

NRG4CAST
FP7-2012-NMP-ENV-ENERGY-ICT-EeB
Contract no.: 600074
www.nrg4cast.org

NRG4CAST

Deliverable D1.3

Early Toolkit architecture specifications

Editor:	Ioannis Chamodrakas, SingularLogic
Author(s):	Ioannis Chamodrakas, Kostas Kalaboukas, SingularLogic; Jasna Skrbec, Klemen Kenda, JSI
Reviewers	Gregor Leban (JSI), Angela Pluchinotta (CSI), Piergiorgio Chiriotti (CSI), Marco Boz (CSI)
Deliverable Nature:	Report (R)
Dissemination Level: (Confidentiality) ¹	Public (PU)
Contractual Delivery Date:	April 2013
Actual Delivery Date:	April 2013
Suggested Readers:	Energy monitoring/forecasting software developers
Version:	1.0
Keywords:	Architecture specification, Service Oriented Architecture, Web Services, N-Tier

¹ Please indicate the dissemination level using one of the following codes:

• **PU** = Public • **PP** = Restricted to other programme participants (including the Commission Services) • **RE** = Restricted to a group specified by the consortium (including the Commission Services) • **CO** = Confidential, only for members of the consortium (including the Commission Services) • **Restreint UE** = Classified with the classification level "Restreint UE" according to Commission Decision 2001/844 and amendments • **Confidentiel UE** = Classified with the mention of the classification level "Confidentiel UE" according to Commission Decision 2001/844 and amendments • **Secret UE** = Classified with the mention of the classification level "Secret UE" according to Commission Decision 2001/844 and amendments

	Architecture
--	--------------

Disclaimer

This document contains material, which is the copyright of certain NRG4CAST consortium parties, and may not be reproduced or copied without permission.

All NRG4CAST consortium parties have agreed to full publication of this document.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the NRG4CAST consortium as a whole, nor a certain party of the NRG4CAST consortium warrant that the information contained in this document is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using this information.

Copyright notice

© 2012-2015 Participants in project NRG4CAST

Executive Summary

The final goal of the Energy Forecasting (NRG4CAST) project is to provide a software toolkit that will enable the provision of a set of generic real-time monitoring and prediction services for energy planning, network failure and energy price prediction as services with specific APIs to external systems and applications of energy consumers and distributors. The monitoring and prediction services access different types of information residing in heterogeneous and distributed information systems and databases. The purpose of this deliverable is to specify and design initial version of software architecture of the NRG4CAST toolkit.

The NRG4CAST early toolkit architecture follows a multi-tiered Web Service architecture approach which includes the following tiers:

Resource Tier: Consists of existing data resources such as relational databases, CSV files and Geographical Information Systems.

Data Access Tier: Consists of low-level data services which provide homogeneous and uniform access to a set of heterogeneous data resources on the Resource Tier. For this purpose the OGSA-DAI (Open Grid Software Architecture – Data Access and Integration) middleware is adopted which supports data access, data transformation, data integration and data delivery through a set of standard and well-documented Web Services.

Integration Tier: Consists of **Data Consolidation Services**, i.e. a set of application-specific data services over the data resources exposed by the Data Access Tier and **Ontology-based Integration Services** which provide semantic access to the data resources through the utilization and mapping of the NRG4CAST ontology.

Application Tier: Consists of alerting and prediction services. The alerting service triggers alerts and warnings based on a set of rules. The prediction service supports different prediction scenarios, such as the prediction of an upcoming measurement according to historical data, the prediction of events (e.g. sensor failure), etc.

The deliverable specifies the interfaces, i.e. the operations (inputs, outputs and faults) for the services at the Data Access, Integration and Application Tiers.

Data Access Tier Services: (1) the *Data Request Execution Service* which submits data-related workflows to the Data Access and Integration (OGSA-DAI) middleware which federates the data resources, (2) the *Data Source Service* which provides clients with access to streamable data resources, (3) the *Data Sink Service* which provides to clients the functionality to push data to the middleware server, (4) the *Data Resource Information Service* which is consumed by clients to query metadata of a data resource and (5) the *Request Management Service* which enables clients to query the execution status of their requests and to retrieve data from asynchronous requests.

Integration Tier Services:

Data Consolidation Services: (1) the *Query Relational Resource Service* which permits clients to submit queries to relational data resources, (2) the *Query File Resource Service* which permits clients to submit queries to flat file data resources, (3) the *Server Information Service* which obtains metadata of the middleware server and (4) the *Subscription Notification Service* which enables a client to subscribe to specific notifications when an event happens.

Ontology Based Integration Services: the *Integration Service* which provides clients with the ability to submit queries on the data resources over their ontological mapping.

Application Tier Services: (1) the Event Processing (Alerting) Service which triggers alerts and warnings based on different rules and (2) the Prediction Service which predicts upcoming measurements of the sensors, other measurable phenomena such as weather forecasting and events.

The implementation of the Web Services provided by the toolkit is the focus of Work Packages 2 to 6. Updated versions of the toolkit architecture will be provided by taking into account the feedback from the further elaborated user requirements of the pilot cases, especially with respect to the revision of the

provided Web Service operations, in deliverables D1.4 and D6.1. Those deliverables will also provide the concrete details of the Subscription & Notification Service.

Table of Contents

Executive Summary	4
Table of Contents	6
List of Figures.....	7
Abbreviations.....	8
1 Introduction	9
2 Background.....	10
2.1 N-Tier Architectures.....	10
2.1.1 Three-tier Architecture.....	11
2.2 Web Service Architectures.....	12
2.2.1 Service Oriented Architecture (SOA) and the WS-* stack	12
2.2.2 Resource Oriented Architecture (ROA) – RESTful Web Services	12
2.2.3 Comparison between SOA and ROA.....	12
3 Overview of the NRG4CAST Toolkit Architecture	13
3.1 Rationale	13
3.2 Resource Tier	14
3.3 Data Access Tier.....	15
3.3.1 Criteria for the adoption of the OGSA-DAI middleware.....	15
3.3.2 OGSA-DAI middleware Description	15
3.4 Integration Tier	17
3.5 Application Tier	17
3.6 Presentation Tier.....	17
3.7 Related Architectures	17
3.7.1 SemSorGrid4Env	18
3.7.2 ENVISION	18
3.7.3 Comparison with the NRG4CAST toolkit architecture.....	20
3.8 Summary.....	20
4 Services.....	22
4.1 Data Access Tier Services	22
4.1.1 Data Request Execution Service	22
4.1.2 Data Source Service	22
4.1.3 Data sink service	23
4.1.4 Data Resource Information Service	24
4.1.5 Request Management Service	24
4.2 Integration Tier Services	25
4.2.1 Data Consolidation Services	25
4.2.2 Ontology-based Integration Services.....	28
4.3 Application Tier Services	29
4.3.2 Prediction Service	31
5 Conclusions	33
References.....	34

List of Figures

Figure 1: Example of N-tier architecture.	10
Figure 2: Three-tier architecture. (Source: Wikipedia).....	11
Figure 3. The NRG4CAST toolkit architecture.....	13
Figure 4. OGSA-DAI component diagram (adapted from [4])	16
Figure 5. The SemSorGrid4Env architecture: the services and their relationships [11]	18
Figure 7: Interaction of components among back-end services and portlets.....	20
Figure 8. Query Relational Resource operation: executeQuery (inputs, outputs and faults).....	26
Figure 9. Query Relational Resource operation: executeQueryWithTimeInterval (inputs, outputs and faults)	26
Figure 10. Query File Resource operation: executeQuery (inputs, outputs and faults)	27
Figure 11. Query File Resource operation: executeQueryWithTimeInterval (inputs, outputs and faults)	27
Figure 12. Server Information Service operation: getResourceIds (inputs, outputs and faults).....	28
Figure 13. Integration Service operation: integrateAs (inputs, outputs and faults)	29
Figure 14. Integration Service operation: executeSPARQLQuery (inputs, outputs and faults)	29
Figure 15. Event Processing Engine within the EnStreamM component.	30

Abbreviations

AMGA	ARDA Metadata Catalogue Project
ARDA	A Realization of Distributed Analysis of Large Hadron Collider
CSV	Comma Separated Values
DBF	DBASE File
DQP	Distributed Query Processing
EPE	Event Processing Engine
ESB	Enterprise Service Bus
GPL	GNU Public License
GUI	Graphical User Interface
OGC O&M	Open Geospatial Consortium Observation and Measurement Standard
OGSA-DAI	Open Grid Software Architecture – Data Access and Integration
QoS	Quality of Service
R2O	Relational to Ontology mapping language
R2RML	RDB to RDF Mapping Language
REST	REpresentational State Transfer
ROA	Resource Oriented Architecture
SOA	Service Oriented Architecture
SOS	Sensor Observation Service
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
SPE	Stream Processing Engine
TSV	Tab Separated Values
UDDI	Universal Description, Discovery and Integration
UI	User Interface
URI	Uniform Resource Identifier
WS-BPEL	Web Services Business Process Execution Language
WS-CDL	Web Services Choreography Description Language
WSDL	Web Services Description Language

1 Introduction

This deliverable provides the functional and technical specification of the early toolkit architecture. The NRG4CAST project aims at developing a set of generic real-time monitoring and prediction services for energy planning, network failure and energy price prediction that will cover the needs of both energy consumers and energy distributors. In this context, the main objective of the NRG4CAST toolkit architecture is to integrate a set of real-time energy monitoring and prediction software components, so that it will enable their provision as services with specific APIs to external systems and applications. A crucial requirement for the specification of the toolkit architecture is the ability to combine different types of information provided by different information systems and databases. Since reasoning, prediction and decision-making methods and services require the mapping of low-level data to a common ontological schema, the toolkit architecture specification should enable the semantic integration of the heterogeneous data resources.

According to the above, and taking into account the requirements of the pilot cases which are presented in Deliverable 1.2, the NRG4CAST toolkit architecture follows a multi-tiered Web Service Architecture approach. A multi-tiered architectural approach is based on the principle of the separation of the concerns of the system into stacked groups. In the case of NRG4CAST there is a distinct data **Resource Tier** which resides on the external information systems or on the NRG4CAST toolkit server file system. The **Data Access Tier** is responsible for access, management, federation, integration and delivery of the heterogeneous shared data resources. The **Integration Tier** is responsible for the translation of queries over the common ontological schema or alternatively for the exposition of a set of application-specific data services over the data resources exposed by the Data Access Tier hiding low-level details and complexity. The **Application Tier** provides monitoring and forecasting functionality as a service to both the NRG4CAST toolkit **Presentation Tier** as well as to external systems.

