

***Metadata and Annotation Two-Stage SSP Provisioning***

***GRA-UML***

**Version 0.6 (Draft for Review)**

|  |  |
| --- | --- |
| Delivery On: | **June 2014** |
|  |  |
| Prepared For:  Prepared By: | **IJIS Institute**  **Model Driven Solutions**  **Open Networks, Inc.** |

Table of Contents

[Objective 4](#_Toc390252078)

[Problem statement 4](#_Toc390252079)

[Approach Overview 4](#_Toc390252080)

[Annotation metadata in GRA-UML models 6](#_Toc390252081)

[The UML “Platform independent model” (PIM) 7](#_Toc390252082)

[Actors 7](#_Toc390252083)

[Use cases 8](#_Toc390252084)

[Interfaces 10](#_Toc390252085)

[Components and ports 10](#_Toc390252086)

[Collaborations & Interactions 11](#_Toc390252087)

[Modeling the GRA Service Specification 13](#_Toc390252088)

[The GRA Annotation Model 17](#_Toc390252089)

[Execution of Phase-2 provisioning 19](#_Toc390252090)

[Appendix A: Supporting Artifacts 20](#_Toc390252091)

[Appendix B: Revision History 20](#_Toc390252092)

Table of Figures

[Figure 1 Two-stage SSP provisioning 5](#_Toc390252093)

[Figure 2 SIRS example – Actors 7](#_Toc390252094)

[Figure 3 Corrections example - Actors 8](#_Toc390252095)

[Figure 4 SIRS Example - Use Cases 8](#_Toc390252096)

[Figure 5 Corrections example - Use Cases 9](#_Toc390252097)

[Figure 6 Corrections example - more Use Cases 9](#_Toc390252098)

[Figure 7 SIRS Example - Interfaces 10](#_Toc390252099)

[Figure 8 Corrections example - interfaces, components and data 10](#_Toc390252100)

[Figure 9 SIRS Example - Components and Ports 11](#_Toc390252101)

[Figure 10 SIRS Example - Collaboration 11](#_Toc390252102)

[Figure 11 Corrections Example - Collaboration 11](#_Toc390252103)

[Figure 12 SIRS Example - Interactions 12](#_Toc390252104)

[Figure 13 Corrections example - interactions 12](#_Toc390252105)

[Figure 14 SIRS Example - Annotations 14](#_Toc390252106)

[Figure 15 SIRS Example – SIP Annotations for Notification Service Interface 15](#_Toc390252107)

[Figure 16 SIRS Example – SIP Annotations for Relocation Service Interface 15](#_Toc390252108)

[Figure 17 Corrections example – annotations 16](#_Toc390252109)

[Figure 18 Corrections example - SIP annotations 17](#_Toc390252110)

[Figure 19 GRA Annotations Model 18](#_Toc390252111)

[Figure 20 GRA Annotations Model for WSDL 19](#_Toc390252112)

# Objective

This document specifies the GRA-UML approach to metadata and provisioning of generated SSPs through the use of model based annotations.

# Problem statement

GRA-UML has two potentially conflicting goals:

* That GRA UML should produce a fully GRA-conformant SSP
* That GRA-UML should provide flexibility in how SSPs are produced, what technologies are used and how these technologies are used.

The reason for this flexibility as that GRA is essentially “open ended” with respect to the service interaction profiles used as well as what specific technologies are used within a SIP. In addition, SSP architects frequently have additional requirements or styles with respect to the generated documentation. The following three alternatives have been considered to solve this dilemma but none are seen as acceptable solutions:

1. That the SSP provisioning would provide for a fixed set of options without substantial extension capability;
2. That the SSP would be modeled in detail at the “technology level” and encompass all the complexity of WSDL, policies and other selected options;
3. That developers would have to modify the QVT that produces a SSP – an unfamiliar technology to most dealing with SSPs and WSDL.

# Approach Overview

The approach in this document provides a 4th alternative that offers full extensibility while retaining simplicity for the SSP developer. The approach introduces another kind of developer, the *SSP template developer*. The template developer uses familiar technologies - primarily XSLT - to tailor SSPs to a particular style and set of technology choices.

In this approach the generation and provisioning of the final SSP is divided into two steps, called *pre-provisioning* and *post-provisioning*. The GRA-UML developer creates a model of the SSP containing two main aspects that separate concerns between the logical services and the GRA-specific requirements and metadata:

* The Logical Service Model - a platform-independent specification of the services automated by the SSP, in terms of SOA constructs such as Actors, Use Cases, Components, Ports, Interfaces etc. It contains “normal” UML structures that are used to define services without any special stereotypes or undue rules. These same UML elements may be used for purposes other than producing GRA specifications, such as producing implementations.
* The Annotation Model – a modeled specification of GRA-specific metadata, including technology and policy choices for the SSP, in cases where these choices cannot be inferred from the Logical Service Model by applying defaults.

The *pre-provisioning* step uses a standardized algorithm to generate an *annotated* *skeleton SSP*. It uses the OMG’s QVT (Query/View/Transformation) to transform the GRA-UML model into a standardized XML-based format that comprises the annotated skeleton SSP.

The *post-provisioning* step tailors the *annotated skeleton* *SSP* using an XSLT transformation template.

The complete process is shown in the figure below.

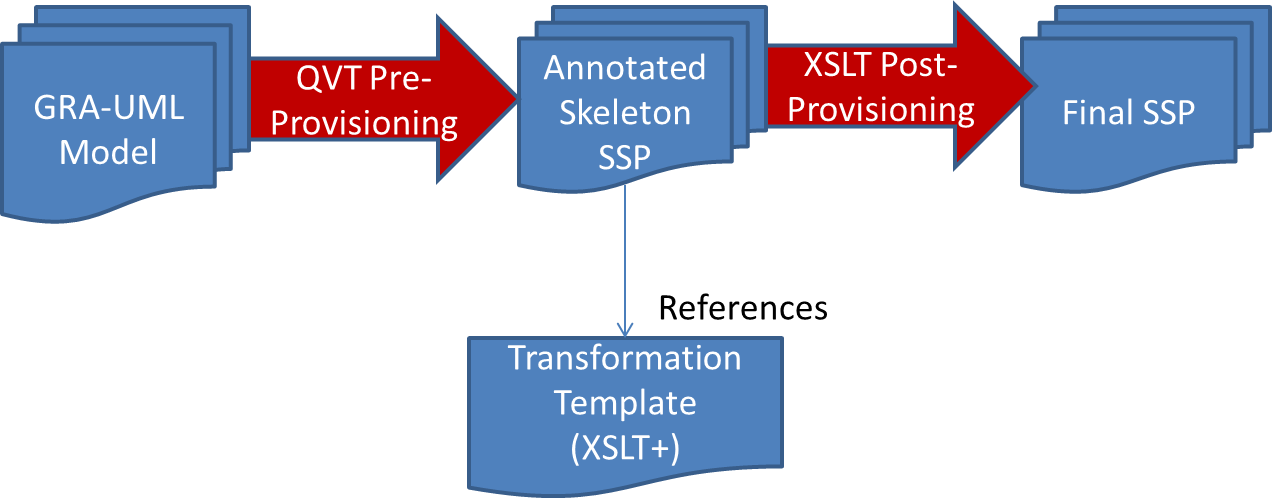


Figure 1 Two-stage SSP provisioning

The Skeleton SSP contains the following generated artifacts material to this discussion. Each of these artifacts is located in the SSP folder structure as defined by the GRA Service Specification Guidelines.

