Project Number 260041
SUPPORTING ACTION

EnRiMa

Energy Efficiency and Risk Management in Public Buildings

Deliverable 5.1: Draft specifications for services and tools

Start date of the project: October 1, 2010

Duration: 42 months

Organisation name of lead contractor for this deliverable: SU

Revision: 20, final public, June 28, 2012

Project funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	X
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)	

Contents

List of figures 4
List of acronyms 5
Executive summary 6
1 Introduction 7
1.1 Architectural views 7
1.2 Architecture requirements and views 8
1.2.1 Functional business requirements 8
1.2.2 Non-functional business requirements 9
2 Part I - System architecture specification 10
2.1 Module view 10
2.1.1 Overview of modules 10
2.1.2 User interface 12
2.1.2.1 User interface module 12
2.1.2.2 Form framework 13
2.1.2.3 Graph tool 14
2.1.3 Engine 15
2.1.3.1 DSS Kernel 15
2.1.3.2 Solver manager 19
2.1.3.3 Scenario generation tool 20
2.1.3.4 Data wrapper 20
2.1.3.5 Database 21
2.2 Information view 21
2.3 Workflow view 24
2.3.1 Stakeholder and user types 24
2.3.2 Sequence of activities 25
3 Part II - System technical design specification 27
3.1 Communication / protocol view 27
3.1.1 Web browser to user interface 27
3.1.2 User interface to DSS Kernel 28
3.1.3 DSS Kernel to solver manager and scenario generation tool 28
3.1.4 Data wrapper to external data sources 29
3.2 Dynamic view 29
3.2.1 Use case “Scenario generation” 29
3.2.2 Use case “Strategic planning” 30
3.2.3 Use case “Data provision for strategic planning” 32

3.2.3.1 Alternative 1 – Automatic upload via an external source 32
3.2.3.2 Alternative 2 – Manual upload of data via the user interface 33
3.3 Deployment view 35
3.3.1 Deployment for development and internal tests 35
3.3.2 Deployment for test sites 35
3.3.3 Possible deployments for production 36
3.4 Validating the architecture by prototypes 37
3.4.1 User interface and kernel communication prototype 37
3.4.2 Data wrapper architectural prototype 39
3.4.3 User interface, graph tool and framework architectural prototypes 40
3.4.4 Solver manager and scenario generation architectural prototypes 42
4 Conclusion 43
Acknowledgements 44
References 45

List of figures

Figure 2-1 Overview of the architecture 10
Figure 2-2 Overview of the user interface architecture 12
Figure 2-3 Overview of DSS Engine architecture 15
Figure 2-4 The DSS Kernel components 16
Figure 2-5 DSS Kernel layers 17
Figure 2-6 DSS Kernel web services 17
Figure 2-7 Examples of technology for the kernel clients 18
Figure 2-8: Information model 23
Figure 2-9: Generic sequence of activities for using EnRiMa DSS 25
Figure 2-10: Screen prototype for displaying results of running the operational module 26
Figure 3-1: Overview of protocols 27
Figure 3-2: Scenario generation 30
Figure 3-3: Strategic planning 32
Figure 3-4: Automatic upload from BMS 33
Figure 3-5: Manual upload of data 34
Figure 3-6: Excerpt of the WSDL used for the communication test 38
Figure 3-7: The communication test, running in the Eclipse development environment 39
Figure 3-8: Excerpt from the file showing weather information from the site 39
Figure 3-9: External data from site, shown in a Sankey diagram 40
Figure 3-10: Form framework and graph tool 41

List of acronyms

AJAX	Asynchronous JavaScript and XML
API	Application Programing Interface
BMS	Building Management System
CSS	Cascading Style Sheets
DoW	Description of Work
DSS	Decision Support System
DTO	Data Transfer Object
FTP	File Transfer Protocol
GAMS	General Algebraic Modeling System
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hyper Text Transfer Protocol
JAXB	Java Architecture for XML Binding
JAX-WS	Java Architecture for Web Services
OXM	Object-to-XML Mapping
SMS	Symbolic Model Specification
UML	Unified Modelling Language
UML	Unified Modelling Language
WS	Web Service
WSDL	Web Service Description Language
XML	Extensible Markup Language
XSD	XML Schema Description

Executive summary

This deliverable specifies the architecture for services and tools to be used within the EnRiMa project in the construction of the EnRiMa Decision Support System (DSS). The architecture describes the main software modules of the DSS and their relations. Furthermore, the type of services provided by each module is specified as well as the tools to be used for their construction. This deliverable consists of two main parts:

Part I, *the system architecture specification*, describes the main modules of the system, their responsibilities within the architecture as well as their inbound and outbound information flows. From a user perspective the system will be used as a highly interactive web application. To ensure an interactive user experience the Vaadin open source web application framework has been selected as a tool for development. The user interface will allow the integration of graph tools, such as Google Visualizations and other tools for displaying information. This allows creating a rich user interface by combing existing tools in a mash-up way. The overall information integrity and optimization algorithms will be handled by the DSS engine, hosted on an open source Apache server. The engine will make use of a solver manager and scenario generation tool for the optimization tasks. The existing building ICT infrastructure will be handled by a data wrapper module. Part I also contains a refined information model and a workflow description.

Part II, *the system technical design specification*, details how the system modules will interact in order to fulfil the business requirements. The interactions are described both in the form of selection of communication protocols for the interconnection of modules and step-by-step descriptions of the types of interactions that the system will handle. The most prominent protocol used in the architecture will be XML based web services, allowing the main modules to interact in a loosely coupled fashion. The architecture was validated using a set of architectural prototypes, each prototype addressing a particular aspect of the architecture design.

The architecture as presented in this deliverable is designed to meet the requirements as defined in deliverable D4.1, Requirement Analysis. The architecture will be further refined as the project progresses.

1 Introduction

The purpose of deliverable is to provide an overview of the software architecture of the EnRiMa decision support system. Based on this document the architecture will be refined and implemented in the final EnRiMa DSS. In this deliverable *software architecture* is referred to as the general structure of the software system, as seen from a software development perspective. Thus, the architecture will serve as an important blueprint when implementing the system. Of special interest when describing software architecture is the use and combination of software modules. This document gives an overview of the software modules constituting the EnRiMa DSS, their responsibilities (i.e. what they should do) and how they should interact with other modules.

This deliverable is structured according to a set of *architectural views*, each view describing a subset of the system architecture. The architecture views are constructed so that the final system will be able to fulfill a set of requirements. In the EnRiMa project the general requirements on the DSS are described in Deliverable 4.1, Requirement Analysis (IIASA et al, 2011). D4.1 describes the requirements on the operational and strategic decisions that the system needs to support, as well as defines scenarios (use cases) outlining the use of the system. This deliverable defines a set of modules that taken together are able to support the requirement as outlined in D4.1. The relation between the architecture and system requirements is further described in Section 1.2, Architecture requirements and views.

The software development process adopted by the EnRiMa development team is highly iterative and incremental, following the principles of modern information system development approaches which embraces agile, incremental and iterative development (Ambler and Lines, 2012). This means that we do not see requirements and designs as complete and frozen. Instead, we consider the current versions of D4.1 and D5.1 merely containing the initial set of requirements that we have planned to implement for the first prototype version of the EnRiMa DSS during the validation of which new requirements and users' needs will be discovered and subsequently implemented. .

1.1 Architectural views

The architecture of a software system can be described using several different views. Each view describes a certain aspect of the architecture. The overall architecture is given by combing the views. In accordance with the DoW this deliverable is divided into two parts; a *system architecture specification* and a *system technical design specification*. Each of these parts and the constituent architectural views are described below.

Part I – System architecture specification

The system architecture specification describes the main modules of the system, the information structures that are to be handled and gives an overview of the activities that the user performs when interacting with the system. For the purpose of describing the software architecture the following views are described in Part I of this deliverable:

- *Module view* – Describes the logical static structure in the form of modules and their input and output, as seen from a software perspective.
- *Information view* – Describes the main information structures that the system need to handle.

- *Workflow view* – Describes the main activities that the user performs when interacting with the system.

Part II – System technical design specification

The system technical design specification describes how the modules of the system will interact in order to produce the desired result. This part of the deliverable is organized according to three views:

- *Communication/protocol view* – Describes the generic way the modules communicate, with a particular focus the used protocols
- *Dynamic view* – Describes how the modules interact.
- *Deployment view* – Describes how the modules are allocated to hardware components, that is, which computers that will run which software components.

The system technical design specification also contains a description of how the modules of the architecture were validated by the use of *architectural prototypes*.

1.2 Architecture requirements and views

An architecture design is developed to fit certain business and technology requirements. The software modules defined in the architecture will interact in order to provide solutions for each requirement. Before going into details on each of the architectural modules and views we discuss the types of requirements that influenced the design of the architecture. We divide this description into functional and non-functional business requirements. Note, that here we use "business requirements" as synonym to "organizational requirements".

1.2.1 Functional business requirements

Functional business requirements refer to requirements put on the functionality of the system. Typically the functional requirements express a goal that the user has in the form of something that the user would like to achieve with the system. The functional business requirements can be expressed as use cases or in plain text. In the case of the EnRiMa project the requirements are described by a combination of text and use cases. This description of functional business requirements is available in Deliverable 4.1, Requirement Analysis (IIASA et al., 2011). These requirements will not be repeated here; instead we provide an overview of how the architectural views relate to the requirements.

While a software architecture is designed to follow the functional requirements, it does not usually contain explicit associations depicting how a certain module addresses each specific requirement. This is due to the fact that a software architecture description, as in this deliverable, describes the main modules of the system and their relations. Adding functional requirements that can be addressed by existing modules in the architecture thus does not change the architecture. For example, the EnRiMa DSS will handle the storage of both operational and strategic symbolic model specifications by the use of the same database system module.

The architecture described in this deliverable is designed to meet the requirement as described in deliverable D4.1. To provide a link between the requirements in D4.1 and the architecture, the *dynamic architectural view* in Section 3.2 describes how use cases (from D4.1), are

realized by the architecture components of the system. Moreover the *information view* contains an updated information model that shows how the information in the system will be structured to meet the data requirements, and the storage of the users choice of objectives and parameters. This is referred to as the handling of decisions and objectives in D4.1. Integration requirements are covered by a specific module in the *module view*, the Data wrapper module. The general requirements as stated in the D4.1 will be discussed as a part of the non-functional requirements section.

1.2.2 Non-functional business requirements

A non-functional requirement refers to *how* the system provides the functionality (Gorton, 2006). Commonly, a non-functional business requirement affects the whole system and the way it is delivered or perceived by its users. In principle, non-functional requirements can be seen as quality properties of the system. For example, the. standard ISO/IEC 9126-1 defines a number of such properties for software products. The most important non-functional aspects that affected the design of the architecture are discussed below.

Modularity and extensibility. The architecture need to be flexible so that it is possible to extend it to cover new needs, for example when adding new test-sites. The architecture should support this by having clearly separated modules (see the *module view*) and by interconnecting these modules with well-known protocols (see the *communication view*).

Performance and scalability. The prototype DSS as produced during the EnRiMa project is addressing two test-sites, with a few users each, thus the current need for scalability is low. However, the architecture and the tool to be used for constructing each module (see the *module view*) is selected with consideration that it can manage at least 100 users per installation (see the *deployment view* for a description of deployment options). The architecture is designed to allow for a responsive performance. That is, a request should take less than 8 seconds to complete, this is in line with guidelines for response times for complex tasks (Shneiderman and Plaisant, 2010). However due to the nature of the DSS, optimizations can take longer (hours). Deliverable D4.1 outlines data sizes (max 250MB per site) and transactional frequencies (max 2/sec) with which the architecture is designed to comply.

Security. The system will handle data that are owned by the building managers, such as the building temperature and energy consumption. The different site data will be kept separated by the use of user authentication and authorization (this is the responsibility of the DSS Kernel module, see the *module view*). Moreover the use of well-known tools such as Vaadin for the Form framework module (see the *module view*), and the options to use secure protocols (see the *communication view*) addresses common Web application vulnerabilities.

In the following the architecture of the EnRiMa DSS will be detailed. First, the System architecture specification (Part I) describes the main modules. Secondly, the System technical design specification (Part II) describes the module interactions, and their deployment.

