SOA - Repository Artifact Model and Protocol Specification (S-RAMP)

Foundation

International Business Machines Corporation
Hewlett-Packard Corporation
Software AG
TIBCO Software Inc

Revision 1.0
October 7, 2010

Preface

Authors (alphabetically)

Tom Bellwood, IBM
Amardeep Bhattal, IBM
Vince Brunssen, IBM
John Colgrave, IBM
Martin Dvorak, HP
Dan Enache, TIBCO
Steve Fanshier, SAG
Eric Johnson, TIBCO
Diane Jordan, IBM
Bernard Kufluk, IBM
Miroslav Novak, HP
Christopher Peltz, HP
Radek Pospisil, HP
Albert Regner, HP
Martin Smithson, IBM
Gary Woods, SAG
Prasad Yendluri, SAG

Copyright Notice

Abstract

Vendors offer tools to facilitate various activities across the life cycle of a SOA artifact, such as design, assembly, quality assurance, deployment and runtime operation of SOA based applications and business processes. The lack of a standardized information model and interaction protocol for artifacts and their metadata residing in a SOA repository, means that tools must be customized for use with each different vendor's SOA repository product. This reduces choice, flexibility and adds costs for customers when choosing tools. This specification defines a SOA artifact data model together with bindings that describe the syntax for interacting with a SOA repository.

Table of Contents

List of Figures

List of Tables

List of Examples

Introduction

1.1 Purpose and Scope

The “SOA - Repository Artifact Model and Protocol” (S-RAMP) specification defines a common data model for SOA repositories to facilitate the use of common tooling and sharing of data. It provides a rich representation data model that supports query. It includes binding(s) which document the syntax for interaction with a compliant repository for create, read, update, delete and query operations within the context of each binding. Initially, only one binding will be defined, but others can be added.

The specification is organized into multiple documents. This document, the SOA - Repository Artifact Model and Protocol Foundation (hereafter referred to as *Foundation*) describes the overall goals of S-RAMP and defines the Artifact Type Model and associated schemas for S-RAMP. It also describes the generic query grammar used in S-RAMP. The Foundation document is not specific to any binding. Other documents in this specification provide material specific to a given binding for S-RAMP, including the syntax for interacting with an S-RAMP compliant repository. Those available at the time of this publication are:

- S-RAMP Atom Binding

When there is a discrepancy between a binding specific document and this Foundation document, the binding specific document always takes precedence. If there is a discrepancy between schema representations provided in this document and the S-RAMP schemas of record on s-ramp.org, the schemas of record SHALL take precedence.

1.2 Problem Statement and Objectives

Service Oriented Architecture (SOA) is an architectural approach to designing applications and business processes by consuming business logic from reusable software components exposed as network accessible services. In today’s environment, vendors offer tools to facilitate various activities across the life cycle of a SOA artifact, such as design, assembly, quality assurance, deployment and runtime operation of SOA based applications and business processes. The SOA repository provides the foundation for all these activities. This specification which we term the SOA - Repository Artifact Model and Protocol (S-RAMP) codifies how SOA information models are represented by artifacts (including their associated metadata) in a SOA repository. This specification also defines bindings to support interaction with the SOA repository, including create, read, update, delete, query, and subscription for notifications. It defines the Artifact Type Model and provides various bindings which define the syntax needed to support interaction with a SOA repository. This approach to providing flexible access to SOA artifacts will facilitate interoperability and provide customers with more choices of tools that can be used to interoperate with any S-RAMP compliant SOA repository implementation.

1.3 Use Case Scenarios

Table 1, Table 2 and Table 3 below provide some examples across different portions of the service lifecycle for which there are various use cases in which an S-RAMP compliant repository could be used. This does not necessarily imply that all vendors would support every scenario, or use an S-RAMP repository in each of these scenarios across all portions of the service lifecycle.

Table 1: Design Time Tool Repository Interaction Use Cases

Tool Category	Activities	S-RAMP Feature	Examples
Integrated Development Environment (IDE)	1. Design WSDL and schemas 2. Publish and consume services 3. Publish SCA into repository 4. Shred SCA for impact analysis 5. Notification to service developer when WSDL changes	WsdlDocument XsdDocument OWL classifications PortType	WID RSA Together VisualStudio Oracle JDeveloper
Business Process Modeling Tools	1. Publish WSDL descriptions for processes 2. Look for process entry points 3. Search/find services to use in their business processes 4. Impact analysis	wsdDocument XsdDocument OWL classifications PortType	WebSphere Business Modeling ARIS Platform Oracle (Collaxa) TIBCO Business Studio

Table 2: Run Time Tool Repository Interaction Use Cases

Tool Category	Activities	S-RAMP Feature	Examples
Testing	1. Search to find a WSDL 2. Understand policies associated with WSDL 3. Understand relationships between SOA components 4. Notification when WSDL changes	WsdlDocument Service PolicyAttachment Policy	HP Service Test Manager Rational Testing Tools Itko Lisa
ESB	1. Dynamic routing based on #services, requestor type, etc. 2. Notifications of new artifacts, changes/deletions 3. Track information on lifecycle and operational state (e.g., using classifications, properties, etc.)	PolicyAttachment WSDL Parts Service properties	DataPower WebSphere ESB Oracle Service Bus SAP XI WebMethods TIBCO ActiveMatrix

Policy Mgmt	1. Edit and store policies 2. Query repository for policies to deploy/provision 3. Execute (enforce) policy 4. Update managed endpoint in repository	PolicyAttachment policy service ServiceEndpoint	AmberPoint Actional HP SOA Policy Enforcer CentraSite TIBCO ActiveMatrix Policy Manager

Table 3: Monitoring Tool Repository Interaction Use Cases

Tool Category	Activities	S-RAMP Feature	Examples
Service Monitoring	1. Retrieve service definitions from repository 2. Update service information with performance and availability data 3. Discover dependencies between business services and web service instance 4. Discover what organizations provide a service 5. Discover operational data for the service for monitoring	Organization Service ServiceInstance ServiceOperation user properties	Tivoli CAM for SOA BAC for SOA AmberPoint Actional WebMethods Insight TIBCO ActiveMatrix Service Performance Manager

1.4 References

This specification shall be used in conjunction with the following publications.

W3C, Extensible Markup Language (XML) 1.0 Specification

W3C, Namespaces in XML 1.0 (Second Edition)

W3C, XML Schema Part 1: Structures Second Edition, version 1.0

W3C, XML Path Language (XPath), version 2.0

W3C, RDF Primer and the associated documents listed therein

W3C, OWL Web Ontology Language Guide

W3C – Web Services Description Language (WSDL), version 1.1

ISO 639.2, Codes for the Representation of Names and Languages

The Open Group, SOA Ontology

Web Services Policy 1.5 – Framework

Web Services Policy 1.5 – Attachment

A Universally Unique IDentifier (UUID) URN Namespace Specification, version 4

1.5 Document History

This document, version 0.8, is the first version of this document.

1.5.1 Conventions and Notation

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC 2119].

The character "|" is used to indicate a choice between alternatives.

The characters "[" and "]" are used to indicate that contained items are to be treated as a group with respect to cardinality or choice.

This document makes use of many UML Diagrams and a summary of the notation used in those diagrams is provided here for convenience

- Italicized artifacts are abstract
- Closed arrows indicate inheritance
- Open arrows indicate non containment association relationships
- Filled diamonds ending in open arrows represent composition relationships
- Names associated with arrows indicate Relationship Type names

NOTE: Attributes listed within an artifact box do not indicate cardinality. Users must refer to the document text and schemas for that information.

1.5.2 XML Namespaces

The XML namespace URI that MUST be used by implementations of this specification is:

http://s-ramp.org/xmlns/2010/s-ramp

Table 4 lists XML namespaces that are used in this specification. The choice of any namespace prefix is arbitrary and not semantically significant.

Table 4: Prefixes and XML Namespaces Used in this Specification

Prefix	XML Namespace	Specification(s)
xp2	http://www.w3.org/2005/xpath-functions	XPath 2.0
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	RDF namespace
rdfs	http://www.w3.org/2000/01/rdf-schema#	RDFS namespace
owl	http://www.w3.org/2002/07/owl#	OWL namespace
s-ramp	http://s-ramp.org/xmlns/2010/s-ramp	S-RAMP namespace
wsdl	http://schemas.xmlsoap.org/wsdl/	WSDL [WSDL 1.1]
wsp	http://www.w3.org/TR/2007/REC-ws-policy-20070904	WS-Policy [WS-Policy]
xsd	http://www.w3.org/2001/XMLSchema	XML Schema [Part 1, 2]

1.5.3 Conformance

An implementation is not compliant with this specification if it fails to satisfy one or more of the MUST or REQUIRED level requirements defined herein.

The XML Schemas published at http://s-ramp.org take precedence over the normative text within this specification which takes precedence over the S-RAMP XML Schema in the appendix of this document. The authoritative S-RAMP XML Schema is published at:

s-ramp.org/2010/specification/schemas

1.6 Terminology

Defined Term	Definition
Artifact Type	The data type of an S-RAMP artifact
Artifact Type Model	The set of all Artifact Types used in the S-RAMP specification
Service Implementation Model	Set of S-RAMP Artifact Types and relationships which describe the service implementation layer associated with the SOA Model.
Core Model	Set of basic Artifact Types
Policy Model	Set of Policy document related derived Artifact Types
Relationship	The logical triple of a Relationship Type, Source and Target. Relationships in S-RAMP are all directed from a source, to a target
Relationship Type	A name which represents the type of the relationship (e.g., “includedXsds”). Multiple relationships can share the same Relationship Type.
SOA Model	Set of Artifact Types and relationships used to link The Open Group’s SOA Ontology artifact types with those in the S-RAMP data model.
WSDL Model	Set of WSDL document related derived data types
XSD Model	Set of XSD document related derived data types
User Defined Model	An S-RAMP model whose content and structure has been defined by the client

1.7 Abbreviations and Acronyms

S-RAMP	SOA – Repository Artifact Model and Protocol
UTC	Coordinated Universal Time
XPath2	XML Path Language (XPath) 2.0

1.7.1 Design Principles

There are several high level design principles to which S-RAMP has adhered:

- Use of existing standards where possible (e.g., XML, XML Schema, OWL, XPath2, APP (Atom Publishing Protocol), ASF (Atom Syndication Format), etc.).
- Vendor neutrality.
- Does not include governance models, but may be used by them.
- Driven by use cases.
- Data model extensibility for new data types, and support for system and user defined metadata.
- Inclusion of an XML Schema based serialization for its data model.
- Use of XPath 2 to describe its query grammar.
- Use of OWL Lite to describe its classification system grammar.
- Separation of the data model from the bindings which describe the interaction APIs clients use to interact with the repository.

1.7.2 S-RAMP Schemas

The schemas for the various S-RAMP Models are provided in the appendices . They closely follow the conceptualized UML diagrams described in this document. The normative S-RAMP schemas of record define the serialization for S-RAMP and may be found on the s-ramp.org web site at:

s-ramp.org/2010/specification/schemas

. Notable points concerning the schemas in S-RAMP include:

- Built-in properties are typically represented as attributes.
- Types based on *BaseArtifactType* use an extension of that type as their base.
- Where practical, Global Element Declarations are provided.
- Extensibility in the Core Model is limited to the ##any attribute on most structures.
- Schemas are provided for serialization purposes and the UML diagrams define the S-RAMP meta-model.

Artifact Type Model

2.1 Introduction

The S-RAMP Artifact Type Model is a strongly typed data model for SOA repositories which allows interoperability among vendor repository implementations and tooling, within the context of a specific binding. It will also later serve as a foundation for developing data exchange and federation models. The Artifact Type Model is presented here using conceptualized UML models. Each of these models also has a XML Schema representation available in the Appendices.

2.2 Artifact Type Models

An artifact in S-RAMP is a container for all of the metadata that describes it. There are 4 major types of artifacts in S-RAMP. Each of these Artifact Types is discussed in more detail in later sections:

1. **Document Artifact**: Those which correspond to a physical document stored in the repository. Several important document types are pre-defined and have special support in S-RAMP (such as XML Schema or WSDL documents). But any document type can be placed in the repository.
2. **Service Implementation Model Artifact**: Those S-RAMP defined artifacts which provide a representation of the service implementation layer associated with the SOA Model (such as a ServiceOperation or ServiceEndpoint).
3. **Derived Artifact**: Derived Artifacts (e.g., WSDL PortType, or WS-Policy PolicyExpression) are dynamically instantiated by the server as a consequence of publishing a document instance whose type is one of those supported with a Derived Model (see Table 5). These artifacts cannot be created or deleted directly, although clients can edit them to add or remove Generic relationships, properties and classifications. Derived Artifact Models are managed by the server and kept in synchronization with the document object with which they are associated. Derived artifacts provide a metadata model of the content components of a particular document. This allows much more powerful query capabilities at a granularity specific to the internal components of a document, when it is of a format supported with a Derived Model. Refer to Section 4 for more information on the query model supported in S-RAMP, as well as to the individual binding document(s) for the query syntax pertaining to each binding.
4. **SOA Model Artifact**: Those S-RAMP defined artifacts and relationships as well as those defined in the "SOA Repository View" of The Open Group's SOA Ontology, which provides a conceptual representation of a SOA environment. The SOA Repository View is defined by The Open Groups SOA Ontology draft version 3.2.
5. **User Defined Artifact Model**: These are created by the client and are part of a User Defined Model. The means by which a client specifies such a model are beyond the scope of this specification, but some provision is made within S-RAMP schema to facilitate basic interoperability for such artifacts. Regardless of the internal definition of these artifacts, they SHALL be serialized in S-RAMP as an instance of *UserDefinedArtifactType*, which extends BaseArtifactType.

The pre-defined S-RAMP Artifact Types are organized into a set of logical models as summarized in Table 5 below. Each of these is discussed further in the sections that follow. Note that Derived Artifact Models are currently specified for each of the XSD, WSDL and WS-Policy document types.

Table 5: Artifact Type Models

Model	Purpose
Core	Defines the base data types used by the other models
SOA	Defines the artifact data types and relationships which are used to integrate The Open Group's SOA Ontology object model into S-RAMP's data model.
Service Implementation	Defines the artifact data types and relationships used to model the service implementation layer of a SOA environment.
Derived: XSD	Defines logical artifact data types for an XML Schema document
Derived: WSDL	Defines logical artifact data types for a WSDL document

Derived: SOAPWSDL	Defines artifact data types for the SOAP binding of a WSDL document
Derived: Policy	Defines artifact data types for an WS-Policy document

2.2.1 Artifact Metadata

An artifact can contain three major types of metadata. Each is discussed in detail in the sections that follow.