* metadata.xml. GRA defines a standard metadata.xsd which all metadata.xml documents must conform to. The metadata.xml in the annotated skeleton SSP is generated via a QVT transform from corresponding content in the GRA-UML model. The target metamodel for the QVT transform is a MOF model that corresponds to metadata.xsd according to the mapping defined by the Eclipse emf tooling.
* catalog.xml. GRA does not define a standard catalog.xsd. We will reverse-engineer such a schema from example catalog.xml documents. The catalog.xml in the annotated skeleton SSP is generated via a QVT transform from corresponding content in the GRA-UML model. The target metamodel for the QVT transform is a MOF model that corresponds to catalog.xsd according to the mapping defined by the Eclipse emf tooling.
* artifacts\service model\behavior model\XXX.xmi. This is the logical service model - a UML platform-independent model of the services, participants, interfaces and interactions involved in the Service Description.
* schemas\SIP\XXX-skeleton.wsdl file. This file is generated from corresponding content of the GRA-UML model. It provides the bare-bones structure of the wsdl file, but is likely to omit extensions and policy choices which will be fleshed-out by post-provisioning.
* artifacts\annotations.xml. This file is a serialized form of the annotation aspect of the GRA-UML model. It contains all of the modeled annotations as well as annotations inferred by applying defaults to the logical service model. It is intended to be used to provide parameter values for post-provisioning. It contains references to the logical service model.

Other artifacts that comprise the SSP are outside the scope of this document.

In post-provisioning the XSLT transformation reads the skeleton SSP and makes whatever adjustments, enhancements and inclusions that are required based on that template’s technology and stylistic choices. Any file in the SSP can be modified and other transformation tools may be called for complex transformation tasks. Template developers may also use the metadata to produce other artifacts in support of their GRA specification or development tasks.

The GRA-UML standard includes a baseline XSLT template that supports basic GRA web services. Templates can then add additional technology content, such as policy implementations. Policy implementations should be based on the *InteractionRequirements* as defined in GRA and modeled in the annotation model.

Within the annotation aspect of a GRA-UML model the architect can select the XSLT template specification to use and may also assign specific templates for use on any individual element of the GRA model; this allows specific technology choices to be made at any level. For example, a template could be provided to support REST web services and select REST services for a specific SIP or interface.

XSLT was chosen for the post-processing steps due to its familiarity and popularity among developers who typically operate at the WSDL/SOAP and integration technology levels. XSLT is also a proven and standard technology. However, XSLT is not well-suited for traversing complex models, so QVT is a better choice for the pre-provisioning step.

# Annotation metadata in GRA-UML models

GRA already defines a substantial amount of metadata. The GRA-UML annotation approach augments the GRA metadata with additional metadata to support post-provisioning. The additional metadata maps well to the structure of WSDL. Some of the metadata may be inferred by applying defaults to the logical service model; the modeler may override these defaults by explicit modeling.

GRA-UML specifies the *GRA Annotation Model*. This is a UML class model, shown in Figure 19 and Figure 20, which defines all of the metadata that can be used for GRA annotations. More details of its contents and examples of its use are provided later in this document.

The GRA annotation model is a NIEM-UML model. This capitalizes on the fact that NIEM-UML defines XSD and XML mappings well-understood by the GRA community.

Annotations in a particular GRA-UML model are represented by UML InstanceSpecifications. Each InstanceSpecification is classified by a Class in the GRA Annotation Model. The serialized form of each InstanceSpecification is an XML document fragment that conforms to the XSD type corresponding to the GRA Annotation classifier.

Note that this is different from the approach taken in NIEM-UML where UML stereotypes are used for metadata. UML InstanceSpecifications provide at least as much expressive power as stereotypes, as well as providing the ability to easily augment the annotations and model complex structures.

The relationships between annotation elements and the elements in the logical model that they annotate are represented by UML *realizations*. Realization is a standard UML construct that shows as an arrow between elements with a dashed line and an open triangular arrowhead.

Annotation elements are serialized as XML fragments conforming to the XSD types that correspond to the GRA Annotation model considered as a NIEM-UML model. Logical service elements are serialized as UML XMI elements. Realizations are serialized as ModelReferences, a form of traceability annotation within the serialized annotations.

# The UML “Platform independent model” (PIM)

GRA-UML is designed to interpret specific UML constructs. These constructs are in keeping with the SoaML services modeling standard profile, but use of SoaML stereotypes is optional. The following elements have meaning to GRA-UML and are shown with examples relating to the SIRS (SORNA Interjurisdictional Relocation Service) specification and to the Corrections specification.

## Actors

Actors are the business participants in the SSP’s subject community. They are typically people or organizations. Figure 2 shows Actors relating to SIRS and Figure 3 shows Actors relating to Corrections.

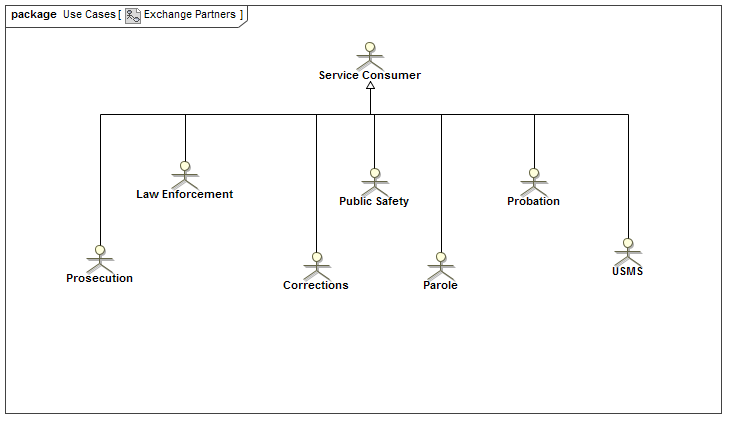


Figure 2 SIRS example – Actors

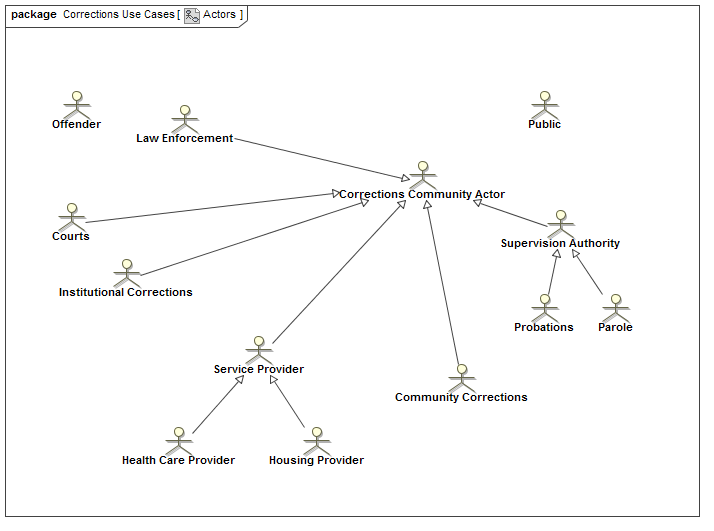


Figure 3 Corrections example - Actors

## Use cases

The “real world effect” of services is modeled as use cases involving actors. Each use case (oval) is a real world effect and a business interaction. Each relationship between a use case and an actor may be marked to indicate whether the actor is a provider or consumer of the service. Figure 4 shows Use Cases from the SIRS example; Figure 5 and Figure 6 show Use Cases from the Corrections example.

