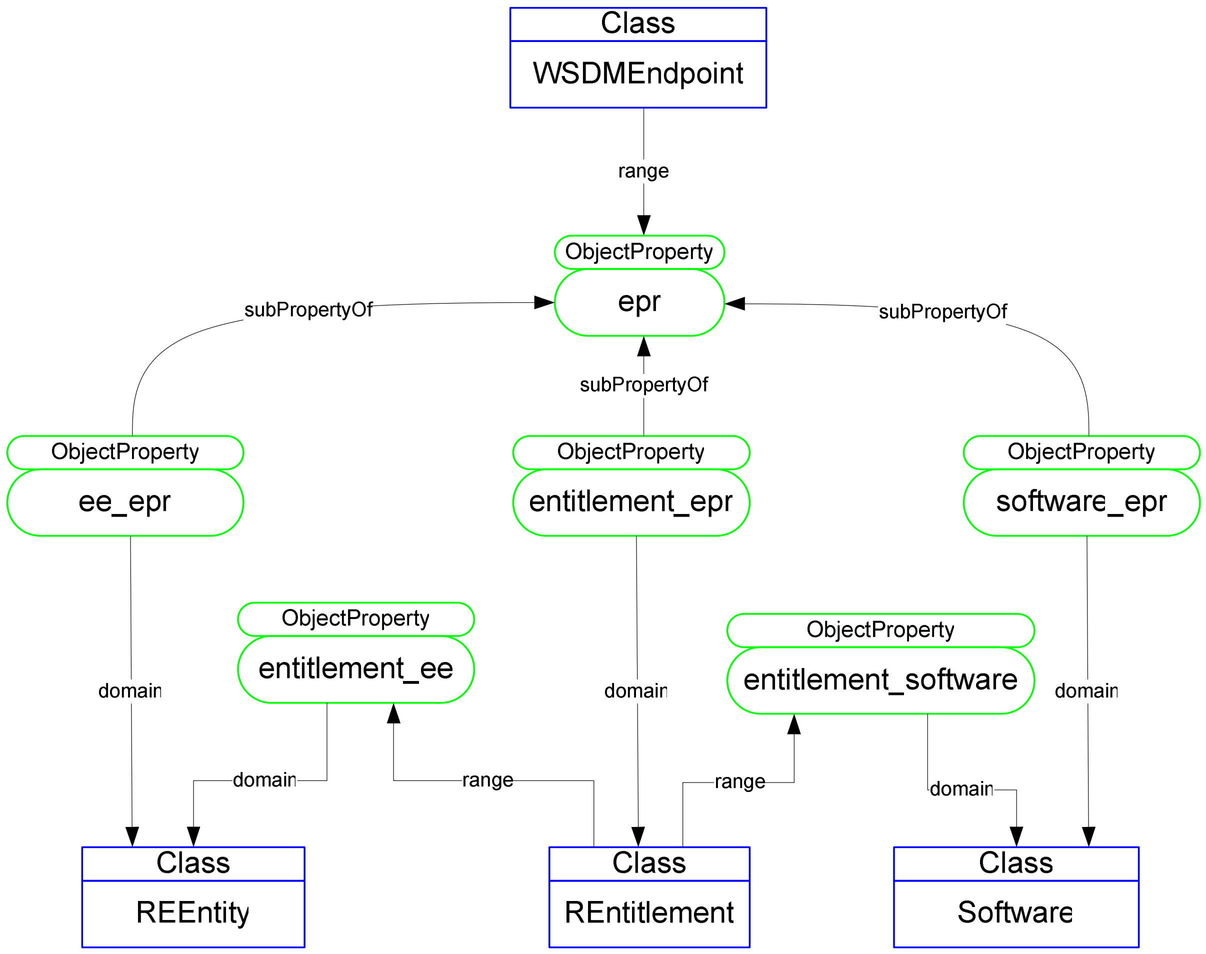


Figure 10: RDF ontology with WSDM linkage

References

[1] HP: Management for the Adaptive Enterprise: Service Delivery Controller, September 2004.

[2] HP: Management for the Adaptive Enterprise: Conceptual Architecture, July 2004.

[3] HP Labs: Quartermaster, http://webnt.hpl.hp.com/quartermaster.

[4] DMTF: Common Information Model (CIM), http://www.dmtf.org.

[5] OASIS WSDM: Management Using Web Services (MUWS 1.0), Working Draft September 29, 2004.

[6] WS-RF: Web Services Resource Framework, http://devresource.hp.com/drc/specifications/wsrf/index.jsp.

[7] WS-ResourceLifetime: http://devresource.hp.com/drc/specifications/wsrf/index.jsp.

[8] RDF Primer: http://www.w3.org/TR/rdf-primer/.

[9] RDF/S: http://www.w3.org/TR/rdf-schema/.

[10] Web Ontology Language (OWL): http://www.w3.org/TR/owl-features/.

[11] Graupner, Thompson, Singhal: SDC Specification, HP internal document, to be published.