2 Part I - System architecture specification

In this part, the system architecture specification, the main constituents of the system in form of software modules will be described. The focus in this part is thus to describe the systems static structure, the dynamic behavior of the system will be described in Part II, system technical design specification.

2.1 Module view

2.1.1 Overview of modules

The overall architecture of the EnRiMa DSS follows the common approach of having three principal layers (Fowler, 2003): presentation, domain logic, and data source. In the Figure 2-1 these layers are realized through the user interface (presentation), DSS Engine kernel (domain logic) and database (data source). The figure illustrates the overall EnRiMa DSS architecture using the Unified Modeling Language (UML) component diagram notation (OMG, 2011). The notation uses small circles and half circles to denote application programming interfaces (APIs), while the arrows show flow of control and information. The users will use web browsers to connect to the user interface module.

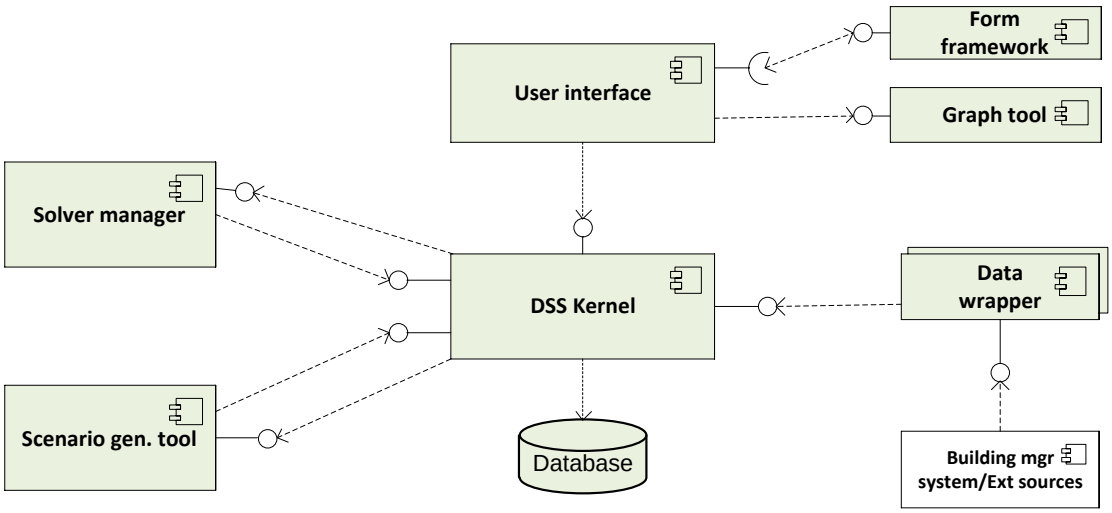


Figure 2-1 Overview of the architecture

Before describing details of each of the modules, the role and contents of each part in the architecture will be briefly summarized below.

The ***user interface*** consists of the modules responsible for the interaction with the user. The user will access the DSS via a Web browser. Central to the user interface layer is the user interface module, which will run on a Web server. The user interface module produces the graphical components as seen by the user. The user interface will consist of regular and custom-made interface elements, such as buttons, tables, tabs etc. To aid in the presentation of these elements the user interface module will make use of a *form engine* module and a *graph*

tool module. By the use of these tools it will be possible to create a composite application that combines various user interface elements.

The ***engine*** is the backbone of the DSS, providing services, each through one of the DSS modules, namely the *kernel*, *solver manager*, and *scenario generator tool*. The kernel provides the set of harmonized services that enable integration of all heterogeneous components into the DSS. The scenario generation tool provides realizations for the stochastic parameters for a scenario tree, which is then used for stochastic optimization. The solver manager provides solutions of the stochastic optimization problems. The kernel will make use of a *database* module to store information.

The DSS will interface with external data sources such as external building management systems as well as other external systems serving weather and energy price data via *data wrapper* modules.

As stated above, each layer consists of several modules. A module, as it is used here, denotes a logical software unit designed for a certain coherent task. The purpose of dividing the system into modules is to make it easier to develop and maintain. One of the goals when dividing a system into modules is that each module should be responsible for a certain well-defined area of functionality. This is called module cohesion and it should be high, meaning that each module is as single purpose as possible. At the same time module dependencies with other modules in the system should be kept to a minimum, i.e. avoiding unnecessary message pathways. This is called module coupling and it should be kept low. We have applied the design principles of low coupling and high cohesion (Ghezzi et al., 2002) to the architecture of the EnRiMa DSS.

For a description of the modules in the architecture the following template is used:

Responsibility. This describes the main idea/function of the module; this should be a cohesive set of functionality.

Input/output. The main information structures that the module works with, and its formats.

Selected technology. The tools/framework/platform chosen for implementation.

In the following sections each module will be described according to the above template.

2.1.2 User interface

The user interface layer consists of three main modules; the user interface module, form framework and the graph tool module. The central part is the user interface module that uses the form framework and the graph tool to create a graphical user interface. The three modules will all run in the same web server. Figure 2-2 illustrates, using a simplified UML component diagram, the main relationships within the user interface. Note that the user interface module is divided into view, controller, model objects and proxy parts, which will be further explained in the next section.

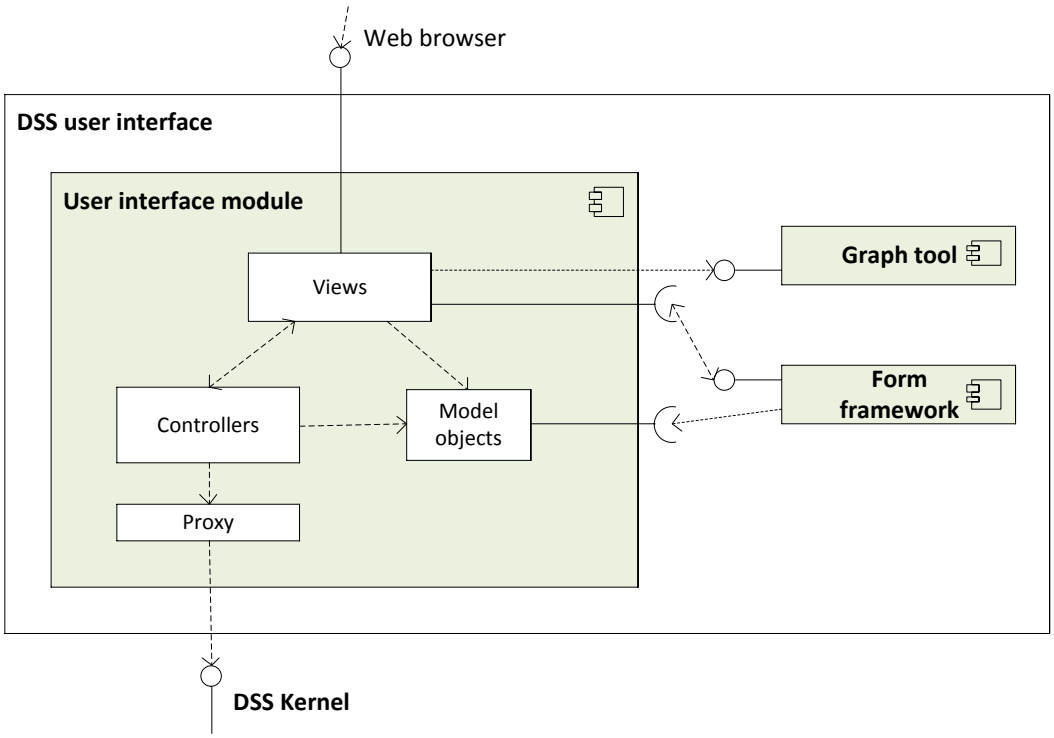


Figure 2-2 Overview of the user interface architecture

2.1.2.1 User interface module

Responsibility. The user interface module will provide a Web based user interface that lets the user view and modify data, as well as navigate between datasets. The user interface triggers the kernel when the user instructs the system to change data, delete data, run optimizations, etc. No calculations or data transformations are performed as part of the user interface module. While the persistent data are handled by the engine, the user interface module needs to handle session states. This means that user-made selections, and other screen states will be handled by the user interface until the information is sent to the engine for permanent storage.

Internally the user interface module will be structured into objects according to the object-oriented architecture patterns Model-View-Controller (MVC) (Krausner & Pope, 1988), and the Proxy pattern (Gamma et al., 1994), see Figure 2-2. The most central pattern to the design is the MVC pattern. The view (V) objects of the user interface module will deal with the graphical presentation (such as panels, tables and fields). The view objects will be making use of the *form framework* and the *graph tool* modules for the actual graphical representation of the user interface in a web browser. The form framework will be able to sent event, for example when the user clicks a button, to the view objects. The controller objects (C) will

coordinate the actions between different view objects, for example when the user switches between different screens of the application. The model (M) objects contain the data that are going to be shown to the user. By the use of a data-binding mechanism, the form framework module can retrieve the information to be displayed from the model objects. The model objects will be fetched by the controller by the use of the proxy. The proxy can retrieve the needed data from the kernel via XML web service calls and then convert it into model objects suitable for handling within the user interface module. Thus the model objects will also be used as a form of Data Transfer Objects (DTOs).

Input/output interface.

The user will access this module by using a web browser. The interface to this module will be via a web browser based user interface only. Examples of user requests includes:

- Displaying, and enabling modification of site information
- Displaying diagrams, with the use of the *graph tool* module
- Enable the user to upload of data, e.g., in the form of text files which are sent to the kernel module.

Technically the web browser will send hypertext transfer protocol (HTTP) requests to the user interface module, and the user interface module will respond with HTML, CSS and JavaScript code. The *form framework* module will produce the actual HTML and CSS sent to the user's web browser.

The user selection and modification to the data will be sent to the kernel (using web services.

Selected technology.

The user interface module will be created using the Vaadin Java framework, as described in the *form framework* module. This means that the user interface module will be implemented using the Java programming language, and deployed as a Java Servlet in a web server. While developing, the user interface module will run in the Jetty servlet environment. When deploying the user interface for access by non-developers the user interface module will be deployed in the open source Apache Tomcat web server.

To convert the user interface internal Java objects (model objects) into XML an Object-to XML Mapping tool (OXM) will be used. Furthermore, Java API for XML Web Services (JAX-WS) will be used for accessing the web services of the *kernel* module.

Java was chosen as the main programming environment for the user interface because it is well suited for creating rich internet applications with the use of the Vaadin framework. Moreover, it is easy to get access to the kernel web services via Java.

2.1.2.2 Form framework

Responsibility. The form framework is responsible for providing generic functionality that displays common user interface components, and for handling the direct input from the user. The framework contains buttons, tables, field, tabs, and other constructs that the user interface puts together to form the specific user interface of the EnRiMa DSS.

Input/output interface.

- Input: The selected framework, Vaadin, contains programming interfaces that make it possible for the user interface module to build forms/screens. These programming interfaces are accessible directly via the framework API: thus, no separate server is needed for deploying the framework.

- Output: The framework produces HTML, CSS and JavaScript code that are sent to the user's web browser for display. Moreover, in order for the user interface module to react to user actions, such as, clicking a button, the *user interface* module can get callbacks on specific Java event interfaces.

Selected technology.

The open-source Vaadin java framework will be used as form framework. Vaadin allows creating advanced, highly interactive web applications without the need for the developers to handle HTML, CSS or JavaScript. In particular, Vaadin contains a large set of pre-made user interface components that can be incorporated in a Vaadin java project. Browser compatibility is ensured because Vaadin supports the main desktop web browsers on the market; Google Chrome, Internet Explorer and Mozilla Firefox. Vaadin is also compatible with the popular mobile browsers Google Android (for Android devices) and Apple Safari (for iOS devices). Vaadin is built on top of Google Web Toolkit (GWT).

Vaadin was selected as framework because it allows the developers to focus on the main logic of the user interface, rather than manually writing HTML, CSS and JavaScript. Moreover Vaadin takes care of differences in web browsers, which could otherwise be a time consuming task.

2.1.2.3 Graph tool

Responsibility. The graph tool module will display advanced graphs in the web based user interface. Example of graphs includes stacked line charts and Sankey diagrams. The module should contain no calculations, i.e., it will be given pre-formatted data suitable for display.