1. **Relationships**: These are directed associations which describe a conceptual link between two artifact instances. There are several types of relationships, which are defined below in Section 2.2.2.
2. **Properties**: These describe various named attributes associated with an artifact instance, and can be built-in or user-defined. Each S-RAMP property MUST have a single name which is unique to the artifact that it decorates. When present, an S-RAMP property SHALL have a single value.
3. **Classifications**: Classification systems in S-RAMP are imported by the server as OWL documents, which define the classification system to the server. The means by which a client imports the system into the server is implementation specific and is beyond the scope of this specification. Clients MAY decorate artifacts with references to specific values in a classification system defined to the server.

Note that Artifact Type and Artifact Model values MUST also be unique.

2.2.2 Relationships in S-RAMP

Relationships in S-RAMP are all directed from a source, to a target. Each relationship instance is the logical triple of the following 3 items of metadata:

1. **Relationship Type**. This is the name for the type of relationship. A number of these are pre-defined by S-RAMP in the various Artifact Models (e.g., “includedXsds”, “appliesTo”, …). There can be multiple relationship instances of the same Relationship Type.
2. **Source.** This is a reference to the artifact which is on the source side of the directed relationship. Relationships are always contained by the Source Artifact which “owns” them.
3. **Target.** This is a reference to the artifact which is on the target side of the directed relationship.

It is possible for a relationship of a given Relationship Type not to have a target, which is termed a “relationship with no targets”. In this case there is only one relationship instance with that Relationship Type for a given Source. Such relationships have a target cardinality of “0”. If there is a relationship instance with a given Relationship Type which does have a target, then there CANNOT also be a relationship instance with that Relationship Type which has no target.

There are 4 types of relationships supported in S-RAMP. Refer to the table in Appendix E for a complete list of pre-defined relationships, an indication of whether the relationship is derived, and the model in which it occurs.

1. **Derived Relationships**. These are pre-defined relationships which cannot be directly created or deleted by the client. Cardinalities for such relationships are defined in the applicable Derived Model. All instances of an Artifact Type for which a Derived Relationship is defined, will always contain that Derived Relationship, even if there are no targets for that relationship. The S-RAMP serialization of a Derived Relationship uses named elements defined in the schema for the appropriate Artifact Type definition(s).
2. **Modeled Relationships.** These refer to the S-RAMP pre-defined relationships which may be edited by the client. Cardinalities for such relationships are defined in the applicable model (e.g., the Service Implementation Model or Core Model). All instances of an Artifact Type for which a Modeled Relationship is defined, will always contain that Modeled Relationship, even if there are no targets for that relationship. The S-RAMP serialization of a Modeled Relationship uses named elements defined in the schema within the applicable Artifact Type. There are several considerations related to target cardinality of Modeled Relationships:

   - **Modeled Relationships with Minimum Cardinality = 0:**
     - Instances of the Source Artifact can be created independently of the Target Artifact.
   - **Modeled Relationships with Minimum Cardinality > 0:**
     - Instances of the Source Artifact cannot be created without the appropriate Target Artifact(s) based upon the required minimum cardinality of the relationship.
     - This can be accomplished by publishing the relevant artifacts at the same time, or by publishing the target artifact(s) first.

- Actions which result in the deletion of a relationship instance are not permitted if that would result in a violation of the minimum cardinality.

- **Modeled Relationships with Maximum Cardinality < unbounded:**
  - Relationship instances cannot be created if that would result in a violation of the maximum cardinality limit for the Modeled Relationship.

Note that in cases where the minimum cardinality equals the maximum cardinality, such a relationship must be created or updated in a single step to avoid intermediate states which would violate these requirements.

3. **Generic Relationships**. These are user-defined ad-hoc directed relationship instances between any two artifacts in S-RAMP. They always have a minimum cardinality of 0 and an unbounded maximum cardinality. The Relationship Type value of a Generic Relationship instance is chosen by the client, but it MUST NOT match any pre-defined Relationship Type values already defined by the S-RAMP Modeled and Derived relationships. (see Appendix E). The S-RAMP serialization of a Generic Relationship uses the *relationship* structure defined in the Core Model.
4. **User Defined Modeled Relationships**. S-RAMP attempts to be compatible with implementations which choose to allow users the ability to define models of their own which consist of new or existing Artifact Types and any defined relationships between them, although how and whether such models are supported is beyond the scope of this specification. Such models are called "User Defined Models". Since pre-defined relationships in a model are termed "Modeled", then in this context they are called "User Defined Modeled Relationships". The S-RAMP serialization of a User Defined Modeled Relationship uses the S-RAMP *relationship* structure defined in the Core Model.

2.3 The Core Model

There is a single "core" model which defines all the basic Artifact Types used throughout S-RAMP. The Core Model contains abstract base artifacts for document artifacts and derived artifacts (which are associated with certain document types). Most Artifact Types in the other S-RAMP models are extensions of *BaseArtifactType*. Figure 1 provides a conceptualized illustration of the Core Model artifacts. In addition, there are a number of support types used by the core and other models. Note that the class attributes in the UML diagram are essentially built-in properties. The remaining sub-sections provided additional details on the Core Model.

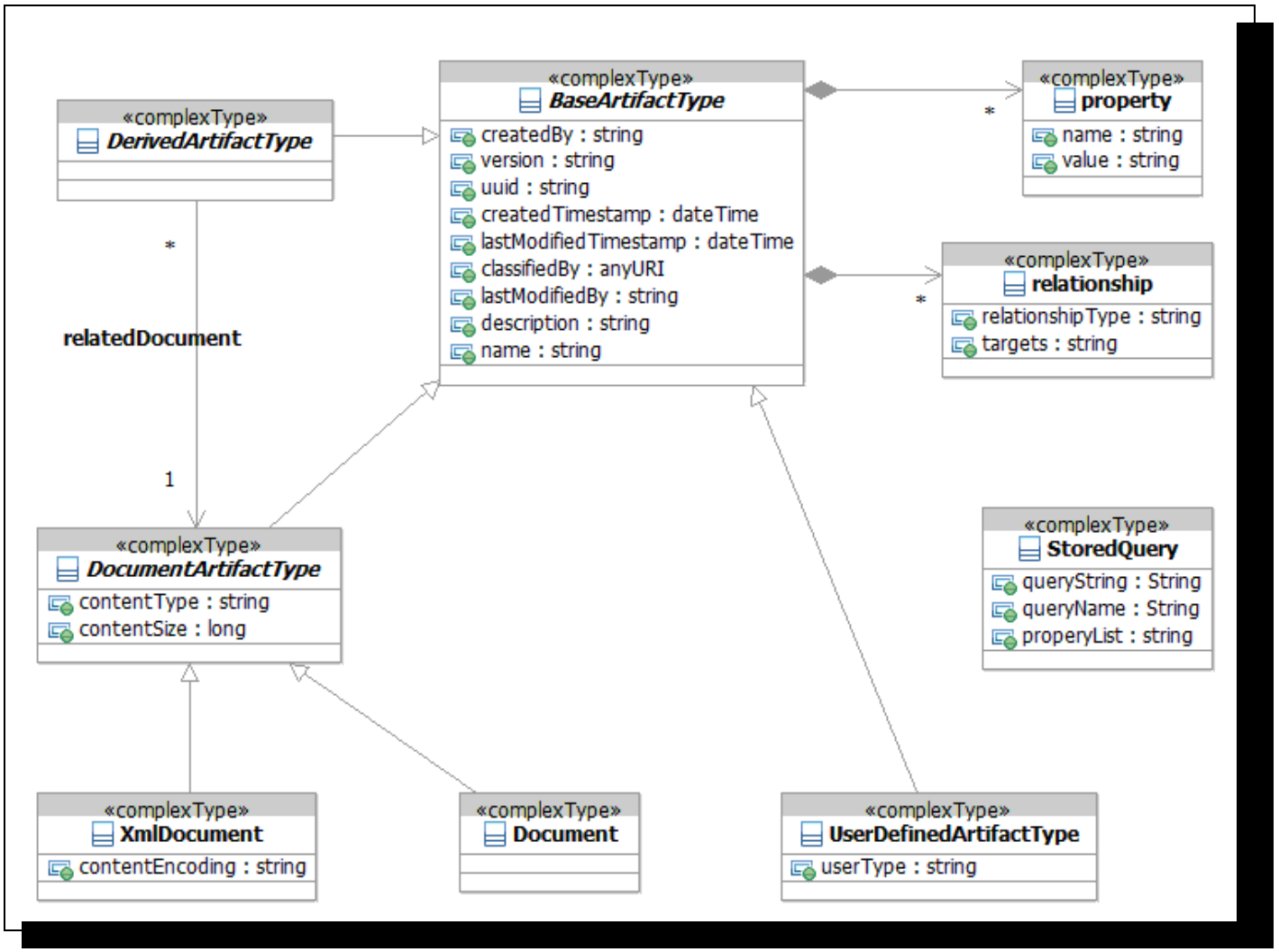

Figure 1: Conceptualized UML Model of Core Model Artifacts

2.3.1 Base Artifact Type

The *BaseArtifactType* is the fundamental abstract type used by all of the artifact models in S-RAMP. It contains all of the common metadata which describes an artifact instance. All artifact instances which are based on the *BaseArtifactType* contain the following metadata:

- **Built-in Properties:**
  - ***createdBy***: A required string assigned by the server identifying the user who created the artifact. S-RAMP does not define requirements on this value. These are implementation specific.
  - ***createdTimestamp***: A required timestamp which is set by the server at the time an artifact is first published. It conforms to xml:dateTime, referenced to UTC.
  - ***description***: This optional property is used to provide a human consumable description of the artifact instance. This value is set by the client for all non-Derived Artifacts (although implementations MAY support setting it automatically using introspection. Derived Artifact descriptions are set by the server.
  - ***lastModifiedBy***: A required string assigned by the server identifying the user who last updated the artifact. S-RAMP does not define requirements on this value. These are implementation specific.
  - ***lastModifiedTimestamp***: A required timestamp which is updated by the server each time an artifact instance is modified. It conforms to xml:dateTime, referenced to UTC.

- ***name***: This required property is used to describe the artifact instance. This value is set by the client for all non-Derived Artifacts (although implementations MAY support setting it automatically using introspection). Derived Artifact names are set by the server.
- ***uuid***: A required unique identifier of an artifact instance in the repository[1]. This value is set for the artifact at the time of its creation. If the value is not provided by the user, the repository will assign one.
- ***version:*** An optional string representing the version of the artifact instance. S-RAMP makes no attempt to define formatting rules for this property, which are implementation specific.

- **Generic Properties:**
  - ***property***: These are optional properties which are defined by the client. They MUST have a single unique name (which SHALL NOT duplicate any other property name of any type, and SHALL NOT duplicate any Relationship Type value). A property name SHALL have 0 or 1 value.
- **Generic Relationships**:
  - ***relationship***: These are optional relationship(s) defined by the client. A relationship contains a Relationship Type identifying the type of the relationship, and 0 or more target artifact references. Relationship Type values within a relationship SHALL NOT duplicate the name of any property.
- **Classifications**:
  - ***classifiedBy***: This is a separate class of metadata. It MAY be set by the client with an unbounded upper cardinality limit. Each value SHALL be a URI which references a specific OWL class from a classification system defined to the repository. For more on OWL classification systems in S-RAMP, see Section 3.

2.3.2 Document Artifact Types

The *DocumentArtifactType* is the fundamental abstract data type for all documents represented in the repository, and it extends *BaseArtifactType*. This Artifact Type includes several built-in properties:

contentType:

- A string indicating the MIME Media type of the content. This is set by the server as part of processing the publication of the document. It cannot be changed by the user.

contentSize:

- An integer representing the size of the content in bytes. This is set by the server as part of processing the publication of the document. It cannot be changed by the user.

The Core Model also includes an *XmlDocument* type which all XML based document data types extend. The *Document* type provides a concrete artifact that can be used to represent arbitrary document types. The *DocumentArtifactType* itself can also be further extended for other document types, such as binary data, etc.

Documents which have a Derived Model associated with them cannot be updated in the repository. They must be removed and republished.

Documents upon which another document has a dependency cannot be deleted.

2.3.3 Miscellaneous Types

There are a few miscellaneous classes in the Core Model:

StoredQuery:

- This is a special Artifact Type which is used to persist queries in the repository. Additional information on this topic is available in Section 4.

UserDefinedArtifactType:

- The *UserDefinedArtifactType* allows clients to create their own artifact type when it is not predefined in an S-RAMP model. The *userType* property is intended to be used to provide an indication of the object type.

[1] S-RAMP uuid values use the *Universally Unique IDentifier (UUID) URN Namespace* specification, Version 4, random number based format.

2.4 Modeling SOA Concepts

S-RAMP supports modeling of business level SOA concepts related to service and process representations and interactions. Since it is not the mission of S-RAMP to define a "SOA Ontology" for the industry, this specification has chosen to reference work being done by The Open Group in their "SOA Working Group" on the "SOA Ontology" (see http://www.opengroup.org/projects/soa-ontology/). That work is compatible with both SCA and BPMN but draws both service and process concepts together at a higher level of abstraction.

S-RAMP supports modeling of SOA concepts using a layered approach:

1. S-RAMP SOA Model
2. S-RAMP Service Implementation Model

The sections below describe how the SOA Ontology work is integrated with S-RAMP as well as the implementation layer underneath it.

2.4.1 The SOA Model

The S-RAMP SOA Model exists to provide a mechanism to link work done by The Open Group SOA Ontology work group with the rest of the S-RAMP internal models. It defines a very minimal set of linkages to artifacts defined in the SOA Ontology.

NOTE: The S-RAMP specification SHALL be referential to draft version 3.1 of the SOA Ontology being developed by The Open Group SOA working group. The SOA Ontology being developed is suitable for use by SOA Repositories. This SOA Ontology SHALL be considered normative for this specification.

Several S-RAMP modeling features have been defined in order to provide linkage between the SOA Ontology and the S-RAMP data model. This is done using the S-RAMP SOA Model. Artifact Types in this model are all user instantiated, and are described in the S-RAMP SOA Model illustrated in Figure 2 below.