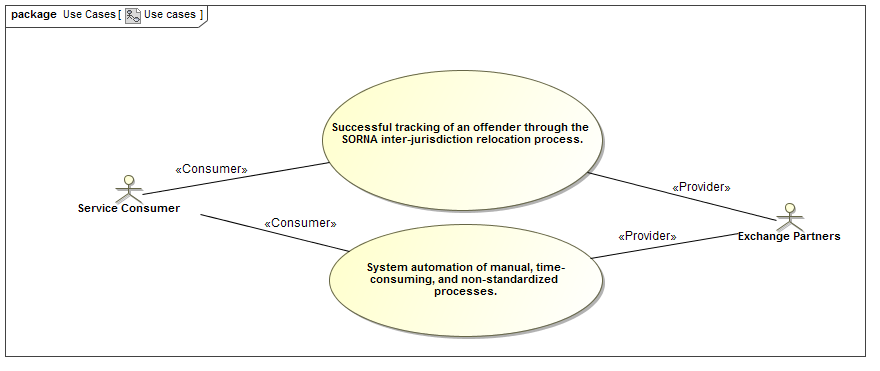


Figure 4 SIRS Example - Use Cases

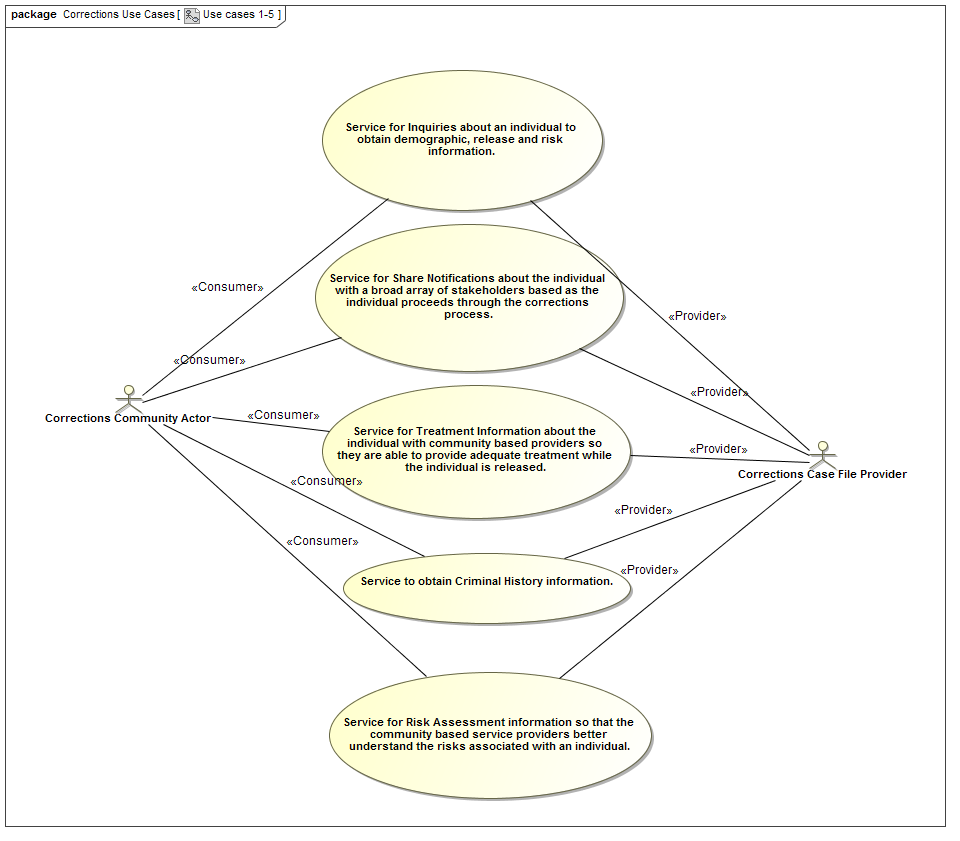


Figure 5 Corrections example - Use Cases

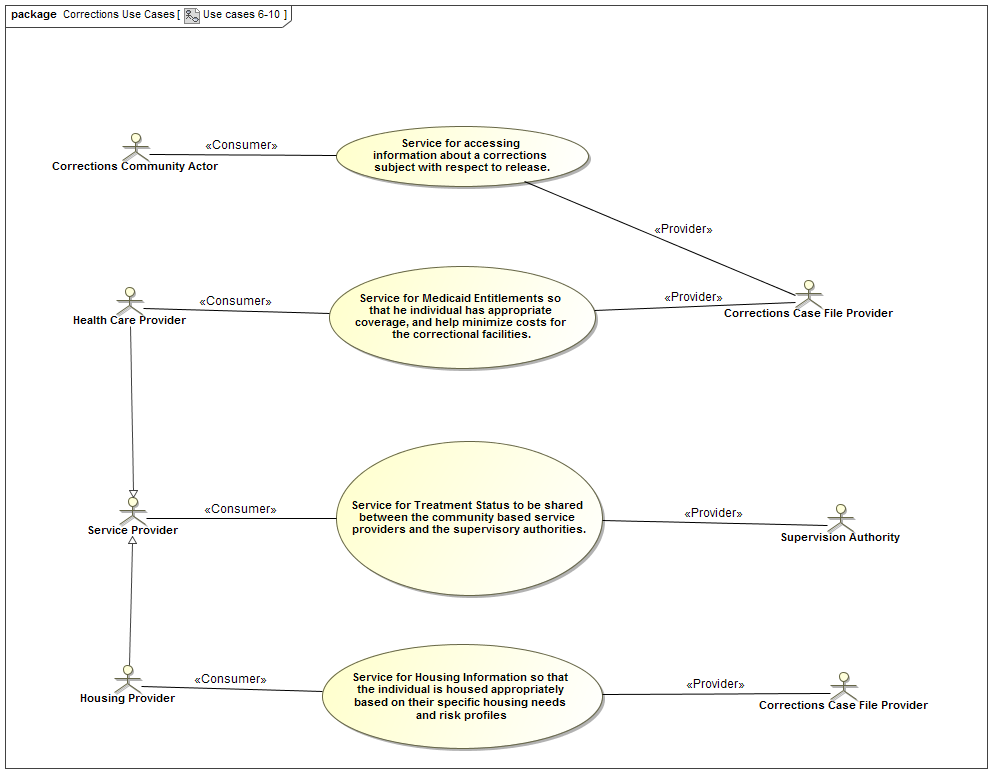


Figure 6 Corrections example - more Use Cases

## Interfaces

UML interfaces define the operations and signal receptions that service providers implement to expose a service. Figure 7 shows interfaces relating to SIRS. Figure 8 shows interfaces relating to Corrections, as well as Components (see below) and NIEM definitions for related message types.

