Engineering Virtual Domain-Specific Service Platforms

Specific Targeted Research Project: FP7-ICT-2009-5 / 257483

Service Platform Infrastructure Repository Concept & Realization

Abstract

The INDENICA project provides infrastructure components and tools to support the effective creation of domain-specific Virtual Service Platforms (VSP). In this report, accompanying the submitted code deliverable, we present the completed prototype and accompanying concepts for a Service Platform Infrastructure repository to support the creation of VSPs. The repository encapsulates design time, deployment time, as well as runtime aspects of VSPs. At design time, the repository allows for the creation of infrastructure component libraries to foster reuse and protect investments in integration actions. Furthermore, it enables the retention of relevant design-time information for the deployment and runtime environment of VSPs in appropriate data stores. At runtime, the repository acts as the central entity responsible for storing and forwarding data to all VSP components using a messaging infrastructure.

Document ID:	INDENICA - D2.3.2
Deliverable Number:	D2.3.2
Work Package:	WP2
Type:	Deliverable
Dissemination Level:	PU
Status:	final
Version:	2.0
Author(s):	*TUV, SUH*

Project Start Date: October 1st 2010, Duration: 36 months

Version History

0.1	02. Nov 2011	Initial version
0.2	06. Dec 2011	Update overview, repository sections
0.3	12. Jan 2012	Prototype implementation notes, usage guide
0.4	14. Jan 2012	Update prototype usage guide
0.5	27. Jan 2012	Incorporate review comments
1.0	27. Jan 2012	Final version for submission (D2.3.1).
1.1	13. Mar 2013	Update overview
1.2	30. Apr 2013	Update description
1.3	24. May 2013	Update usage guide, description
1.4	28. May 2013	Incorporate review comments
2.0	28. May 2013	Final version for submission

Document Properties

The spell checking language for this document is set to UK English.

Table of Contents

Table of Figures

1 Introduction

The INDENICA project aims at providing infrastructure components and tools to support the efficient and effective creation of domain-specific Virtual Service Platforms (VSP). The Service Platform Infrastructure Repository represents a central component to achieve these goals at design time, deployment time, and runtime. At design time, the repository acts as infrastructure component library to foster reuse, decrease development and integration time, as well as protect investments in the INDENICA platform. At deployment time, the repository holds deployment descriptors and environment configuration information enabling easy (re-) deployment of VSPs and their components. Furthermore, the repository allows for the transfer of relevant design and deployment time information to the runtime environment. This allows runtime components, such as monitoring and adaptation facilities, to take additional information about the deployed VSPs and their infrastructure into account, facilitating greater control over the runtime characteristics of VSPs and their environment.

In this prototype deliverable document we will discuss the concept and architecture for the Indenica Service Platform Infrastructure Repository, provide an overview of the prototype implementation, along with a usage guide for the provided tools, and discuss interactions with other tools in the INDENICA ecosystem.

2 Overview

The INDENICA service platform infrastructure consists of a number of components that help to build and run virtual service platforms. At runtime, a messaging infrastructure is used, providing unified communication interfaces for all infrastructure components. As shown in Figure 1, control interfaces provide for a unified mechanism for a bidirectional communication with domain-specific application services and components. The platform provides a publish/subscribe mechanism, enabling complex communication patterns, and allows components to receive information they are interested in. The infrastructure enables efficient handling of large amounts of streaming data (messaging infrastructure), persistent storage of relevant information and its retrieval (repository), as well as aggregation of online and offline data.