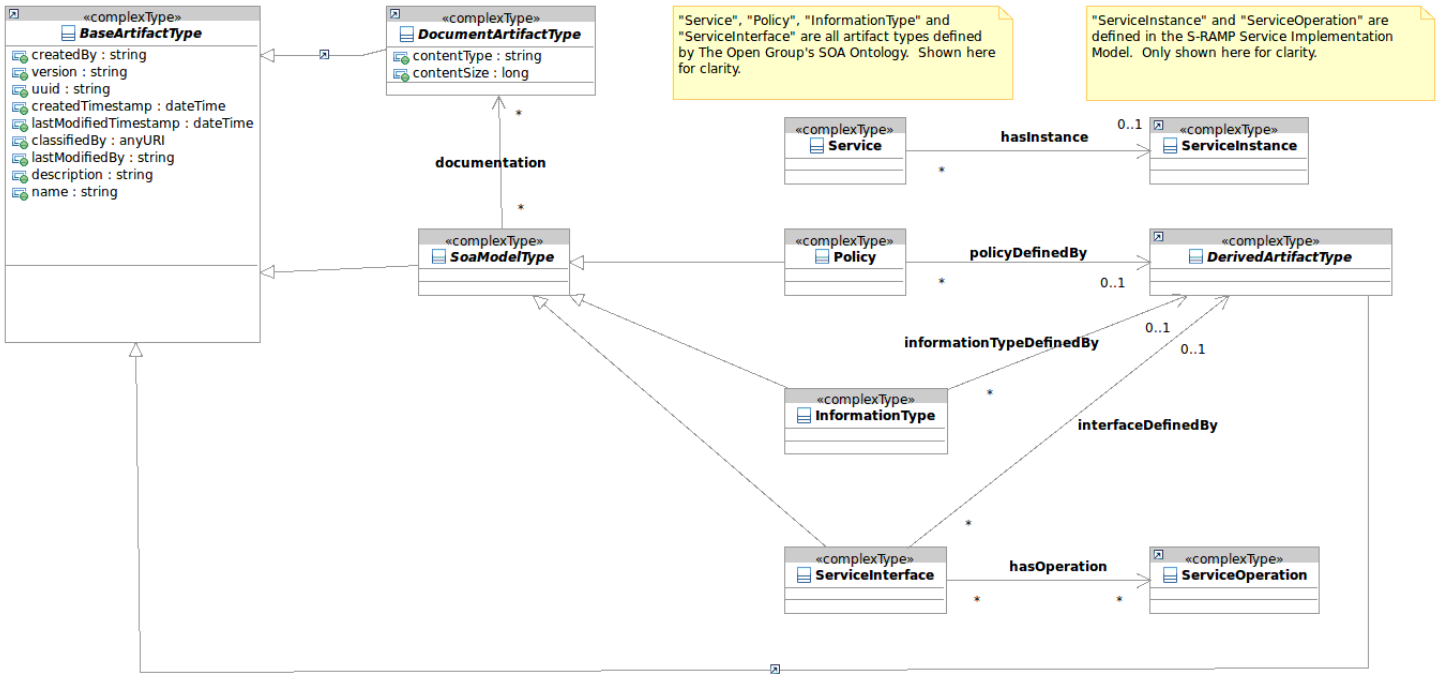

Figure 2: Conceptualized UML Model of SOA Model Artifacts

SOA Model artifacts and the SOA Ontology artifacts which the SOA Model references are designed to provide clients the ability to create conceptual SOA representations. SOA Model Artifacts are all logical artifacts and do not represent or correspond directly to a document instance as do those in the Derived Models.

S-RAMP provides an XML Schema representation of the S-RAMP SOA Model, including elements corresponding to the base version of The Open Groups SOA Ontology's defined artifacts, within the context of the S-RAMP data model. It can be found in Appendix B.

2.4.1.1 SOA Model Artifact Types and Relationships

The abstract *SoaModelType* Artifact Type implicitly acts as a super class for ALL top level SOA Ontology artifacts when they are used within the context of S-RAMP. This imbues all of them with the properties built into all S-RAMP Artifact Types. S-RAMP adds several relationships to SOA Ontology Artifacts used in the SOA Model in order to provide a connection from the SOA Ontology artifacts into the implementation level artifacts described in the Service Implementation Model in Section 2.4.1.1.

SOA Model Artifacts MAY have relationships to Document Artifacts and/or Derived Artifacts in other models, as well as among themselves. All the relationships defined in the SOA Model are Modeled Relationships. See Section 2.2.2 for behavioral details associated with Modeled Relationships. The relationships which have been added to artifacts from the SOA Ontology are summarized in Figure 3 below.

Relationship Name	**Source Artifact Type (from SOA Ontology)**	**Target Artifact Type (from S-RAMP Business & Core Models)**	**Notes**
hasInstance	Service	ServiceInstance	
hasOperation	ServiceInterface	ServiceOperation	
interfaceDefinedBy	ServiceInterface	*DerivedArtifactType*	This allows one to indicate the Derived Artifact instance which defines this service interface. For example, this could be *PortType* artifact instance from the WSDL Model.
policyDefinedBy	Policy	*DerivedArtifactType*	This allows one to link the SOA Model Policy artifact with a concrete S-RAMP derived artifact type (e.g., PolicyAttachment)
informationTypeDefinedBy	InformationType	*DerivedArtifactType*	This allows one to link the SOA Model InformationType artifact with a concrete S-RAMP derived artifact type (e.g., PolicyAttachment)

Figure 3: SOA Model Relationships

2.4.2 The Service Implementation Model

S-RAMP defines a “Service Implementation Model” which describes the service implementation layer underneath the SOA Model. Artifact Types in this model are all user instantiated and most are extensible. Each of these artifacts derives from the *ServiceImplementationModelType* and may be created, changed, updated and deleted by the client. There are a number of pre-defined Modeled Relationships between artifacts of particular types. Figure 4 below illustrates the conceptualized Service Implementation Model artifacts.

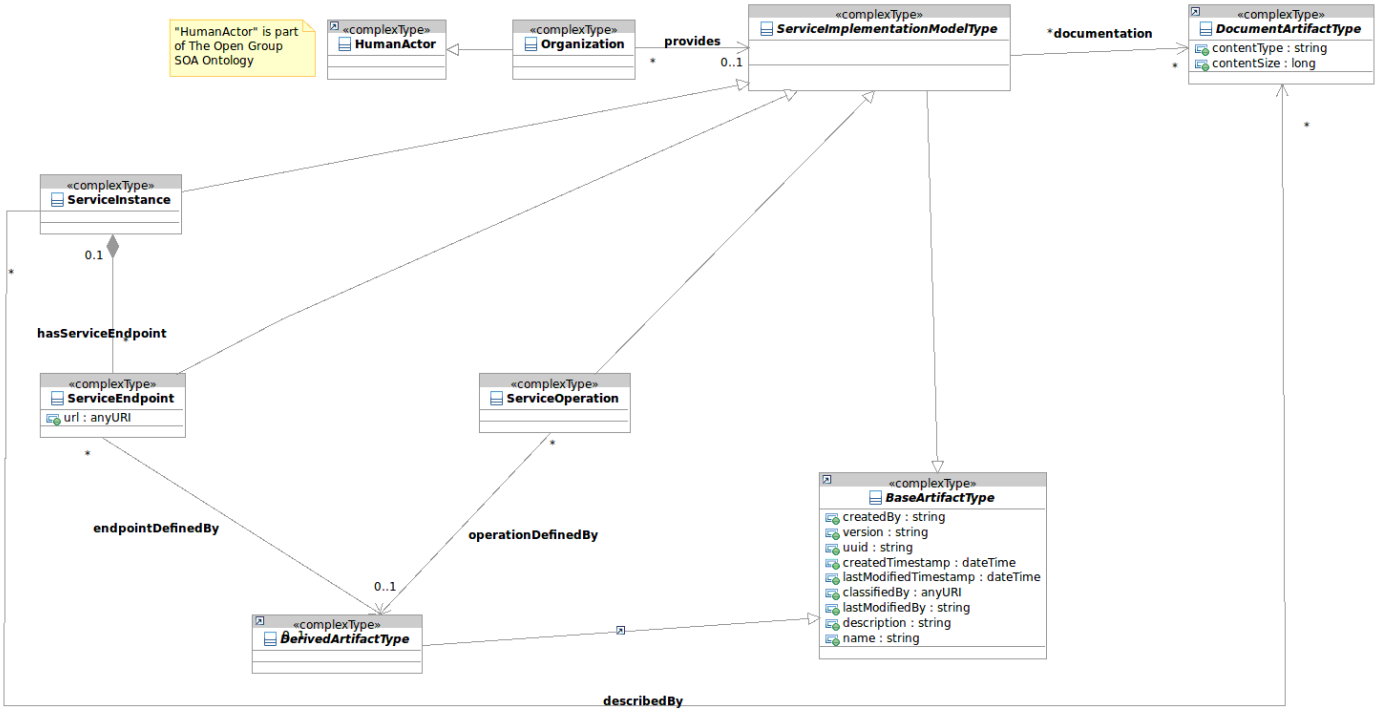

Figure 4: Conceptualized UML Model of Service Implementation Model Artifacts

S-RAMP provides an XML Schema representation of the Service Implementation Model. It can be found in Appendix C. The sub-sections that follow discuss each of the Service Implementation Artifact types in more detail.

2.4.3 Service Implementation Model Artifact Types

The primary Artifact Type from which all Service Implementation Model Artifacts extend is the abstract *ServiceImplementationModelType*. The concrete Service Implementation Model Artifacts which extend it are designed to provide the implementation layer below the SOA Model which allows clients to build SOA representations. Service Implementation Model Artifacts are all logical artifacts and do not represent or correspond directly to a document instance as do those in the Derived Models.

Service Implementation Model Artifacts MAY have relationships to Document Artifacts and/or Derived Artifacts in other models, as well as among themselves. All the relationships shown in the Service Implementation Model are Modeled Relationships. See Section 2.2.2 for behavioral details associated with Modeled Relationships. Of note for the *ServiceImplementationModelType* is the *documentation* Modeled Relationship which allows any artifact in the Service Implementation Model to reference a document describing it.

The concrete Service Implementation Model Artifact Types are then:

Organization

- The *Organization* type is used to describe an organizational entity. It is a subclass of the "Human Actor" artifact defined in the SOA Ontology. Clients can define an unlimited number of organizations and can use the *provides* relationship to link an Organization to any Service Implementation Model Artifact

ServiceInstance

- The *ServiceInstance* Artifact Type represents deployed instance(s) of a service. For example, a Web service running in WebSphere, or Oracle Fusion, etc. The *describedBy* Modeled Relationship can be used to reference document artifact(s) that describe it.

ServiceEndpoint

- The *ServiceEndpoint* Artifact Type represents a physical location at which the Service instance can be invoked, using its *url* Modeled Property. The *endpointdefinedBy* Modeled Relationship allows indicating the Derived Artifact instance which defines this Service endpoint is defined. For example, this could be a *Port* artifact instance from the WSDL Model.

ServiceOperation

- The *ServiceOperation* Artifact Type represents the specific operation performed by the Service. The *operationDefinedBy* Modeled Relationship can be used to link a *ServiceOperation* with a Derived Artifact which defines it. For example this could be an *Operation* artifact instance from the WSDL Model.

2.5 Derived Models

The sections which follow describe the logical models which are constructed by the repository in response to publication of a document for which a Derived Model is defined. The server parses these documents upon publication, dynamically constructing Derived Artifacts corresponding to the major data type components of the document. Every Derived Artifact instance has a *relatedDocument* relationship to the document from which it was created. The artifacts and relationships in a Derived Model provide a powerful tool for searching the repository based on specific components of such a document (e.g., a WSDL PortType, etc.). In most cases, the UML model diagrams illustrated in this section are adequate to define the Artifact Types and the Derived Relationships between them, so these sections are correspondingly brief.

Appendix D describes all of the Derived Model schemas defined in S-RAMP.

2.5.1 The Policy Model

The Policy Model describes the Artifact Types which correspond to the primary components of a WS-Policy document. Policy expressions can be in a stand alone policy document, or embedded in another document such as a WSDL. Figure 5 below is a conceptual UML based illustration of the Policy Model:

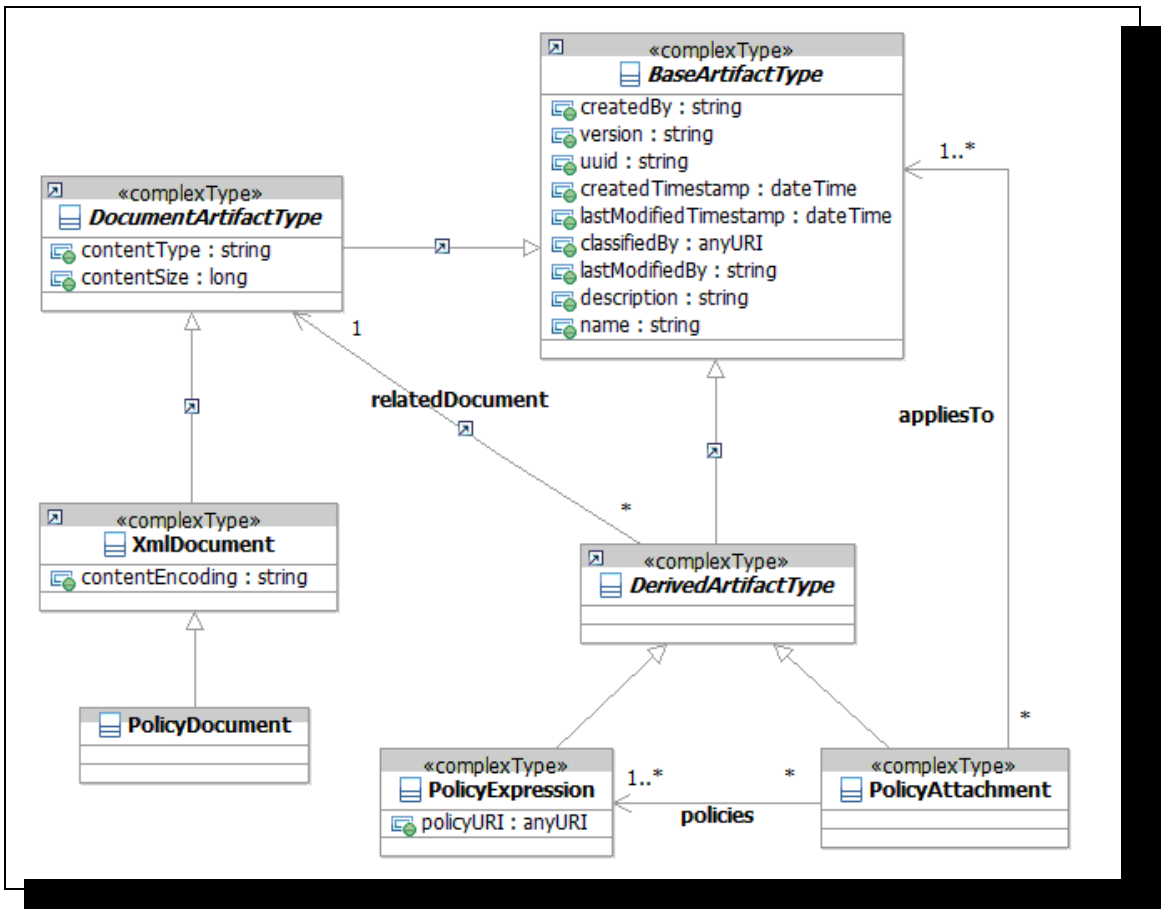

Figure 5: Conceptualized UML Model of Policy Model Artifacts

All *appliesTo* relationships are instantiated in the repository based on the content in the policy attachment document. There is no restriction on the artifacts it can reference.