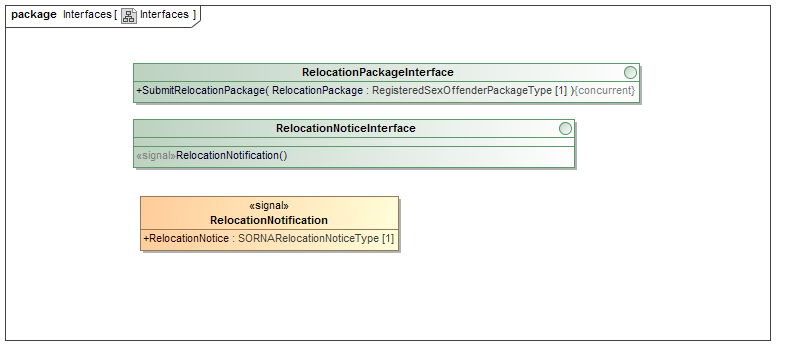


Figure 7 SIRS Example - Interfaces

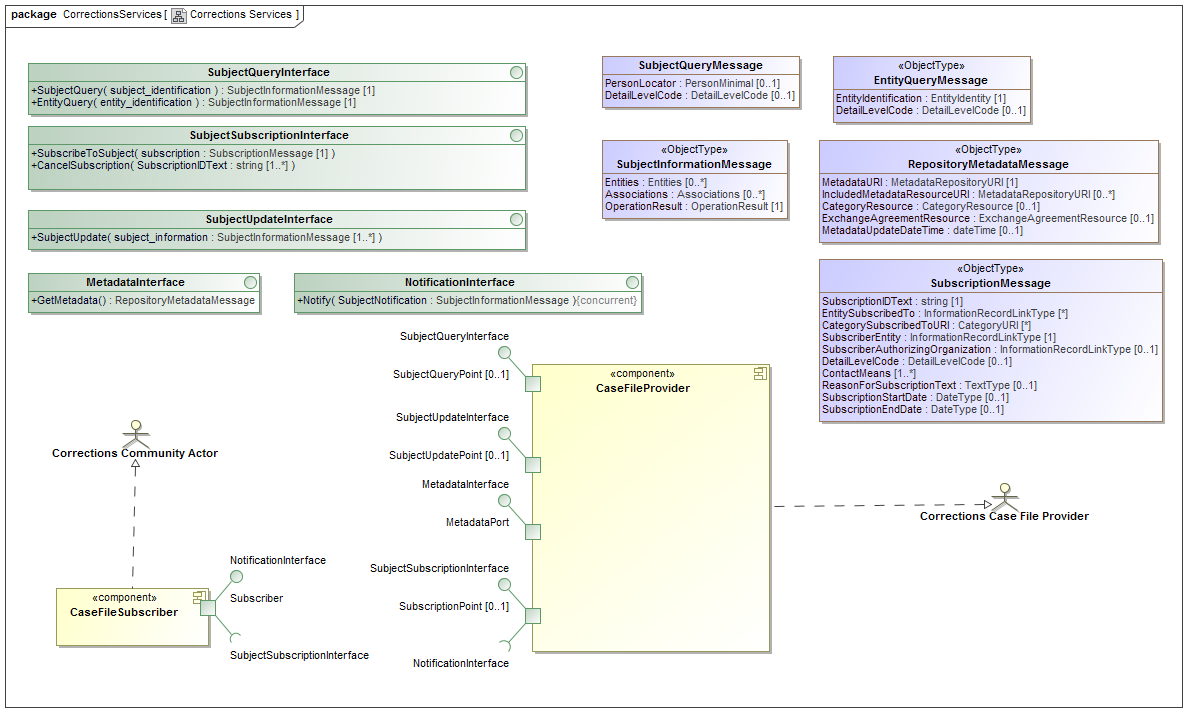


Figure 8 Corrections example - interfaces, components and data

## Components and ports

UML components represent the service provided by an actor. Each component has a set of ports that provide or use UML interfaces. A component may realize an Actor, to signify in the model that it represents the services provided by that Actor. SIRS components are shown in Figure 9, and Corrections components in Figure 8 which also shows Actor realizations.

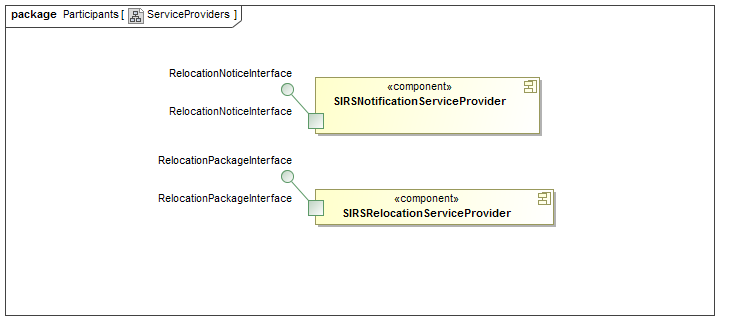


Figure 9 SIRS Example - Components and Ports

## Collaborations & Interactions

Collaborations are used to define the “community” for which the SSP is intended, providing a property for each actor and service component playing a role in that community.

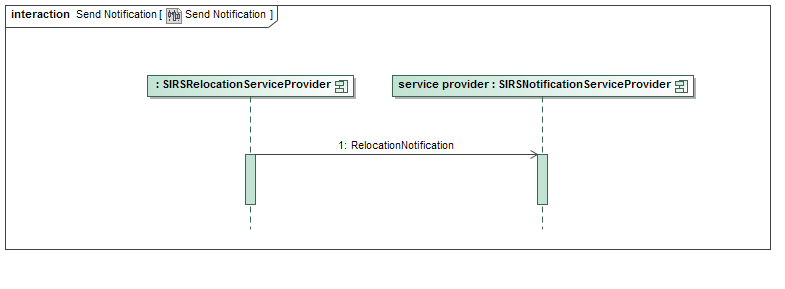


Figure 10 SIRS Example - Collaboration



Figure 11 Corrections Example - Collaboration

The collaboration that represents the community owns a set of interactions that define the choreography of exchanges between the actors to implement the use cases. Figure 12 shows interactions in the SIRS community. Figure 13 shows interactions in the Corrections community.