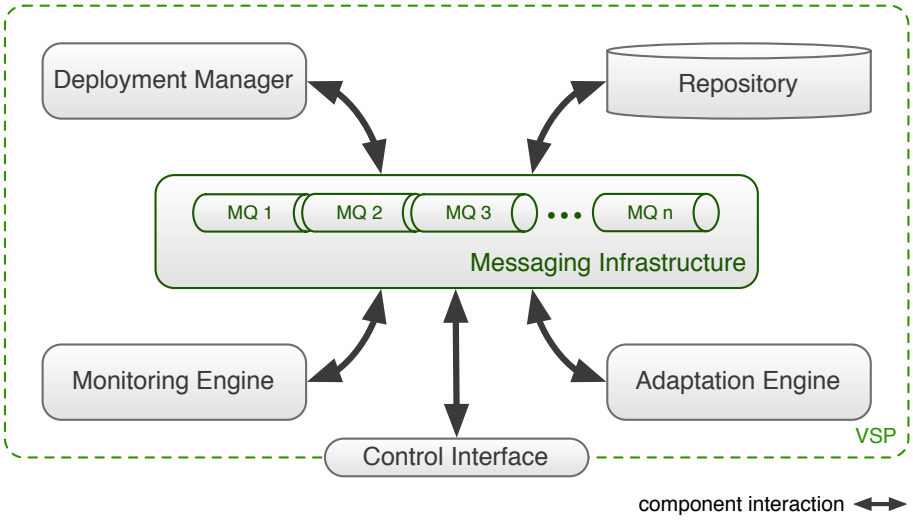

Figure 1: INDENICA Runtime Infrastructure Architecture Overview

The repository contains data about deployment configurations needed by the deployment manager, which is responsible for instantiating and assembling virtual service platforms. The monitoring engine combines online data about the domain-specific runtime components with historical data to draw conclusions about the system's status. Based on this information, the adaptation engine reasons about and triggers reconfigurations. The rules and policies determining the monitoring and adaptation engines are also stored in the repository.

In addition to the runtime and deployment time concerns mentioned above, the repository represents a central part of the design time tool suite. Artefacts from design time tools, such as the requirements engineering tool IRENE (cf. D1.2.2), the architecture decision modelling and support framework ADvISE (cf. D1.3.2), the variability management tool EASy-Producer (cf. D2.4.2), and the view-based modelling framework VbMF (cf. D3.3.2), are stored in the repository. The INDENICA tools use the repository as a storage and integration backend, allowing for unified provenance of relevant platform assets, as well as easy sharing of artefact data between tools. Furthermore, the repository contains platform infrastructure template definitions used to provision physical infrastructure for running VSPs. Using

the repository along with a lifecycle evolution model allows for the realisation of adaptation solutions spanning the complete application development lifecycle, from design and development to runtime [1].

The repository currently supports the following artefacts, and provides specialized storage and retrieval methods tailored to the specifics of each artefact type:

- IRET requirements models (cf. D1.2.2)
- Architecture decision models (cf. D1.3.2)
- Variability engineering information (cf. D2.4.2)
- View-based modelling framework models (cf. D3.3.2)
- Deployment descriptors (cf. D4.2.2)
- Monitoring and adaptation rules (cf. D4.2.2)
- Platform infrastructure configuration (e.g. Chef[1] cookbooks)

[1] http://opscode.com/chef

3 Repository

The repository is the entity in INDENICA responsible for retention and retrieval of persistent data. It manages data that needs to be persisted or be available for retrieval using query mechanisms. The key design goals for the repository are availability (provide a scalable architecture that is able to cope with high volumes of data and frequent requests), data store abstraction (technological flexibility with respect to the technology used for storing data in the backend), and domain-specific data retrieval modes (different query methods tailored to the specifics of various types of information managed by the repository).

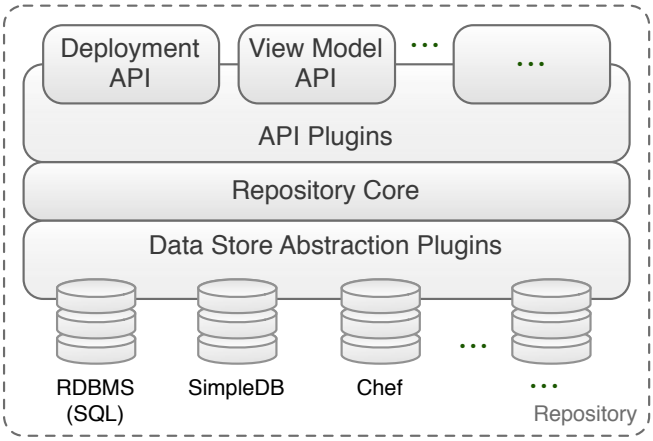

Figure 2: Repository Architecture Overview

An overview of the repository architecture is shown in Figure 2. The repository core provides basic infrastructure such as default service interfaces, as well as initializing the API and data store abstraction facilities. This enables the repository to be data store agnostic, accommodating for different requirements of service platform installations. The API plugin facility allows for the retrieval of stored data via custom interfaces for different types of information (deployment descriptors, monitoring rules, log data, etc.). The repository's capabilities range from storing simple key/value entries for global configuration values to providing structured and interconnected data formats, such as platform-specific variability models.