2.5.2 The XSD Model

The XSD Model describes the Artifact Types which correspond to components of an XSD document stored in the repository, and yields structural metadata useful in performing queries. The conceptual UML model describing the XSD Model is illustrated in Figure 6.

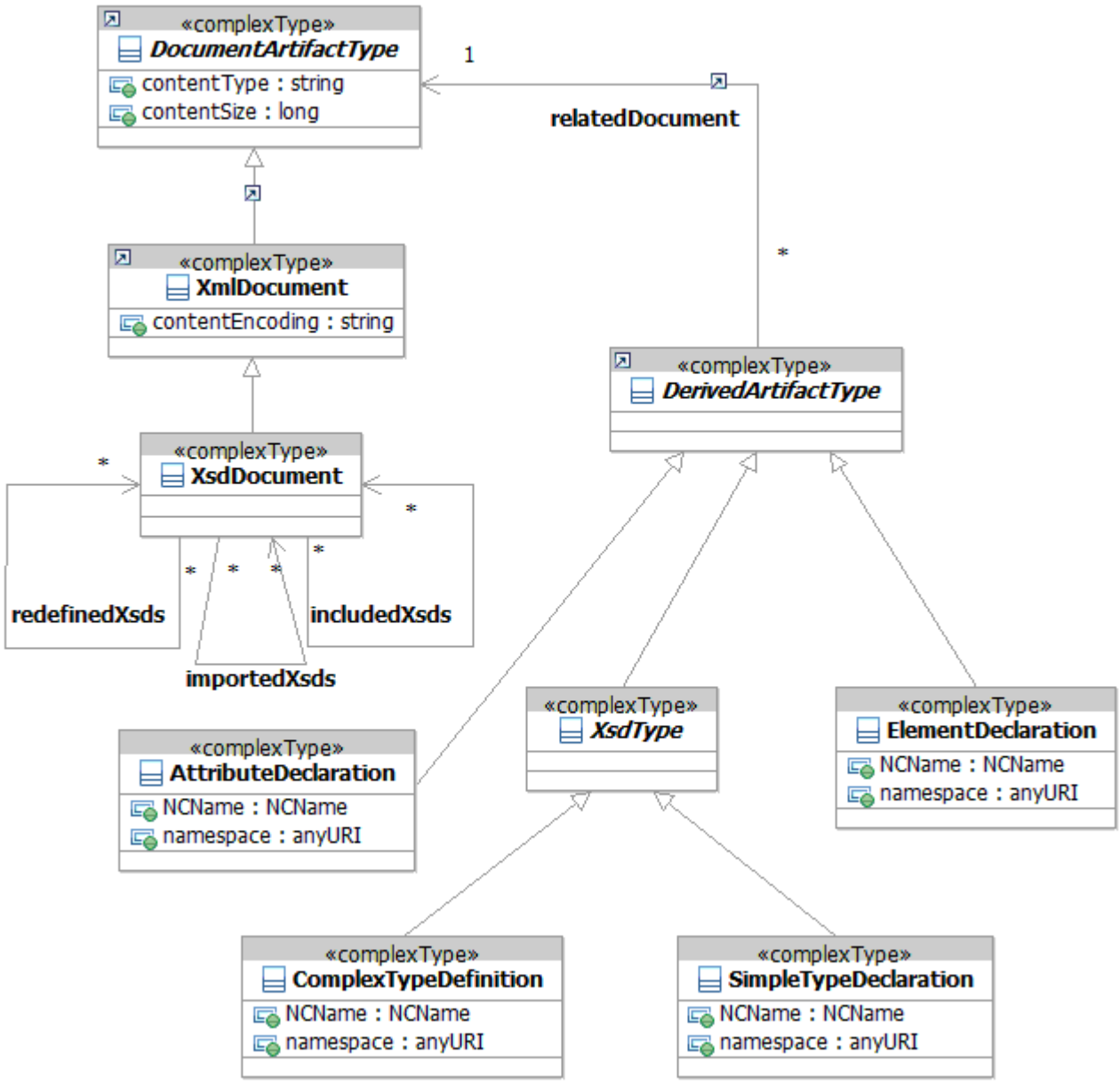

Figure 6: Conceptualized UML Model of XSD Model Artifacts

Note that an XSD document MAY include, import, or redefine other XSD documents. These capabilities are modeled using the *includedXsds*, *importedXsds* and *redefinedXsds* Derived Relationships, respectively.

A *SimpleTypeDeclaration* artifact instance is generated for each global simple type defined in the associated XML Schema document. Similarly, a *ComplexTypeDefinition* instance is generated for each global complex type defined in that XML Schema document. Each global attribute declared in the XML Schema document will generate a corresponding *AttributeDeclaration* artifact instance, and

finally, each global element declared in the XML Schema document will generate a corresponding *ElementDeclaration* artifact instance.

While it is possible to construct a more fine-grained model of an XML Schema document, this level of modeling provides a suitably rich context for discovery queries without creating an overly complex model.

2.5.3 The WSDL Model

The WSDL Model is one of the most complex Derived Models, owing to the complexity of WSDL itself. The WSDL Model contains substantial richness because of the value in being able to perform highly refined queries using logical artifact representations of most of a WSDL document's constituent components.

Figure 7 and Figure 8 below provides a conceptual UML model representing the logical artifacts and their Derived Relationships in the WSDL Model. This model is intended to mirror the structure of a WSDL file. In most cases, the artifact names exactly match corresponding data types in WSDL (e.g., *Message*, *PortType*, *Operation*, *Binding*, *Service*, *Port*, and so on). These figures are actually one diagram presented in two pieces for legibility.

Note that aggregation relationships in these diagrams are modeled as Derived Relationships in the WSDL Model schema found in Appendix D.3.

Note that the WsdlExtension Artifact Type illustrated in Figure 8 below is used to model extensibility elements for any given WSDL element. This supports WSDL extensions which the repository does not recognize.

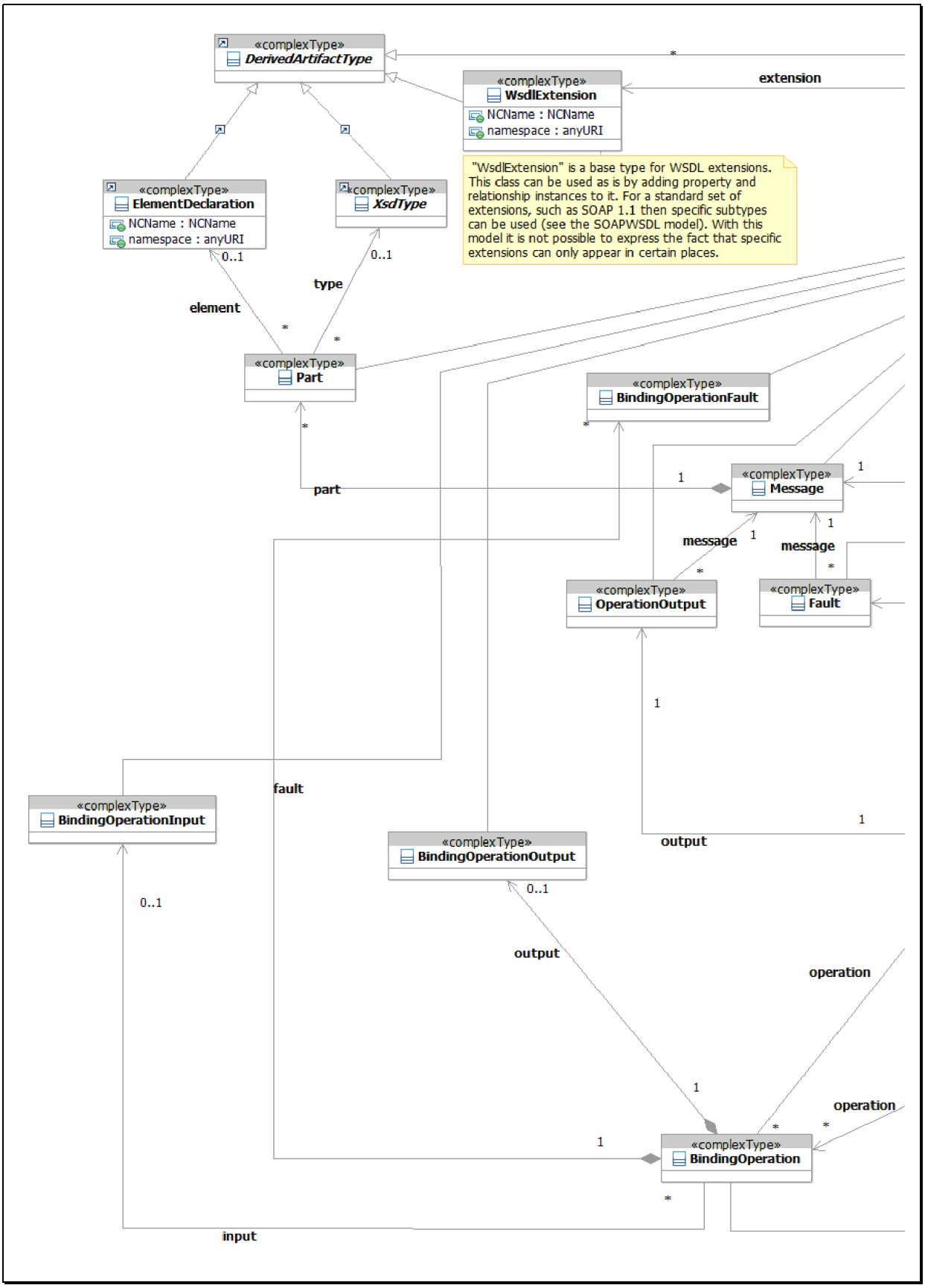

Figure 7: Conceptual UML Diagram of WSDL Model: Part 1

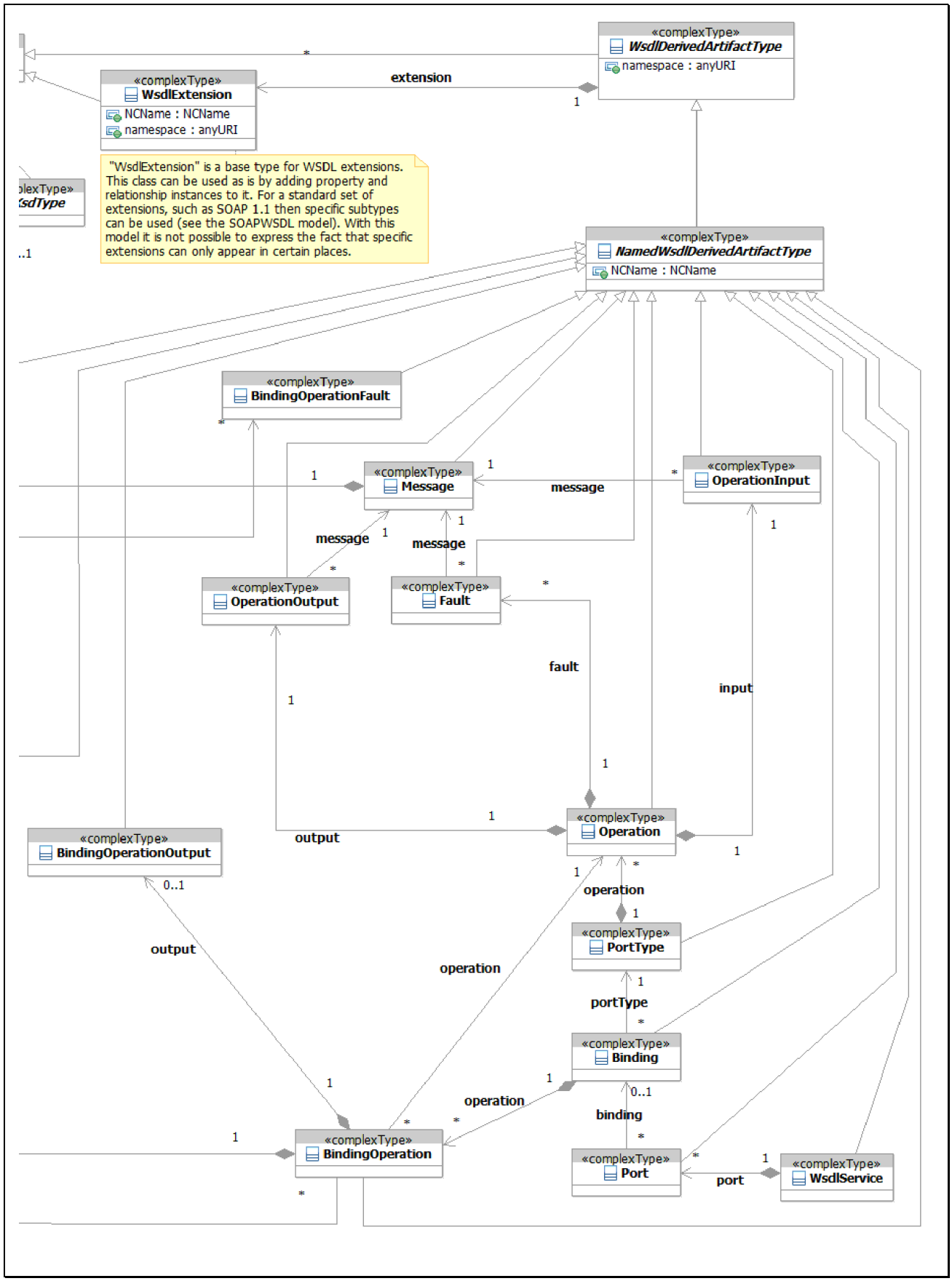

Figure 8: Conceptual UML Diagram of WSDL Model: Part 2

The WsdlDerivedArtifactType is a modeling convenience from which all WSDL related metadata artifacts extend. It contains a *namespace* attribute which is the namespace of the WSDL document and it applies to every Derived Artifact instance for that Document. It also has an *extension* Derived Relationship which serves to identify the set of WSDL extensions which apply to the Derived Artifact.

