iPlant Semantic Web Services
Use Case Document

Clark & Parsia
University of Arizona

May 20, 2010

Abstract

Contents

Todo list

1 Introduction

This document describes use cases for the main components of SSWAP.

2 Actors

Figure 1 shows the typical setup of the main components of SSWAP. The setup involves a client node, a provider's node, a discovery server, and one or more ontology servers. A client typically initiates the activity by first interacting with a discovery server to find the provider for the desired service, and then interacts with the provider's node directly. During the interactions between the client and the discovery server or the provider, messages in SSWAP protocol are exchanged. The messages contain semantic data which uses vocabulary from ontologies, which are accessible by one of the ontology servers.

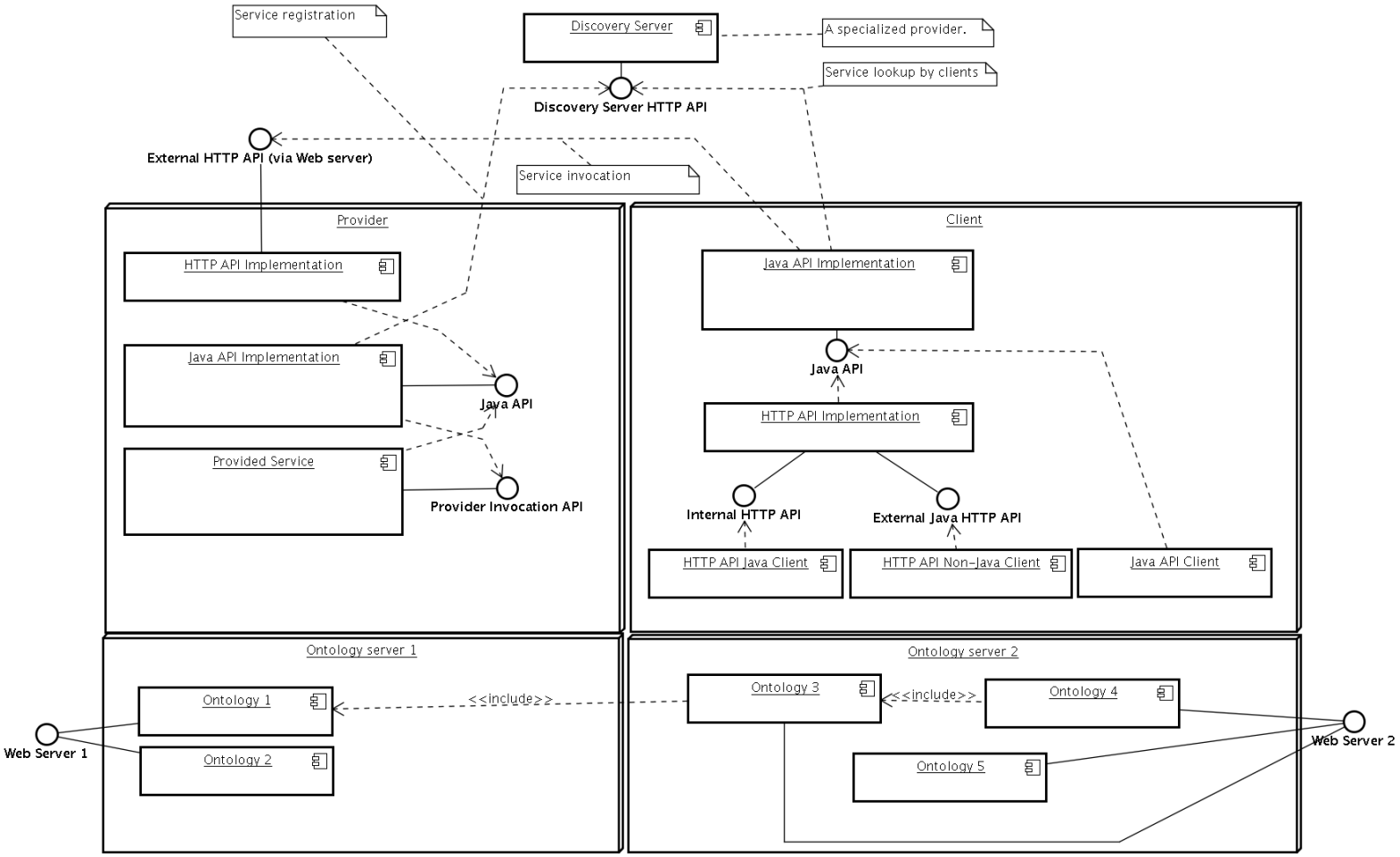

Figure 1: High-level diagram showing dependencies between basic SSWAP components, and their typical deployment across nodes on a computer network. The internal structure of the top node (Discovery Server) is not shown because it is equivalent to the Provider's. Also the calls to the ontology servers are not shown since all the other components rely on the vocabulary provided by these servers.