With the exception of the Resource and the Presentation tiers each tier of the architecture provides its functionality through the provision of Web Services. The Web Service approach decouples the tiers and therefore promotes extensibility, reusability and adaptability of the toolkit architecture. In particular, the Data Access Tier exposes virtualized and federated data resources via RESTful services hiding database heterogeneities. The Integration tier provides higher-level application specific data access services such as subscription / notification, semantic queries, etc. which consume the lower-level Data Access services. Finally, the Application Tier provides alerting and prediction Web services which take as an input the output of the services of the lower tiers. The provision of alerting and prediction functionality as Web Services is equivalent with the definition of well-defined APIs to both external systems and the UI components of the Presentation tier.

Apart from the high-level view of the architecture, this deliverable describes in detail the services provided in each relevant tier through the presentation of the operations they provide, their inputs, outputs and faults.

The remainder of this deliverable is structured as follows: Section 2 summarizes the literature with regard to N-Tier and Service-oriented software architectures. Section 3 presents a high-level view of the NRG4CAST toolkit architecture in detail. The provided RESTful and WS-Services are described in Section 4. Finally, Section 5 presents the main conclusions and results of this deliverable.

2 Background

2.1 N-Tier Architectures

The N-tier architecture, also referred as the multi-tier architecture, organizes different software components in tiers or layers. Tiers are divided based on functionality, from lower tiers around raw data to higher tiers around business logic. Historically, first applications had one-tier architecture and from one-tier architecture a development of client/server architecture arose, which divided the architecture into two layers; a client layer ensures for the user's applications and a server layer takes care of data [1]. The evolution of this two-tier architecture went into two different directions: fat client/thin server architecture, where the client also includes business logic and thin client/fat server, where the business logic is included in the server side. Next logical step is the evolution of business layer as its own layer; this is so called the three-tier architecture (client tier, business tier, data management tier). In architectures with more than three tiers, additional tiers are added considering the needs of an application, its structure and purpose (e.g. data access tier – between database and business tier, web tier – on top of presentation tier, together they are instead of the client tier).

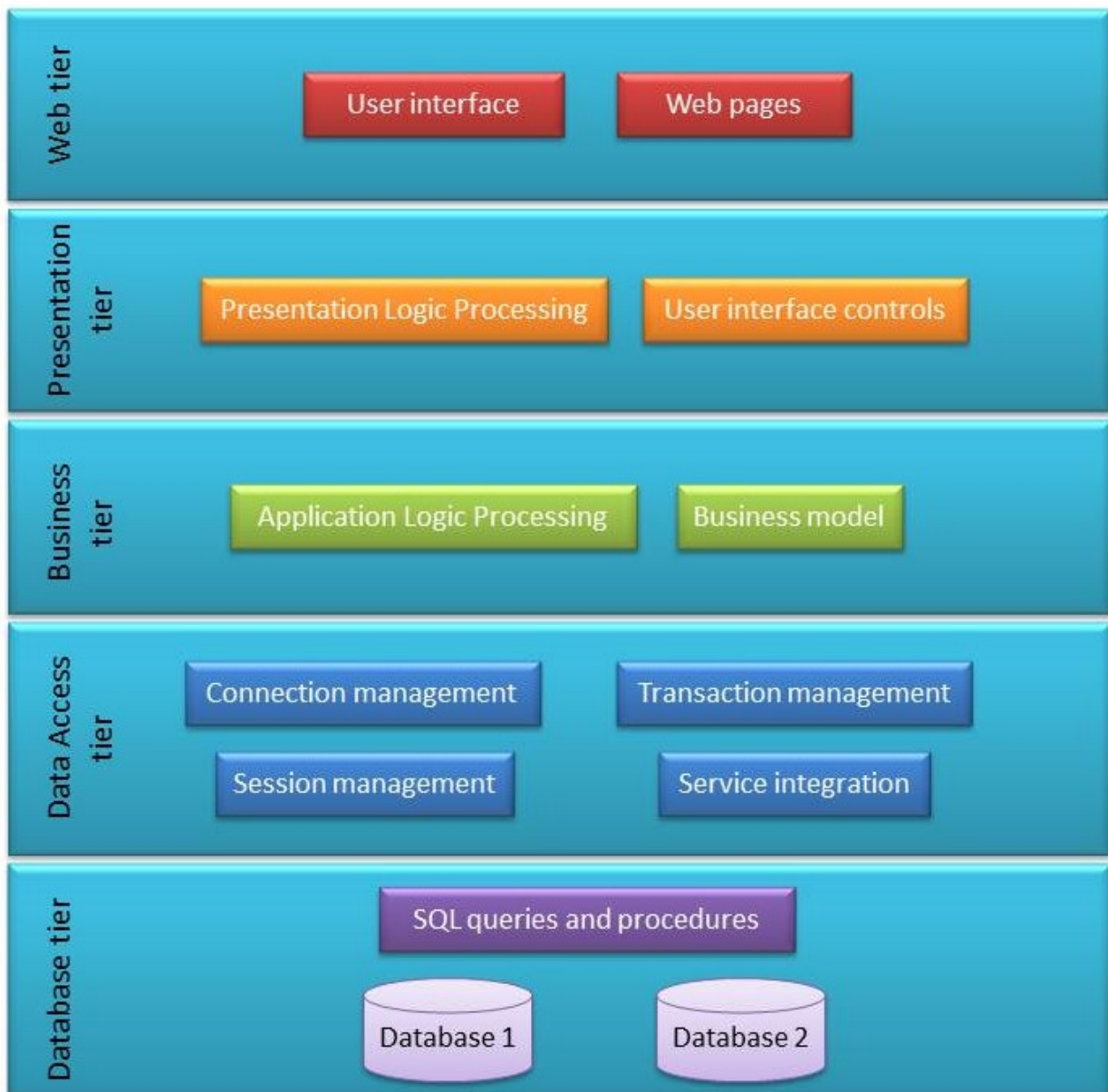


Figure 1: Example of N-tier architecture.

The number of tiers in the N-tier architecture is defined by a vertical decomposition of functionality and it should be appropriate to enable developers to create a flexible and reusable application. Tiers by itself should be logically implied to the application. Consecutive tiers should follow the client/server model, each lower tier acts as server to the next, higher tier. Each tier can be distributed horizontally, if needed. Connections and data transfer between tiers are part of the architecture and must be intently planned. Tiers communicate only with its neighbour tiers, lower and higher, and not with much higher or much lower tiers.

Planning software in line with N-tier architecture has more benefits; besides already mentioned flexibility for developers, it also means that different platforms can be included on different layers. This also makes a lot easier to correct or improve parts of software and simplify adding new parts to the existing application, which results in more powerful applications and more useful services for many clients.

2.1.1 Three-tier Architecture

The three-tier architecture model consists of the presentation tier, the logic tier and the data tier (see Figure 2). The presentation tier is for presentation the data to the client and for manipulation and entry of the data from client’s side; therefore it handles all the interaction with a client or user. The logic tier provides all the information for the client tier, so it plays a role of a server to this tier. Moreover it takes all the data it needs from the data tier, so it plays a role of a client to the data tier. This tier is also referred as the application tier or simply the middle tier, and it controls the application’s functionality. The data tier keeps information, which is stored in database servers. Keeping data away and independent from logic improves scalability and performance. The three tiers present a logical grouping of components which may or may not be the same in actual physical form. Three tiers are the most widespread used type of N-tier architecture model.

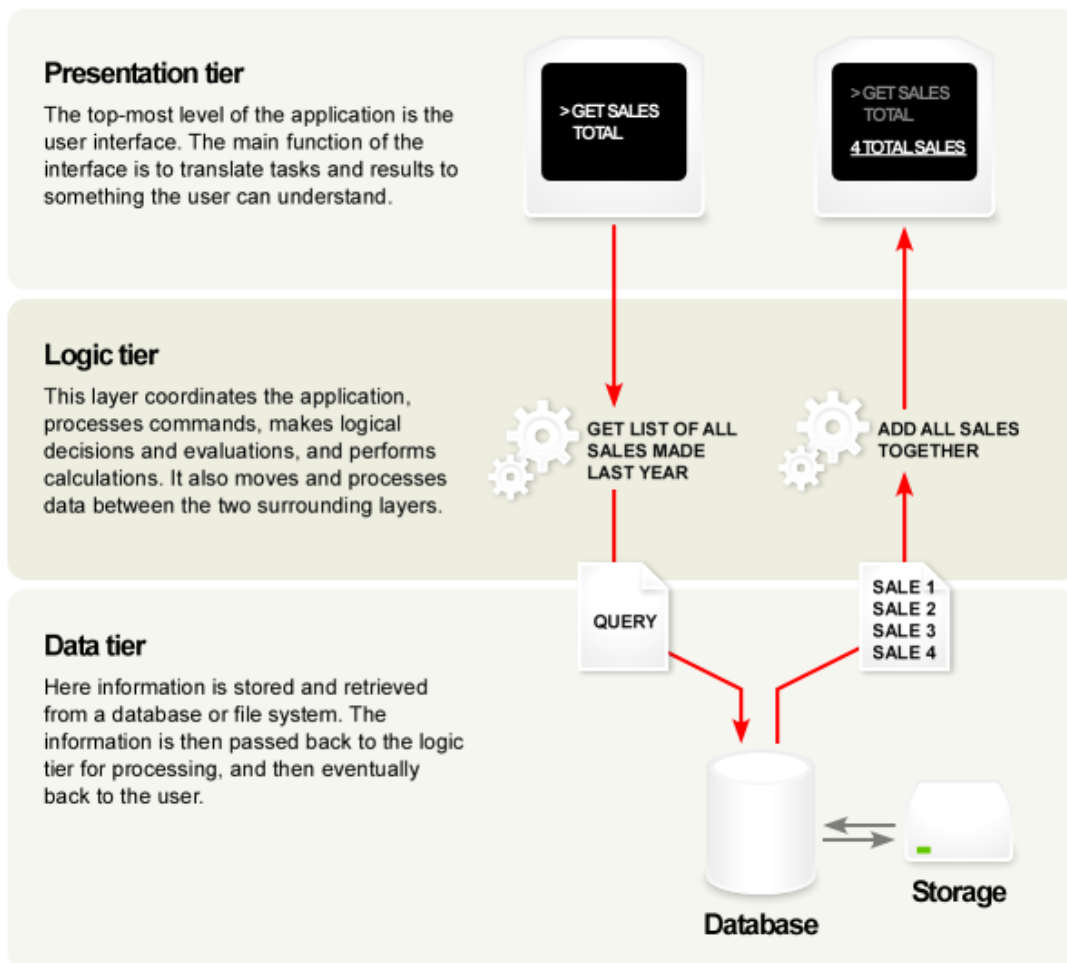


Figure 2: Three-tier architecture. (Source: Wikipedia)

2.2 Web Service Architectures

There are two main Web Service Architecture approaches: Service Oriented Architecture (SOA and the WS-* stack) and Resource Oriented Architecture (ROA and the REpresentational State Transfer – REST) [2].