Figure 9 below illustrates a conceptual UML model representing the portions of the WSDL Model which describe the relevant document artifacts and Derived Relationships for the associated WSDL document. A WSDL document can import, include and redefine XSD document(s), as well as import other WSDL documents. Similarly, an XSD document can import, include and redefine other XSD documents. The model supports each of these capabilities with the Derived Relationships shown in the diagram.

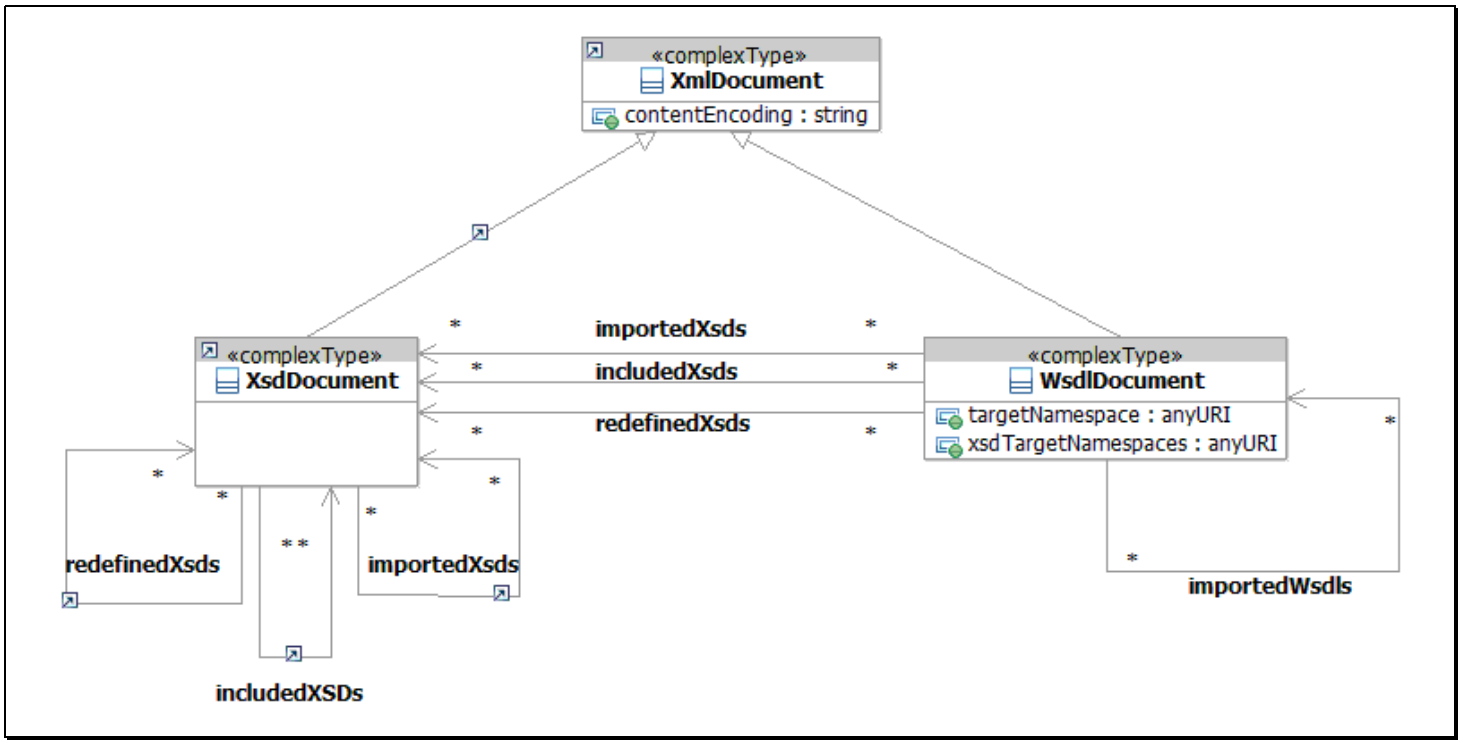

Figure 9: Conceptual UML Diagram of WSDL Model: Part 3

2.5.4 The SOAPWSDL Model

The SOAPWSDL Model contains the SOAP 1.1 binding specific WSDL Model artifacts for WSDL 1.1. It is separated into its own Model and schema to simplify its use. Figure 10 below illustrates a conceptual UML model of the relevant SOAP WSDL Model artifacts.

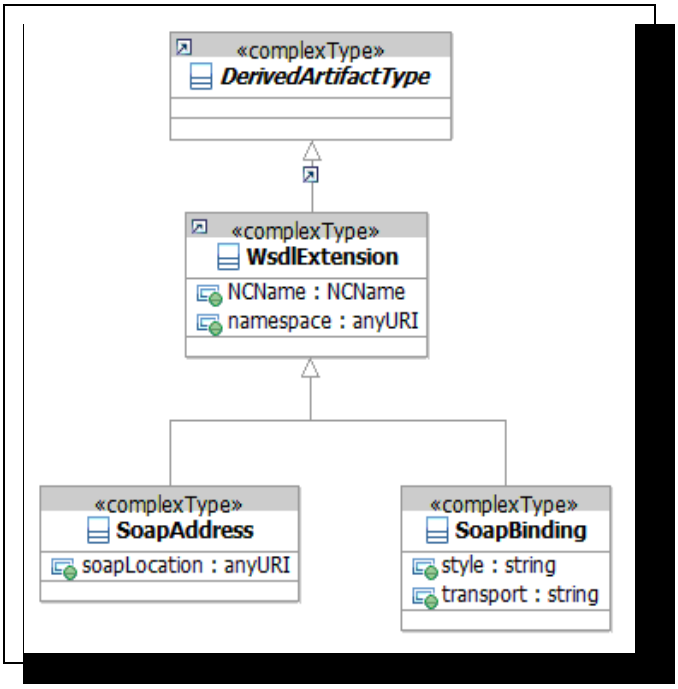

Figure 10: Conceptualized UML Diagram of the SOAP WSDL Model

2.6 Referencing S-RAMP Artifacts

This section describes the syntax for referencing Artifact Type(s) in the S-RAMP bindings. The use of the references described in this section to operate on S-RAMP artifacts is specific to the binding used (e.g., the S-RAMP Atom Binding). Please refer to the appropriate binding specific document of this specification for details.

The following referential syntax applies:

```
/s-ramp/{ArtifactModel}/{ArtifactType}
```

Note that each successive component of this syntax is optional. Specifically

- A reference of "/s-ramp" refers to all Artifact Types in all models
- A reference of "/s-ramp/{Artifact Model}" refers to all Artifact Types within the specified Artifact Model.
- References of the form "//{ArtifactModel}" or "//{ArtifactType}" are also permitted.

Table 6 below provides the valid values for Artifact Model and Artifact Types. Abstract types are not included since they cannot be instantiated.

Table 6: Artifact Models and Types

Artifact Model	Artifact Type
core	Document
	XmlDocument
xsd	XsdDocument
	AttributeDeclaration
	XsdType
	ElementDeclaration
	SimpleTypeDeclaration
	ComplexTypeDeclaration
policy	PolicyDocument
	PolicyExpression
	PolicyAttachment
soapWsdl	SoapAddress
	SoapBinding
wsdl	WsdlDocument
	WsdlService
	Port
	WsdlExtension
	Part
	Message
	Fault
	PortType
	Operation
	OperationInput
	OperationOutput
	Binding
	BindingOperation
	BindingOperationInput
	BindingOperationOutput
	BindingOperationFault
serviceImplementation	Organization
	ServiceEndpoint
	ServiceInstance
	ServiceOperation

user	{User Defined Artifact Type}
soa	{See The Open Group's SOA Ontology for the normative list of Artifact names. They are reproduced here for clarity.}
	HumanActor
	Choreography
	ChoreographyProcess
	Collaboration
	CollaborationProcess
	Composition
	Effect
	Element
	Event
	InformationType
	Orchestration
	OrchestrationProcess
	Policy
	PolicySubject
	Process
	Service
	ServiceContract
	ServiceComposition
	ServiceInterface
	System
	Task

Below are some examples of valid S-RAMP model and Artifact Type references.

Example 1: Artifact Model and Type References

```
/s-ramp

    o   References all Artifact Types in the repository

/s-ramp/xsd   or    //xsd

    o   References all Artifact Types in the XSD Model (e.g., XsdDocument, XsdType, …)

/s-ramp/xsd/XsdType or    //XsdType

    o   References the XsdType Artifact Type in the XSD Model

/s-ramp/soa/Service
```

- References all *Service* Artifacts Types in the SOA Model (included by reference to The Open Group's SOA Ontology)

/s-ramp/wsdl/Port

- References the Port Artifact Type in the WSDL Model

3. Classification Systems in S-RAMP

A classification system allows classification values to be organized in a hierarchy, allowing artifacts to be grouped into sets and to identify subsets within those sets. Query and display of all artifacts in a set then becomes possible and provides a simple yet powerful means of classifying and finding related objects. Artifact instances in the repository MAY have classification values applied to them.

A simple example of a classification system hierarchy is a geographical region hierarchy that could, for example, be used to indicate which location produced an artifact. Using '/' to delineate levels in the hierarchy and ',' to separate members at the same level, one part of an example hierarchy could be World / Asia / Japan, China. The hierarchy could also contain World / Europe / United Kingdom, Germany. In other words, subsets of World are Asia and Europe, subsets of Asia are Japan and China, and subsets of Europe are United Kingdom and Germany. To find artifacts produced by teams in Asia, a search can be issued to return artifacts classified by the Asia classification. To give the behavior desired, a repository implementation interprets this query as requiring the retrieval of artifacts classified by Asia and its sub-groups i.e. Asia or Japan or China.

Classification systems used in S-RAMP are expressed in the Web Ontology Language (OWL) which builds upon the Resource Description Framework (RDF). The RDF/XML serialization format is commonly used for files containing OWL constructs and is a format that SHALL be understood by an S-RAMP repository. OWL consists of three increasingly expressive sublanguages, OWL Lite, OWL DL and OWL Full. The W3C "OWL Web Ontology Language Overview" document indicates that "OWL Lite supports those users primarily needing a classification hierarchy and simple constraints", and accordingly the OWL elements that S-RAMP uses to define classification systems come from OWL Lite. To enable the classification capabilities required in S-RAMP, a restricted set of OWL Lite elements is sufficient. Also, it should be noted that multiple inheritance will not be supported in S-RAMP.

The elements that SHALL be supported and a brief description of each follow (refer to the W3C documents detailing OWL and RDF for further detail on these elements).

rdf:ID

- In an element which requires a resource to be identified, an *rdf:ID* attribute MAY be used. This attribute has slightly different behavior to *rdf:about*, including the requirement that the value only appear once in the scope of a document, so it provides a useful check when defining distinct resources. In RDF and OWL, resources are identified by using URIs. The attribute value is a string indicating a URI fragment relative to the currently in-scope base value (typically set using the xml:base attribute).

rdf:about

- As an alternative to using an *rdf:ID* attribute to identify a resource, an *rdf:about* attribute MAY be used. The attribute value is either a string indicating an absolute URI, or a string indicating a relative URI resolved against the currently in-scope base value (typically set using the xml:base attribute).

owl:Ontology

- In OWL, classification systems are represented by an OWL ontology. The ontology groups a number of related classifications together. In the example the geographical regions classification system would be defined using an *owl:Ontology* element.

owl:Imports

- owl:Imports elements are permitted under owl:Ontology. This means that classes declared in the ontology MAY subclass classes declared in any imported ontologies, although multiple inheritance is not permitted. How owl ontologies are imported is vendor specific and therefore the resolution of owl:import references is also vendor specific.

owl:Class

- In OWL, classifications are represented by OWL classes. In the example, the World, Europe, Asia, United Kingdom, Germany, Japan and China classifications would be defined using an *owl:Class* element.

rdfs:subClassOf

- To define the hierarchy, classes are related to each other via the *rdfs:subClassOf* element. If class B is declared to be a subclass of class A, then the instances of class B represent a subset of the instances of class A. In the example, Asia would be declared to be a subclass of World, Japan a subclass of Asia, and so on.

rdfs:label

- Ontologies and classes MAY be given a human readable name using the *rdfs:label* element. Names in multiple languages are supported by this element, using the xml:lang attribute.

rdfs:comment

- Ontologies and classes MAY be given a human readable comment or description using the *rdfs:comment* element. Comments in multiple languages are also supported by this element using the xml:lang attribute.

All other OWL elements are not supported (in terms of OWL Lite this means that property-related elements, and the *owl:equivalentClass*, *owl:imports*, *owl:intersectionOf*, and versioning elements are not supported).