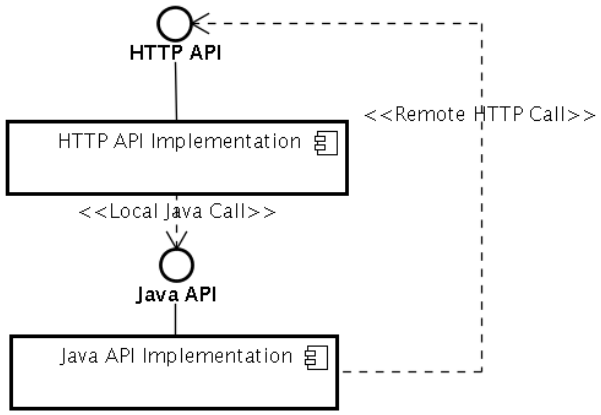

Figure 2: Dependency between HTTP API and Java API. HTTP API always calls Java API locally (using Java calls), while Java API calls HTTP API only when HTTP API is located on a remote computer (using HTTP network request).

It is worth noting the generic pattern in the dependencies between HTTP API and Java API (regardless whether they are located on the client side or on the provider's side). Figure 2 visualizes that pattern.

In this document, all components that are active (they can initiate interaction with another component) are considered actors in the use-cases. In fact, most of the components fit this definition – the only components that are passive are ontology servers.

Figure 3 shows the hierarchy of the actors. The actors can be divided into two separate groups: actors that are called by HTTP API (HTTP API Clients) and actors that are called by Java API (Java API Clients). It is also worth noting that both Java API and HTTP API exist in two different contexts: provider-side or client side.

2.1 Client actors

Client actor is the code or a person that initiates an interaction on its own (usually with the intention of calling a service provided as a SSWAP Resource). Clients can interact with both HTTP API (basic set of functionalities) or with Java API (full set of functionalities for advanced users).

2.1.1 HTTP API clients

HTTP API serves two main purposes: to expose an interface that can be access via a network call (HTTP protocol), and to provide a simplified interface for typical use-cases (e.g., for less advanced users).

Client side The HTTP API on the client side should be usable by both Java and non-Java clients. In order to facilitate non-Java clients, the HTTP API on the client side can be called locally using HTTP protocol (e.g., by using a loopback connection). At the same time, since HTTP API is implemented in Java, it would introduce unnecessary overhead to force Java clients to connect over loopback, if they can call the HTTP API methods directly (*internal HTTP API* on Figure 1). Therefore, we can distinguish two