Input/output interface.

- Input: The module will have an interface for each type of diagram that should be displayed. As input parameters java objects will be used. The Java objects will be either the same as used in the main user interface module (that is, its model classes) or generic objects representing arrays of data to be displayed.

- Output: The output will be presentation objects suitable for inclusion in the forms produced by the Vaadin framework as used in the *user interface* module. For example, the result of the graph module can be a Vaadin Custom Component that is easy to display when using the Vaadin framework.

Selected technology.

The graph tool will partially be built on top of products that are integrated into the DSS. The necessary configuration and extension of these products will be done within the project.

For displaying most graphs the VisualizationForVaadin add-on will be used. It is a Vaadin integrated variant of Google Visualization. The reason for choosing VisualizationsForVaadin is its integration with Vaadin, and that VisualizationForVaadin supports different graph types as needed in the project.

For displaying Sankey diagrams a number of tools have been examined. Most likely to be used are SankeyVis or Tamc Sankey tools. SankeyVis is a Sankey diagram generator written in Java, while Tamc Sankey is a JavaScript library. Depending on the project needs, there also is an option not to generate the Sankey diagrams dynamically, but to use more generic static Sankey diagrams instead. For example, in the architectural prototype for the data wrapper module a static diagram was used (see Section 3.4.2, Data wrapper architectural prototype).

2.1.3 Engine

The DSS engine has a modular structure illustrated in Figure 2-3, and is designed to make it reusable for other buildings without substantial software modifications. The engine's components are described in subsequent sections.

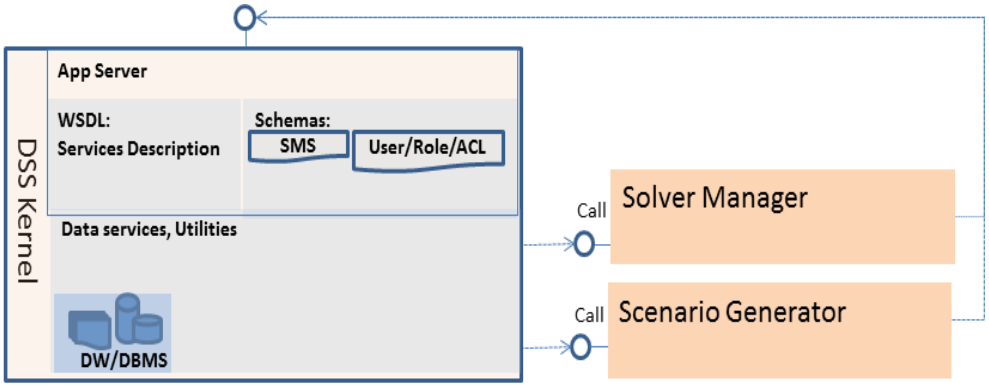


Figure 2-3 Overview of DSS Engine architecture

2.1.3.1 DSS Kernel

Responsibility: The DSS Kernel (hereafter referred to as the *kernel*) is responsible for providing functionality needed by the Web Services (WSs) specified below. Therefore, the kernel is responsible for providing state-less services requested by the DSS users through the user interface or through the *data wrapper*, *solver manager* and *scenario generation tool* modules. The kernel also includes "back-office" applications needed for actual implementation of WSs, such as, checking consistency of the provided data with the SMS, organizing the data provided as Data Transfer Objects (DTOs) into structures suitable for effective handling by the data warehouse (implemented in a DBMS), organizing data from the data warehouse into DTOs used by WSs, user handling, access control, and preparing data for diverse reports for developers and users.

The kernel, in order to process the wide scope of WSs effectively and be reusable, has modular structure illustrated in Figure 2-4. We first summarize the communication with and within the kernel. The kernel is accessed by the external components through WSs; components of the kernel developed in Java programming language communicate through Java methods, while those developed in C++ communicate through WSs. Below, we briefly summarize the function and characteristics of each component:

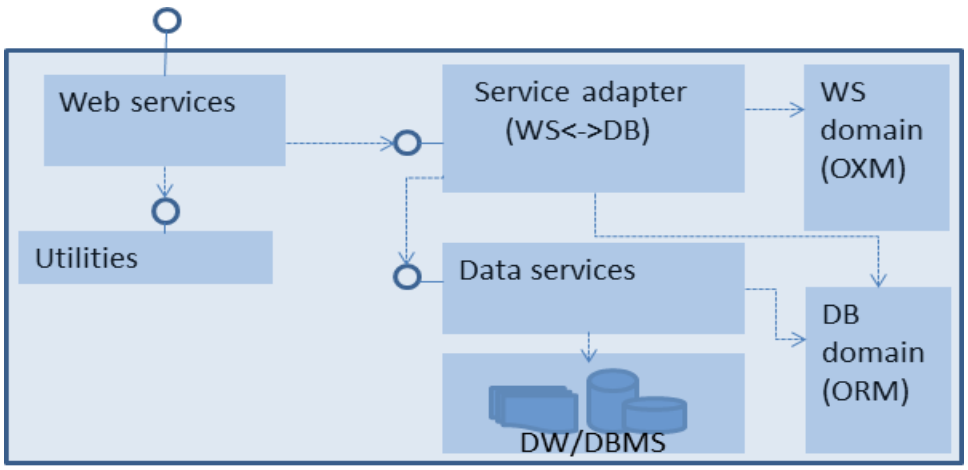


Figure 2-4 The DSS Kernel components

1) *Web services*: The web services publish the contract (as the WSDL file) through lightweight application service (Tomcat) and the endpoints implement the web-services through calling the service adapter.
2) *Service adapter*: Adapt web services with the internal data services; transform formats between web services domain objects and database domain objects; make the data service to be transparent (a black box) for other components
3) *Web services domain*: The DTOs based on the contract (WSDL file) are used for generating objects through an Object/XML mapping (OXM) tool (e.g., JAX-WS, JIBX, XStream, gSoap). The tools also generate classes for applications consuming WSs in other components (including user interface, solver, scenario generator, and wrapper on external data sources).
4) *Database domain*: The Java Persistence API (JPA) entity objects generated based on the database schema through Object-relational mapping tools (Hibernate, Eclipse link, Toplink, etc)
5) *Data services*: Transactional business logic, read/store data from/to DBMS through JPA which will handle conversion of DTOs into the Data Warehouse schema that will be independent of the DTOs thus remaining transparent for clients and stable, i.e., not requiring modifications when DTOs will be modified.
6) *Data warehouse*: It will be implemented within a DBMS, and will be accessed external (to the kernel) clients only through the kernel data services. Therefore a DBMS choice does not influence other DSS components. Currently, PostgreSQL is used for prototyping, and is planned to be used for the final version.
7) *Utilities*: This is container of diverse applications that support various functions needed by the kernel.

The multi-layered representation of the kernel is illustrated in Figure 2-5. The top layer contains the WSs published within the WS-domain in the WSDL format, and endpoints indicate a location for accessing the service. This domain is shared by all components of the EnRiMa DSS. The middle layer contains the service adapter that maps the requested operations into the domain data services. The data services layer provides stateless transactional business logic, while the data access layer provides interface to the data warehouse built on a DBMS. The data access and services layers share cross-cutting applications, e.g., for handling logging, transaction management and security.

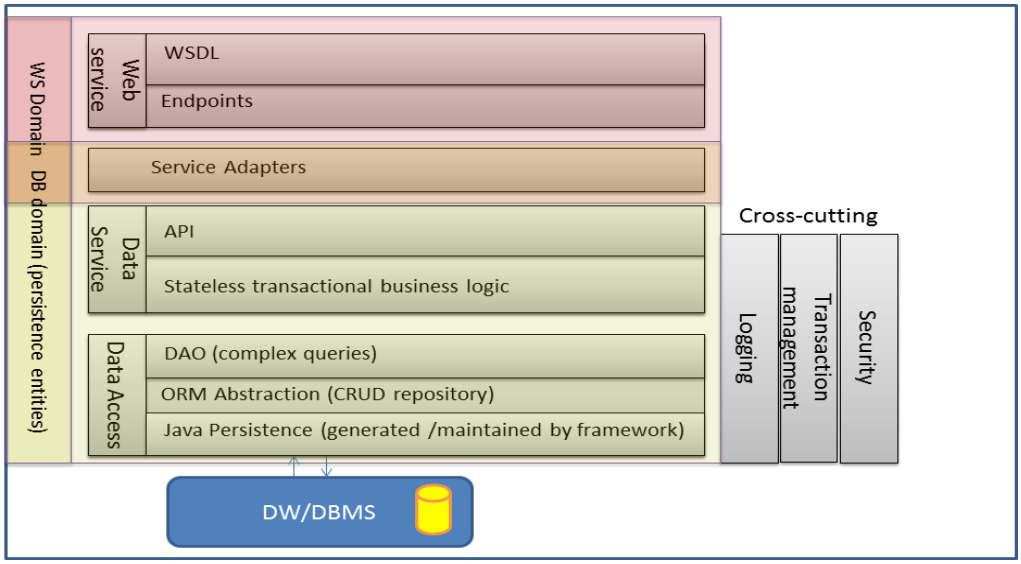


Figure 2-5 DSS Kernel layers

Figure 2-6 shows the main principles when developing and using the WSs. One starts with the requirements from clients, i.e., other DSS components (user interface, solver manager, etc.), and develops the XML schema and the WSDL file that meet the requirements. After this is completed, the clients can use the web services directly. Marshalling the WS-domain objects to XML, unmarshalling XML to WS-domain objects, and translations between WS-domain and DB-domain are totally transparent for the clients.

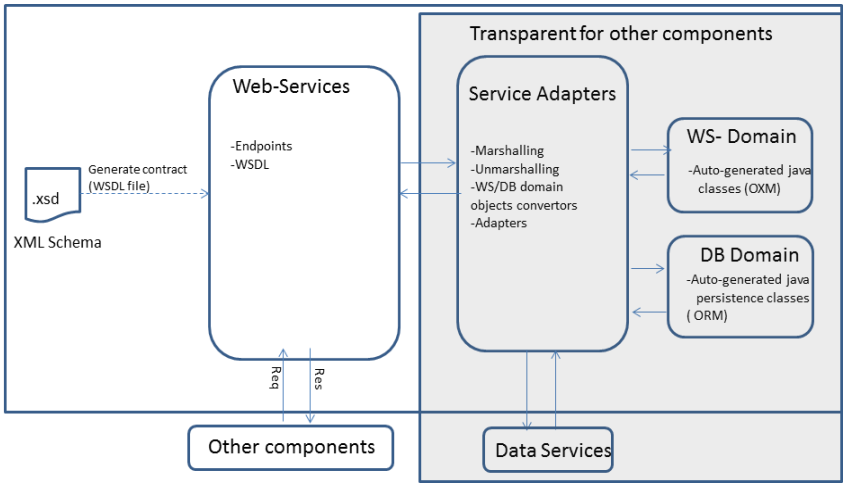


Figure 2-6 DSS Kernel web services

Moreover, there are tools for generating definitions of objects (e.g., classes) directly from the XML schema for all widely used programming languages and tools. A selection of such tools is illustrated in Figure 2-7. Therefore, using the philosophy outlined above, along with the corresponding tools reduces substantially the resources needed for the development of applications in diverse components, enables parallel development, supports consistency between these applications. Thus, it contributes to the effectiveness of the software development and to the reliability of the applications.