OWL files used in S-RAMP MUST conform to the following rules, which result in ontologies that are self-contained in a single OWL file:

- There MUST be exactly one *owl:Ontology* element in any OWL file
- A class MUST only be defined in one ontology

The example ontology can be expressed in OWL as follows:

Example 2: An OWL Ontology

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

<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY ns_region "http://www.regions.com/geographicalregion">
]>

<rdf:RDF
  xmlns:xsd="&xsd;"
  xmlns:rdf="&rdf;"
  xmlns:rdfs="&rdfs;"
  xmlns:owl="&owl;"
  xmlns:ns_region="&ns_region;"
  xml:base="&ns_region;"
>

  <!-- ontology -->

  <owl:Ontology rdf:ID="">
    <rdfs:label>Geographical Regions</rdfs:label>
    <rdfs:comment>An ontology used to provide classifications describing geographical
regions.</rdfs:comment>
  </owl:Ontology>

  <!-- root class -->
```

```
  <owl:Class rdf:ID="World">
    <rdfs:label>World</rdfs:label>
    <rdfs:label xml:lang="en">World</rdfs:label>
    <rdfs:label xml:lang="fr">Monde</rdfs:label>
  </owl:Class>

  <!-- sub classes -->

  <owl:Class rdf:ID="Asia">
    <rdfs:subClassOf rdf:resource="&ns_region;#World"/>
    <rdfs:label>Asia</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="Europe">
    <rdfs:subClassOf rdf:resource="&ns_region;#World"/>
    <rdfs:label>Europe</rdfs:label>
  </owl:Class>

  <!-- sub sub classes -->

  <owl:Class rdf:ID="Japan">
    <rdfs:subClassOf rdf:resource="&ns_region;#Asia"/>
    <rdfs:label>Japan</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="China">
    <rdfs:subClassOf rdf:resource="&ns_region;#Asia"/>
    <rdfs:label>China</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="UnitedKingdom">
    <rdfs:subClassOf rdf:resource="&ns_region;#Europe"/>
    <rdfs:label>United Kingdom</rdfs:label>
  </owl:Class>

  <owl:Class rdf:ID="Germany">
    <rdfs:subClassOf rdf:resource="&ns_region;#Europe"/>
    <rdfs:label>Germany</rdfs:label>
  </owl:Class>

</rdf:RDF>
```

Some points of interest in the foregoing example:

- An *rdfs:comment* element is shown within the Geographical Regions *owl:ontology* element, but comments may be used in *owl:Class* constructs if desired.
- *rdfs:label* elements have generally been used with no xml:lang attribute specified. However, multiple *rdfs:label* elements MAY be specified within an *owl:Ontology* or *owl:Class* element, and these may contain a variety of xml:lang attribute values (and an *rdfs:label* element with no xml:lang attribute may be specified in addition to elements with such attributes present). This is illustrated by the World class in the ontology defined above. An *owl:Ontology* or *owl:Class* element SHOULD contain only one *rdfs:label* element with a given xml:lang value to ensure well-defined behavior when requesting a label for a specific language.
- The behavior of *rdfs:comment* elements with respect to xml:lang attributes is identical to the behavior for *rdfs:label* elements.

- Although the example shows a single root class which the other classes' subclass either directly or indirectly, multiple root classes are permitted in an ontology. Also permitted are stand-alone classes that neither subclasses some parent class nor are themselves parents to other child classes.
- Multiple inheritance (a class being a subclass of multiple parent classes) is NOT permitted.

4. S-RAMP Query Model

4.1 Overview

The S-RAMP specification supports a robust query interface which compliant implementations MUST support. It provides a way to find repository artifacts using a rich set of constraints. The query expression in S-RAMP uses an XPath 2.0 based dialect. Refer to Section 4.5 for additional information on the S-RAMP query grammar. The expression predicates act as filters to identify matching artifact instances. Filtering can be done based upon artifact metadata, including properties, relationships and classifications. Complex criteria can be formed. One can for example, search for "all services which are categorized as being in production", or find "all the XML Schema documents which are referenced by any WSDL document", and so on. Queries are executed against a set of S-RAMP artifacts using the criteria provided and the result returned is a set of S-RAMP artifacts. Furthermore, all artifact types both concrete and abstract are eligible for use in the query expressions.

Specific request and response syntax for executing query requests is, however, binding specific. Details can be found in the relevant binding document(s) of this specification. This document covers the common features of an S-RAMP query, independent of their invocation syntax.

4.2 Query Dialect (XPath2) Context

Since the S-RAMP query model is based on XPath2, this specification defines a static context for the information available during static analysis of a query expression prior to its evaluation, as well as a dynamic context for the information available when the expression is evaluated.

Static analysis of query expressions is performed using the following static context:

Table 7: Static Context for S-RAMP Query Expressions

Component	Value
XPath 1.0 Compatibility Mode	False
Statically known namespaces	See Table 1
Default element/type namespaces	"s-ramp"
Default function namespace	"s-ramp"
In-scope schema definitions	S-RAMP schemas are defined on the s-ramp.org web site. A copy is included in the Appendices of this document.
In-scope variables	Only the variables used in a template based query expression (typically applies to Stored Query)
Context item static type	element([namespace of this specification], record)
Function signatures	Functions defined in XQuery 1.0 and XPath 2.0 Functions and Operators document (http://www.w3.org/TR/xquery-operators/), plus functions defined in Section 4.4
Statically known collations	none
Default collation	http://www.w3.org/2005/xpath-functions/collation/codepoint
Base URI	none
Statically known documents	none
Statically known collections	none
Statically known default collection type	none

The following dynamic context defines the default values which are available during evaluation of an S-RAMP query expression:

Table 8: Dynamic Context for S-RAMP Query Expressions

Component	Value
context item	dynamic; node changes during evaluation of the expression
context position	1
context size	1
Variable values	none
Function implementations	function implementations are on a per-server basis
Current dateTime	date and time on server against which request is made
Implicit time zone	UTC+/-0
Available documents	none
Available collections	none
Default collection	“s-ramp”

The domain of an S-RAMP query is the structure of the data model defined by its schemas, using the Artifact Model and Artifact Type reference syntax described in Table 6.

Additional features and limitations of S-RAMP query support:

- Uses the XPath2 default Axis of ancestor / child
- The XPath abbreviated paths below are not supported
  - ‘..’
- XPath2 Node types are not supported
  - Comment, text and so on are not relevant in an S-RAMP query

4.3 Query Expression Predicates

The default namespace assumed in an S-RAMP query is “s-ramp”. S-RAMP query expressions allow filtering of the path expression using XPath 2.0 predicates, in the form:

```
{path-expression}[{predicate}]
```

where:

```
{path-expression} = /s-ramp/{ArtifactModel}/{ArtifactType}
```

which corresponds to the S-RAMP Artifact Model and Type reference syntax (see Table 6); and:

```
{predicate}
```

is an XPath2 predicate which conforms to the S-RAMP query grammar (see Section 4.5).

When evaluated, the predicate returns true or false, and thus filters the set of artifacts returned from the query. Predicates can use combinations of artifact properties, relationships and classifications. The predicate sets the context for evaluating the query, making it wider or narrower. If a query predicate uses a property which is not defined on an artifact instance whose scope is within the path-expression, then the predicate is false for that artifact and it is not returned from the query. Several examples follow:

Example 3: Query Expressions Using Properties

```
/s-ramp/xsd/XsdDocument[@someProperty]
```

- Returns *XsdDocument* artifact instances which have a generic property named “someProperty”.

/s-ramp/xsd/XsdDocument[@name = 'bob']

- Returns *XsdDocument* artifact instances which have an artifact *name* whose value is “bob”.

/s-ramp/serviceImplementation/ServiceInstance[@someProperty =‘high’]

- Returns *ServiceInstance* artifact instances having a generic property called *someProperty* whose value equals ‘high’.

- If the requested property is not defined on the artifact within the set of artifacts selected for search, then the predicate is false and that artifact is therefore not returned.

Example 4: Query Expression Using Relationships

/s-ramp/wsdl/WsdlDocument[includedXsds]

- Returns all the *WsdlDocument* artifact instance(s) representing WSDL documents for which the query predicate evaluates to a non-empty context. The context here consists of the targets of the *includedXsds* relationship, therefore any *WsdlDocument* artifact which has at least one *includedXsds* relationship is returned.
- If the relationship in the predicate is not defined on a given artifact within the set of artifacts selected for search, then the predicate expression is false and that artifact is not returned.

Example 5: Query Expressions Using Relationships and Properties

/s-ramp/wsdl/WsdlDocument[includedXsds[@someProperty=’true’]]

- Only *WsdlDocument* artifact instances corresponding to WSDL documents which have at least one *includedXsds* relationship instance, that has a property name of “someProperty”, whose value is “true” will be returned.

- If the relationship in the predicate is not defined on a given artifact whose scope is within the path expression, then the predicate evaluates to false and that artifact is not returned.

4.4 Query Functions

S-RAMP defines a number of its own query functions in addition to using some already defined in XPath 2.0.

This includes functions which provide several useful simplifications associated with how an artifact is classified. These query functions use the following syntax:

```
s-ramp:{function-name}({artifact}, {category-value-1}, {category-value-2}, …)
```

Table 9 below lists the functions used in S-RAMP. Each of the examples for the classification based functions it describes assumes the following conceptual OWL ontology:

- class color
  - class red
  - class white
- class taste
  - class sour

- class sweet
- class spicy

where this ontology is used to classify a set of WsdlService artifact instances:

- name = ”Bread” classifiedBy = [“white”, “sweet”]
- name = ”Wine” classifiedBy = [“red”, “sweet”]
- name = ”Chili” classifiedBy = [“red”, “spicy"]

and each of the classifiedBy URI references to these will be abbreviated simply as the class name here for brevity:

Table 9: Query Functions Used in S-RAMP

Function	Descriptions & Examples

s-ramp:classifiedByAnyOf	Returns all artifact instances classified by at least one of the specified OWL URIs or their subtypes.
	
	`Syntax: classifiedByAnyOf({artifact}, {classifiedBy_1}, {classifiedBy_2},…)`
	
	Using the classification example setup preceding this table, this example returns all 3 *WsdlService* artifact instances:
	
	`/s-ramp/wsdl/WsdlService[classifiedByAnyOf(., ‘taste’)]`
	
	This example returns the Wine and Chili *WsdlService* artifact instances:
	
	`/s-ramp/wsdl/WsdlService[classifiedByAnyOf(., ‘red’, ‘spicy’)]`

s-ramp:classifiedByAllOf	Returns all artifact instances classified by every one of the specified OWL URIs or their subtypes.
	
	`Syntax: classifiedByAllOf({artifact}, {classifiedBy_1}, {classifiedBy_2},…)`
	
	This example returns all 3 *WsdlService* artifact instances:
	
	`/s-ramp/wsdl/WsdlService[classifiedByAllOf(., ‘taste’)]`
	
	This example returns only the Wine *WsdlService* artifact instance:
	
	`/s-ramp/wsdl/WsdlService[classifiedByAllOf(., ‘red’, ‘sweet’)]`

s-ramp:exactlyClassifiedByAnyOf	Returns all artifact instances classified by at least one of the specified OWL URIs. Subtypes are not considered:
	
	`Syntax: exactlyclassifiedByAnyOf({artifact}, {classifiedBy_1}, {classifiedBy_2},…)`
	
	This example returns none of the *WsdlService* artifact instances:
	
	`/s-ramp/wsdl/WsdlService[exactlyClassifiedByAnyOf(., ‘taste’)]`
	
	This example returns only the Wine and Chili *WsdlService* artifact instances:

	`/s-ramp/wsdl/WsdlService[exactlyClassifiedByAnyOf(., 'red', 'taste')]`

s-ramp:exactlyClassifiedByAllOf	Returns all artifact instances classified by every one of the specified OWL URIs. Subtypes are not considered:
	
	`Syntax: exactlyclassifiedByAllOf({artifact}, {classifiedBy_1}, {classifiedBy_2},…)`
	
	This example returns none of the *WsdlService* artifact instances:
	
	`/s-ramp/wsdl/WsdlService[exactlyClassifiedByAllOf(., 'taste')]`
	
	This example returns only the Wine *WsdlService* artifact instance:
	
	`/s-ramp/wsdl/WsdlService[exactlyClassifiedByAllOf(., 'red', 'sweet')]`

xp2:matches	This is the XPath 2.0 matches function, which returns an xsd:boolean indicating whether {value} of its first argument matches the regular {expression} that is its second argument.
	
	`Syntax: matches( {value}, {expression} )`
	
	The expression may use wildcards, but only in the form '.*'
	
	This example returns all artifact instances (of any Artifact Type) from the Service Implementation Model, whose artifact *name* matches '.*account.*', and a user-defined *version* property whose value is '1'.
	
	Note: if a property or relationship has no namespace, the default namespace is "s-ramp":
	
	`/s-ramp/serviceImplementation[xp2:matches(@name, '*account.*') and @version = '1']`

4.5 S-RAMP Query Grammar

This section describes the XPath2 based query grammar used in S-RAMP. It is based on a redacted subset of the XPath 2.0 grammar.

QName ::=

[http://www.w3.org/TR/REC-xml-names/#NT-QName]

s-ramp query ::=

artifact -set
| artifact -set '[' predicate '] |
| artifact -set '[' predicate ']' '/' subartifact-set

artifact- set::=

location-path

location-path::=

/s-ramp

|/s-ramp/<artifact-model>

| /s-ramp/<artifact-model>/<artifact-type>

| //<artifact-type>

subartifact-set:: =

relationship-path

| relationship-path '[' predicate ']'

| relationship-path '[' predicate ']' '/' subartifact-set

| FunctionCall

relationship-path::=

any-outgoing-relationship

| <s-ramp-relationship-type>

any-outgoing-relationship::=

outgoing

any-incoming-relationship:: =

incoming

Predicate::=

Expr

Expr::=

AndExpr

AndExpr::=

OrExpr

| AndExpr 'and' OrExpr

OrExpr::=

EqualityExpr

| OrExpr 'or' EqualityExpr

EqualityExpr::=

subartifact-set

|ForwardPropertyStep

| ForwardPropertyStep '=' PrimaryExpr

| ForwardPropertyStep '!=' PrimaryExpr

| ForwardPropertyStep '<' PrimaryExpr

| ForwardPropertyStep '>' PrimaryExpr

| ForwardPropertyStep '<=' PrimaryExpr

| ForwardPropertyStep '>=' PrimaryExpr

| '(' Expr ')'

PropertyQName ::=

QName

PrimaryExpr ::=

Literal
| Number
|'$' <PropertyQName>

ForwardPropertyStep ::=

Subaartifact-set'/'@<PropertyQName>|@<PropertyQName>

FunctionCall ::=

FunctionName '(' (Argument (',' Argument)*)? ')'

Argument ::=

Expr

Literal ::=

'"' [^"]* '"' | "'" [^']* "'"

Number ::=

Digits ('.' Digits?)? | '.' Digits

Digits ::=

[0-9]+

FunctionName ::=

QName – NodeType

NodeType ::=

'comment'
| 'text'
| 'processing-instruction'
| 'node'

4.6 Stored Queries

S-RAMP provides support for storing queries in the repository using the *StoredQuery* Artifact Type (refer to Section 2.3 for UML representation). This can be convenient because it allows quick execution of a frequently performed query. The syntax associated with creation, retrieval, update and deletion of a Stored Query is binding specific. Refer to the appropriate binding document of this specification for details.

The *StoredQuery* Artifact Type does NOT extend *BaseArtifactType* as do most other Artifact Types in S-RAMP, which means it is simpler and possesses only these built-in attributes:

- queryName: The name of the Stored Query instance. This must be unique.
- queryExpression: The specification of the query expression.

A *StoredQuery* MAY also contain a list of propertyName values. These are used to indicate to the server that the results returned from the execution of the query SHALL include values for those property names when they are present in the artifact instance(s) returned. This can be valuable in bindings which may not necessarily return the complete artifact in query results. The actual format of the query response is binding specific.

A Core Model Schema

For convenience, an S-RAMP Core Model Schema xsd file is also provided at:

http://s-ramp.org/2010/specification/schemas/coremodel.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   targetNamespace="http://s-ramp.org/xmlns/2010/s-ramp" version="1.0"
   xmlns:tns="http://s-ramp.org/xmlns/2010/s-ramp"
   xmlns:s-ramp="http://s-ramp.org/xmlns/2010/s-ramp"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified">
  <xsd:import namespace="http://www.w3.org/1999/xlink" schemaLocation="http://www.w3.org/1999/xlink.xsd" />
  <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
              schemaLocation="http://www.w3.org/2009/01/xml.xsd"/>

  <!--                 Core data types:                     -->

  <!-- Base type for almost all Artifacts in S-RAMP.  Most other types in S-RAMP extend
      this one. Extensions of BaseArtifactType are limited to attributes. -->
  <xsd:complexType abstract="true" name="BaseArtifactType">
    <xsd:sequence>
      <xsd:element ref="tns:classifiedBy" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="tns:relationship" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="tns:property" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="description" type="xsd:string" use="optional"/>
    <xsd:attribute name="createdBy" type="xsd:string" use="required"/>
    <xsd:attribute name="version" type="xsd:string" use="optional"/>
    <xsd:attribute name="uuid" type="xsd:string" use="required"/>
    <xsd:attribute name="createdTimestamp" type="xsd:dateTime" use="required"/>
    <xsd:attribute name="lastModifiedTimestamp" type="xsd:dateTime" use="required"/>
    <xsd:attribute name="lastModifiedBy" type="xsd:string" use="required"/>
   <xsd:anyAttribute namespace="##any"/>
  </xsd:complexType>

  <!-- Base type for all Derived Artifacts in S-RAMP -->
  <xsd:complexType abstract="true" name="DerivedArtifactType">
    <xsd:complexContent>
      <xsd:extension base="s-ramp:BaseArtifactType">
          <xsd:sequence>
         <xsd:element name="relatedDocument" type="tns:target" minOccurs="1" maxOccurs="1"/>
          </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- Base type for all Documents in S-RAMP -->
  <xsd:complexType abstract="true" name="DocumentArtifactType">
    <xsd:complexContent>
      <xsd:extension base="s-ramp:BaseArtifactType">
         <xsd:attribute name="contentType" type="xsd:string"/>
        <xsd:attribute name="contentSize" type="xsd:long"/>
        <xsd:anyAttribute namespace="##any"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- Base Type for all XML documents.  Specific document types extend XmlDocument -->
  <xsd:complexType name="XmlDocument">
    <xsd:complexContent>
      <xsd:extension base="s-ramp:DocumentArtifactType">
        <xsd:attribute name="contentEncoding" type="xsd:string"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- Base type for all User Defined Artifacts in S-RAMP -->
  <xsd:complexType name="UserDefinedArtifactType">
```

```
    <xsd:complexContent>
      <xsd:extension base="s-ramp:BaseArtifactType">
        <xsd:attribute name="userType" type="xsd:string"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- Relationship target artifact's UUID.  Used by all types of relationships -->
  <xsd:complexType name="target">
   <xsd:simpleContent>
      <xsd:extension base=”xsd:string”>
         <xsd:anyAttribute namespace=”##any”/>
      </xsd:extension>
   </xsd:simpleContent>
  </xsd:complexType>

<!--                 Global Element Declarations                    -->

  <!-- Relationship element used for all GENERIC (user-defined) Relationships -->
  <xsd:element name="relationship">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="tns:relationshipType" minOccurs="1" maxOccurs="1" />
         <xsd:element name="relationshipTarget" type="tns:target" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##any"/>
   </xsd:complexType>
  </xsd:element>

  <!-- Stored Queries Artifact element -->
  <xsd:element name="StoredQuery">
   <xsd:complexType>
     <xsd:sequence>
        <xsd:element ref="tns:propertyName" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="tns:queryExpression" />
     </xsd:sequence>
      <xsd:attribute name="queryName" type="xsd:string"/>
      <xsd:anyAttribute namespace="##any"/>
   </xsd:complexType>
  </xsd:element>

  <!--  Property -->
  <xsd:element name="property">
     <xsd:complexType>
        <xsd:sequence>
           <xsd:element ref="tns:propertyName" minOccurs="1" maxOccurs="1" />
           <xsd:element ref="tns:propertyValue" minOccurs="1" maxOccurs="1" />
        </xsd:sequence>
        <xsd:anyAttribute namespace="##any"/>
     </xsd:complexType>
  </xsd:element>

  <xsd:element name="relationshipType" type="xsd:string"/>
  <xsd:element name="classifiedBy" type="xsd:anyURI"/>
  <xsd:element name="propertyName" type="xsd:string" />
  <xsd:element name="propertyValue" type="xsd:string"/>
  <xsd:element name="queryExpression" type="xsd:string"/>
  <!-- The sourceId and targetId contain the UUID's corresponding to a
      relationship's source and target  -->
  <xsd:element name="sourceId" type="xsd:string"/>
  <xsd:element name="targetId" type="xsd:string"/>


  </xsd:schema>
```