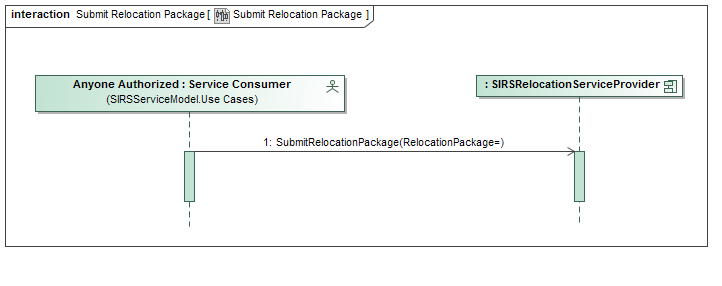


Figure 12 SIRS Example - Interactions

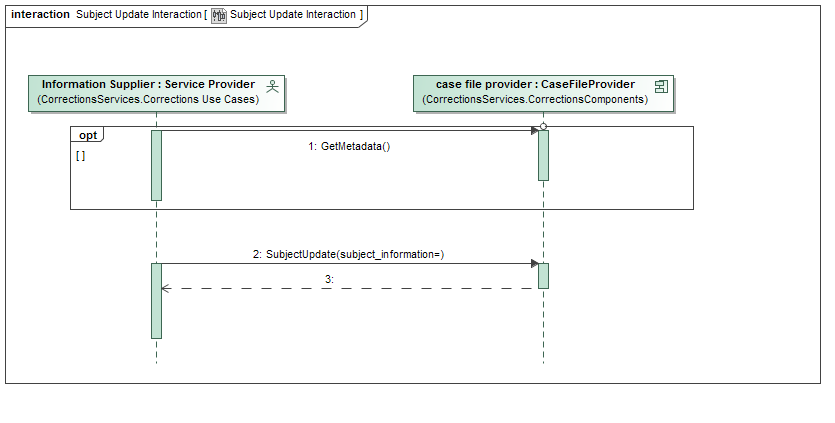
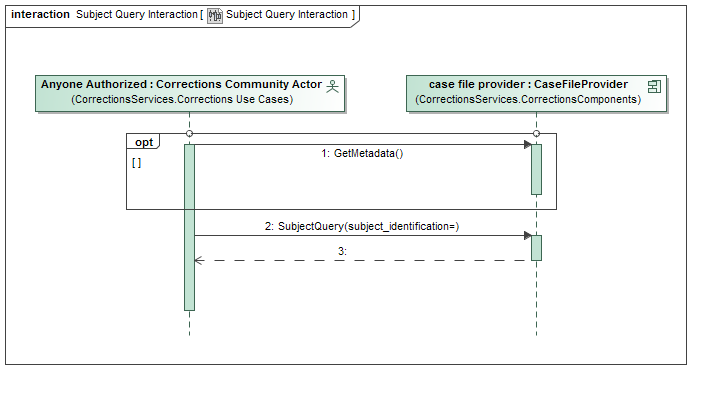


Figure 13 Corrections example - interactions

# Modeling the GRA Service Specification

The GRA Service Specification and its constituents are modeled as InstanceSpecifications classified by GRA annotation classes. Where appropriate these InstanceSpecifications realize the UML element that provides the foundation for that GRA element.

InstanceSpecifications need only be introduced in the user’s model to represent information that can’t be determined from the logical service model or defaults. The stage-1 provisioning process “fills out” missing annotation metadata elements and generates the annotations.xml file. The annotations cross-reference the annotated elements, providing the stage-2 provisioning with sufficient information to tailor each element and create the final SSP.

Figure 14 and Figure 15 below show an example SSP and Service Interface specification for the SIRS example using this approach. It may also be helpful to refer to the GRA Annotation Model shown in Figure 19 and Figure 20. Here are some points to note:

* **SIRS Service Description** represents an instance of the GRA Annotation class ServiceDescription. It *realizes* (i.e. annotates) the collaboration called **SIRS community**.
* Many of the constituents of SIRS Service Description are also represented by InstanceSpecifications, such as **Purpose** and **Scope** (instances of Description), **SLA** (an instance of ServiceLevelAgreement), and so on.
* The **RealWorldEffect** is modeled as an instance of UseCase, which realizes (annotates) the Use Cases package in the PIM.

The transformationURI slot in SIRS Service Description is intended to reference a single XSLT file that handles the transformation rules for an SSP. Without further marking such a transformation should be able to produce something valid.