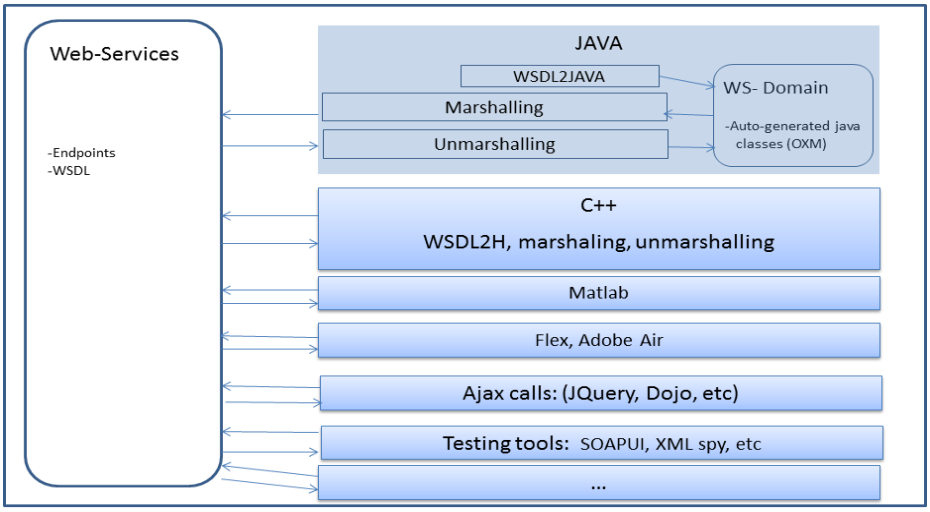


Figure 2-7 Examples of technology for the kernel clients

Input/output interface. Web Services (WS) receiving and sending messages composed of:

- Envelope containing data needed to identify the user, model id, data version, etc.
- Request (specification of the service)
- Response (specification of the response to the service)
- DTO (Data Transfer Object) containing structured data needed for processing the request or the response. DTO optionally contains specification of objects transferred as attachments for requests of storing/retrieving objects (files) that are not processed by the kernel.

The main types of the WSs handled by the DSSE are:

- Storing the provided data (parameters of the model, specification of model analysis tasks, results of model analysis, data needed for administration of the DSS)
- Checking consistency of the provided data with the SMS (Symbolic Model Specification)
- Providing the stored data to the DSS components in the requested DTOs.
- Generating scenarios for stochastic optimization
- Performing model analysis, including stochastic optimization, and possibly other methods of integrated model analysis
- Management of the DSS users, including authentication, specification of roles, access rights

Selected technology

For the implementation of the kernel the Java programming language will be used. For hosting the services Apache Tomcat open source web server will be used. A JPA-compliant framework will be used for the database access.

2.1.3.2 Solver manager

Responsibility.

The Solver Manager is responsible for providing the solution of the optimization problem to the kernel. To achieve this objective, it fetches the symbolic model specification, the generated scenarios and the model instance data from the kernel. Next, the data are prepared in the appropriate format needed by the particular solver that is used for the specific instance. Then, a call to the solver is made, passing the information it needs. Finally, the output of the solver is prepared again in order to be sent to the kernel for storage.

Input/output interface

Input:

- The symbolic model specification: symbols and descriptions for sets, variables, parameters and equations. For more details on the parameter data, see deliverable D4.2, "Symbolic Model Specification" (URJC et al., 2012).
- The model instance: parameter values, sets elements, variables and equations to use, and objective (e.g. minimize a certain variable, risk term value, etc.).
- The scenario tree: values and probabilities for stochastic parameters at each tree node. For more information on the scenario tree, see deliverable D3.2, "Scenario generation software tool" (SINTEF, 2012).
- Other configuration or user information: e.g. default solver

Output:

- The optimal values for the variables
- Further analysis, depending on the solver used, e.g. sensitivity analysis

Selected technology

The technology to be selected depends on the solvers and algorithms analysed and selected in task T4.5, which has started just before the delivery date of this deliverable (D5.1) and continues until the end of the project. Some of the needed features are:

- Capability to consume the web services provided by the kernel.
- Capability to generate different file formats (e.g. MPS, GAMS, AMPL, etc.).
- Capability to communicate with different third party software, both stand-alone solvers and optimization software.
- Specific solvers and optimization software including stochastic optimization capabilities.

Even though decisions have not been made at the time this deliverable is being written, some examples of the technologies that are being considered are:

- Matlab has been used for prototyping and examples on WP2 and WP4.
- GAMS is being used for first implementations of the models in the SMS.
- The open source R language and statistical software is being considered as an integrated framework to interface the kernel and the solvers. See http://www.R-project.org.
- The resources at the COIN-OR (COmputational INfrastructure for Operations Research) project are being explored in order to include open source solvers, as well as open standards (e.g. Optimization Services) when possible. See http://coin-or.org.

2.1.3.3 Scenario generation tool

Responsibility. The scenario generation tool provides realizations for the stochastic parameters for all nodes in a scenario tree, based on input data about the parameters with their statistical properties and about the structure of the scenario tree. An overview of the scenario generation tool is given here; it is in more detail described in deliverable D3.2, "Scenario generation software tool" (SINTEF, 2012).

Input/output interface.

Input:

The scenario generation tool will have three main inputs; module type, parameter data and scenario tree:

- Module type (operational or strategic)
- Parameter data and their statistic properties: weather data (for operational model: predicted, for strategic model: observed), electricity and district heating prices from the test sites (as available), electricity prices from exchange (for example from the European Energy Exchange, EEX) (hourly for Time-Of-Use tariffs); long-term trends of energy prices, technology development and prices, government subsidies (as needed). For more details on the parameter data, see deliverable D3.2.
- Scenario tree structure: time periods, number of stages, number of branchings.

Output:

Tables with values of the stochastic parameters for each node in the scenario tree, together with information about the tree structure (parent/predecessor node, stage and probability). This basically consist of a set of realizations of each stochastic parameter for each scenario tree node – and "directions" how to put them in the multistage stochastic model to be built.

Selected technology.

The scenario generation tool is implemented as a C++ application. The XML input and output will be parsed and generated using the Boost C++ libraries. The use of web services will be implemented through the gSOAP framework.

2.1.3.4 Data wrapper

Responsibility.

The data wrapper module is responsible for communicating with external data sources, and converting relevant data into a format suitable for storage by the kernel. The data wrapper module can be triggered by an FTP file upload from an external source (push), or being set to fetch information from external sources on a regular interval (pull). The data wrapper module can also be triggered from the user interface (e.g. by requesting an upload of the latest energy consumption data of the building) or on an automated way (e.g. FTP upload started by the BMS). This is described in the dynamic view.

Within the project it is necessary to convert input data from building information systems (BMS) (e.g. DESIGO format, CSV format), weather data from weather.com (e.g. XML, JSON, CSV), energy prices from EEX (e.g. WebAccess Eurex/Xetra). The target format of the data conversion is predefined by the XML format of the kernel or the predefined database structure.

Input/output:

Input depends on the source of data. Within the project it is necessary to convert some input data from building information systems (BMS) (e.g. DESIGO™ format, CSV format), weather data from weather.com (e.g. XML, JSON, CSV), energy prices from EEX (e.g. WebAccess Eurex/Xetra).

Output: the DTO of the WS used for uploading the data in the kernel.

Selected technology. There might exist several wrappers, depending on how specific the formats are for each external data source. If possible wrappers could be deployed at each site, thus taking care of the site-specific data formats. If this impossible or undesirable, for example when interfacing with external sources for weather forecasts, the wrapper will be deployed at the same location as the kernel. The wrapper will be developed using the Java programming language; DTOs will be generated by JAX-WS import tool based on the data management schema which published by the kernel; for organizing different data formats, the OpenCSV, Apache POI maybe needed; for calling the data management web services, Apache axis2 or JAX-WS will used. An open source too, e.g. Apache Camel can be used for assisting in format conversions.

2.1.3.5 Database

Responsibility. The database will be storing the information given by the kernel. The database will ensure consistent data by the use of mechanisms such as transaction handling.

Input/output interface.

The input/output interface of the database is a interface allowing for the querying, updates and deletes of table based data according to the structure defined in the project.

Selected technology.

Currently, the open source PostgreSQL database management system is used for prototyping, and is planned to be used for the final version. The database management system will be configured with the appropriate structure that can hold the data for the DSS. Moreover, during the project the database will be populated with data used for the two test sites.

2.2 Information view

To have an overview of the information handled by the system an *information model* can be used. The information model can then be divided into fragments that can be used in the communication between the modules of the system. For example, when displaying information about technologies in the user interface only the information model parts related to technologies will be of use.

Figure 2-8 shows the information model for the EnRiMa DSS, using a simplified UML class diagram notation. In a class diagram each rectangle represents the main information concepts used. The content of each of the rectangles lists the attributes/properties of each concept, while the lines in the information model indicate which concepts that are related. To make it simpler to map the information model concepts and attributes to the symbolic model specification the letters used for parameters and indices in the model specification are included in the information model.

As can be seen in the information model, the concepts of energy markets (EnergyMarket in Figure 2-8), technologies (Technology) and energy types (EnergyType) are central concepts. While some relations are shared between different types of technologies, we can also note that

the energy creating technologies (ECreatingTech) and energy absorbing technologies (EAbsorbingTech) need to be described using separate attributes and relations. The set of concepts listed on the left in Figure 2-8 are concepts related to building information, such as the volume and the area of glass. Moreover, the general input and output of the optimization to be performed are shown in the upper respective the lower part of the information model.

The information model as included in this deliverable is based on the deliverables defining the symbolic model specification, D4.2 (URJC et al., 2012) and D2.2 (UCL et al., 2012). During further design this model need to be extended. For example, information that makes it easier for the user to handle the information, such as names of buildings and technologies need to be added. As the symbolic model evolves, so will this model, and the message structures sent between the modules of the system.

Input: Paramenters & indices

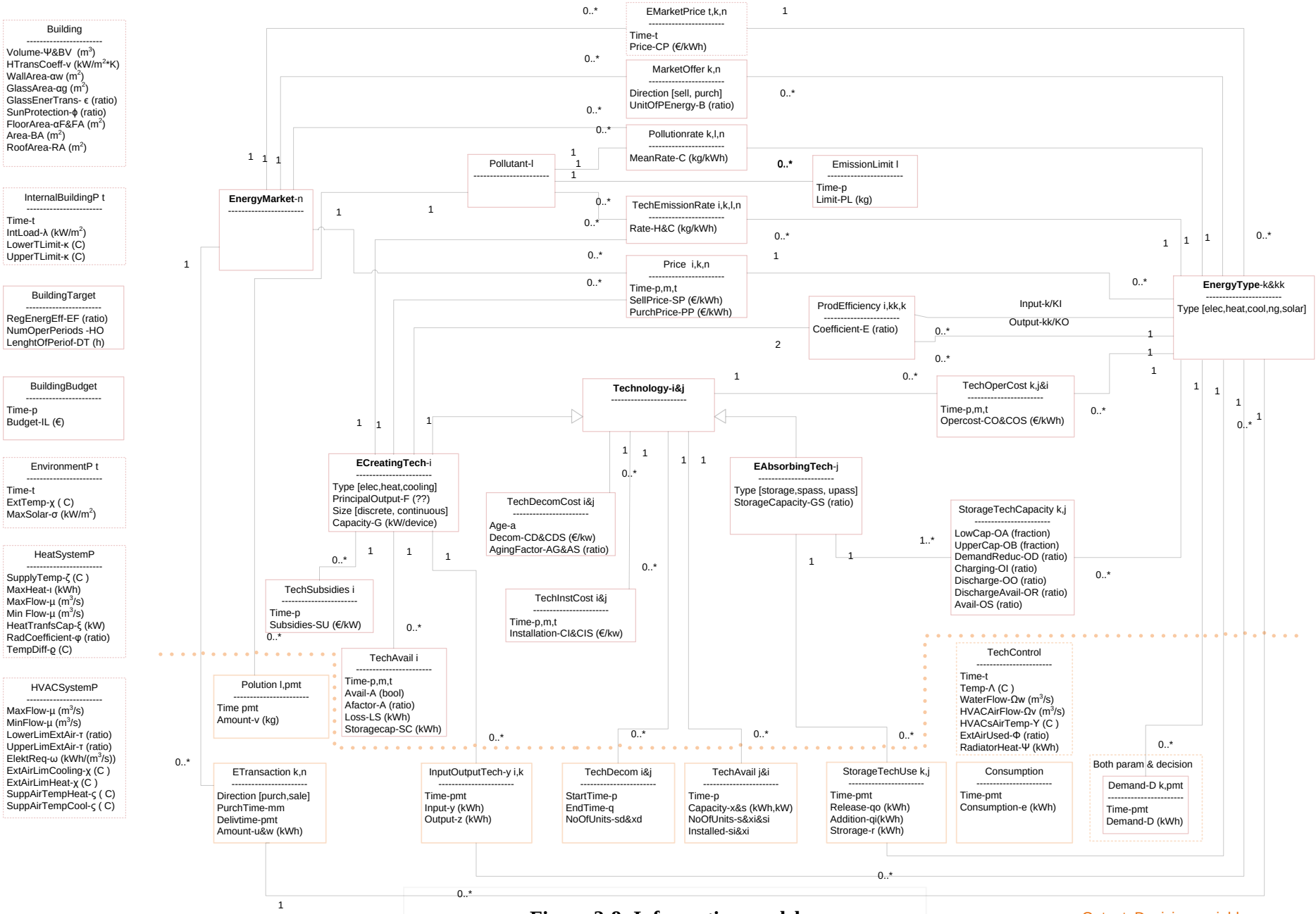


Figure 2-8: Information model

Output: Decision variables

2.3 Workflow view

The workflow view describes the types of users that the architecture should serve and a generic workflow that defines the main activities the system should support. The purpose with the workflow view is to give an overview of the work context in which the system will be used.