B SOA Model Schema

For convenience, an S-RAMP SOA Model Schema xsd file is also provided at:

http://s-ramp.org/2010/specification/schemas/soamodel.xsd

NOTE: this schema is based upon the state of the SOA Ontology work at the time of publication. It will be revised as necessary to reflect any changes made by the TOG SOA Ontology group upon final publication.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   xmlns:tns="http://s-ramp.org/xmlns/2010/s-ramp"
   xmlns:s-ramp="http://s-ramp.org/xmlns/2010/s-ramp"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://s-ramp.org/xmlns/2010/s-ramp"
   elementFormDefault="qualified" version="1.0">

   <xsd:include schemaLocation="coremodel.xsd"/>
   <xsd:include schemaLocation="serviceimplementationmodel.xsd"/>

   <xsd:complexType abstract="true" name="SoaModelType">
      <xsd:complexContent>
         <xsd:extension base="s-ramp:BaseArtifactType">
            <xsd:sequence>
               <!-- Modeled Relationship to abstract DocumentArtifactType-->
               <xsd:element minOccurs="0" name="documentation"
                  type="s-ramp:target" maxOccurs="unbounded">
               </xsd:element>
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ServiceInterface">
      <xsd:complexContent>
         <xsd:extension base="tns:SoaModelType">
            <xsd:sequence>
               <!-- Modeled Relationship to abstract DocumentArtifactType: -->
               <xsd:element minOccurs="0" maxOccurs="1" name="interfaceDefinedBy"
                  type="s-ramp:DerivedArtifactType">
               </xsd:element>
               <!-- Modeled Relationship to ServiceOperation: -->
               <xsd:element minOccurs="0" name="hasOperation"
                  type="s-ramp:target">
               </xsd:element>
               <!-- Modeled Relationship to InformationType: -->
               <xsd:element minOccurs="0" name="hasOutput"
                  type="s-ramp:target" maxOccurs="unbounded">
               </xsd:element>
               <!-- Modeled Relationship to InformationType: -->
               <xsd:element minOccurs="0" name="hasInput"
                  type="s-ramp:target" maxOccurs="unbounded">
               </xsd:element>
               <!-- Modeled Relationship to Service: -->
               <xsd:element minOccurs="0" name="isInterfaceOf"
                  type="s-ramp:target" maxOccurs="unbounded">
```

```
            </xsd:element>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Service">
   <xsd:complexContent>
      <xsd:extension base="tns:Element">
         <xsd:sequence>
            <!-- Modeled Relationship to ServiceContract: -->
            <xsd:element minOccurs="0" name="hasContract"
               type="s-ramp:target" maxOccurs="unbounded">
            </xsd:element>
            <!-- Modeled Relationship to ServiceInterface: -->
            <xsd:element minOccurs="1" name="hasInterface"
               type="s-ramp:target" maxOccurs="unbounded">
            </xsd:element>
            <!-- Modeled Relationship to ServiceInstance: -->
            <xsd:element minOccurs="0" maxOccurs="1"
               name="hasInstance" type="s-ramp:target">
            </xsd:element>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Effect">
   <xsd:complexContent>
      <xsd:extension base="tns:SoaModelType">
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Event">
   <xsd:complexContent>
      <xsd:extension base="tns:SoaModelType">
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="InformationType">
   <xsd:complexContent>
      <xsd:extension base="tns:SoaModelType">
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Policy">
   <xsd:complexContent>
      <xsd:extension base="tns:SoaModelType">
         <xsd:sequence>
            <!-- Modeled Relationship to PolicySubject: -->
            <xsd:element minOccurs="0" name="appliesTo"
               type="s-ramp:target" maxOccurs="unbounded">
            </xsd:element>
```

```
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="PolicySubject">
   <xsd:complexContent>
      <xsd:extension base="tns:SoaModelType">
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="HumanActor">
   <xsd:complexContent>
      <xsd:extension base="tns:Element">
         <xsd:sequence>
            <!-- Modeled Relationship to Task: -->
            <xsd:element minOccurs="0" name="does"
               type="s-ramp:target" maxOccurs="unbounded">
            </xsd:element>
            <!-- Modeled Relationship to Policy: -->
            <xsd:element minOccurs="0" name="setsPolicy"
               type="s-ramp:target" maxOccurs="unbounded">
            </xsd:element>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Element">
   <xsd:complexContent>
      <xsd:extension base="tns:PolicySubject">
         <xsd:sequence>
            <!-- Modeled Relationship to Element: -->
            <xsd:element minOccurs="0" name="represents"
               type="s-ramp:target" maxOccurs="unbounded">
            </xsd:element>
            <!-- Modeled Relationship to Element: -->
            <xsd:element minOccurs="0" name="uses"
               type="s-ramp:target" maxOccurs="unbounded">
            </xsd:element><!-- Modeled Relationship to Service: -->
            <xsd:element minOccurs="0" name="performs"
               type="s-ramp:target" maxOccurs="unbounded">
            </xsd:element>
            <!-- Modeled Relationship to Orchestration: -->
            <xsd:element minOccurs="0" name="directsOrchestration"
               type="s-ramp:target" maxOccurs="1">
            </xsd:element>
            <!-- Modeled Relationship to OrchestrationProcess: -->
            <xsd:element minOccurs="0" name="directsOrchestrationProcess"
               type="s-ramp:target" maxOccurs="1">
            </xsd:element>
            <!-- Modeled Relationship to Event: -->
            <xsd:element minOccurs="0" name="generates"
               type="s-ramp:target" maxOccurs="unbounded">
            </xsd:element>
```

```
            <!-- Modeled Relationship to Event: -->
            <xsd:element minOccurs="0" name="respondsTo"
               type="s-ramp:target" maxOccurs="unbounded">
            </xsd:element>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ServiceContract">
   <xsd:complexContent>
      <xsd:extension base="s-ramp:PolicySubject">
         <xsd:sequence>
            <!-- Modeled Relationship to HumanActor: -->
            <xsd:element minOccurs="0" name="involvesParty"
               type="s-ramp:target" maxOccurs="unbounded">
            </xsd:element>
            <!-- Modeled Relationship to Effect: -->
            <xsd:element minOccurs="1" name="specifies"
               type="s-ramp:target" maxOccurs="unbounded">
            </xsd:element>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="System">
   <xsd:complexContent>
      <xsd:extension base="tns:Element">
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Composition">
   <xsd:complexContent>
      <xsd:extension base="tns:System">
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Choreography">
   <xsd:complexContent>
      <xsd:extension base="tns:Composition">
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Collaboration">
   <xsd:complexContent>
      <xsd:extension base="tns:Composition">
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Orchestration">
   <xsd:complexContent>
```

```
         <xsd:extension base="tns:Composition">
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>

   <xsd:complexType name="Process">
      <xsd:complexContent>
         <xsd:extension base="tns:Composition">
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>

   <xsd:complexType name="ChoreographyProcess">
      <xsd:complexContent>
         <xsd:extension base="tns:Process">
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>

   <xsd:complexType name="CollaborationProcess">
      <xsd:complexContent>
         <xsd:extension base="tns:Process">
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>

   <xsd:complexType name="OrchestrationProcess">
      <xsd:complexContent>
         <xsd:extension base="tns:Process">
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>

   <xsd:complexType name="Task">
      <xsd:complexContent>
         <xsd:extension base="tns:Element">
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>

   <xsd:complexType name="ServiceComposition">
      <xsd:complexContent>
         <xsd:extension base="tns:Composition"></xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
</xsd:schema>
```

C Service Implementation Model Schema

For convenience, an S-RAMP Service Implementation Model Schema xsd file is also provided at:

http://s-ramp.org/2010/specification/schemas/serviceimplementationmodel.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   xmlns:tns="http://s-ramp.org/xmlns/2010/s-ramp"
   xmlns:s-ramp="http://s-ramp.org/xmlns/2010/s-ramp"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://s-ramp.org/xmlns/2010/s-ramp"
   elementFormDefault="qualified" version="1.0">

   <xsd:include schemaLocation="coremodel.xsd"/>
   <xsd:include schemaLocation="soamodel.xsd"/>

   <!--  All Service Implementation Model artifacts inherit from ServiceImplementationModelType. Service
Implementation Model Artifacts
      can have associated documents. Service Implementation Model Artifacts can have dependencies upon other
Service
      Implementation Model Artifacts.  -->
   <!-- Service Model Artifact Type -->
   <xsd:complexType name="ServiceImplementationModelType" abstract="true">
      <xsd:complexContent>
         <xsd:extension base="s-ramp:BaseArtifactType">
            <xsd:sequence>
               <!-- Modeled Relationship(s) to documentArtifactType: -->
               <xsd:element name="documentation" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
            </xsd:sequence>
            <xsd:anyAttribute namespace="##any"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>

   <xsd:complexType name="ServiceInstance">
      <xsd:complexContent>
         <xsd:extension base="tns:ServiceImplementationModelType">
            <xsd:sequence>
               <!-- Modeled Relationship(s) to ServiceInstance(s): -->
               <xsd:element name="uses" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
               <!-- Modeled Relationship(s) to ServiceEndpoint(s): -->
               <xsd:element name="describedBy" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
               <xsd:element name="extension" type="tns:ExtensionType" minOccurs="0"/>
               <xsd:element name="end" type="xsd:string" minOccurs="1" maxOccurs="1"/>
               <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
            </xsd:sequence>
            <xsd:anyAttribute namespace="##any"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>

   <xsd:complexType name="ServiceOperation">
      <xsd:complexContent>
         <xsd:extension base="tns:ServiceImplementationModelType">
            <xsd:sequence>
               <!-- Modeled Relationship to the DerivedArtifactType that defines this ServiceOperation: -->
               <xsd:element name="operationDefinedBy" type="s-ramp:target" minOccurs="0" maxOccurs="1" />
               <xsd:element name="extension" type="tns:ExtensionType" minOccurs="0"/>
               <xsd:element name="end" type="xsd:string" minOccurs="1" maxOccurs="1"/>
               <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:anyAttribute namespace="##any"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>

   <xsd:complexType name="ServiceEndpoint">
      <xsd:complexContent>
         <xsd:extension base="tns:ServiceImplementationModelType">
            <xsd:sequence>
```

```
            <!-- Modeled Relationship with DerivedArtifactType which defines this ServiceEndpoint: -->
            <xsd:element name="endpointDefinedBy" type="s-ramp:target" minOccurs="0" maxOccurs="1" />
            <xsd:element name="extension" type="tns:ExtensionType" minOccurs="0"/>
            <xsd:element name="end" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="url" type="xsd:anyURI"/>
         <xsd:anyAttribute namespace="##any"/>
      </xsd:extension>
   </xsd:complexContent>
 </xsd:complexType>

 <!-- Organization artifact Type. -->
 <xsd:complexType name="Organization">
   <xsd:complexContent>
      <xsd:extension base="s-ramp:HumanActor">
         <xsd:sequence>
            <!-- Modeled Relationship(s) to ServiceImplementationModelType: -->
            <xsd:element name="provides" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="extension" type="tns:ExtensionType" minOccurs="0"/>
            <xsd:element name="end" type="xsd:string" minOccurs="1" maxOccurs="1"/>
            <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
         </xsd:sequence>
         <xsd:anyAttribute namespace="##any"/>
      </xsd:extension>
   </xsd:complexContent>
 </xsd:complexType>

<!-- Generic Extension Type for all un-used element extension points in tns -->
<xsd:complexType name="ExtensionType">
  <xsd:sequence>
    <xsd:any processContents="lax" minOccurs="1"
            maxOccurs="unbounded" namespace="##targetNamespace"/>
  </xsd:sequence>
  <xsd:anyAttribute namespace="##any"/>
</xsd:complexType>

</xsd:schema>
```

D Derived Model Schemas

The subsections which follow define the schemas associated with each of the defined Derived Models in S-RAMP.