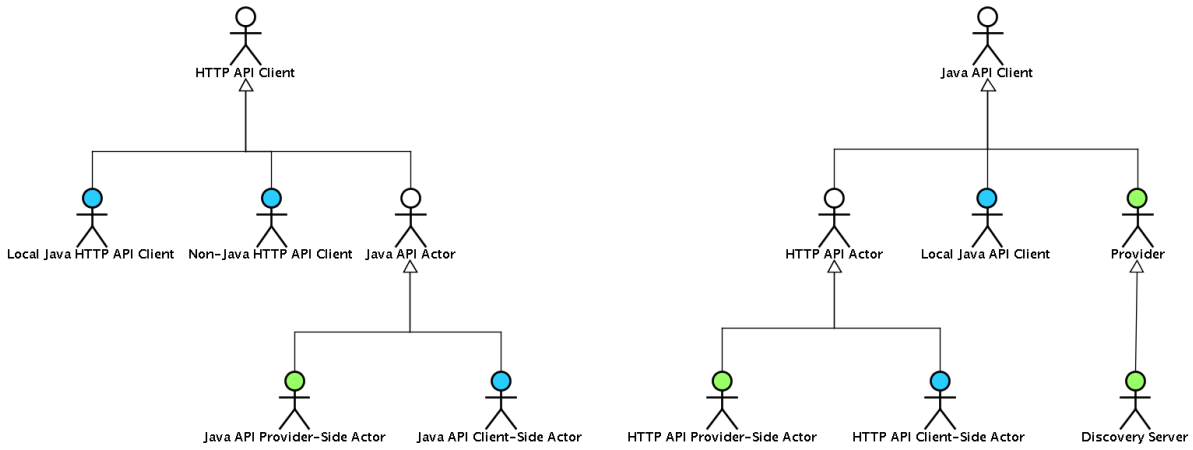

Figure 3: Actor hierarchy in SSWAP. Actors that can exist only on the client side are denoted by the blue color, and actors that can only exist on the provider side are denoted by the green color.

kinds of client actors (denoted as *Local Java HTTP API Client* and *Non-Java HTTP API client* on Figure 3), which can access the same set of use cases, but through different interfaces.

Provider side The main purpose of HTTP API on the provider side is to allow access to the underlying Java API by remote clients. In the current design, these remote clients are Java API Client-side actors (as shown on Figure 2).

2.1.2 Java API client actors

Java API, similarly to HTTP API, exists both on the client side and provider side.

Client side Java API on the client side is accessed either by local Java API Clients (i.e., custom Java code prepared by advanced users) or by HTTP API (*HTTP API Client-side Actor* on Figure 3).

Provider side On the provider side, the job of Java API is to process an external request (passed via HTTP API), contact the actual Provider implementation, allow the Provider to examine the request, reason about it, and then prepare the response. Therefore, the clients of Java API on the provider side are HTTP API Provider-Side Actor, and Provider Actor (including its specialized version of Discovery Server).

2.2 Provider actor

The provider's code (providing an actual service in SSWAP) can be considered an actor when a request comes, and the provider's code uses Java API to examine the contents of RIG and prepares the RRG to be returned in response. Additionally, the Java API must prepare means to couple the provider's code with Java API.

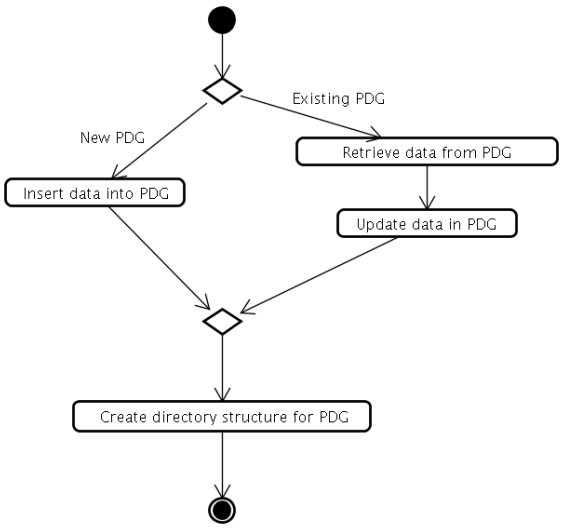

Figure 4: Basic course of events for UC2.1 Create or edit PDG

A special case of a Provider Actor is a Discovery Server, which allows clients to discover services in SSWAP framework.

3 HTTP API Use cases

Place for HTTP API use cases.

4 Java API Use cases

UC2.1 Create or edit PDG (publishing provider information)

Goal: Create a new PDG, or modify an existing one, and then publish it.

Basic flow of events: Figure 4 shows the basic flow of events for this use case.

Based on these events, it is possible to identify the following structure for this use case, as shown on Figure 5.

UC2.2 Create or edit RDG (publishing resource/service)

Goal: Create a new RDG, or modify an existing one, and then publish it.

Basic flow of events: Figure 6 shows the basic flow of events for this use case.

Based on these events, it is possible to identify the following structure for this use case, as shown on Figure 7.