2.3.1 Stakeholder and user types

Deliverable D4.1 identified the following organizational stakeholder types: building owner's financial manager, building owner's operations manager, outsourced maintenance manager, energy service company, utility, energy consultants, policy makers, and energy auditors. Many of these roles will use the same functionality of the DSS but for different purposes. This is similar to what was described in the module view; some requirements will be handled by the architecture in a similar way. To simplify the description of stakeholder types we have defined a set of user types that have the same access needs to the system. Below is a list with the main types of users that need to be supported with specific functionality and user interface components. In comparison to D4.1, we have added the roles of system administrator and product provider to cover the users involved in system installation and maintenance.

User types of the system:

- System administrator – configuring the system, importing data from sites, weather and pricing, as well as managing users
- EnRiMa product provider, such as the DSS developers – installing and configuring the system according to customer requirements (building set up, external data sources, user data).
- External users, such as energy auditors, consultants, policy makers and technology providers – running operational and strategic optimizations, viewing past optimization results and other historical data, possibly restricted to a specific subset of technologies and or building configurations.
- Building managers, such as building owners operations manager, – running operational and strategic optimizations, viewing past optimization results and other historical data.
- Building operators, such as building owners operations manager and the maintenance manager, – running the operational optimizations, viewing past optimization results and other historical data.

Other stakeholder types such as, controllers, analysts, and politicians aiming to use the DSS for seeking advice regarding, for instance, subsidies and CO_2 reduction measures, should assume one of the user types listed above.

We envision that not all of the data providers will not initiate communication with the EnRiMa DSS from outside. Instead the EnRiMa DSS will check for updated data at predefined times. Use cases for the external data management are further described in the *module view* and *dynamic view*.

2.3.2 Sequence of activities

A generic sequence of activities for users performing operational and strategic optimization tasks is:

- Setting up the data for building, the environment, specifying internal load, budget limitations, etc. Viewing building energy flows in the form of Sankey Diagrams.
- Setting the optimization goal (e.g. reduce CO_2 emissions, reduce costs), target temperature interval
- View summary of optimization settings and start optimization
- View results, save and/or export to a predefined file format if desired.

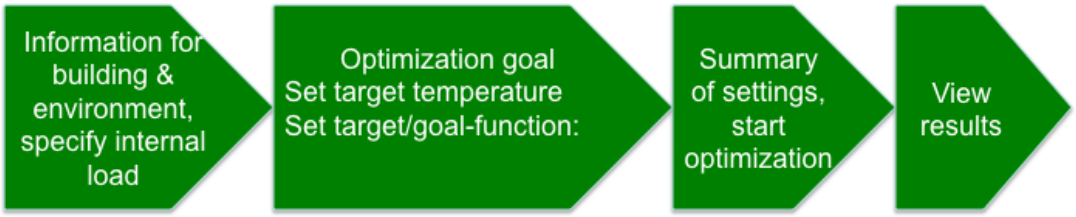


Figure 2-9: Generic sequence of activities for using EnRiMa DSS

The generic workflow will be customised in terms of input data types (defined in the information model), and output layouts for each site defined during the system installation and configuration phase. Each step in the process will be supported by a user interface screen, se e.g. figure 2-10 showing a prototype screen containing results from running the operational module.

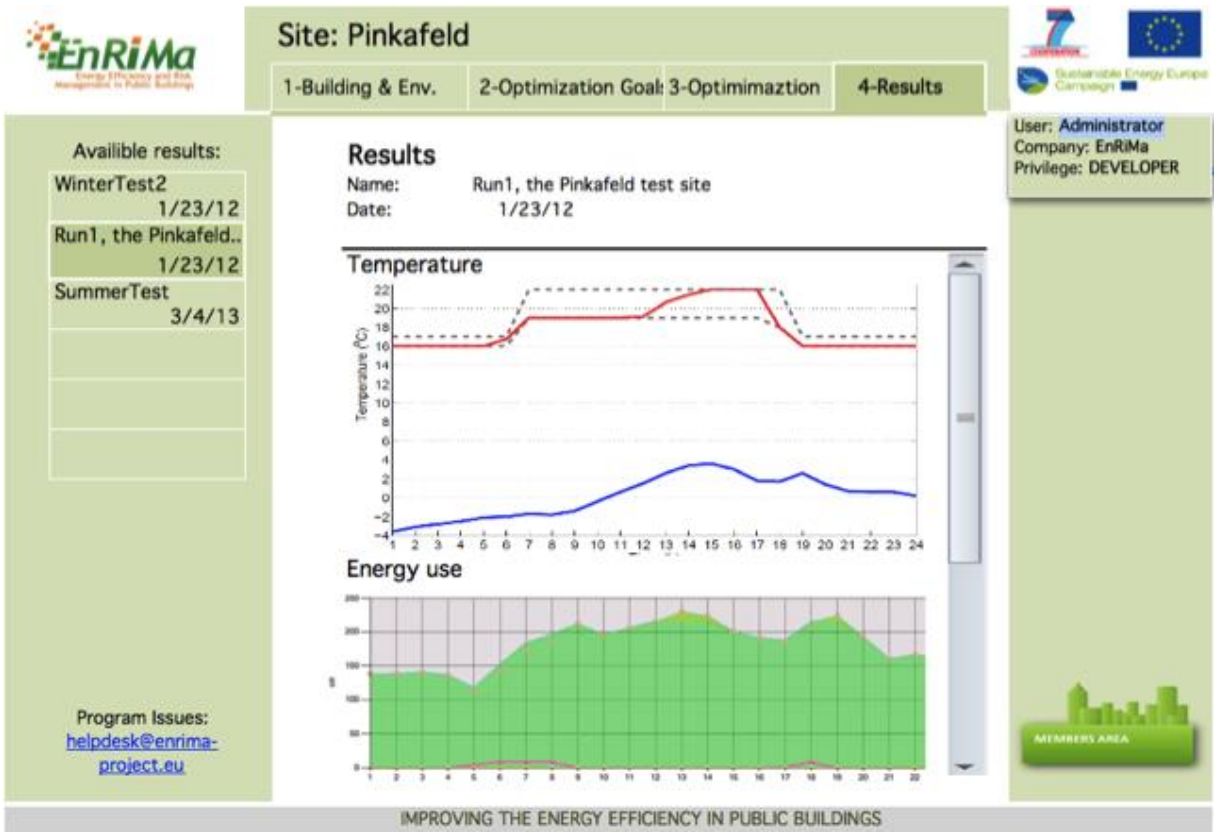


Figure 2-10: Screen prototype for displaying results of running the operational module

In addition to the optimization functionality supported by the workflow, the EnRiMa DSS should also be customizable by experts, for example energy consultants. Examples of customizations are creating a new way of representing optimization results, and extending the DSS with new analysis models and services. These activities will be performed independently of the optimization workflow. Currently we foresee the following activities:

- Upload/define/save historical data
- Upload/define new equipment details (e.g. in an predefined XML format) supplied by an equipment manufacturer
- Configure and/or extend available analysis models (e.g. cost or CO_2 minimization)
- Create a new analysis model (e.g. NOx minimization)
- Change/extend available parameter definitions (e.g. weather details). This could potentially require customizing the data import module.
- Set-up and configure a new site
- Find the best medium-term/long-term investment plan. This could require running the strategic model independently and making a comparison of results.

These tasks require more detail control over the DSS than what the user interface of the DSS can provide. These tasks will instead be performed by using suitable development tools, for example using the DBMS data import tools to import large sets of historical data into the database, or using a development environment to change the logic of the solver manager and/or user interface modules.

3 Part II - System technical design specification

In this part, the system technical design specification, the modules interactions, their deployment and the validation of the architecture will be described.

3.1 Communication / protocol view

The communication / protocols view describes the protocols used when communicating between the modules in the architecture. The following UML component diagram gives an overview of the protocols that will be utilizes by the DSS:

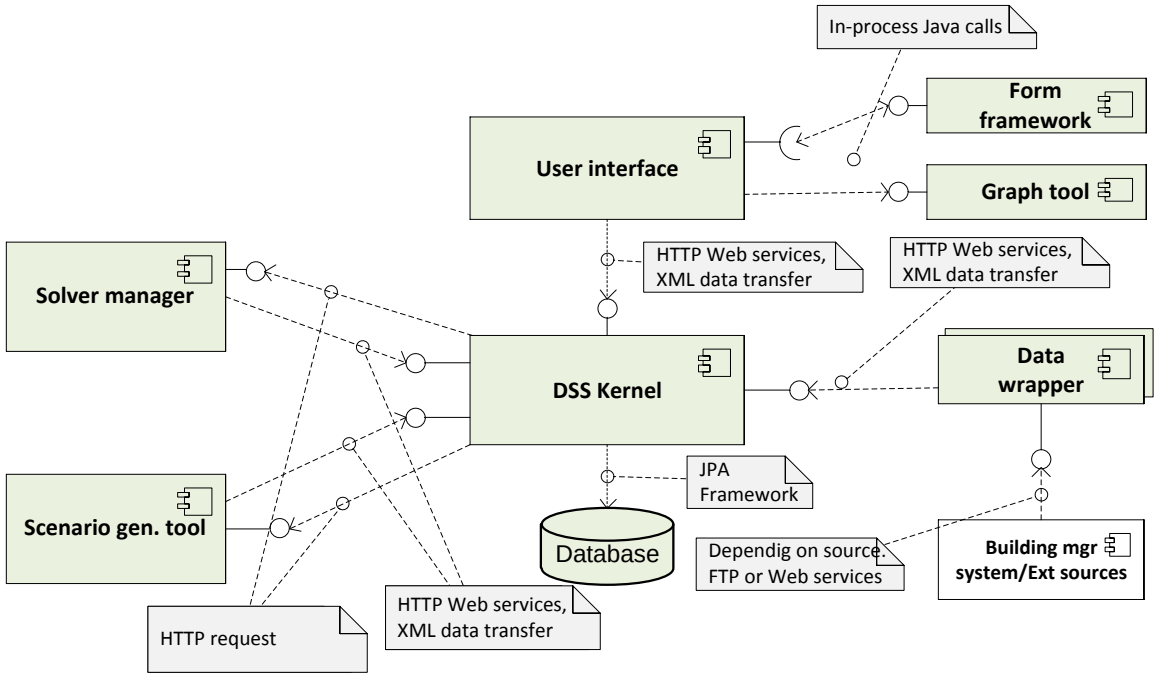


Figure 3-1: Overview of protocols

In the following subsections the use of the protocols will be described.

3.1.1 Web browser to user interface

The user interface module will create HTML, CSS and JavaScript that is executed in the browser. When executing in the browser the application will send request to the server via the Vaadin User Interface Description Language (UIDL). The UIDL is transferred on top of HTTP using JSON structures, employing the same principles as AJAX applications.

The UIDL protocol is designed in such a way that only changes to the interface are sent to the browser, which minimizes the information that need to be transferred. However the application will be interactive, i.e. for each interaction performed by the user (for example clicking a button) a request will be sent to the server. To keep the application responsive a normal latency (below 200ms) is preferred.

When needed the HTTPS secure protocol will be used when communicating from the browser to the user interface server.

The choice of protocol for the web browser to user interface server is based on the need for having an interactive application, rather than a web page. The choice of the Vaadin framework for the user interface constructions also mandates the use of the protocols, since they are built into the Vaadin framework. The Vaadin framework contains logic that handles data transfer security threats, such as cross scripting.