D.1 Policy Model Schema

For convenience, an S-RAMP Policy Model Schema xsd file is also provided at:

http://s-ramp.org/2010/specification/schemas/policymodel.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   xmlns:tns="http://s-ramp.org/xmlns/2010/s-ramp"
   xmlns:s-ramp="http://s-ramp.org/xmlns/2010/s-ramp"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://s-ramp.org/xmlns/2010/s-ramp"
   elementFormDefault="qualified" version="1.0">
   <xsd:include schemaLocation="coremodel.xsd"/>
   <xsd:complexType name="PolicyAttachment">
      <xsd:complexContent>
         <xsd:extension base="s-ramp:DerivedArtifactType">
            <xsd:sequence>
               <!-- Derived Relationship to any other artifact (BaseArtifactType): -->
               <xsd:element name="appliesTo" type="s-ramp:target" minOccurs="1" maxOccurs="unbounded" />
               <!-- Derived Relationship to a Policy Expression artifact: -->
               <xsd:element name="policies" type="s-ramp:target" minOccurs="1" maxOccurs="unbounded" />
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="PolicyExpression">
      <xsd:complexContent>
         <xsd:extension base="s-ramp:DerivedArtifactType"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="PolicyDocument">
      <xsd:complexContent>
         <xsd:extension base="s-ramp:XmlDocument"/>
      </xsd:complexContent>
   </xsd:complexType>
</xsd:schema>
```

D.2 XSD Model Schema

For convenience, an S-RAMP XSD Model Schema xsd file is also provided at:

http://s-ramp.org/2010/specification/schemas/xsdmodel.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   xmlns:tns="http://s-ramp.org/xmlns/2010/s-ramp"
   xmlns:s-ramp="http://s-ramp.org/xmlns/2010/s-ramp"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://s-ramp.org/xmlns/2010/s-ramp"
   elementFormDefault="qualified" version="1.0">

   <xsd:include schemaLocation="coremodel.xsd"/>

   <xsd:complexType name="XsdDocument">
      <xsd:complexContent>
         <xsd:extension base="s-ramp:XmlDocument">
            <xsd:sequence>
               <!-- Derived Relationships with (other)XmlDocument artifact(s): -->
               <xsd:element name="importedXsds" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
               <xsd:element name="includedXsds" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
               <xsd:element name="redefinedXsds" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="AttributeDeclaration">
      <xsd:complexContent>
         <xsd:extension base="s-ramp:DerivedArtifactType">
            <xsd:attribute name="NCName" type="xsd:NCName"/>
            <xsd:attribute name="namespace" type="xsd:anyURI"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ElementDeclaration">
      <xsd:complexContent>
         <xsd:extension base="s-ramp:DerivedArtifactType">
            <xsd:attribute name="NCName" type="xsd:NCName"/>
            <xsd:attribute name="namespace" type="xsd:anyURI"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="XsdType" abstract="true">
      <xsd:complexContent>
         <xsd:extension base="s-ramp:DerivedArtifactType"/>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="ComplexTypeDeclaration">
      <xsd:complexContent>
         <xsd:extension base="tns:XsdType">
            <xsd:attribute name="NCName" type="xsd:NCName"/>
            <xsd:attribute name="namespace" type="xsd:anyURI"/>
         </xsd:extension>
```

```
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="SimpleTypeDeclaration">
      <xsd:complexContent>
         <xsd:extension base="tns:XsdType">
            <xsd:attribute name="NCName" type="xsd:NCName"/>
            <xsd:attribute name="namespace" type="xsd:anyURI"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
</xsd:schema>
```

D.3 WSDL Model Schema

For convenience, an S-RAMP WSDL Model Schema xsd file is also provided at:

http://s-ramp.org/2010/specification/schemas/wsdlmodel.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:tns="http://s-ramp.org/xmlns/2010/s-ramp"
   xmlns:s-ramp="http://s-ramp.org/xmlns/2010/s-ramp"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://s-ramp.org/xmlns/2010/s-ramp"
   elementFormDefault="qualified" version="1.0">

   <xsd:include schemaLocation="coremodel.xsd"/>
   <xsd:include schemaLocation="xsdmodel.xsd"/>
   <xsd:include schemaLocation="policymodel.xsd"/>

   <xsd:complexType name="WsdlDocument">
      <xsd:complexContent>
         <xsd:extension base="s-ramp:XmlDocument">
            <xsd:sequence>
               <!-- Derived Relationships: -->
               <xsd:element name="importedXsds" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
               <xsd:element name="includedXsds" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
               <xsd:element name="redefinedXsds" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
               <xsd:element name="importedWsdls" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
            </xsd:sequence>
            <xsd:attribute name="targetNamespace" type="xsd:anyURI" use="optional"/>
            <xsd:attribute name="xsdTargetNamespaces" type="xsd:anyURI" use="optional"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="WsdlDerivedArtifactType" abstract="true">
      <xsd:complexContent>
         <xsd:extension base="s-ramp:DerivedArtifactType">
            <xsd:sequence>
               <!-- Modeled "extension" relationship to any wsdlExtension artifact(s) -->
               <xsd:element name="extension" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
            </xsd:sequence>
            <xsd:attribute name="namespace" type="xsd:anyURI"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="WsdlService">
      <xsd:complexContent>
         <xsd:extension base="tns:NamedWsdlDerivedArtifactType">
            <xsd:sequence>
               <!-- Derived Relationship to Port(s) this Service has: -->
               <xsd:element name="port" type="s-ramp:target" minOccurs="1" maxOccurs="unbounded" />
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="Port">
      <xsd:complexContent>
```

```
      <xsd:extension base="tns:NamedWsdlDerivedArtifactType">
         <xsd:sequence>
            <!-- Derived Relationships with Binding artifact: -->
            <xsd:element name="Binding" type="s-ramp:target" />
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Binding">
   <xsd:complexContent>
      <xsd:extension base="tns:NamedWsdlDerivedArtifactType">
         <xsd:sequence>
            <!-- Derived Relationship to BindingOperation(s): -->
            <xsd:element name="bindingOperation" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
            <!-- Derived Relationship with this Binding's PortType: -->
            <xsd:element name="portType" type="s-ramp:target" />
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="PortType">
   <xsd:complexContent>
      <xsd:extension base="tns:NamedWsdlDerivedArtifactType">
         <xsd:sequence>
            <!-- Derived Relationship to this PortType's Operation(s): -->
            <xsd:element name="operation" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="BindingOperation">
   <xsd:complexContent>
      <xsd:extension base="tns:NamedWsdlDerivedArtifactType">
         <xsd:sequence>
            <!-- Derived Relationship to BindingOperationFault(s) for this BindingOperation: -->
            <xsd:element name="fault" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
            <!-- Derived Relationship to BindingOperationInput for this BindingOperation: -->
            <xsd:element name="input" type="s-ramp:target" minOccurs="0" maxOccurs="1" />
            <!-- Derived Relationship to BindingOperationOutput for this BindingOperation: -->
            <xsd:element name="output" type="s-ramp:target" minOccurs="0" maxOccurs="1" />
            <!-- Derived Relationship with Operation artifact: -->
            <xsd:element name="operation" type="s-ramp:target"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="BindingOperationInput">
   <xsd:complexContent>
      <xsd:extension base="tns:NamedWsdlDerivedArtifactType">
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="BindingOperationFault">
   <xsd:complexContent>
      <xsd:extension base="tns:NamedWsdlDerivedArtifactType"/>
   </xsd:complexContent>
```

```
</xsd:complexType>
<xsd:complexType name="Operation">
   <xsd:complexContent>
      <xsd:extension base="tns:NamedWsdlDerivedArtifactType">
         <xsd:sequence>
            <!-- Derived Relationships to OperationInput and OperationOutput for this Operation: -->
            <xsd:element name="input" type="s-ramp:target" />
            <xsd:element name="output" type="s-ramp:target" />
            <!-- Derived Relationship for fault(s) associated with this Operation: -->
            <xsd:element name="fault" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
   <!-- Only request / response is modeled -->
</xsd:complexType>
<xsd:complexType name="OperationInput">
   <xsd:complexContent>
      <xsd:extension base="tns:NamedWsdlDerivedArtifactType">
         <xsd:sequence>
            <!-- Derived Relationship with Operation artifact: -->
            <xsd:element name="operation" type="s-ramp:target" />
            <!-- Derived Relationship with Message artifact: -->
            <xsd:element name="message" type="s-ramp:target" />
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Fault">
   <xsd:complexContent>
      <xsd:extension base="tns:NamedWsdlDerivedArtifactType">
         <xsd:sequence>
            <!-- Derived Relationship with Message artifact: -->
            <xsd:element name="message" type="s-ramp:target" />
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Message">
   <xsd:complexContent>
      <xsd:extension base="tns:NamedWsdlDerivedArtifactType">
         <xsd:sequence>
            <!-- Derived Relationship to Part(s) of this Message: -->
            <xsd:element name="part" type="s-ramp:target" minOccurs="0" maxOccurs="unbounded" />
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Part">
   <xsd:complexContent>
      <xsd:extension base="tns:NamedWsdlDerivedArtifactType">
         <xsd:sequence>
            <!-- Derived Relationships with ElementDeclaraion and XSDType artifacts: -->
            <xsd:element name="type" type="s-ramp:target" minOccurs="0" maxOccurs="1" />
            <xsd:element name="element" type="s-ramp:target" minOccurs="0" maxOccurs="1" />
         </xsd:sequence>
      </xsd:extension>
```

```
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="BindingOperationOutput">
      <xsd:complexContent>
         <xsd:extension base="tns:NamedWsdlDerivedArtifactType">
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="OperationOutput">
      <xsd:complexContent>
         <xsd:extension base="tns:NamedWsdlDerivedArtifactType">
            <xsd:sequence>
               <!-- Derived Relationship with Message for this OperationOutput: -->
               <xsd:element name="message" type="s-ramp:target"/>
            </xsd:sequence>
            <xsd:attribute name="outputNCName" type="xsd:NCName" use="optional"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="NamedWsdlDerivedArtifactType" abstract="true">
      <xsd:complexContent>
         <xsd:extension base="tns:WsdlDerivedArtifactType">
            <xsd:attribute name="NCName" type="xsd:NCName" use="optional"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="WsdlExtension">
      <xsd:complexContent>
         <xsd:extension base="s-ramp:DerivedArtifactType">
            <xsd:attribute name="NCName" type="xsd:NCName" use="optional"/>
            <xsd:attribute name="namespace" type="xsd:anyURI"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
</xsd:schema>
```

D.4 SOAPWSDL Model Schema

For convenience, an S-RAMP SOAP WSDL Model Schema xsd file is also provided at:

http://s-ramp.org/2010/specification/schemas/soapwsdlmodel.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   xmlns:tns="http://s-ramp.org/xmlns/2010/s-ramp"
   xmlns:s-ramp="http://s-ramp.org/xmlns/2010/s-ramp"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://s-ramp.org/xmlns/2010/s-ramp"
   elementFormDefault="qualified" version="1.0">

   <xsd:include schemaLocation="coremodel.xsd"/>
   <xsd:include schemaLocation="wsdlmodel.xsd"/>

   <xsd:complexType name="SoapAddress">
      <xsd:complexContent>
```

```
         <xsd:extension base="s-ramp:WsdlExtension">
            <xsd:attribute name="soapLocation" type="xsd:anyURI"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
   <xsd:complexType name="SoapBinding">
      <xsd:complexContent>
         <xsd:extension base="s-ramp:WsdlExtension">
            <xsd:attribute name="style" type="xsd:string"/>
            <xsd:attribute name="transport" type="xsd:string"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
</xsd:schema>
```

E Pre-defined Relationships

This appendix summarizes all of the pre-defined relationships in S-RAMP.

Table 10: Pre-defined Relationships

Model Name	Derived Model?	Artifact Type Name	Relationship Type Name
Core	Yes	DerivedArtifactType	relatedDocument
SOA	No	SoaModelType	documentation
		Service	hasInstance
		Policy	policyDefinedBy
		InformationType	informationTypeDefinedBy
		ServiceInterface	hasOperation
		{See SOA Ontology for remaining Artifacts and relationships}	
Service Implementation	No	ServiceImplementationModelType	documentation
		Operation	provides
		ServiceInstance	uses
			hasServiceEndpoint
		ServiceOperation	operationDefinedBy
		ServiceEndpoint	endpointDefinedBy
Policy	Yes	PolicyAttachment	appliesTo
			policies
		XsdDocument	importedXsds
			includedXsds
			redefinedXsds
WSDL	Yes	WsdlDocument	importedXsds
			includedXsds
			redefinedXsds
			importedWsdls
		WsdlDerivedArtifactType	extension
		WsdlService	port
		Port	binding
		Binding	bindingOperation
			portType

PortType	operation
BindingOperation	fault
	input
	output
	operation
Operation	input
	output
	fault
OperationInput	operation
	message
Fault	message
Message	part
Part	type
	element
OperationOutput	message