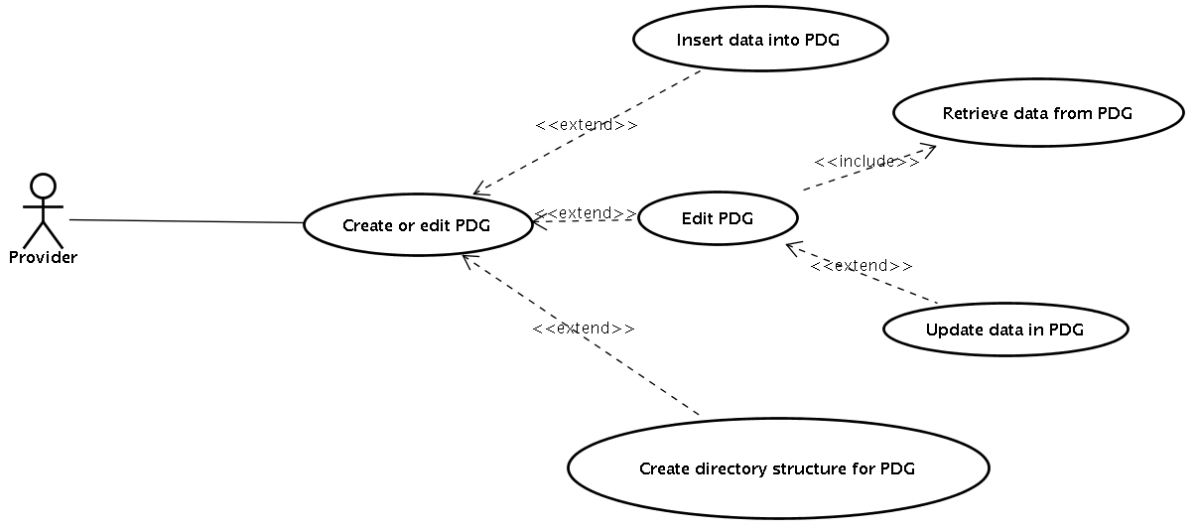

Figure 5: UC2.1 Create or edit PDG (Use Case diagram)

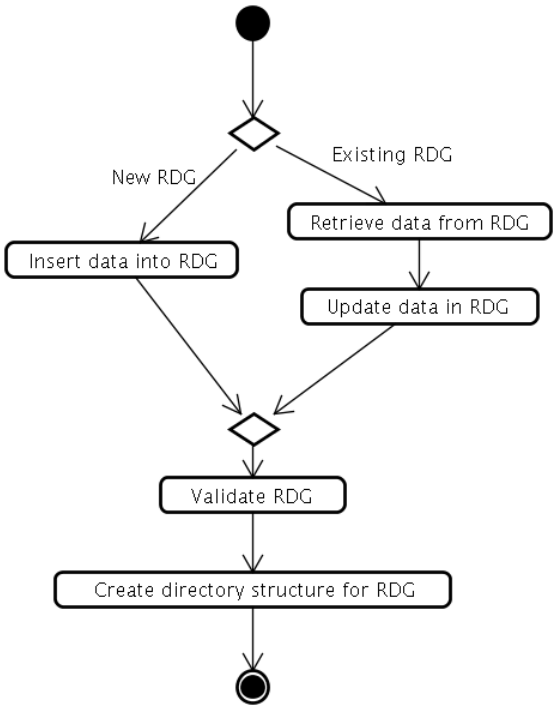

Figure 6: Basic course of events for UC2.2 Create or edit RDG

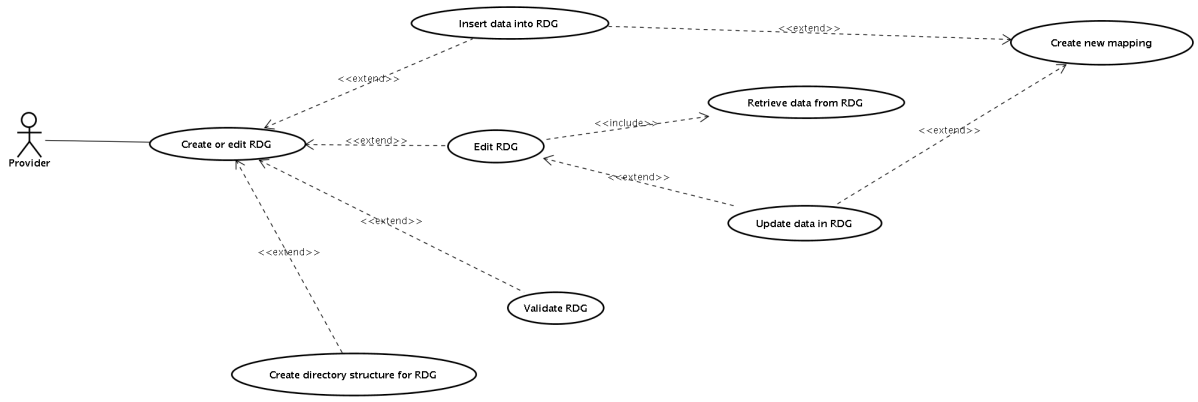

Figure 7: UC2.2 Create or edit RDG (Use Case diagram)