3.1.2 User interface to DSS Kernel

The user interface server will connect to the engine via XML web services on top of the HTTP protocol. The interface provided by the kernel will be based on WSDL defining defines a set of methods that the user interface server can call. Data passed to/from the kernel will be in XML format, as defined in an XML Schema. The XML format supports the inclusion of binary data, for example by using base 64 encoding.

The user interface server will use the XML structures to convert to/from java objects. This means that from the user interface server side the web services will the used as a mechanism to serialize java object structures. The kernel will be able to deliver complex non-cyclic object structures to the client side via this mechanism.

While it is possible to use the secure HTTPS protocol for the communication, another option is to deploy the user interface server and the kernel in the same machine, thereby addressing security threats to the communication. A description of deployment options can be found in the *deployment view*.

The decision to use web services as the protocol for the user interface to kernel interaction are based on the desire to create an flexible modular solution where the user interface implementation is independent of the implementation technology used in the kernel.

The proposed use of the communication protocol has been validated in a prototype, see Section 3.4.1, User interface and kernel .

3.1.3 DSS Kernel to solver manager and scenario generation tool

The interfaces between the kernel and the solver manager and scenario generation tool will make use of the same protocols, therefore they are described together here.

Both the scenario generator and the solver manager will be implemented as standalone tools that can be started via a HTTP request. This means that the interface of these two modules will consist of a few parameters passed via the HTTP request.

To fetch the required data (scenario tree respective a model instance) the modules will reach the kernel by a sequence of XML based web services calls via HTTP/HTTPS. The web service interface of the kernel will expose methods that the modules can call in order to retrieve data.

The decision that the solver manager and scenario generator will fetch data from the kernel by the use of web services, rather than the other way around, is based on that the modules need to fetch data that are appropriate for the task to be computed. Moreover, the use of simple HTTP request to start the tools eliminates the need for the modules to implement more complex interfaces, like web services. However the possibility of using other protocols (such as RPC) for starting both the solver manager and the scenario tool will be investigated further during the implementation.

3.1.4 Data wrapper to external data sources

There will be several different external data sources that the DSS needs to communicate with. This means that several different data wrappers need to be developed, each addressing the specific protocols and format as handled by each data source. To describe the general structure of the data wrappers we here describe the options for *triggering* and *formats*.

Triggering. The data wrapper module can be triggered by an external source (push), or being set to fetch information from external sources on a regular interval (pull). The data wrapper modules can also be triggered from the user interface (e.g. by requesting an upload of the latest energy consumption data of the building) or on an automated way (e.g. FTP upload started by the BMS). Use cases showing how this is handled are described in the dynamic view.

Formats. Within the project it is necessary to convert some input data from building information systems (BMS) (e.g. DESIGO format, CSV format), weather data from weather.com (e.g. XML, JSON, CSV), energy prices from EEX (e.g. WebAccess Eurex/Xetra). The target format of the data conversion is predefined by the XML format of the kernel or the predefined database structure.

Note that the data wrappers can be deployed at the sources locations or at the same location as the engine. For example, the architectural prototype that was done to test the integration capabilities deployed a small data wrapper at a site to collect data from the building management system.

3.2 Dynamic view

To exemplify the relationship of the modules this section includes three scenarios. Each scenario corresponds to a use case in deliverable D4.1, thus they are mapped to the business requirements. The three scenarios are chosen to represent interaction patterns among the modules:

- Use case “Data provision for strategic planning”: Involves user interface, kernel and data wrapper module interaction.
- Use case “Scenario generation”: Involves user interface, kernel, scenario generation tool interaction.
- Use case “Strategic planning”: Involves user interface, kernel and solver manager interaction.

In this deliverable the use cases are used to highlight the type of interaction that the system architecture needs to support. More use cases can be found in deliverable D4.1.

3.2.1 Use case “Scenario generation”

(Use case 9.4.3. in deliverable D4.1, page 72)

The use case “Scenario generation” is here extended to provide a more detailed description of how the modules in the DSS will interact. The use case involves the user interface, kernel, and the scenario generation tool modules. The following steps are included:

1) The user starts an action that requires a scenario tree to be generated. The user interface module sends a triggering request to the kernel web services. The request contains an

identifier that can be used by the kernel to fetch the data to be used by the scenario generator.

2) The kernel reads the necessary symbolic model specification and the parameters needed, including the scenario tree structure, from the database. For example the parameters can include weather data and electricity prices. For a description of parameters, please refer to the description of the scenario generation tool module.
3) The scenario generator tool is started by the kernel. The engine passes identifiers that identify the needed parameters.
4) The scenario generator fetches the stored parameters and scenario tree structure from the engine.
5) The scenario generator generates values for the nodes in the tree.
6) When the generation of the scenario tree is done, the scenario generator forwards the result to the engine. The result consists of values for each node in the scenario tree. Please see the description of the scenario generation tool module for a full description of the result.
7) The kernel stores the information in the database for further use, for example when it is needed for strategic planning (See use case “Strategic Planning”).

The following communication diagram shows the interaction among the modules.

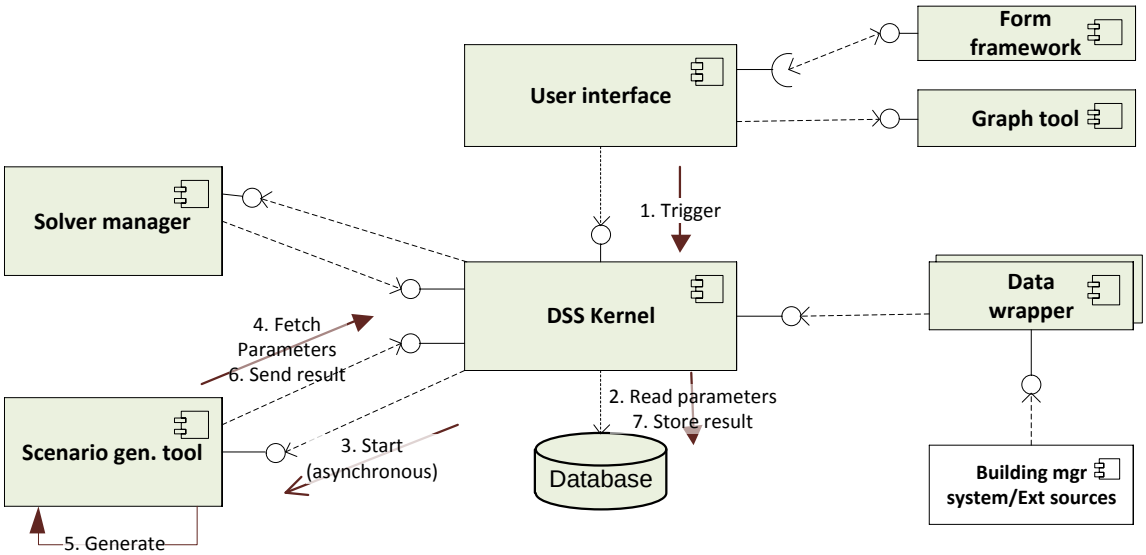


Figure 3-2: Scenario generation

3.2.2 Use case “Strategic planning”

(Use case 9.3.2 in D4.1, page 70)

This use case describes how the user starts an analysis and views the results. Involved main modules are user interface, kernel and solver manager. Compared to the use case description in deliverable D4.1 the use case here is extended to include more details on how the modules interact. The use case contains the following steps:

Steps 1-6, setting parameters:

1) The user requests the user interface to review parameters. For example, the user can review the building information.
2) The user interface fetches the needed subset of parameters from the kernel, via the kernel web services.
3) The kernel gets the desired parameters from the database.
4) The user changes the parameters, for example the desired temperature intervals. The user also selects the type of analysis to be performed, for example an operational or strategic analysis, and the parameters of this analysis that represent his/her preferences.
5) The user interface sends the changed values to the kernel, thought the engine web services.
6) The kernel stores the values in the database.

Steps 7-16, running the analysis:

7) User triggers the optimization.
8) The user interface requests an optimization to be started by forwarding identifiers of the selected parameters, and identifiers (such as name and date) for the analysis to be performed to the engine.
9) The kernel prepares an instance of the model to be analyzed by reading and parsing;
    a. Reading the symbolic model specification
    b. Reading the parameter values for the models, such as weather forecasts and desired temperature intervals.
    c. Reading the prepared scenario tree specifications, and associated stochastic parameters (see results from use case "Scenario generation").
10) The kernel starts the solver manager. The engine passes identifiers that identify the needed data set.
11) The solver manager fetches the model instance from the kernel.
12) The solver manager starts the optimization. This can take a couple of minutes, or hours depending on the task to optimize. If needed for solving the task the solver manager employs several solvers.
13) The solver manager forwards the result to the kernel.
14) The kernel performs processing of the result, if needed.
15) The kernel stores the result (the decision variables) in the database. The result can for example be the estimated energy flows of the building, and the hourly use of technologies.
16) -17) The user interface, upon user request, displays the result. For example, this can be done by using the graph tool to display the optimization result in a chart. During the optimization process the user interface can use a kernel web service to check information provided by the solver about the optimization progress and status.

The following communication diagram shows the module interaction. Note that the call to the solver manager is asynchronous.

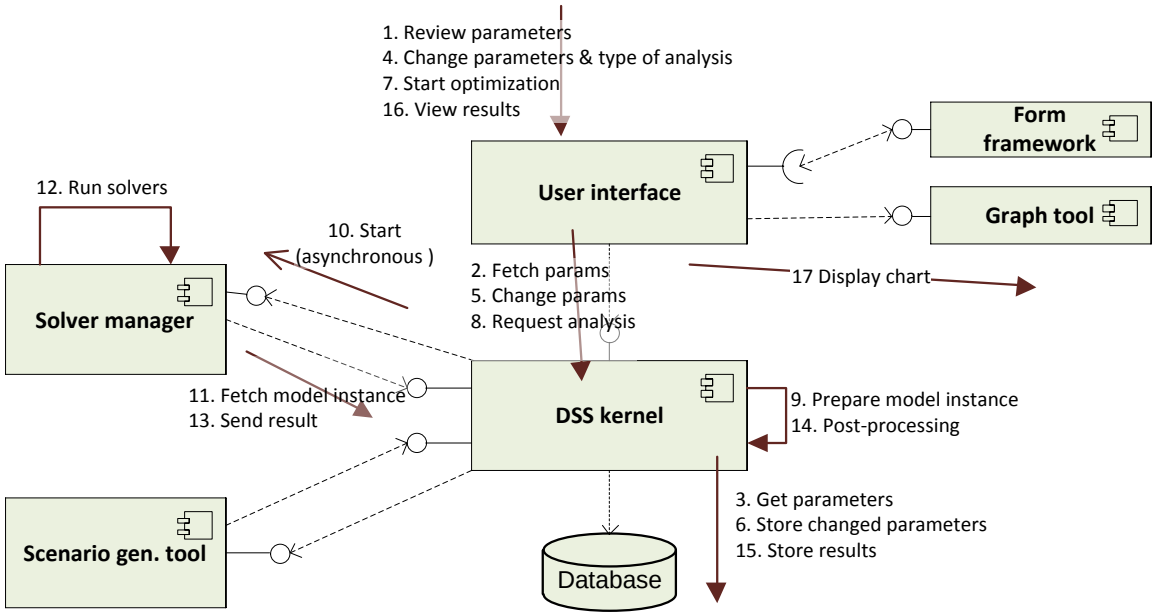


Figure 3-3: Strategic planning

3.2.3 Use case "Data provision for strategic planning"

(Use case 9.3.3 in D4.1, page 71)

This use case is an example that involves the user interface, kernel, data wrappers modules and data external sources. Examples of data provisioning includes getting weather forecast (temperatures, solar irradiation) and energy prices from external sources as well as getting building data (energy consumption, airflows) from the sites building management system. The use case in D4.1 contains the following generic steps:

1) Upload data
2) Check consistency
3) Commit data
4) Send notification.

As the above use case is generic is has been specialized into two variants in this deliverable, *manual upload* and *automatic upload*. This specialization into two use cases is done to show how the modules interact in order to handle data from external sources.