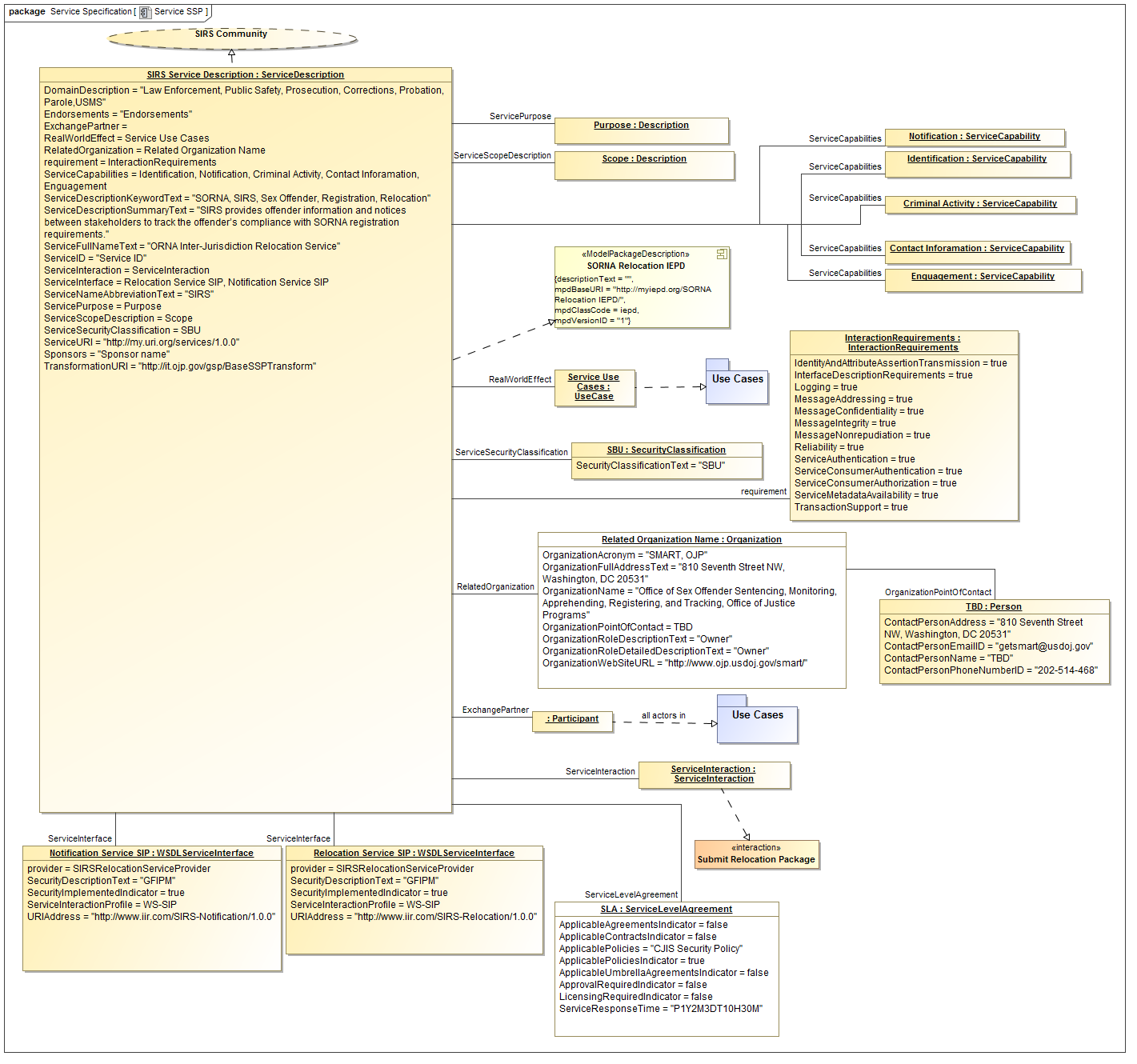
**

Figure 14 SIRS Example - Annotations

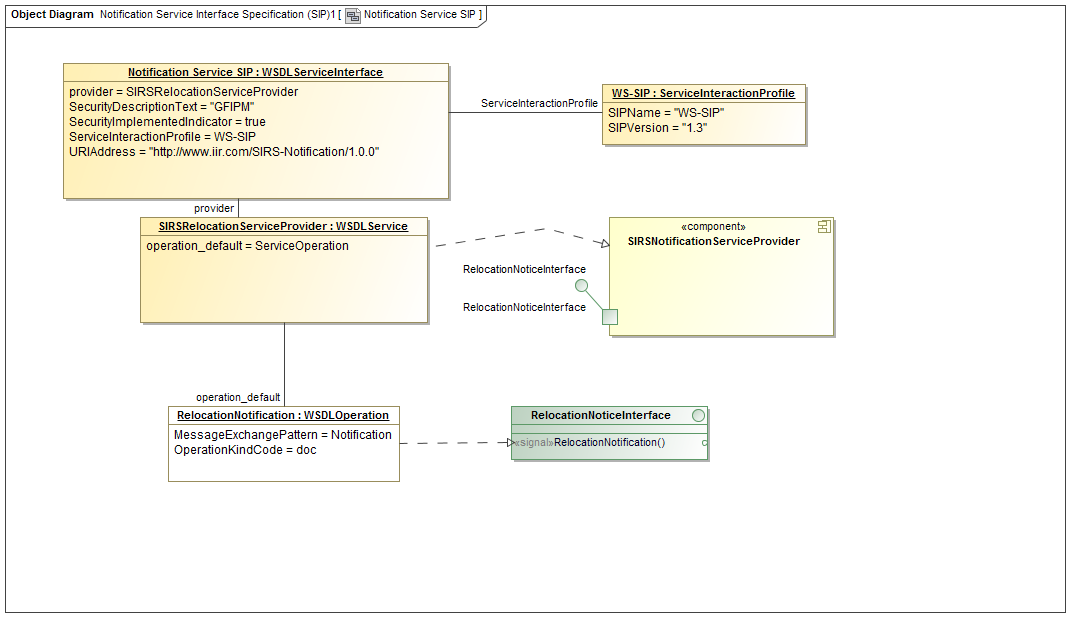


Figure 15 SIRS Example – SIP Annotations for Notification Service Interface

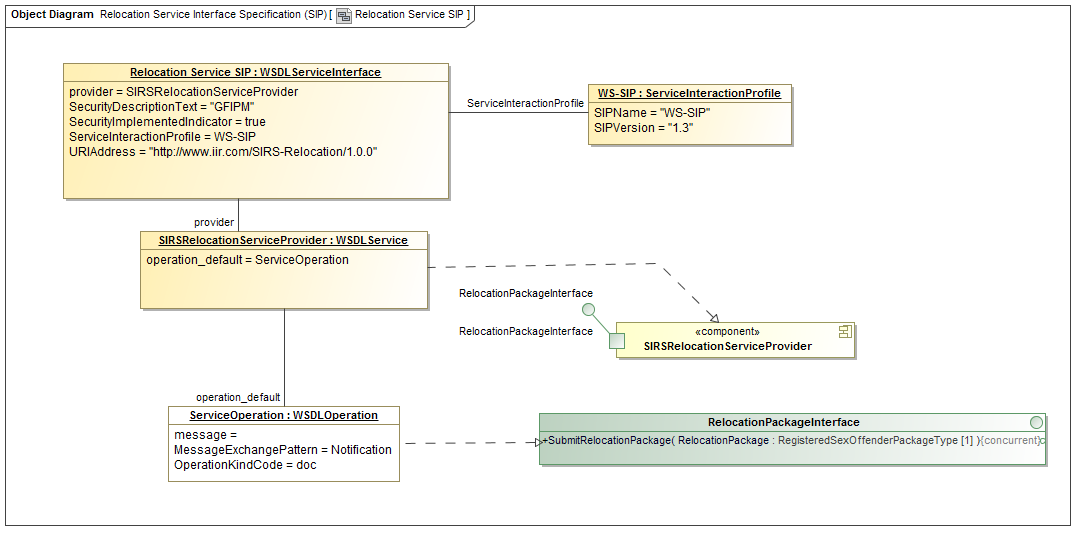
**

Figure 16 SIRS Example – SIP Annotations for Relocation Service Interface

The GRA architect has the option of providing additional parameters to further customize the phase-2 transformation. These are the *Template* and *flag* properties of each metadata element. The Template parameter “calls” a specific template in the transformation instead of using the default matching. This allows complete control over how an element is interpreted by invoking custom new templating logic.

The *flag* parameter defines a simple tag that can provide additional parameterization to the template. Flags are anything the user would like them to be to indicate technology mapping or business relevant choices. There are no flags shown in the example.

The chosen template must interpret the requirements represented by InteractionRequirements, annotations and choices represented in flags to produce the wsdl, policies, instances and documentation that satisfies those requirements. Unknown flags are ignored.

*Note*: Support “mode” as well?

Figure 19 below show an example SSP for the Corrections example. Here are some additional points to note about this example:

* The use of the Template property in the two WSDLServiceInterface instances, to signify the use of local templating logic for these instances.
* UML Usage («use») dependencies between the ServiceDescription and the related NIEM ModelPackageDescription elements.