2.2.1 Service Oriented Architecture (SOA) and the WS-* stack

The Service Oriented Architecture is described in Section 2.2.1.1 of deliverable D1.1.

2.2.2 Resource Oriented Architecture (ROA) – RESTful Web Services

The Resource Oriented Architecture is described in Section 2.2.1.2 of deliverable D1.1.

2.2.3 Comparison between SOA and ROA

As it was pointed out in D1.1 the SOA approach is preferred in *enterprise application integration* scenarios with a longer lifespan and advanced QoS requirements whereas the ROA approach is simpler and more flexible but requires a lot of low-level coding and does not provide the enterprise-level features of SOA (security, subscription-notification, orchestration, choreography, transactions, etc). However, in the context of multi-tiered software architecture it is possible to combine both architectural approaches. For example, the web services at the Data Access Tier can provide low-level functionality behind a Firewall. Therefore, Data Access Services may follow the ROA approach since enterprise-level issues will not be tackled on this level.

3 Overview of the NRG4CAST Toolkit Architecture

The NRG4CAST toolkit architecture will follow a multi-tiered Web Service Architecture approach which is depicted in Figure 3. The services provided by the platform are classified into the following tiers: the **Data Access Tier**, the **Integration Tier**, and the **Application Tier**. Furthermore, the NRG4CAST toolkit federates a number of external and/or local Data Resources which reside on the **Resource Tier** and provides Web UI / Rich Client UI which resides on the **Presentation Tier**.

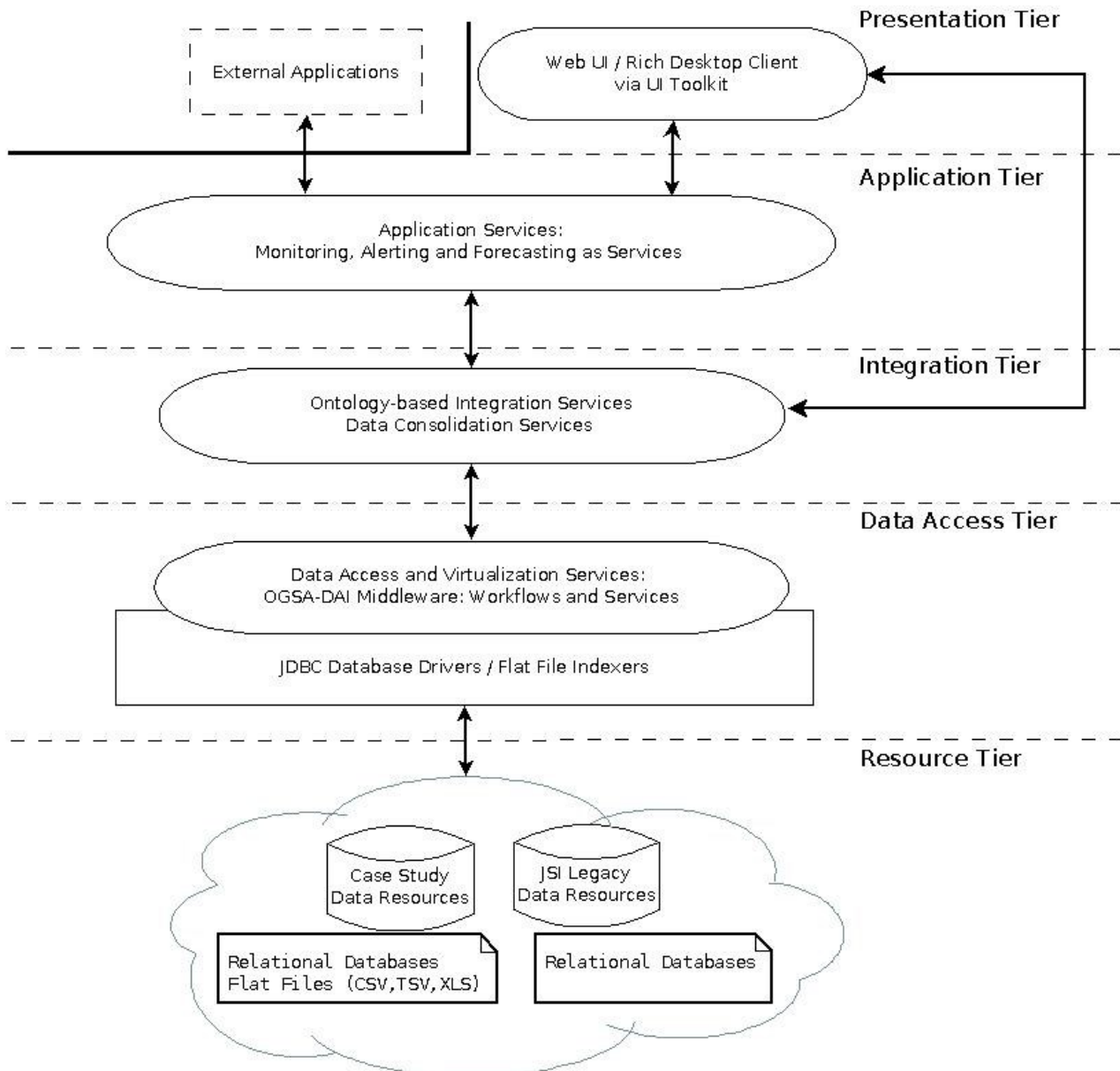


Figure 3. The NRG4CAST toolkit architecture

3.1 Rationale

The main objectives and innovation challenges NRG4CAST addresses from the architectural perspective are the following:

- The NRG4CAST project aims at providing a toolkit that will integrate a set of real-time energy monitoring and prediction components provided as services with specific APIs to external systems and applications.

- In order to provide energy monitoring and forecasting functionality, it has to combine different types of information obtained from different and heterogeneous local infrastructures, information systems and databases.
- Both data-driven prediction methods as well as reasoning and decision-making methods necessitate mapping of low-level data as well as knowledge formalization and representation through the use of an ontological schema, i.e. the NRG4CAST ontology which will be produced by integrating and merging a number of existing domain sub-ontologies from relevant areas like energy efficiency, environmental monitoring, etc.

A **multi-tiered architectural approach** is based on the principle of the separation of the concerns of the system into stacked groups. In the case of NRG4CAST there is a distinct data Resource Tier which resides on the external information systems or on the NRG4CAST toolkit server file system. The Data Access Tier is responsible for access, management, federation, integration and delivery of the heterogeneous shared data resources. Next comes the Integration Tier which is responsible for the translation of queries over the common ontological schema or alternatively for the exposition of a set of application-specific data services (e.g. distributed queries) over the data resources exposed by the Data Access Tier in order to hide low-level details and complexity. The Application Tier provides monitoring and forecasting functionality as a service to both the NRG4CAST toolkit Presentation Tier as well as to external systems.

Apart from being suitable to the NRG4CAST requirements according to the previous discussion, multi-tiered architectures exhibit a number of general advantages: improved scalability and availability; finer control of security which may be enforced independently on each tier (e.g. Resource Tier and Data Access Tier may reside behind firewall for protection); decoupling of tiers / ability to make changes to a tier without affecting the rest; more efficient development and maintenance; easy new feature addition; improved reusability [3].

Web Services can be present in most of the tiers of the multi-tiered architecture except for the Presentation tier. Moreover, it is recommended to decouple a Tier from another Tier as much as possible through the use of a decoupling technique such a Web Service approach [3].

Starting at the bottom and going up, a data resource often exposes its contents through a Web Service Interface, as for example is the case of the Energy 3D Cadastre Geographical Information System provided by CSI Piemonte in the context of the NRG4CAST Turin pilot scenario. The Data Access Tier exposes virtualized and federated data resources via web services. The OGSA-DAI platform which is used in NRG4CAST provides 6 types of web services (Data request execution service, Data resource information service, Data sink service, Data source service, Session management service, Request management service) discussed in Section 4.1 of this document. Through these services OGSA-DAI provides a layer in which to hide database heterogeneities which can yield thinner clients and allows users to focus on application-specific features, while getting standard functionalities out-of-the-box. Specifically, these services act as a middleware for building higher-level application-specific data services which are located at the Integration Tier. Finally, in the context of NRG4CAST, the Application Tier provides monitoring and forecasting APIs to existing systems therefore their design and implementation as Web Services is highly recommended.

The need on the one hand to integrate information coming from heterogeneous data resources and the need on the other hand to provide advanced monitoring and forecasting functions as APIs to existing energy management systems render the multi-tiered Web Services architectural approach highly appropriate.

3.2 Resource Tier

This Tier includes existing data resources such as information systems and databases. In the context of the NRG4CAST pilot scenarios data resources include CSV, TSV, DBF and Excel files and Relational Databases. Relational databases are accessed through the connectivity infrastructure of the respective Relational Database Management Systems whereas CSV, TSV, DBF and Excel files are accessed through HTTP or FTP. In the specific case of the Turin pilot, GIS access will be initially provided through DBF file delivery. In a next phase of the project GIS will expose information through a web service.

3.3 Data Access Tier

The services belonging to the Data Access Tier provide homogeneous and uniform *access* to a number of heterogeneous data resources through the interfaces which they expose. For this purpose the OGSA-DAI (Open Grid Software Architecture – Data Access and Integration) middleware is adopted [4].

3.3.1 Criteria for the adoption of the OGSA-DAI middleware

In order to select a middleware for distributed data access, integration and management a number of different software offerings were reviewed. Commercial products such as Informatica, SAS Institute, Jitterbit, IBM Infosphere and Oracle Data Integrator were excluded from the review process due to the high cost of royalties and licensing fees.

The main remaining alternatives can be classified into two categories: (1) **Open Core** products: Talend ESB and Pentaho and (2) **Open Source** products: OGSA-DAI and ARDA Metadata Catalogue Project (AMGA).