UC2.3 Invoke SSWAP Service

Goal: Invoke a published SSWAP service (i.e., its RDG is published online).

Basic flow of events: Figure 8 shows the basic flow of events for this use case.

This use case shares some activities with the next use case (UC2.4 Accept SSWAP Service invocation): the structure for this use case is presented on Figure 10.

UC2.4 Accept SSWAP Service invocation

Goal: Accept an invocation of a published SSWAP service, extract parameters from the RIG, invoke actual implementation of service, and return results to the client.

Basic flow of events: Figure 9 shows the basic flow of events for this use case.

The structure of this use case is shown on Figure 10.

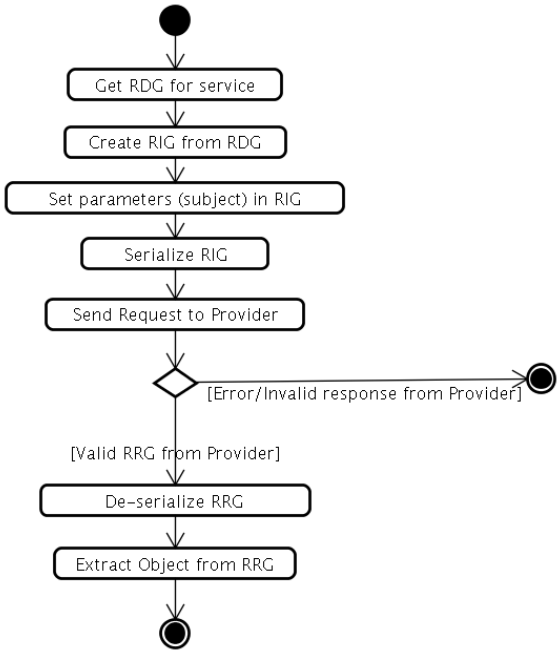

Figure 8: Basic course of events for UC2.3 Invoke SSWAP Service