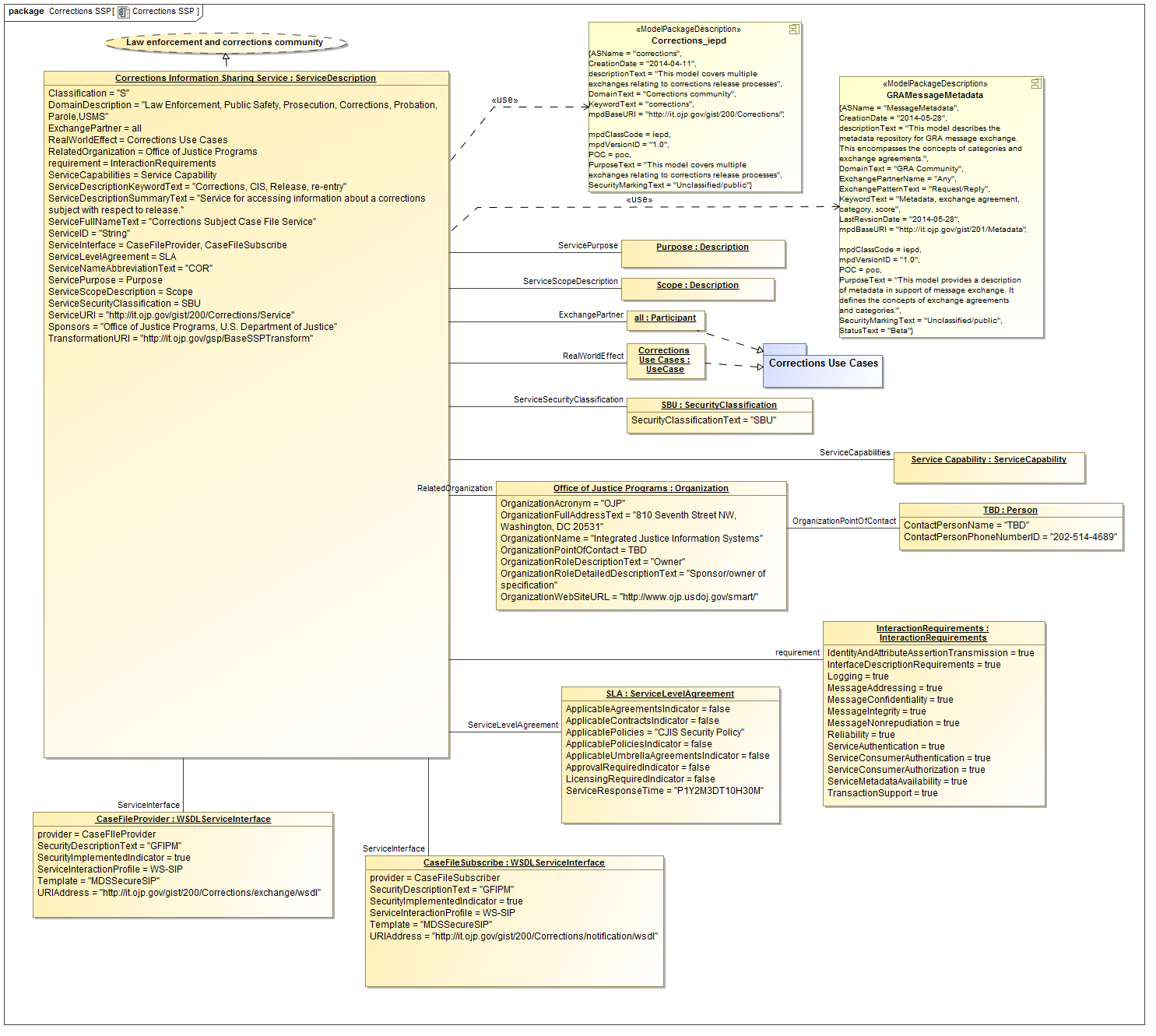


Figure 17 Corrections example – annotations

Figure 18 illustrates SIP annotations for the Corrections example. A point to note here is that the model only contains information that cannot be inferred from defaults. In Figure 18 a technology choice is made to use http\_get to implement the metadata interface instead of the default (SOAP). This choice is explicitly represented in the annotation model.

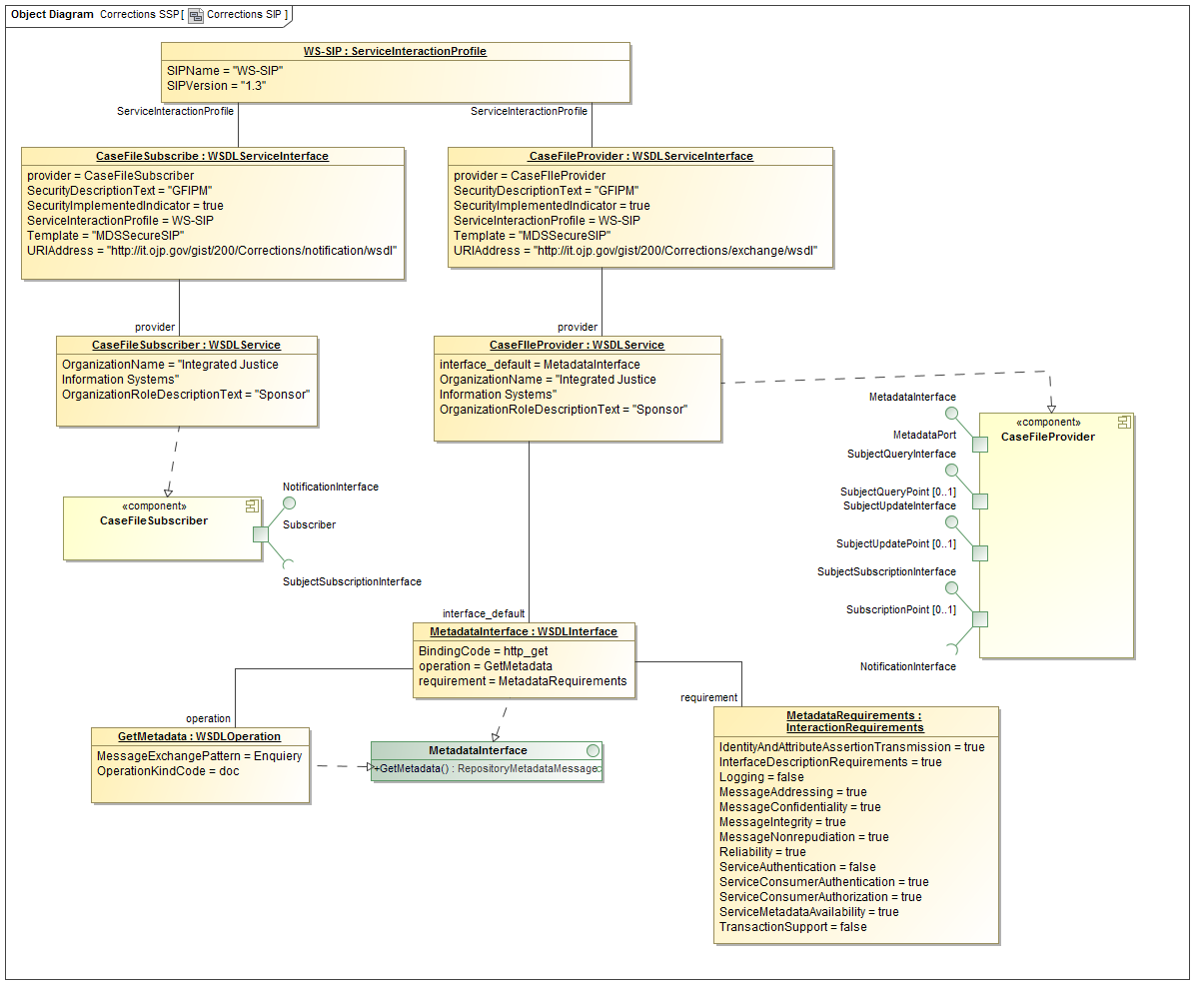


Figure 18 Corrections example - SIP annotations

# The GRA Annotation Model

As already observed, all of the InstanceSpecifications used to annotate a GRA-UML model represent instances of classes in the GRA Annotation Model. This is defined in UML and is also available as an XML schema. The annotations.xml file contains XML elements that conform to the GRA Annotations XML schema, and that are generated either directly from the modeled InstanceSpecifications or by applying default generation rules to the GRA-UML model.

The GRA Annotation Model is shown in Figure 19.