Talend ESB and Pentaho products provide a rich set of functions and features for publishing data access and integration services which integrate and virtualize heterogeneous data resources, such as support of the whole stack of WS-* standards, visual tools for integrating data and exposing them through the creation and deployment of RESTful and SOAP Web Services, administration and testing tools, connectors to any data resource type, etc. In this respect, the performed tests have shown that these products are as comprehensive and complete as closed-source commercial solutions. However, in the case of Talend Open Studio for ESB the licensing model would oblige the NRG4CAST toolkit to adopt the GPL license which is very restrictive with respect to the exploitation of the results of the NRG4CAST project. On the other hand, in the case of Pentaho the core package excludes most of the necessary functionality which is available only in the commercially licensed edition. Therefore, the Open Core alternatives were also excluded from the selection process.

The tests that were performed in the context of the NRG4CAST project in combination with the published comparative evaluation of AMGA and OGSA-DAI [5] have exhibited the following results, as far as the Open Source products are concerned: (1) the OGSA-DAI middleware supports many different type of data resources (relational, XML, RDF, files) whereas AMGA supports only relational databases; (2) OGSA-DAI supports the submission of complex data-centric workflows and transformations on the data whereas AMGA supports only a subset of SQL, i.e. it does not even support complex queries including inner and outer joins; (3) SQL queries are slightly slower on OGSA-DAI than AMGA; (4) OGSA-DAI can be easily extended to support new types of data resources in contrast with AMGA; (5) it is not simple to deal with more than one database at the same time in the case of AMGA.

Further, the OGSA-DAI middleware has emerged as an industrial standard since it has been used as a Data Access and Integration middleware by a large number of projects and organisations around the world, in sectors including medical research, geographical information systems, meteorology, transport, computer-aided design, engineering and astronomy, environmental management, etc. [6]. Specifically, it has been widely used in software platforms which access, exploit and integrate streaming **sensor data** as it is the case with the NRG4CAST project, i.e. the Global Earth Observation Grid project (sharing satellite, geological and ground-sensed data) [7] and the Mobile Environmental Sensing System Across Grid Environments (monitoring traffic pollution) [8]. Moreover, it has been used as the data integration platform in projects focusing on **forecasting** such as the Linked Environments for Atmospheric Discovery project [9].

According to the above, the OGSA-DAI platform has been selected as the most suitable middleware for the Data Tier of the NRG4CAST toolkit.

3.3.2 OGSA-DAI middleware Description

OGSA-DAI supports: (1) **Data access** - access to structured data in distributed heterogeneous data resources; (2) **Data transformation**, e.g. data in schema A are transformed into schema B to be exposed to the users; (3) **Data integration**, e.g. multiple databases are exposed to the users as a single virtual

database; (4) **Data delivery** - the most appropriate means, e.g. web service, e-mail, HTTP, FTP, GridFTP, can be used to deliver the data to where it is necessary [4].

The OGSA-DAI middleware achieves this functionality through the execution of workflows which are submitted by clients to OGSA-DAI Web Services. OGSA-DAI workflows are comprised by a number of individual units of work in the workflow called activities. Activities perform a well-defined data-related task such as running an SQL query, performing a data transformation or delivering data. A workflow connects a set of activities: the output of an activity is connected to the input of another activity. Data flow is unidirectional. Further, data may be transformed by a Transformation Activity when the input and output data formats of different activities are not the same. Apart from data access and transformation, OGSA-DAI workflow can carry out data updates and data delivery. An important feature of OGSA-DAI is that it can index and lookup flat file data resources via the Apache Lucene library [10].

The OGSA-DAI middleware provides 5 types of Web Services: (1) **Data request execution service** consumed by clients to submit workflows and get the request status of requests; (2) **Data resource information service** consumed by clients to query information about a data resource, e.g. product name, vendor, version; (3) **Data sink service** consumed by clients to push data to data sinks; (4) **Data source service** consumed by clients to pull data from data sources; and (5) **Request management service** consumed by clients to query request execution status and get data associated with asynchronous requests.

Through these services, the OGSA-DAI middleware layer **hides database heterogeneities**. This approach exhibits several advantages: (1) thinner clients can be implemented; (2) users can focus on application-specific features since standard functionalities are provided out-of-the-box; (3) complex data integration scenarios are supported; (4) control of the data is retained by their providers.

There are several alternative implementations of the OGSA-DAI middleware which support different Web Service presentation layers such as: Axis 1.4, Globus Toolkit, WS-DAIR and WS-DAIX, Jersey RESTful, Jersey RESTful. NRG4CAST adopts the OGSA-DAI 4.2 Jersey presentation layer which is the most recent, bug free and reliable providing simplicity and flexibility in consuming the OGSA-DAI services.

An overview of the OGSA-DAI middleware components is presented in Figure 4.

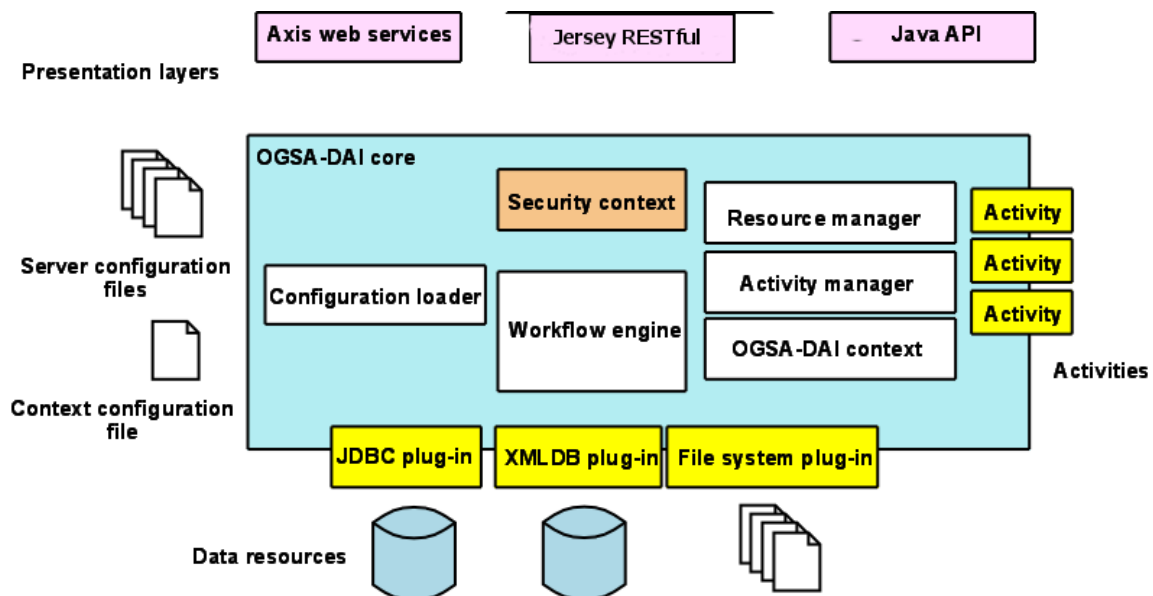


Figure 4. OGSA-DAI component diagram (adapted from [4])

The main components are the following:

- Activities which are objects that perform data-related operations.
- The configuration loader which is responsible for loading the configuration files which specify the available activities, resources, database credentials, etc.

- The resource manager which provides access to the data resources to the activities and the Presentation layer.
- The activity manager which provides access to the activities to the workflow engine.
- The workflow engine which executes workflows through the creation of activity objects, monitors workflow execution and updates the execution status.
- The Data resource plug-ins which are used in order to communicate with data resources.

3.4 Integration Tier

The services belonging to the Integration tier are classified into two main types:

- **Data Consolidation Services** which provide of a set of application-specific data services (e.g. distributed queries) over the data resources exposed by the Data Access Tier in order to hide low-level details and complexity. With the creation of this layer of services which use OGSA-DAI as an abstraction layer, functionality is further decoupled since the Application Tier Services should not bother with low level SQL queries or Apache Lucene text searches. Data Consolidation Services expose a standard but extensible set of simple “business” queries.
- **Ontology-based Integration Services** which provide integrated access to the data through a **single ontological representation** in order to achieve *syntactical and semantic homogeneity* of the heterogeneous data resources. In order to create a single ontological representation the structure of the flat files, the database schemas as well as the content of the data resources will be syntactically and semantically aligned to the NRG4CAST ontology. The ontology alignment will utilize appropriate languages, constructs and tools such as SPARQL, R2O, ODEMapster, Virtuoso, etc. However, the ontology alignment is necessarily a semi-automated process since the input from domain experts may become necessary in complex alignment definitions.

The services of the Integration Tier consume the services of the Data Access Tier in order to gain access to the data resources.

3.5 Application Tier

The services belonging to the Application tier provide monitoring, alerting and forecasting functionality according to the requirements of the pilot cases. The application services consume the integration services in order to gain access to the data and utilize advanced knowledge technologies, such as machine learning, data and text mining, stream mining, link analysis, information extraction, knowledge formalization and reasoning methods and algorithms. The functionality of the services will be exposed through Web application User Interface components. Moreover, the application services may be consumed by external applications according to their RESTful interfaces.

3.6 Presentation Tier

The detailed design and implementation of the Presentation Tier will be performed in the “Initial Roll-out” phase of the project and will be included in D1.4 “Final Toolkit Architecture Specification” which is due on M15. A UI toolkit will be designed and implemented which will enable fast prototyping of rich desktop clients and Web front-ends on the Presentation Tier of the NRG4CAST toolkit.

3.7 Related Architectures

In this Section we describe the architectures of two software platforms accessing sensor data and heterogeneous data sources in order to provide domain-specific application services (decision support, monitoring, etc.). These descriptions highlight the similarities and differences with the NR4CAST toolkit architecture in relation to the different objectives of each platform.

3.7.1 SemSorGrid4Env

The Semantic Sensor Grid Rapid Application Development for Environmental Management (SemSorGrid4Env) IST FP7 project developed a software platform for the rapid development of applications over heterogeneous environmental data sources, including both sensor networks and traditional databases [11].

The SemSorGrid4Env software architecture is comprised of a set of Web services. The SemSorGrid4Env Web services are divided into three tiers according to their functionality (Figure 5):

- **Application Tier:** Services that provide domain specific support to applications;
- **Middleware Tier:** Services which use of semantic techniques, e.g. service discovery and data integration;
- **Data Tier:** Services that provide access to streamed or stored data.