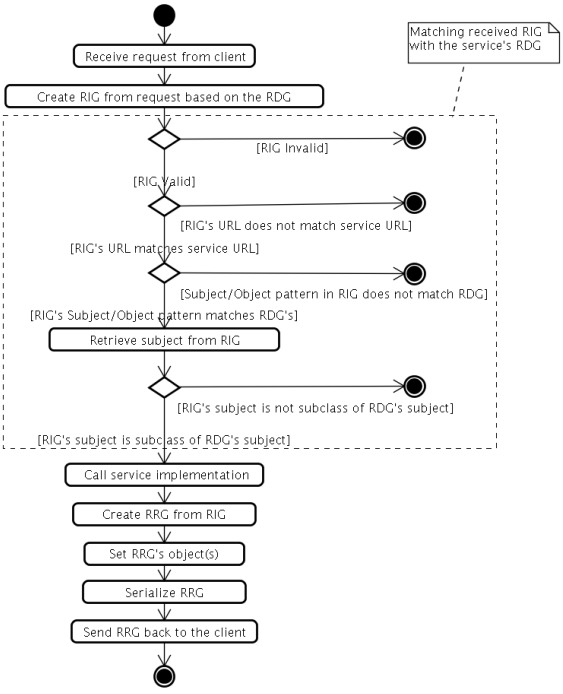

Figure 9: Basic course of events for UC2.4 Accept SSWAP Service invocation

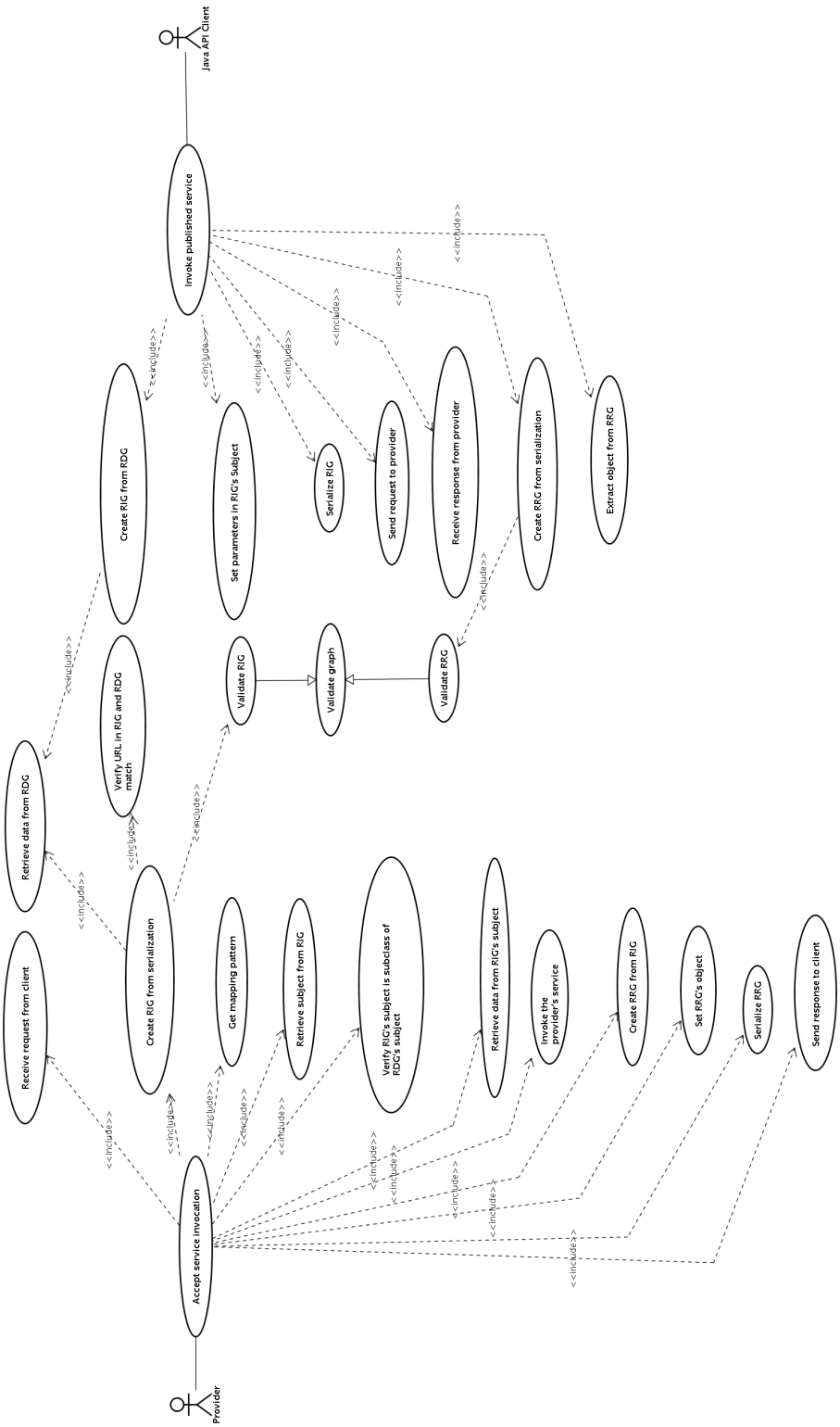

Figure 10: UC2.3 Invoke SSWAP Service and UC2.4 Accept SSWAP Service invocation (Use Case diagram)

5 Use case table

`UCG1` HTTP API Use cases

`UCG2` Java API Use cases

- `UC2.1` Create or edit PDG (publishing provider information)
- `UC2.2` Create or edit RDG (publishing resource/service)
- `UC2.3` Invoke SSWAP service
- `UC2.4` Accept SSWAP service invocation

6 Acronyms

This chapter lists all the acronyms used in this document:

DS Discovery Server

PDG Provider Description Graph

RDG Resource Description Graph

RIG Resource Invokation Graph

RRG Resource Response Graph

RQG Resource Query Graph