PADERBORN UNIVERSITY
The University for the Information Society

Department of Computer Science
Computer Networks Research Group

Architecture Design

S C r A M b L E

Management of ServiCes Across MultipLE clouds

Authors:

Arkajit Dhar
Ashwin Prasad Shivarpatna Venkatesh
Bhargavi Mohan
Deeksha Mysore Ramesh
Harshitha Pandithanahalli Somashekaraiah
Sanket Kumar Gupta
Suheel Shrirangapura Nazeersab
Vivek Jaganath

Supervisors:

Prof. Dr. Holger Karl | Sevil Dräxler | Hadi Razzaghi Kouchaksaraei

Paderborn, January 26, 2019

Contents

1 Architecture Design

1.1 Technology Choices

1.1.1 Python

We choose python as our programming language as the majority of the software written in the MANO community is in python.

1.1.2 Nameko

Nameko[1] is a micro-service framework which makes writing micro-services in python easy. It encapsulates connections, transports and concurrency making it easy for us to start implementing our application logic without the concern of underlying complexity.

1.1.3 RabbitMQ

RabbitMQ is an open source message broker software that facilitated communication between services. This is the default choice for Nameko.

1.1.4 MongoDB

MongoDB is a document-oriented database, we are using this as our persistent storage.

1.1.5 Docker

Docker is a virtualization software, which helps in easy distribution and management of applications. We are using docker to containerize our application and its dependencies.

1.2 Proposed Architecture

This section describes the architecture of SCrAMbLE (figure 1.1). The SCrAMbLE software suite comprises of five independent micro-services (1) Translator, (2) Splitter, (3) Adaptor, (4) Gateway and (5) Main Engine.

[1] `https://github.com/nameko/nameko`

Communication between MANO frameworks (OSM, Sonata) and SCrAMbLE is achieved through plugins which are integrated inside the MANOs, the plugins communicate through REST calls to SCrAMbLE.

The Gateway micro-service is responsible only for receiving a service request and forwarding it to the Main Engine. The Main Engine facilitates interoperability between Translator, Splitter and Adaptor micro-services, it identifies the type of service request and channels the request to the appropriate micro-service, with the help of a message broker, RabbitMQ.

All micro-services use Nameko framework to facilitate easy to use encapsulated communication between micro-services. The micro-services are all individually containerized using docker, SCrAMbLE as a whole is also containerized for easy distribution and scaling. SCrAMbLE uses MongoDB database for persistent storage.

The workflow of the individual service modules is explained in the sections below.