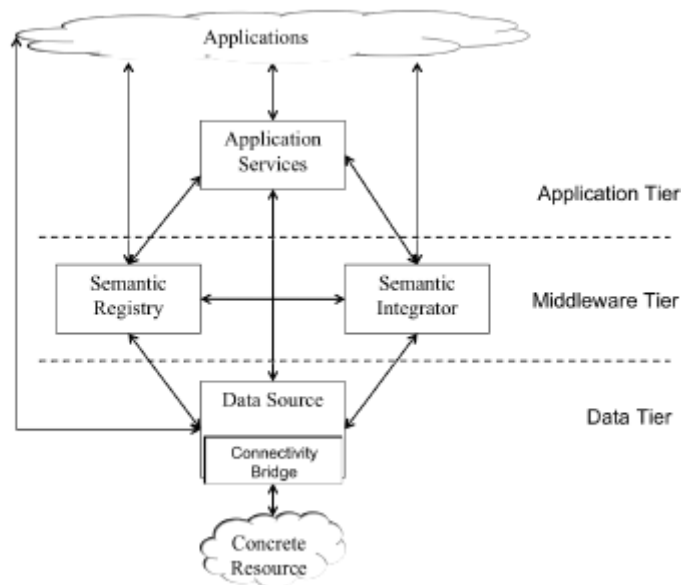


Figure 5. The SemSorGrid4Env architecture: the services and their relationships [11]

The infrastructural backbone of the architecture consists of the specification of five core service-oriented Web services: the **Registration and Discovery Service** and the **Integration and Querying Service** at the Middleware Tier as well as the **Stored Data Service**, the **Streaming Data Service**, and the **Distributed Query Processing Service** at the Data Tier.

3.7.2 ENVISION

ENVISION (Environmental Services Infrastructure with Ontologies) project (FP7, ICT for Environmental Services and Climate Change Adoption) aimed to provide a platform, that would enable technically unskilled users to take advantage of advanced information technologies built on top of environmental sensing platforms. The platform provides easy semantic discovery of environmental data sources and services and prepares adaptive chaining and composition of environmental models provided as services.

ENVISION platform [12] provides a suitable infrastructure to develop environmental decision portals.

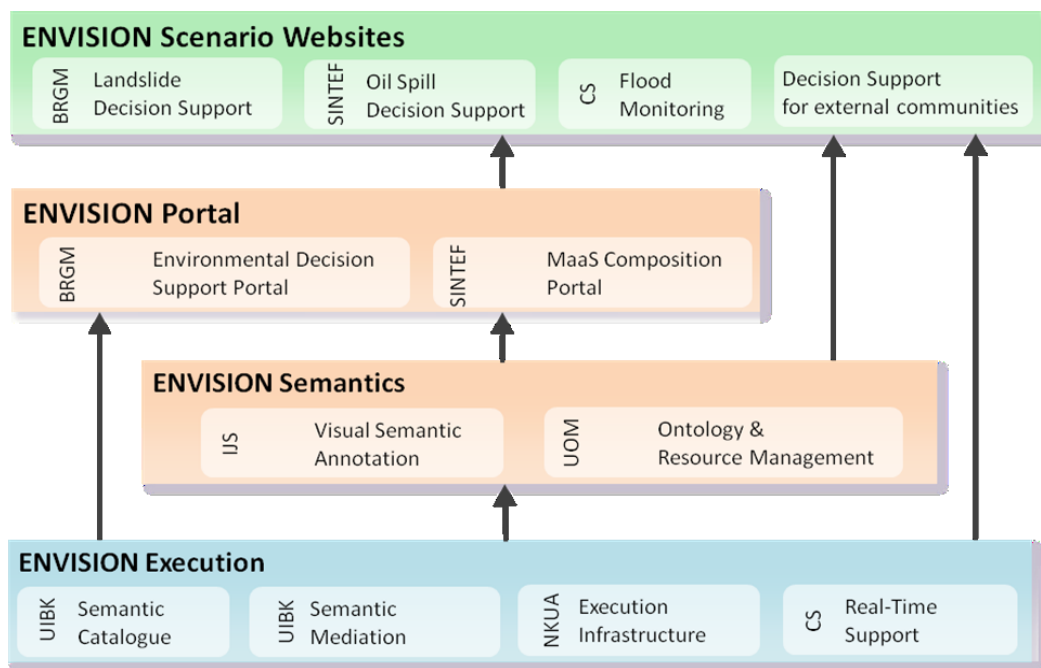


Figure 6: ENVISION project architecture.

Figure 6 depicts ENVISION project's service-oriented architecture for front- and back-end. Objective of the architecture is a distributed service infrastructure and a pluggable user interface. The platform is not integrated, only distributed components are served. The goal is to provide building blocks for an expert user, to build an interface, which provides a specific needed functionality.

Platform provides many universal components that can be reused in different scenarios. Focus is on standardized communication between the components to ensure that individual components can be easily swapped and integrated.

Back-end services in the platform (blue) are doing the number crunching, the portlets are managing the visualizations (red), and finally the client-side websites (green) are showing the results and enabling user interaction with the system.

Prerequisite for all the services on the top of the data is a standard and universal solution to provide access to the sensor data. As the project was dealing with mostly environmental data the most common solution was the usage of Open Geospatial Consortium (OGC) standards.

Sensor Observation Service (SOS) was chosen to handle the collection and distribution of sensor meta-data and sensor measurements. SOS implements different standards. It uses SensorML standard to describe sensors and their capabilities and it uses OGC O&M (observation and measurement) standard for data exchange. In theory SOS service can be polled for its capabilities (what sensors, physical phenomena, what is the measurements metadata, like units of measurements or accuracy, time-span of the measurements, geolocation etc.) and then it can be queried for data (measurements) for certain properties. In practice the result of ENVISION project was also testing of the SOS servers. The project consortium concluded that many implementations or even installations of the same implementation of an SOS server use plenty of different schemas, which makes it difficult to implement a general-purpose parser for such a service. A simpler schema (REST for example) might be much more effective both in the sense of physical and human resources.

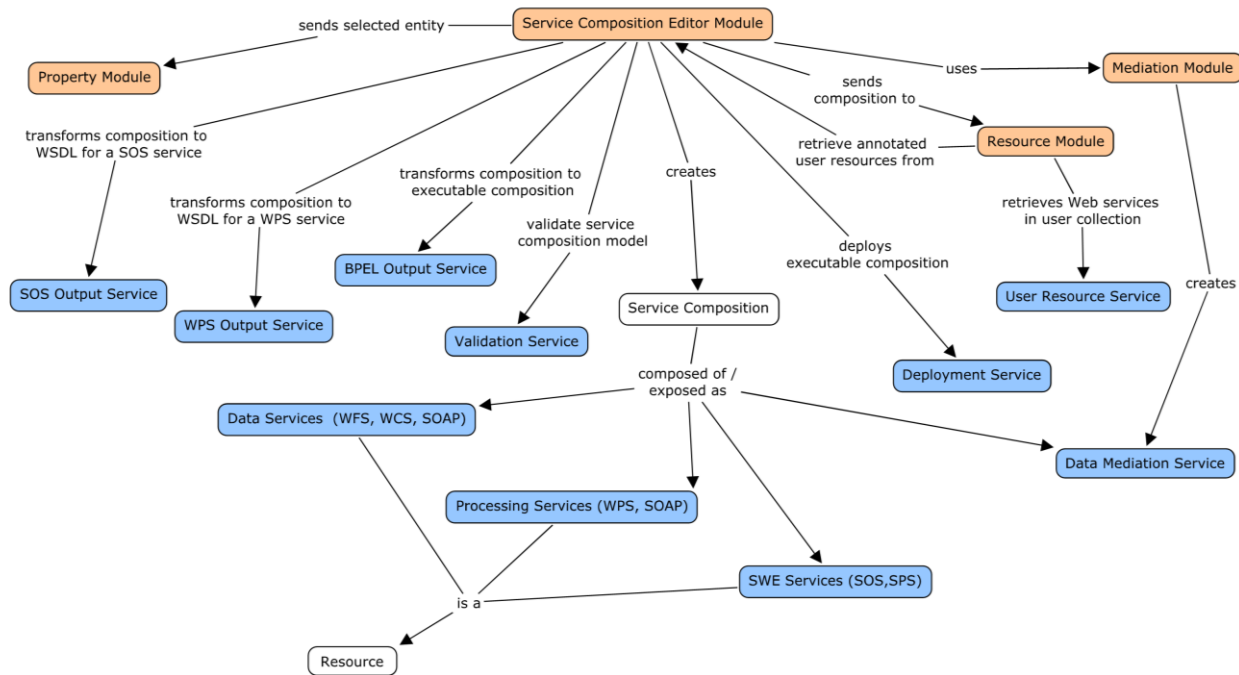


Figure 7: Interaction of components among back-end services and portlets.

Components within ENVISION platform are loosely coupled. Any component within the platform can be replaced with another as long as it respects the communication schema of the platform. Figure 7 shows that the web services are using standard protocols like SOAP and SOS, which are described with a standard way (WSDL for SOAP services).

The main module of the platform is the Service Composition Module, where service compositions are created. A service composition is composed of a set of interacting resources and data mediation services to connect the resources in a proper manner. The service composition editor may naturally be described as a four-layer architecture: Web-Client-Server-DB. The Web interface layer provides a model view to the end-user in Web browser. The client layer provides a link between the Web interface and the back-end server code. The server side code provides the application. Finally, the database (DB) layer enables persistent storage of the models.

3.7.3 Comparison with the NRG4CAST toolkit architecture

Both of the presented architectures follow an N-Tier service oriented approach. The different number and type of tiers stems from the different functionality provided by each platform which is therefore vertically decomposed in different ways. In the case of Envision, the user scenarios are not exhaustively defined a priori. The goal of the platform is to provide an expert user with a set of building blocks which can be combined in order to provide relevant functionality in a flexible way. Therefore, the architecture utilizes the WS-BPEL language so as to provide different service compositions through the respective module. On the contrary, the SemSorGrid4Env architecture does not support the composition of service workflows even if it aims at providing a platform for sensor-aware applications which are not defined a priori. In this respect it is more similar with the NRG4CAST approach. WS-BPEL should be used when the requirements are process-centric and the logic of the processes is complex [13]. Therefore, the NRG4CAST architecture follows the simpler approach, according the principle of parsimony, since the project objectives and the user requirements are mainly centred on the capabilities of the forecasting and monitoring algorithms as well as on the ability to semantically represent and map low-level data and not on process-wise flexibility.