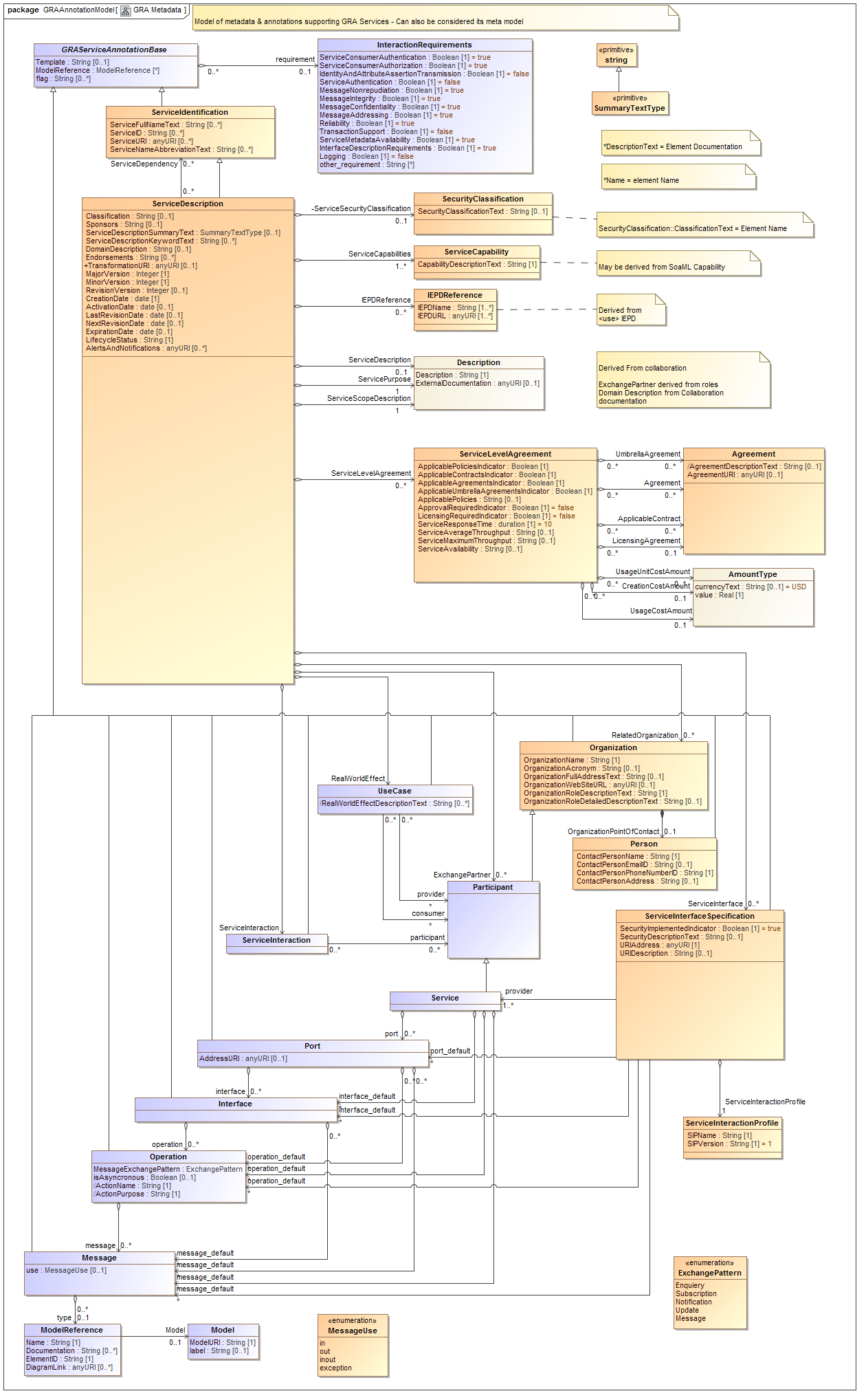


Figure 19 GRA Annotations Model

The orange classes are primarily defined by the GRA metadata structure whereas the blue classes extend this structure to include information required to populate and extend the technology artifacts (e.g. WSDL).

The model above defines the metadata for “generic” SSPs; however most SSPs use WSDL and perhaps SOAP. Specific extensions to the metadata classes are provided to define WSDL/SOAP specific choices. Use of these extensions is optional; defaults may also be used. The WSDL/SOAP extensions also illustrate how template developers can introduce specific metadata elements for their technology choices. The following model defines the WSDL/SOAP metadata classes.

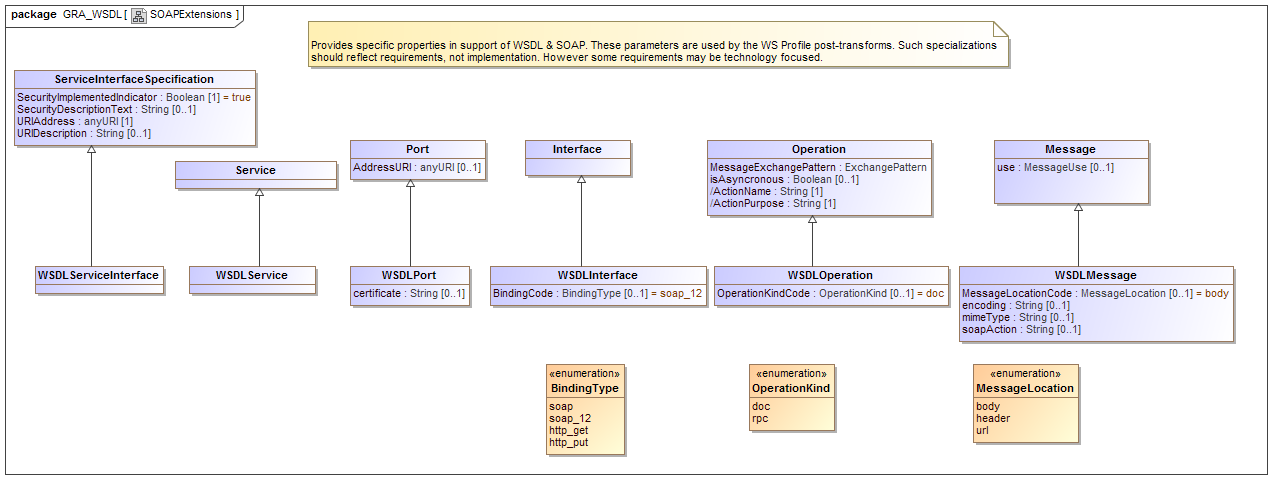


Figure 20 GRA Annotations Model for WSDL

# Execution of Phase-2 provisioning

As part of the GRA specification a Transformation template is specified. In the example case it is “http:it.ojp.gov/gsp/BaseSSPTransform”. This is an XSLT transform that is executed on each artifact of the skeleton SSP to produce the final SSP. The skeleton SSP combined with the annotation metadata provide the parameters for these transforms. There are essentially no rules about what can or cannot be done in a phase-2 transformation. If a user transformation is specified it is the responsibility of the provider to ensure that a GRA conformant SSP is produced.

# Appendix A: Supporting Artifacts

The following artifacts are provided as ancillary to this design document:

* GRAAnnotationModel.mdzip – the models in Figure 19 and Figure 20, together with a small UML profile that defines the stereotypes used to decorate the PIM as in Figure 4.
* GRAAnnotationIEPD.mdzip – a NIEM-UML IEPD that imports the annotation model and defines the production of an NIEM IEPD for GRA annotations.
* GRAAnnotationIEPD – The generated IEPD with schemas for the GRA annotation model. The generated IEPD is in a folder called GRAAnnotationIEPD-1.0.iepd and the generated schemas are in the GRAUML subfolder.
* SIRS-Service.mdzip – the SIRS example SSP modeled using GRA-UML according to this approach.
* Corrections\_SSP\_v\_1.0.0.mdzip – the Corrections SSP modeled using GRA-UML.

*Note*: The .mdzip files are MagicDraw 17.0.5 zipped project files.

# Appendix B: Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Editor | Change |
| 5/26/2014 | 0.1 | Cory Casanave | Initial rev |
| 6/4/2014 | 0.2 | Steve Cook | Substantial rewriting |
| 6/9/2014 | 0.5 | Steve Cook | Replacement of example |
| 6/11/2014 | 0.6 | Steve Cook | Addition of Corrections example |