3.2.3.1 Alternative 1 – Automatic upload via an external source

Data could be provided by the building management system in an automatic fashion, for example via FTP upload. This has been successfully tested for the ENERGYbase site. A benefit with automatic uploading is that the user does not have to manually upload the data regularly. Note that the interaction between the EnRiMa DSS and the external system can also

be reversed. This is that the EnRiMa system could fetch data from the external system, for example via FTP or via XML web services. The following use case illustrates the actions needed for automated upload.

1) The external building management system uploads data to the data wrapper module. Alternatively, the wrapper downloads the data from the building management system. The data exchange can also be triggered by a scheduler based on either status of the available data or time intervals.
2) The wrapper converts the data into the corresponding XML structure.
3) The wrapper module submits the data to the kernel by the use of web services.
4) The kernel checks the data consistency with the symbolic model specification and stores the data.
5) The kernel commits the data to the database, if it is consistent. The information can later be accessed via the web services of the kernel, for example for displaying it in the user interface.

The following communication diagram shows how the modules can interact in order to support the use case when the upload is triggered by the building management system.

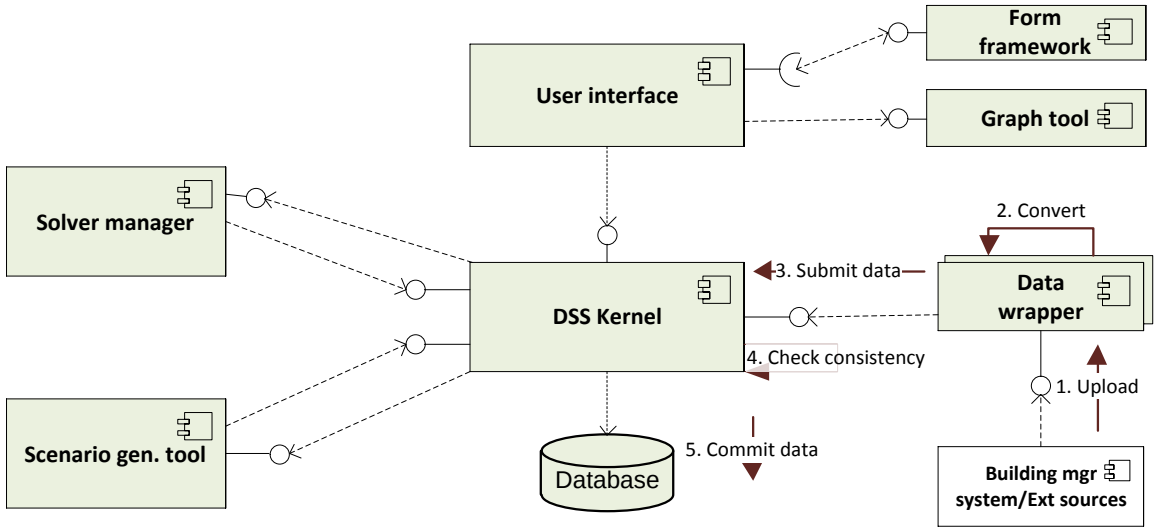


Figure 3-4: Automatic upload from BMS

3.2.3.2 Alternative 2 – Manual upload of data via the user interface

This specialized use case describes the case when the user themselves fetches the data from external sources, as files, and upload the files to the DSS. For example manual upload can be necessary if the building management system is not connected to a network.

1) The user requests an upload via the user interface and provides a file to upload. The file can for example contain a weather forecast, or information on a building's energy consumption.

2) The file is forwarded to the kernel.
3) The kernel checks the data consistency according to the symbolic model specification.
4) The kernel commits the data to the database. If the data is to be displayed by the user interface (see next step) the kernel returns the data using the ordinary exchange format for user interface to kernel communication. If complex conversion is needed, the kernel might use a data wrapper module to perform the conversion.
5) The user interface displays the uploaded data, for example as a table of values.

The following communication diagram shows how the modules can interact in order to support the use case when the upload is triggered via the user interface. Note that all interactions are synchronous; the calling module waits until the call is finished.

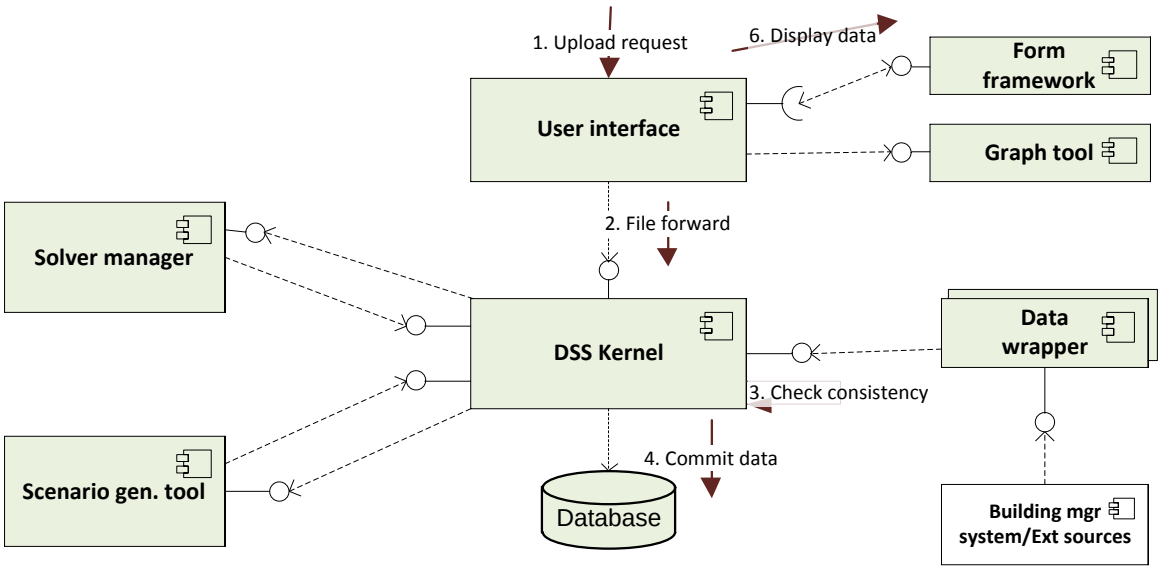


Figure 3-5: Manual upload of data

3.3 Deployment view

The developed DSS needs to be deployed on computers. Since the architecture is modular and protocols such as web services are used it is possible to deploy the system in several different configurations. For example, the user interface layer of the system can be deployed to the same computer as the engine layer, or on another computer, possible at another location. To determine the components of the system that can run at another location it is important to determine possible distribution boundaries, and thereby units of deployment. In essence, the components that use frequent or large data interchange, and those components that communicate using protocols inappropriate for distribution should be in the same deployment unit, deployed within a fast network.

With an architecture as presented in this deliverable it is reasonable to divide the system into three deployment units;

- User interface, including the form framework and the graph tool modules. These modules are tightly integrated and have frequent data exchange.
- Engine, including the kernel, database, solver manager and scenario generation tool. These modules communicate using protocols that are not suitable for distribution. However, both the scenario generation tool and solver manager could be deployed to separate computers if needed during development.
- Data wrappers. These modules can run in a computer where it is readily accessible for external systems.

In the following sections different deployment options for development, test-sites and production use will be described. The tool selection as described in the *module view* makes it possible to run the system on both Microsoft Windows and UNIX operating systems.

3.3.1 Deployment for development and internal tests

During development it is convenient to run the system in a computer close to the developer, so that the time for deployment is short, and so that any introductions of errors do not affect the other developers or end-users. This means that all of the deployment units can run on different computers during development.

3.3.2 Deployment for test sites

Deployment for test sites within the frame of the project will be done in the same way as the deployment for internal tests. I.e. the main parts of the system will run on computers hosted by the project partners. Another option that will be considered is deployment in one computer, this might give slightly better performance, as latencies are lower and bandwidth higher within a single computer. Co-location of the engine and the user interface has the advantage of not having to use encrypted protocols for their communication. The solver manager module requires a lot of computing power, thus it might be reasonable to deploy that module to a separate computer.

All the test-sites can use the same deployment. That is, information about the different sites can be stored in the same database. Each of the users that are making use of the system is authenticated and authorized to access just their site information. This way of “sharing” a single installation is sometimes referred to as a multi-tenant installation. All access of the users will be done via web browsers over the Internet.

3.3.3 Possible deployments for production

When making use of the system after the project is completed, there can be several deployment options depending on the business model of the system exploitation:

Deployment at a site, this might be a viable option if the site needs a lot of customizations to the system, or that there is a need for a very tight integration with existing building management systems. Furthermore this could be the mandated option if the site has restrictions concerning the exchange of data with external systems, or the site lacks an Internet connection.

Deployment at external company. This entails installing the system at a company that sells configuration services for the DSS. The external company needs to handle user access, and configurations specific for each new site.

Cloud service deployment. This entails modifying the system to be able to run on a cloud platform service, such as Google app engine or Amazon EC2. This type of installation can be done by as site owner, or by an exploitation company for a multiple sites (multi-tenant installation).

3.4 Validating the architecture by prototypes

The architecture as described in this deliverable has been validated by the use of prototypes. This form of prototypes are sometimes referred to as “architectural prototypes”, or “spike solutions”. Each of these architectural prototypes validates a part of the architecture. The idea with validation using architectural prototypes is to test the architectural parts that have the highest risk. Typically these parts contain elements that are challenging, or make use of technologies that are unknown to the development team.

The purpose of these architectural prototypes is to validate the architecture, that is, test that the chosen architecture is feasible and will allow constructing the final system in accordance with the functional and non-functional business requirements. The program code used in these prototypes can be, but is not necessarily, used in the final product. It is rather the gained knowledge that is transferred into the construction of the final system. In this deliverable the initial set of architectural prototypes that has been performed are briefly described. During the continuation of the project more prototypes will be created when deemed necessary, for example if there is a need to include new tools or protocols in the architecture.

3.4.1 User interface and kernel communication prototype

The communication between the *kernel* module and the *user interface* module has been tested by retrieving and sending object structures thought the web service protocols. The purpose of this architectural prototype was to ensure that the user interface and kernel could make use of the selected protocols and tools to interpret information from the kernel. Moreover the use of XML schemas as the foundation for creating declarations of the needed XML structures was tested.

The architectural prototype was created by involving the following development steps:

- Creation of the XML schema describing the kernel interface. The XML schema description (XSD) defines the structure of the information to be sent to/from the kernel. Moreover the XML based Web Service Description Language (WSDL) was used to declare the operations that the kernel supported.
- Implementation of a kernel prototype with access to the database. The implementation followed the operations as declared by the WSDL file. The kernel was deployed in an Apache Tomcat server at IIASA (Laxenburg, Austria).
- The user interface Java model classes was created by using XSD and WSDL to generate Java classes. This was done by the use of an open source JAX-WS tool.
- A user interface was created to access the kernel using the generated Java classes. This application had a simple user interface created using the Vaadin framework. The application was deployed as a Java Servlet in an Apache Tomcat server at SU (Stockholm, Sweden).

The following figure below shows an excerpt of the created WSDL file:

```
▼<xs:element name="getModelDescriptionResponse">
  ▼<xs:complexType>
    ▼<xs:all>
       <xs:element minOccurs="0" name="model" type="m:model"/>
     </xs:all>
   </xs:complexType>
  </xs:element>
▼<xs:complexType name="model">
  ▼<xs:sequence>
     <xs:element name="id" type="xs:int"/>
     <xs:element name="name" type="xs:string"/>
     <xs:element name="version" type="xs:string"/>
     <xs:element name="description" type="xs:string"/>
     <xs:element name="note" type="xs:string"/>
     <xs:element name="auditable" type="m:auditable"/>
     <xs:element maxOccurs="unbounded" name="revision" type="m:modelRev"/>
   </xs:sequence>
```

Figure 3-6: Excerpt of the WSDL used for the communication test

When running the architectural prototypes the following steps are executed:

1) Starting the user interface by opening a web browser and point if to the start address. The user interface module (Vaadin application) starts and displays a web interface where the user can start the communication with the engine.
2) The user interface module initiates the connection to the kernel via the generated Java classes and sends a request to get a list of model objects to the kernel. The JAX-WS (Java API for XML web services) translates the request into a XML based web service request.
3) The kernel retrieves the XML based request and re-translates in to Java objects suitable for internal processing. The kernel fetches the requested information from the database and returns the resulting data structure.
4) The client gets the result as XML and translates that into Java objects. The resulting Java objects are sent to the Vaadin framework for display in the users web browser.