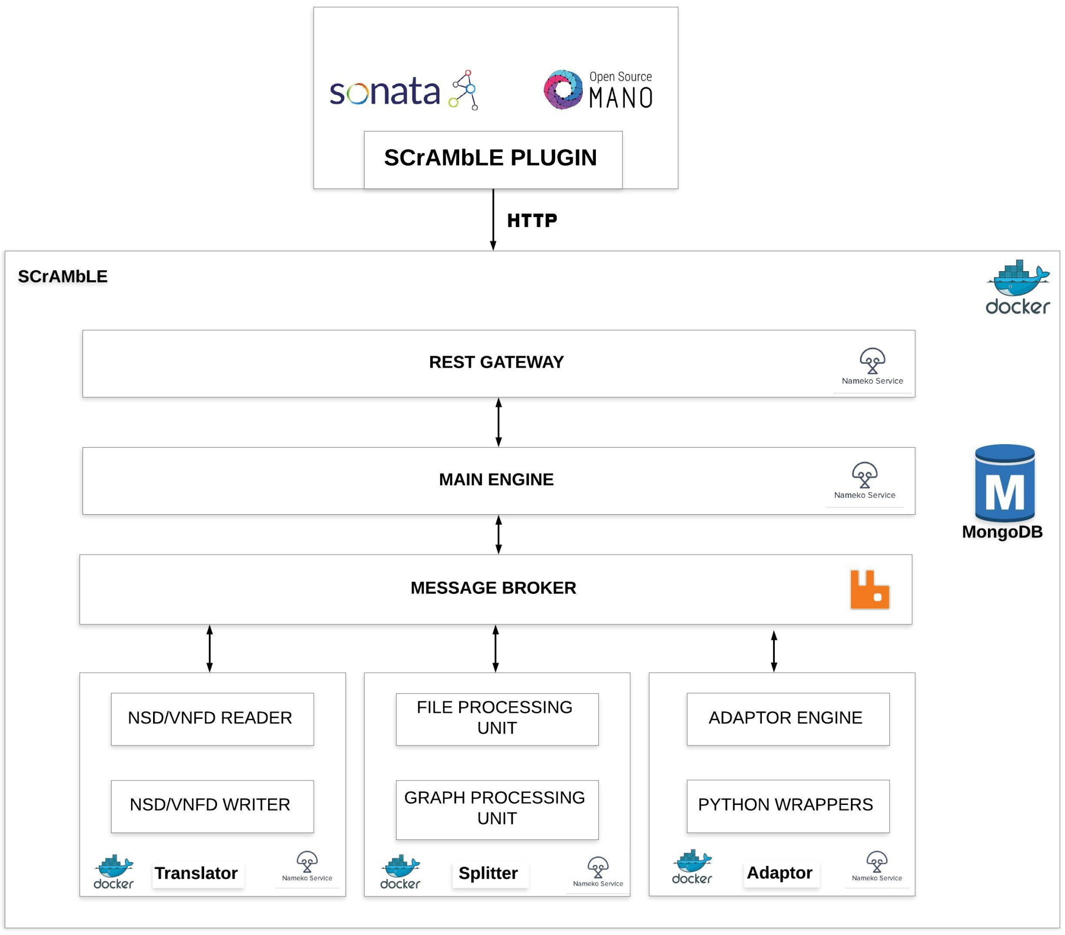

Figure 1.1: Project reference architecture

1.2.1 Adaptor Workflow

When there is a service request from any MANO to the adaptor, the request first flows to the adaptor engine (adaptor.py). Depending on the service request the adaptor engine makes a decision about the appropriate python wrapper to choose to complete the request. The adaptor engine uses the right wrapper (OSM or SONATA wrapper) to make a REST call to the target MANO. The required information is retrieved and is finally delivered to the MANO that requests the service.