3.8 Summary

This section has provided an overview of the NRG4CAST toolkit specifications. The following sections present an overview of the functionality of the Web Services pertaining to each respective tier and their

relationships. The detailed specification of the Web Services comprising the toolkit architecture includes a full description of the interfaces exposed by the services, i.e. of the set of operations they comprise and the metadata that they make available, of their inputs, outputs and possible faults.

4 Services

This section defines the Web Services offered by the NRG4CAST toolkit on the corresponding Tiers. The definition includes the dependencies of each service on other services as well as the functionality they provide through their interfaces.

4.1 Data Access Tier Services

The Data Access Tier Services are RESTful Services provided by the OGSA-DAI 4.2 Jersey Technology middleware (see Section 3.3).

4.1.1 Data Request Execution Service

The Data Request Execution Service lists existing Data Request Execution Resources (DRERs) and submits OGSA-DAI workflows to them. After the submission of a workflow the request status is returned.

4.1.1.1 Interfaces

`http://{server.url}/dai/services/drers`

Description: DRER parent

POST: no-op

GET: lists existing DRERs eg DataRequestExecutionResource
* Accept: text/html and application/json

PUT: no-op

DELETE: no-op

`http://{server.url}/dai/services/drers/{drer}`

POST: Takes an OGSA-DAI workflow which should include
RequestExecutionType.SYNCHRONOUS or ASYNCHRONOUS in the request,
returns request status which includes the request url.

* Content-Type: application/xml

* Accept: application/xml

GET: lists existing properties of the resource

* Accept: text/html and application/json

PUT: no-op

DELETE: no-op

`http://{server.url}/dai/services/drers/{drer}/async`

`http://{server.url}/dai/services/drers/{drer}/sync`

POST: Takes an OGSA-DAI workflow (sync or async accordingly),
returns request status which includes the request url.

* Content-Type: application/xml

* Accept: application/xml

GET: lists existing properties of the resource

* Accept: text/html and application/json

PUT: no-op

DELETE: no-op

4.1.1.2 Dependencies

None

4.1.2 Data Source Service

The Data Source Service is consumed by clients to pull data from data sources. In the context of the OGSA-DAI middleware, Data Sources are Resources which provide a streamable, and hence scalable, alternative

to deliver data. A request can be sent to stream data into them via the OGSA-DAI WriteToDataSource activity. A client can then stream data from the data source via the OGSA-DAI data source service. Alternatively they can request that another server stream the data by sending a request that contains the OGSA-DAI ObtainFromDataSource activity. Data sources support a "pull" mode of data delivery. Data Sources support both synchronous and asynchronous requests but are better suited for asynchronous requests since their use is appropriate when large data sets are transferred.

4.1.2.1 Interfaces

`http://{server.url}/dai/services/dataSources`

Description: data sources parent

POST: creates a new data source returning the ID of the new resource
* Accept: text/plain

GET: lists existing data sources
* Accept: text/html and application/json

PUT: no-op

DELETE: no-op

`http://{server.url}/dai/services/dataSources/{dataSource}`

POST: consumes from the data source (the data disappears as consumed)
* Accept: application/octet-stream

GET: lists existing properties of the resource
* Accept: text/html and application/json

PUT: no-op

DELETE: deletes the data source

4.1.2.2 Dependencies

None

4.1.3 Data sink service

The Data Sink Service is consumed by clients to push data to data sinks. In the context of the OGSA-DAI middleware, Data Sinks support a "push" mode of data delivery. A request can be sent to stream data from them via the OGSA-DAI ReadFromDataSink activity. A client can also stream data into the sink, and so into a workflow via the OGSA-DAI data sink service. Alternatively, they can request that another server stream the data by sending a request that contains the OGSA-DAI DeliverToDataSink activity.

4.1.3.1 Interfaces

`http://{server.url}/dai/services/dataSinks`

Description: data sink parent

POST: creates a new data sink returning the ID of the new resource
* Accept: text/plain

GET: lists existing data sinks
* Accept: text/html and application/json

PUT: no-op

DELETE: no-op

`http://{server.url}/dai/services/dataSinks/{dataSink}`

POST: no-op

GET: lists existing properties of the resource
* Accept: text/html and application/json

PUT: uploads data to the sink

* Accept: application/octet-stream
DELETE: deletes the data sink

http://{server.url}/dai/services/dataSinks/{dataSink}/status
Description: status

POST: no-op
GET: gets the read status of the sink
* Accept: text/html and application/json
PUT: changes the write status of the sink to one of the following:
DataSinkStatus.COMPLETE or DataSinkStatus.ERROR
* Content-Type: text/plain
DELETE: no-op

4.1.3.2 Dependencies

None

4.1.4 Data Resource Information Service

The Data Resource Information Service is consumed by clients to query information about a data resource, e.g. product name, vendor, version.

4.1.4.1 Interfaces

http://{server.url}/dai/services/dataResources
Description: data resources parent

POST: no-op
GET: lists existing data resources
* Accept: text/html and application/json
PUT: no-op
DELETE: no-op

http://{server.url}/dai/services/dataResources/{dataResource}

POST: no-op
GET: lists existing properties of the resource
* Accept: text/html and application/json
PUT: no-op
DELETE: no-op

4.1.4.2 Dependencies

None

4.1.5 Request Management Service

The Request Management Service is consumed by clients to query request execution status and get data associated with asynchronous requests.

4.1.5.1 Interfaces

http://{server.url}/dai/services/requests
Description: requests parent

POST: no-op
GET: lists existing request
* Accept: text/html and application/json
PUT: no-op

DELETE: no-op

http://{server.url}/dai/services/requests/{request}

POST: no-op

GET: lists existing properties of the resource
* Accept: text/html and application/json

PUT: no-op

DELETE: deletes the request

http://{server.url}/dai/services/requests/{request}/results

POST: no-op

GET: retrieves request status property
* Accept: application/xml

PUT: no-op

DELETE: no-op

4.2 Integration Tier Services

The Integration Tier services are WS-Services and are classified into two categories: (1) **Data Consolidation services** which provide simple application specific data queries as a service through the consumption of Data Access Tier services and thus hide low level complexity and (2) **Ontology-based Integration Services** which provide access to the data through a **single ontological representation**, i.e. the NRG4CAST Ontology. The Ontology Services may also consume OGSA-DAI services and specifically through the submission of workflows which perform federated queries onto the **SPARQL-DQP** data resource extension of the OGSA-DAI middleware which utilizes the **R2O** (Relational to Ontology mapping language) engine, an implementation of the **R2RML** recommendation for mapping ontologies to relational databases [14]. The R2O engine may be also used for mapping CSV / Excel flat files to the ontology.

Further, security concerns such as authentication and authorization of the clients can be enforced on this Tier through the use of the WS-Security functionality.

Interfaces of the Integration Tier Services are described schematically according to the WSDL specification.

4.2.1 Data Consolidation Services

4.2.1.1 Query Relational Resource Service

The Query Relational Resource Service provides operations for retrieving tuples according to an SQL query from a specific data resource which abstracts a Relational Database.

4.2.1.1.1 Interfaces

Operation executeQuery: This operation returns a data set according to an SQL query on a specific relational data resource. It is performed **synchronously** and it is appropriate for small result sets. Inputs, outputs and faults are depicted in Figure 8.

Dependencies: *Data Resource Execution Service* (Data Access Tier)

Operation executeQueryWithTimeInterval: This operation returns a portion of the data that the query requests depending on the interval time that the client submits. The query is performed **asynchronously** and is appropriate for large result sets and streaming data. Inputs, outputs and faults are depicted in Figure 9.

Dependencies: *Data Resource Execution Service, Data Source Service* (Data Access Tier)

Parameters (2)	
Parameter Name	Parameter Type
resourceId	java.lang.String
query	java.lang.String

Output
Return type: gr.singularlogic.nrg4cast.data.ResultData

Faults (5)	
Parameter Name	Parameter Type
MalformedURLException	gr.singularlogic.nrg4cast.webservices.faults.MalformedURLException
IOFault	gr.singularlogic.nrg4cast.webservices.faults.IOFault
SQLFault	gr.singularlogic.nrg4cast.webservices.faults.SQLFault
ResourceUnknownFault	gr.singularlogic.nrg4cast.webservices.faults.ResourceUnknownFault
UnexpectedDataValueFault	gr.singularlogic.nrg4cast.webservices.faults.UnexpectedDataValueFault

Figure 8. Query Relational Resource operation: executeQuery (inputs, outputs and faults)

Parameters (4)	
Parameter Name	Parameter Type
resourceId	java.lang.String
query	java.lang.String
time_interval	int
more_data	boolean

Output
Return type: gr.singularlogic.nrg4cast.data.ResultData

Faults (6)	
Parameter Name	Parameter Type
MalformedURLException	gr.singularlogic.nrg4cast.webservices.faults.MalformedURLException
IOFault	gr.singularlogic.nrg4cast.webservices.faults.IOFault
SQLFault	gr.singularlogic.nrg4cast.webservices.faults.SQLFault
ResourceUnknownFault	gr.singularlogic.nrg4cast.webservices.faults.ResourceUnknownFault
UnexpectedDataValueFault	gr.singularlogic.nrg4cast.webservices.faults.UnexpectedDataValueFault
DataStreamErrorFault	gr.singularlogic.nrg4cast.webservices.faults.DataStreamErrorFault

Figure 9. Query Relational Resource operation: executeQueryWithTimeInterval (inputs, outputs and faults)

4.2.1.2 Query File Resource Service

The Query File Resource Service provides operations for retrieving rows according to a Lucene query from a specific data resource which abstracts a flat file (CSV, Excel, TSV, etc).

4.2.1.2.1 Interfaces

Operation executeQuery: This operation returns a data set according to a Lucene query on a specific flat file data resource. It is performed **synchronously** and it is appropriate for small result sets. Inputs, outputs and faults are depicted in Figure 10.

Dependencies: *Data Resource Execution Service* (Data Access Tier)

Operation executeQueryWithTimeInterval: This operation returns a portion of the data that the Lucene query requests depending on the interval time that the client submits. The query is performed **asynchronously** and is appropriate for large result sets and streaming data. Inputs, outputs and faults are depicted in Figure 11.