Such tailored interfaces in the form of domain-specific query languages, exposed through custom APIs, or GUIs enable efficient interaction with the repository. The data store abstraction allows for the transparent use of multiple storage back ends, ideally suited for the stored data and the particular application environment. The according plugin architectures are discussed below.

3.1 Data Store Abstraction Plugin Architecture

In the following, we discuss the repository data storage and retrieval plugin architecture used to enable customized persistence and query mechanisms tailored to each supported artefact type and usage scenario.

Due to the diverse nature of data artefacts produced during the creation of VSPs, the repository needs to be able to easily accommodate various storage back ends. This is

achieved by the data store abstraction plugin mechanism, specifying a thin interface layer to be implemented by platform integrators. As shown in Figure 3, plugins announce storage and retrieval capabilities by implementing the according `canHandle` methods. The repository core infrastructure provides a generic artefact container, `Data`, to wrap arbitrary XML content for storage. Furthermore, a generic data retrieval specification, `Filter`, is provided to query data according to the capabilities of the implemented storage back end.

```
public interface IDataStorePlugin {
   boolean canHandle(Filter f);
   boolean canHandle(Data d);

   Data getData(Filter f);
   Data storeData(Data d);
}
```

Figure 3: Basic Plugin Interface

The repository does not mandate any additional specific semantics from storage back ends, so that plugins are free to define custom query and storage semantics according to the capabilities of the employed storage technology. By convention, all plugins support queries by name and artefact identifier.

To further illustrate the merits of this approach, we discuss a concrete storage plugin in more detail. The Chef plugin accepts `infrastructure` data structures representing the physical infrastructure available for a given VSP instance, as well as required software packages. The schema of the used document type is provided with the plugin documentation and is used by consumers either directly through the generic repository API or through custom API providers, such as the Integration API provider discussed below. The plugin transforms received data into appropriate constructs to store in the underlying Chef instance. Furthermore, the plugin supports a number of filter constructs to query the stored infrastructure data by, e.g., hardware properties (RAM, CPU, disk), installed software, or available network interfaces. In addition to storing infrastructure data in an appropriate storage back end, this plugin enables automatic provisioning of physical infrastructure through the associated Chef configuration management (CM) system according to data managed by INDENICA tools. The infrastructure storage plugin is based on Chef for tighter integration with the deployment manager presented in D4.2.2, but the repository can easily support different CM systems, such as Puppet[2] or cfengine[3], by providing appropriate plugins.

The presented storage abstraction allows for easy integration of custom storage back ends tailored to the stored data. Furthermore, switching storage back ends is intrinsically supported, avoiding vendor lock-in effects.

[2] http://puppetlabs.com/

[3] http://cfengine.com/

3.2 API Plugin Architecture

In the following, we discuss the repository API plugin architecture enabling tailored interfaces using domain-specific query languages.

In addition to the generic query API, exposing storage and filter capabilities as provided by data store abstraction plugins, the repository supports domain-specific API plugins to gather data from multiple back ends in an application-specific way.

API plugins can be implemented as API service endpoints or repository client libraries. Service endpoints extend the running repository instance, offering domain-specific API to all repository clients. Repository client libraries are deployed with tools consuming repository data and are especially suitable if a custom API will only benefit a single or very few client applications. Plugins implemented as client libraries can furthermore be used to realise repository proxies, allowing for clear separation of core repository functionality and domain-specific add-ons.

4 Prototype Implementation

In this section, we discuss the prototype implementation of the Service Platform Infrastructure Repository, along with a guide on how to use the repository within the INDENICA tool ecosystem.

The prototype infrastructure is realized using the Jetty[4] servlet engine, providing the core repository service interface. The previously discussed plugin infrastructure allows for the addition of arbitrary storage backend plugins. The currently provided plugins allow storage of INDENICA assets using Chef (a configuration management system, as mentioned above), MongoDB[5] (a document-oriented database), as well as conventional file systems. These technologies have been chosen based on a careful evaluation of requirements of the INDENICA tool suite and the artefacts they produce, as well as the current state of the art in database and data processing technology. In the following we provide documentation for the core repository API, as well as a usage guide for the provided example scenario. Additional demonstrations of repository usage throughout INDENICA will be provided with the deliverables describing the according tools, such as the Architecture Decision Framework (D1.3.2), the Variability Management Tool (D2.4.2), the VSP engineering tool suite (D3.3.2), as well as the runtime infrastructure tools (D4.2.2).