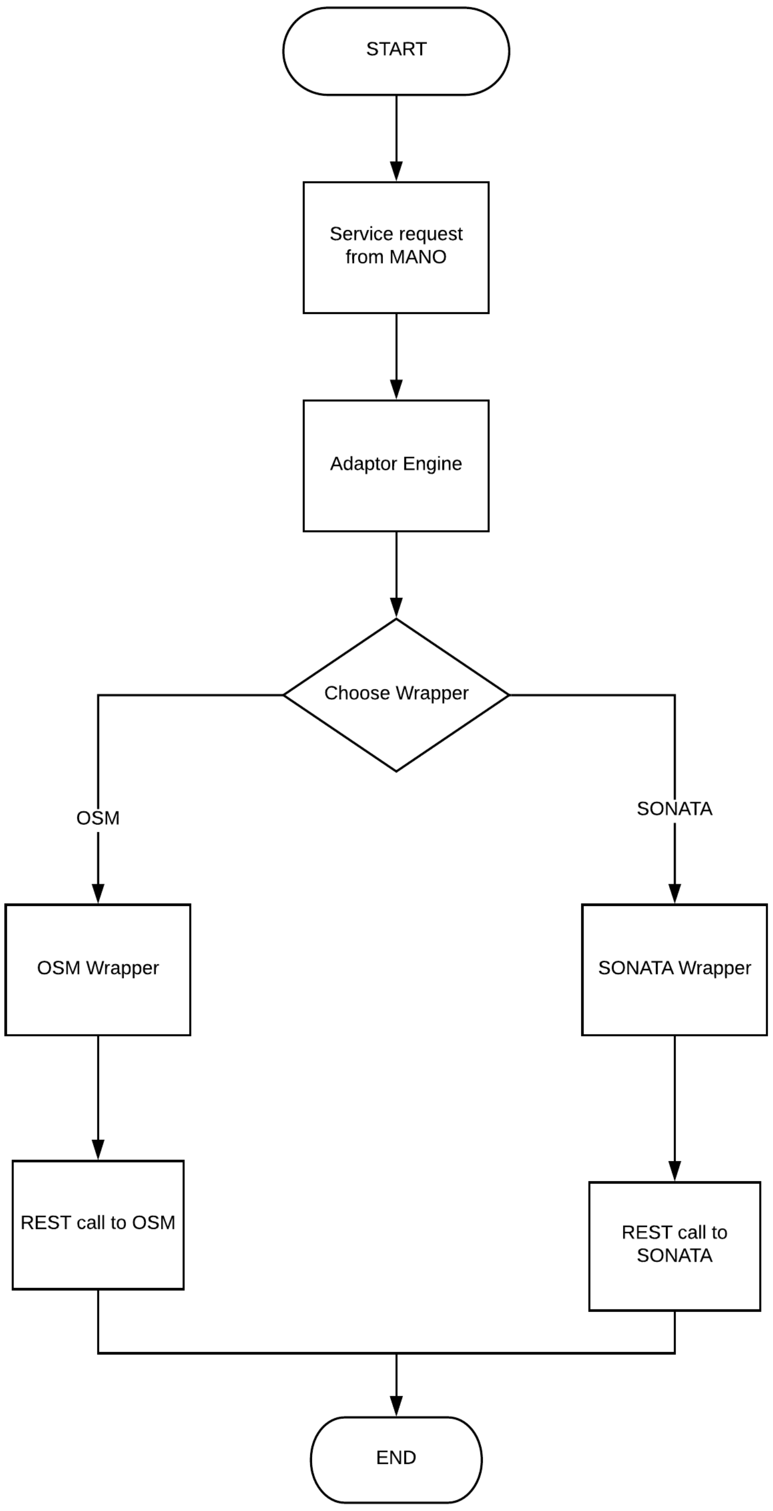

Figure 1.2: MANO Adaptor Workflow

1.2.2 Service Descriptor Splitter Workflow

1.2.3 Service Descriptor Translator Workflow

In a scenario where a network service is to be deployed among different MANO frameworks, having their own respective schema, SDT translates a NSD schema of one MANO framework to that of other MANO framework. On receiving a service request to translate a NSD and/or VNFD to schema of a different MANO, the main engine stores the file(s) into the MongoDB as a document and sends a request in the form of a message to SDT usinfg Nameko services. The message contains the information of DB document name and document id, source type and destination type. SDT comprises of two main services, namely, a Reader and a Writer. On receiving the message from main engine, the Reader fetches the file/document from DB and validates it. After validating the file will be converted to intermediate format, which will be sent to Writer. The Writer converts the file from intermediate format to the destination file format mentioned in the request. On completion of translation, the translated file will be stored into MongoDB and the reference to the same will be passed to main engine with the service completion message. This is an overview of the workflow for Service Descriptor Translator. The details of the message parameters communication between the components are subjected to changes in development phase.

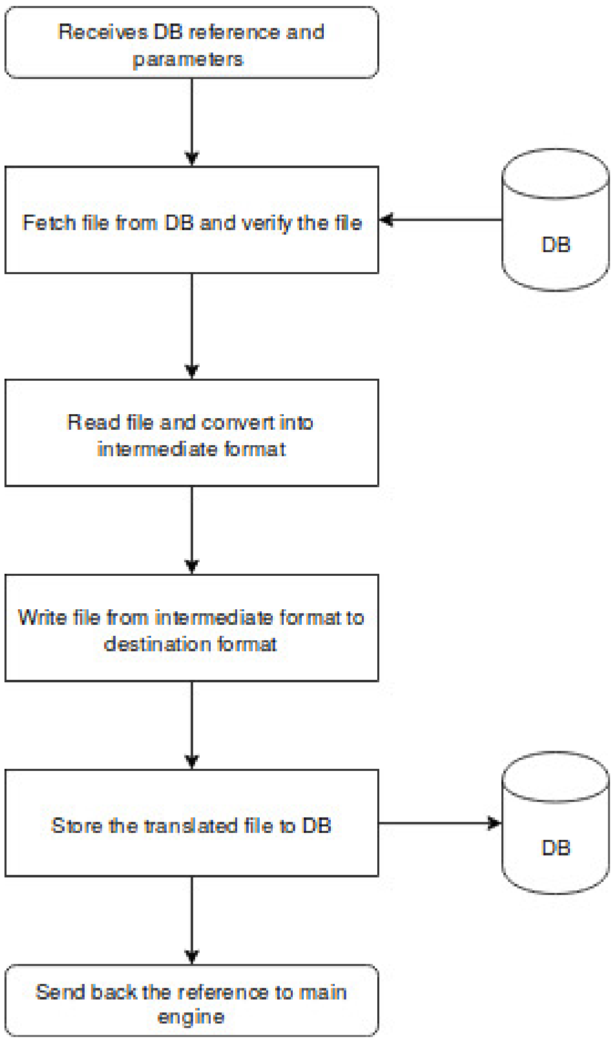
Receives DB reference and parameters
Fetch file from DB and verify the file
DB
Read file and convert into intermediate format
Write file from intermediate format to destination format
Store the translated file to DB
DB
Send back the reference to main engine

Figure 1.3: Service Descriptor Translator Workflow