Dependencies: *Data Resource Execution Service, Data Source Service (Data Access Tier)*

The screenshot shows the 'executeQuery' operation interface. It is divided into three main sections: Parameters, Output, and Faults. The Parameters section has a table with two columns: Parameter Name and Parameter Type. The Output section shows the return type. The Faults section has a table with two columns: Parameter Name and Parameter Type.

Parameters (2)	
Parameter Name	Parameter Type
resourceId	java.lang.String
query	java.lang.String

Return type: gr.singularlogic.nrg4cast.data.ResultData

Faults (6)	
Parameter Name	Parameter Type
MalformedURLException	gr.singularlogic.nrg4cast.webservices.faults.MalformedURLException
IOFault	gr.singularlogic.nrg4cast.webservices.faults.IOFault
LuceneParseException	gr.singularlogic.nrg4cast.webservices.faults.LuceneParseException
ResourceUnknownFault	gr.singularlogic.nrg4cast.webservices.faults.ResourceUnknownFault
UnexpectedDataValueFault	gr.singularlogic.nrg4cast.webservices.faults.UnexpectedDataValueFault
DataStreamErrorFault	gr.singularlogic.nrg4cast.webservices.faults.DataStreamErrorFault

Figure 10. Query File Resource operation: executeQuery (inputs, outputs and faults)

The screenshot shows the 'executeQueryWithTimeInterval' operation interface. It is divided into three main sections: Parameters, Output, and Faults. The Parameters section has a table with four columns: Parameter Name and Parameter Type. The Output section shows the return type. The Faults section has a table with two columns: Parameter Name and Parameter Type.

Parameters (4)	
Parameter Name	Parameter Type
resourceId	java.lang.String
query	java.lang.String
time_interval	int
more_data	boolean

Return type: gr.singularlogic.nrg4cast.data.ResultData

Faults (6)	
Parameter Name	Parameter Type
MalformedURLException	gr.singularlogic.nrg4cast.webservices.faults.MalformedURLException
IOFault	gr.singularlogic.nrg4cast.webservices.faults.IOFault
LuceneParseException	gr.singularlogic.nrg4cast.webservices.faults.LuceneParseException
ResourceUnknownFault	gr.singularlogic.nrg4cast.webservices.faults.ResourceUnknownFault
UnexpectedDataValueFault	gr.singularlogic.nrg4cast.webservices.faults.UnexpectedDataValueFault
DataStreamErrorFault	gr.singularlogic.nrg4cast.webservices.faults.DataStreamErrorFault

Figure 11. Query File Resource operation: executeQueryWithTimeInterval (inputs, outputs and faults)

4.2.1.3 Server Information Service

The Server Information Service provides operations for listing Data Resources deployed in a OGSA-DAI server. Further operations may be added in the next phases of the project.

4.2.1.3.1 Interfaces

Operation getResources: This operation returns a list of the Data Resources ids. Inputs, outputs and faults are depicted in Figure 12.

Dependencies: *Data Resource Information Service (Data Access Tier)*

Parameters (0)	
No Parameters.	

Output	
Return type: java.util.ArrayList<gr.singularlogic.nrg4cast.data.ResourceInfo>	

Faults (1)	
Parameter Name	Parameter Type
MalformedURLException	gr.singularlogic.nrg4cast.webservices.faults.MalformedURLException

Figure 12. Server Information Service operation: getResourcelds (inputs, outputs and faults)

4.2.1.4 Subscription Notification Service

The Subscription Notification Service provides operations by which (1) a client can subscribe to receive notifications when an event happens (**subscribe**, **getCurrentMessage** operation); (2) a client can manage its subscription for receiving notifications: e.g. alter the filtering condition, end the subscription, etc. (**renew**, **unsubscribe** operations); (3) a client receives notifications (**notify** operation). The service follows the specification of the Ws-BaseNotification standard [15].

4.2.1.4.1 Interfaces

Operation subscribe: a client is registered to receive a subset of the notification messages sent by the service. Inputs, outputs and faults will be described in deliverable 1.4.

Dependencies: *none*

Operation renew: enables a client to renew its subscription (lifetime, filtering condition). Inputs, outputs and faults will be described in deliverable 1.4.

Dependencies: *none*

Operation getCurrentMessage: enables a new client to retrieve the most recent notification (lifetime, filtering condition). Inputs, outputs and faults will be described in deliverable 1.4.

Dependencies: *none*

Operation unsubscribe: enables a client to end its subscription. Inputs, outputs and faults will be described in deliverable 1.4.

Operation notify: enables a client to receive notifications according to its subscription. Inputs, outputs and faults will be described in deliverable 1.4.

Dependencies: *Data Resource Execution Service, Data Source Service (Data Access Tier)*

4.2.2 Ontology-based Integration Services

4.2.2.1 Integration Service

The Integration Service provides operations by which a client can gain access to the data through a single ontological representation. The operations utilize the **SPARQL-DQP** extension of the OGSA-DAI middleware in order to access data from multiple resources through a **SPARQL** query which utilizes an **R2O** - enabled mapping of the data resources to the ontology.

Operation integrateAs: This operation creates a SPARQL-DQP data resource which unifies a set of existing OGSA-DAI data resources into a single ontological representation. The set of existing data resources may be discovered by the client through the *Server Information Service* (Integration Tier). Inputs, outputs and faults are described in Figure 13.

Dependencies: *Data Resource Execution Service (Data Access Tier),*

Operation executeSPARQLQuery: This operation returns a data set according to a SPARQL query on a specific SPARQL-DQP data resource. Inputs, outputs and faults are described in Figure 14.

Dependencies: *Data Resource Execution Service* (Data Access Tier)

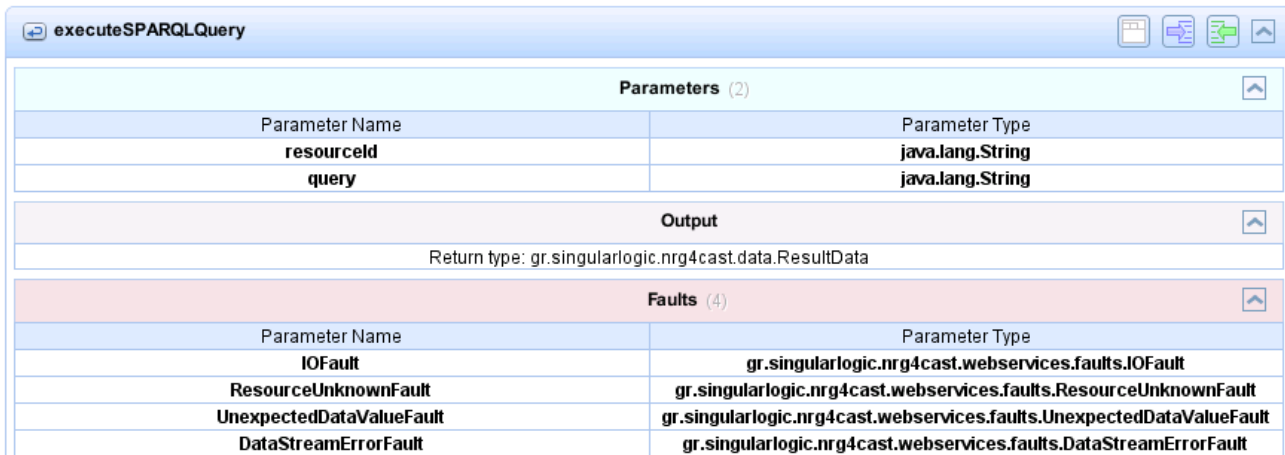


Parameters (3)	
Parameter Name	Parameter Type
listOfResources	java.util.List<java.lang.String>
SPARQLDQPResourceId	java.lang.String
mappingDocument	java.lang.String

Output
Return type: java.lang.String

Faults (2)	
Parameter Name	Parameter Type
IOFault	gr.singularlogic.nrg4cast.webservices.faults.IOFault
ResourceUnknownFault	gr.singularlogic.nrg4cast.webservices.faults.ResourceUnknownFault

Figure 13. Integration Service operation: integrateAs (inputs, outputs and faults)



Parameters (2)	
Parameter Name	Parameter Type
resourceId	java.lang.String
query	java.lang.String

Output
Return type: gr.singularlogic.nrg4cast.data.ResultData

Faults (4)	
Parameter Name	Parameter Type
IOFault	gr.singularlogic.nrg4cast.webservices.faults.IOFault
ResourceUnknownFault	gr.singularlogic.nrg4cast.webservices.faults.ResourceUnknownFault
UnexpectedDataValueFault	gr.singularlogic.nrg4cast.webservices.faults.UnexpectedDataValueFault
DataStreamErrorFault	gr.singularlogic.nrg4cast.webservices.faults.DataStreamErrorFault

Figure 14. Integration Service operation: executeSPARQLQuery (inputs, outputs and faults)

4.3 Application Tier Services

Application tier services assume lower tier services to be functional.

Input data is obtained directly from the integration tier. This data can be twofold: static and streaming. Static data inputs do not represent a problem for an engine that is being able to handle streaming data. EnStream described in [16] is a Stream Processing Engine (SPE), designed and tested for working with high load of streaming data. EnStream has a built-in data layer, which does not duplicate any services from Data Tier, but stores aggregates and other data relevant for machine learning methods and other algorithms.

4.3.1 Event Processing Service (Alerting)

Event Processing Service triggers alerts and warnings based on different rules. There are various possible scenarios for discovery of the rules:

1. Expert user has sufficient knowledge of the system and is able to create a rule without any support.
2. Expert user knows about a certain type of events appearing and is using a graphical user interface to analyze the sensor data, different aggregates to get an idea about the behavior of the system

prior to the event. Also a rule suggestion method would be useful in this scenario (e. g. a method providing overview of the relevant sensors and creating rules based on those).

3. Expert user provides list of events of a certain type and the system tries to build a model, based on the timestamp of events and corresponding data.

A simple and effective GUI should be available for creating the rules and visualizing the data.

System should be able to evaluate rules on large amounts of data (several hundreds of sensors) in a predefined timespans. Effective indexing and pre-processing of sensor measurement would be needed therefore.

Event detection, where rules are provided by expert user, can only return true or false, therefore priority of the event (case that the rule is fulfilled) should be entered by the expert user. In the case of event prediction based on the model, system returns probability of the event happening based on the learning process. Expert user should define a probability threshold for triggering such an event; probability can be used also as a priority parameter, which should be determined by an expert user.