4.1 API

As mentioned above, the core repository API serves as a generic layer exposing the underlying storage plugins. Hence, it provides a very simple interface to pass data storage, retrieval, and subscription requests to the underlying storage infrastructure. An excerpt of the WSDL description of the generic API is shown in Figure 4 below.

```
<definitions
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://impl.services.runtime.indenica.eu/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://impl.services.runtime.indenica.eu/"
  name="RepositoryImplService">
  <binding xmlns:ns1="http://services.runtime.indenica.eu/"
    name="RepositoryImplPortBinding" type="ns1:IRepository">
    <soap:binding
      transport="http://schemas.xmlsoap.org/soap/http"
      style="document"/>
      <operation name="getData">
        <input><soap:body use="literal"/></input>
        <output><soap:body use="literal"/></output>
      </operation>
      <operation name="subscribeToData">
        <input><soap:body use="literal"/></input>
        <output><soap:body use="literal"/></output>
      </operation>
      <operation name="publishData">
        <input><soap:body use="literal"/></input>
        <output><soap:body use="literal"/></output>
      </operation>
    </binding>
```

[4] http://www.eclipse.org/jetty/

[5] http://www.mongodb.org/

```
  <service name="RepositoryImplService">
    <port name="RepositoryImplPort"
      binding="tns:RepositoryImplPortBinding">
      <soap:address location="http://0.0.0.0:45689/repo"/>
    </port>
  </service>
</definitions>
```

Figure 4: Excerpt of Generic API WSDL

The generic API does not make any assumptions on the data structure, but allows storage plugins to define their own processing semantics using the generic `Filter` and `Data` containers mentioned in Section 3.1.

In the following, we briefly illustrate the MongoDB plugin filter semantics. Further documentation can be found in the according Javadoc. The MongoDB plugin allows storing arbitrary XML documents. Artefacts to be stored are converted to JSON prior to storage, and the plugin exposes the complete MongoDB query API[6] for retrieving stored documents. An exemplary query retrieving hosts with more than 1024MB RAM is shown in Figure 5.

```
<query>{ node: { ram: { $gt: 1024 } }</query>
```

Figure 5: Exemplary MongoDB filter query

As mentioned above, the repository prototype currently ships with additional storage plugins for Chef, as well as file system storage. These plugins expose storage and retrieval semantics similar to the MongoDB plugin, offering the underlying store's native persistence and querying capabilities. API plugins, such as the provided `IvmlRepository` API plugin are used to provide an application-specific interface for consumers.

4.2 Usage Guide

In this section we provide a step-by-step guide for executing the provided simple example application, illustrating the interaction of a simplified client extracted from the variability management tool set with the Service Platform Infrastructure Repository.

In order to successfully execute the provided samples, a MongoDB instance must be available on port 27017.

To start a repository instance with the Ivml API plugin, run

```
java –jar \
  IvmlRepository-1-SNAPSHOT-with-dependencies.jar
```

The repository will perform some basic sanity checks and listen on port 45689 by default.

The exemplary client application can be launched using

```
java –jar IvmlRepositoryClient-1-SNAPSHOT-with-
dependencies.jar
```

The client will retrieve a resolved variability model from the repository, check its consistency and report its properties.

[6] http://docs.mongodb.org/manual/core/read-operations/

Note that this prototype deliverable contains only the core repository infrastructure and a simple example. The repository is extensively used throughout the project, and its usage is documented in the according deliverables (as mentioned above).

5 Conclusion

The Service Platform Infrastructure Repository takes a key role in the Virtual Service Platform, providing global access to various types of information with diverse requirements concerning storage and retrieval. In this document we describe the interim version of the INDENICA Service Platform Infrastructure Repository prototype, including documentation on its usage. The key design goals of the repository are scalability, data store abstraction and domain-specific data retrieval possibilities. We present the repository architecture, its main components, along with a set of tools developed to support platform infrastructure configuration using the Service Platform Infrastructure Repository. The repository implementation builds on state-of-the-art technologies that have been carefully evaluated and chosen to fulfil their particular purpose. It provides a solid basis for extensibility and is integrated with other tools in the INDENICA ecosystem that rely on the storage and retrieval capabilities of the repository.

References

[1] C. Inzinger, W. Hummer, I. Lytra, P. Leitner, H. Tran, U. Zdun, and S. Dustdar, “Decisions, Models, and Monitoring – A Lifecycle Model for the Evolution of Service-Based Systems,” presented at the 17th IEEE International Enterprise Distributed Object Computing Conference, 2013, p. to appear.