A good practice with stream data handling is seamless transfer from historical to current data. Event Processing Engine (EPE) should therefore be able to process the rules, created on historical data. If possible, same rules should be used. Also automatic translation of the rules would be an acceptable solution.

Relative position of EPE within the EnStream component is depicted in Figure 15. Sensor data would be pushed to the aggregator, which would produce current aggregates, keep recent aggregates (relevant to the rules) and store older aggregates for rule discovery and evaluation. Aggregator is the component that aggregates sensor measurements (calculates for example MIN, MAX, AVG, STD, SUM, CNT) for a certain time window and sensor. Such aggregates are time-wise aligned and can be then used for rules without any pre-processing.

Rules, generated by an expert user, are kept in separate store. Rule is triggered for evaluation when a particular measurement is pushed into the aggregator (which is taken care by Optimizer).

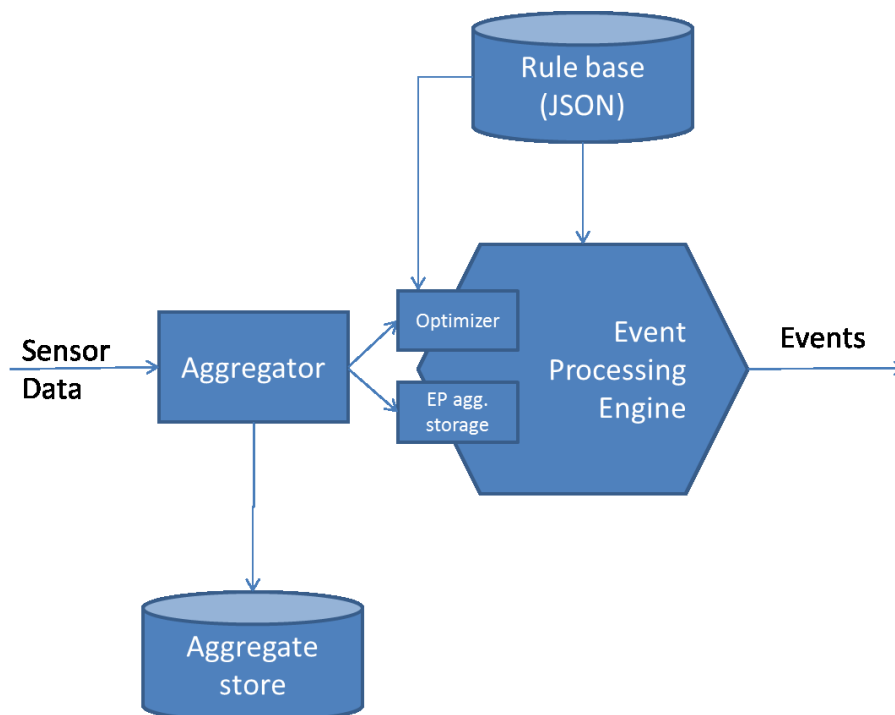


Figure 15. Event Processing Engine within the EnStream component.

EPS outputs would be messages with timestamp and location (geo-location or some other unique id of a position), type and description of an alert or warning detected.

4.3.1.1 Interfaces

Graphical User Interface: For input of the rules a dedicated GUI that will satisfy the requirements from parent section needs to be developed.

Subscription to events via REST interface

```
http://{server.url}/{xml|json}/events-subscribe
```

```
Description: data sink parent
```

```
Parameters:
```

```
id - event type id (0 for all)
```

```
POST: no-op
```

```
GET: returns success report on subscription
```

```
* Accept: text/xml or application/json
```

```
PUT: no-op
```

```
DELETE: no-op
```

4.3.2 Prediction Service

Prediction Services are to be built on top of the EnStreamM data layer. One of the standard operations on streaming data is prediction, based on the current and past data, in our case coming from the sensor networks. Prediction scenarios can range from prediction of upcoming measurement (e.g. temperatures for tomorrow based on current), prediction of some other measurable phenomenon, not acquired directly from sensor network (e.g. temperatures for tomorrow, based on a general weather prognosis), to prediction of events (e.g. predicting failure likelihood based on sensor measurements [17]).

In general, prediction scenarios can be formulated as a standard machine learning classification (for events) or regression (for numeric values) tasks [18]. In order to achieve this, we need to define what are the training examples, their attributes, and their target variable. The target variable can be, in the case of classification, a binary value encoding whether the event happened, or, in the case of regression, a measurement value.

4.3.2.1 Interfaces

```
http://{server.url}/{xml|json}/sensor-prediction
```

```
Description: data sink parent
```

```
Parameters:
```

```
id - sensor id
```

```
ts - timespan (eg. in seconds)
```

```
POST: no-op
```

```
GET: returns prediction for certain sensor for certain timespan
```

```
* Accept: text/xml or application/json
```

```
PUT: no-op
```

```
DELETE: no-op
```

http://{server.url}/{xml|json}/event-prediction

Description: data sink parent

Parameters:

id - event type id

POST: no-op

GET: returns prediction for certain event type on latest data

* Accept: text/xml or application/json

PUT: no-op

DELETE: no-op

5 Conclusions

This deliverable has presented the specification of the early NRG4CAST toolkit architecture which follows a multi-tiered Web Service approach. The need for the NRG4CAST toolkit architecture arises from the main objectives of the NRG4CAST project: (1) to provide monitoring and forecasting as services with specific APIs to external systems; (2) to federate and integrate heterogeneous data resources; (3) to enable the *semantic* representation and querying of the data resources.

The infrastructural backbone of the architecture comprises of the following tiers: *Resource*, *Data Access*, *Integration*, *Application* and *Presentation*. The functionality of Data Access, Integration and Application Tiers is provided through Web Services.

Data Access Tier Services: (1) the *Data Request Execution Service* which submits data-related workflows to the Data Access and Integration (OGSA-DAI) middleware which federates the data resources, (2) the *Data Source Service* which provides clients with access to streamable data resources, (3) the *Data Sink Service* which provides to clients the functionality to push data to the middleware server, (4) the *Data Resource Information Service* which is consumed by clients to query metadata of a data resource and (5) the *Request Management Service* which enables clients to query the execution status of their requests and to retrieve data from asynchronous requests.

Integration Tier Services:

Data Consolidation Services: (1) the *Query Relational Resource Service* which permits clients to submit queries to relational data resources, (2) the *Query File Resource Service* which permits clients to submit queries to flat file data resources, (3) the *Server Information Service* which obtains metadata of the middleware server and (4) the *Subscription Notification Service* which enables a client to subscribe to specific notifications when an event happens.

Ontology Based Integration Services: the *Integration Service* which provides clients with the ability to submit queries on the data resources over their ontological mapping.

Application Tier Services: (1) the *Event Processing (Alerting) Service* which triggers alerts and warnings based on different rules and (2) the *Prediction Service* which predicts upcoming measurements of the sensors, other measurable phenomena such as weather forecasting and events.

The development of the Web Service implementations according to the description of their interfaces as it is provided in Section 4 is likely to lead to a refinement and a revision of the service interfaces. Updated versions of the toolkit architecture will be provided by taking into account the feedback from the further elaborated user requirements of the pilot cases, especially with respect to the revision of the provided Web Service operations, in deliverables D1.4 and D6.1. Those deliverables will also provide the concrete details of the Subscription & Notification Service.

References

- [1] H. Schuldt, "Multi-Tier Architecture", Encyclopedia of Database Systems, pp. 1862-1865, Springer, 2009.
- [2] C. Pautasso, O. Zimmermann, F. Leymann, "RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision", in Proceedings of the 17th international conference on World Wide Web, pp. 805-814. ACM, 2008.
- [3] Code Project, "N-Tier Architecture and Tips", available at: <http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips> (last accessed at 15/4/2013).
- [4] OGSA-DAI Documentation, available at: <http://sourceforge.net/apps/trac/ogsa-dai/wiki/UserDocumentation> (Last accessed at 16/4/2013).
- [5] G. Donvito, "Comparative evaluation of tools providing access to different types of data resources exposed on the Grid", available at <http://indico.cern.ch/getFile.py/access?contribId=71&sessionId=43&resId=1&materialId=slides&confId=18714> (Last accessed at 27/4/2013).
- [6] OGSA-DAI Applications, available at <http://www.ogsadai.org.uk/applications/> (Last accessed at 16/4/2013).
- [7] Global Earth Observation Grid project, available at: <http://www.geogrid.org/en/first.html> (Last accessed at 26/4/2013).
- [8] Mobile Environmental Sensing System Across a Grid Environment project, available at: <http://www.commsp.ee.ic.ac.uk/~wiser/message/> (Last accessed at 26/4/2013).
- [9] Linked Environments for Atmospheric Discovery project, available at <http://d2i.indiana.edu/leadII-home> (Last accessed at 26/4/2013).
- [10] Lucene Java Documentation, available at: http://lucene.apache.org/core/old_versioned_docs/versions/2_9_1/index.pdf (Last accessed at 26/4/2013).
- [11] SemSorGrid4Env Architecture – Phase II Deliverable D1.3v2, available at <http://www.semsorgrid4env.eu/files/deliverables/wp1/D1.3v2.pdf> (Last accessed at 16/4/2013).
- [12] R. Grønmo, D. Roman, A. Llaves, P. Maué, "MaaS Composition Portal – Architecture specification", FP7 ENVISION project consortium, 2010.
- [13] M. Fasbinder, "BPEL or ESB: Which should you use?", IBM white paper, available at: http://www.ibm.com/developerworks/websphere/library/techarticles/0803_fasbinder2/0803_fasbinder2.html (Last accessed at 26/4/2013).
- [14] F. Priyatna, C.B. Aranda, and O. Corcho, "Applying SPARQL-DQP for federated SPARQL querying over Google Fusion Tables", to appear in the Proceedings of the 10th European Semantic Web Conference, Montpellier, 2013.
- [15] Web Services Base Notification 1.3 Standard, OASIS, available at http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf (Last accessed at 18/4/2013).
- [16] J. Škrbec et al., "Report & library on existing technology and data", FP7 NRG4CAST project consortium, 2013.
- [17] A. Sheth, K. Tejaswi, P. Mehta, C. Parekh, R. Bansal, S. Merchant, T. Singh, U. B. Desai, C. A. Thekkath, and K. Toyama, "SenSlide: a sensor network based landslide prediction system.," in *Proceedings of the 3rd international conference on Embedded networked sensor systems (SenSys '05)*. ACM, New York, NY, USA, 2005.

-
- [18] J. Shawe-Taylor, N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge University Press, 2004.