By the use of the architectural prototypes the communication between the kernel and user interface was tested. It was shown that the communication could be done by the use generic web services. At the same time it was tested that the user interface internally could use Java objects without the need to manually code the translation between XML and Java objects. This kind of automatic conversion can also save time during development.

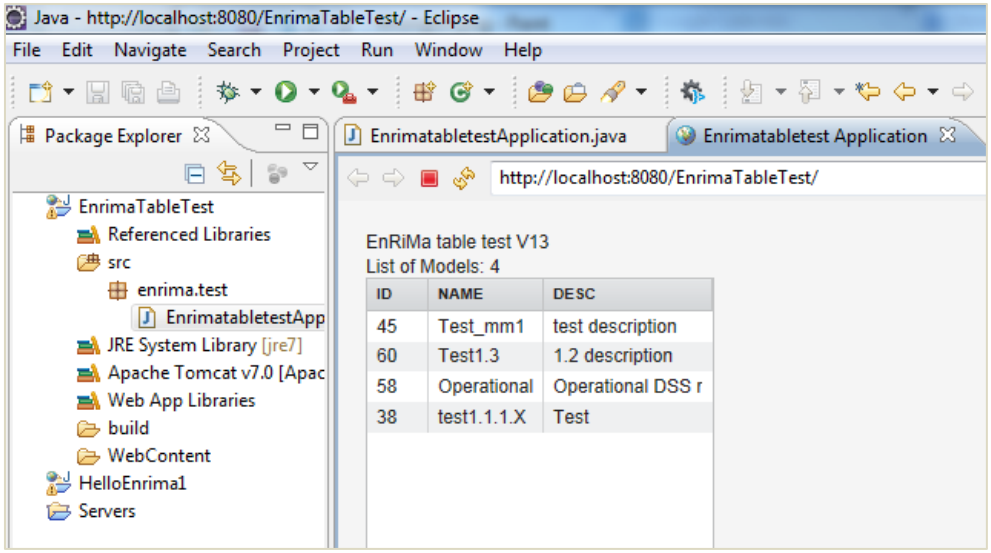
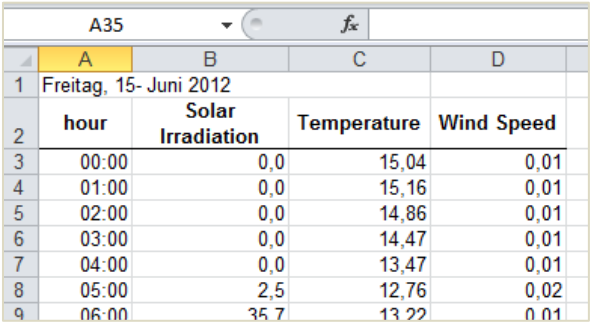


Figure 3-7: The communication test, running in the Eclipse development environment

3.4.2 Data wrapper architectural prototype

An architectural prototype was built to test the interconnection of a building management system and the DSS. The purpose of this architectural prototype was to test the access to building information, and how to handle format conversion and in order to get enough information to calculate the energy flow of a building. Thus this prototype validates part of the architecture of the *data wrapper* module. In order to make use of the prototype the collected information was displayed in a Sankey diagram. Because of readily available data the ENERGYbase site was selected for this architectural prototype.

The data is collected at the site and then transferred to an FTP server where it can be picked up and processed in order to create the Sankey diagram. The transferred file includes the daily energy consumption and weather details, which are stored at 15-minute intervals. The figure below shows a sample file for the architectural prototype:



A35

	A	B	C	D
1	Freitag, 15- Juni 2012			
2	hour	Solar Irradiation	Temperature	Wind Speed
3	00:00	0,0	15,04	0,01
4	01:00	0,0	15,16	0,01
5	02:00	0,0	14,86	0,01
6	03:00	0,0	14,47	0,01
7	04:00	0,0	13,47	0,01
8	05:00	2,5	12,76	0,02
9	06:00	35,7	13,22	0,01

Figure 3-8: Excerpt from the file showing weather information from the site

The sequences of actions to get the site data and process it are the following:

1) At the site, data are collected and stored continuously within the building management system (BMS). The site uses the DESIGO system from Siemens.
2) With the help of the BMS, a pre-defined set of data is stored within a MS Excel file.
3) Each day at the building operator's computer, an MS Windows script (cmd) is executed to transfer the collected data (*.xls file) to a pre-defined FTP server. This FTP server is managed by CET.

Once at the FTP server the *data module* can pick up the file and convert it into a format suitable for handling by the *kernel* module. To demonstrate that the file was successfully retrieved the architectural prototype creates a Sankey diagram that can be displayed to show that the data was correctly received. This architectural prototype shows one way of communicating with external systems. For other ways, for example that the DSS initiates contact with external systems via web services, more architectural prototypes need to be built.

The following figure depicts the final Sankey diagram, as created based on the data read from the site at June 14, 2012 (the latest Sankey diagram can be found at http://enrima-project.eu/dss/sankey-diagrams):

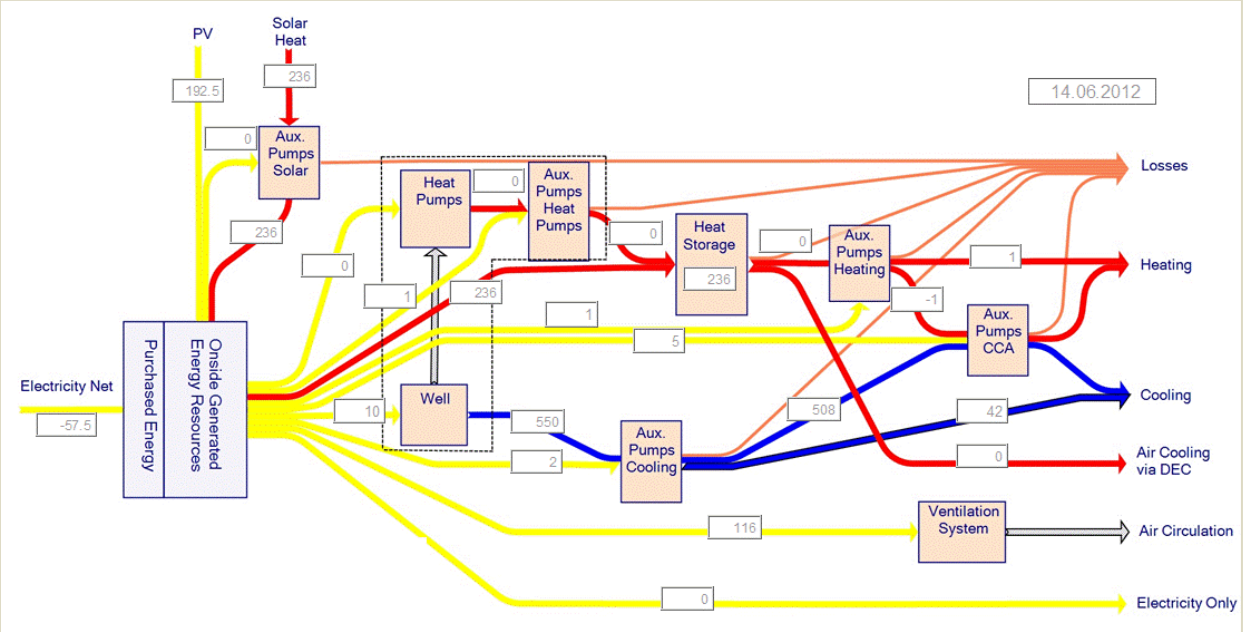


Figure 3-9: External data from site, shown in a Sankey diagram

3.4.3 User interface, graph tool and framework architectural prototypes

Several architectural prototypes were created to try out the technologies that were selected for use in the user interface layer of the architecture. The purpose was to see how the *user interface* module could make use of the *form framework* module and the *graph tool* module. Furthermore the composition of common user elements was tested, as well as how to bind the model objects as received by the kernel to user interface elements. Binding entails both allowing the user to view and update the model objects via the user interface.

To illustrate the types of architectural prototypes that were built the figure below shows a simple form containing a data table and a graph. In this case the prototype was run in the Google Chrome web browser.

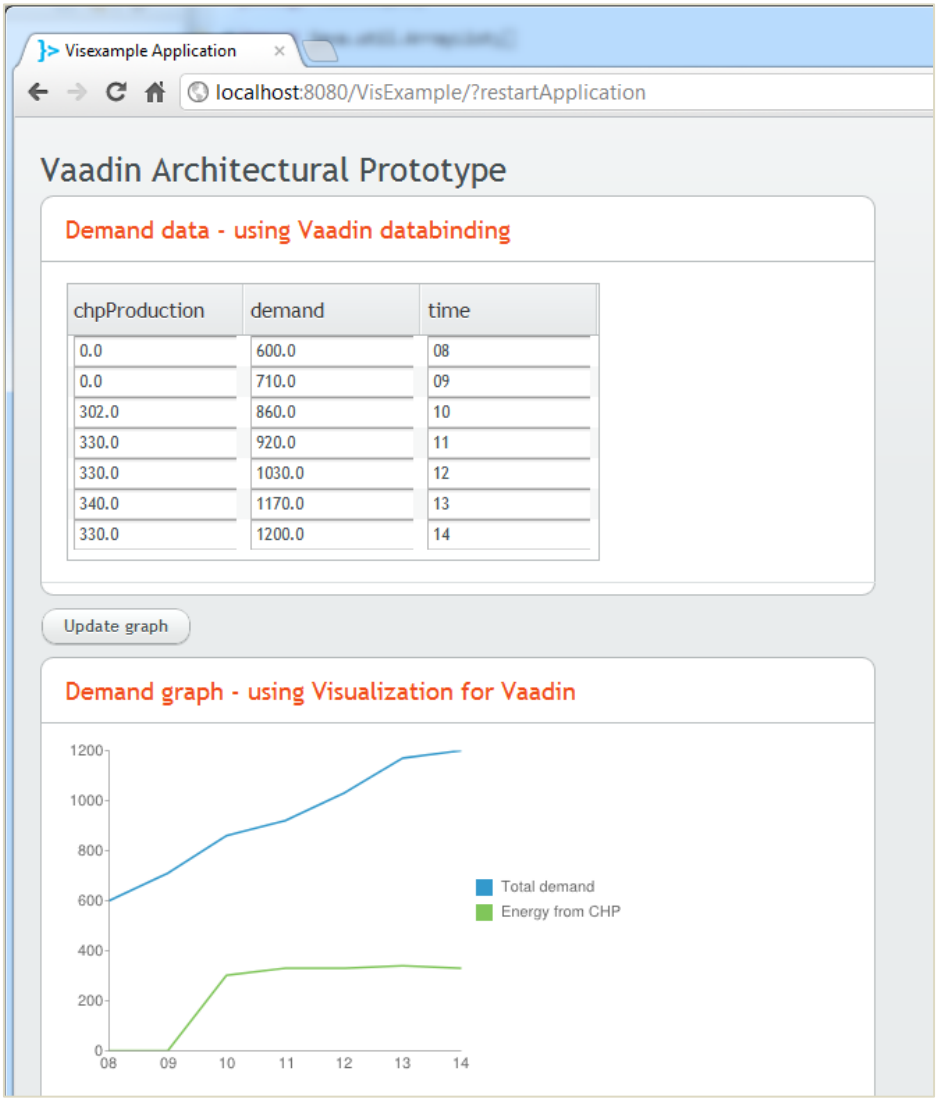


Figure 3-10: Form framework and graph tool

The above prototype tests the following elements:

- Data binding of model classes. The table at the top is bound to a list of model objects by the use of the form frameworks (Vaadin) built-in functions. Data binding work two-ways, that is, the model objects are updated when the user changes the values in the table.
- Visualization using the VisualizationForVaadin add-on. This add-on, showing the graph above, is utilizing the Google visualization toolkit.
- A rich, interactive web application. The architectural prototype is making use of AJAX calls to quickly update the graphs when requested. The events are handled by the server, however the Vaadin form framework takes care of updating the screen in the users web browser. The Vaadin framework creates the needed HTML, CSS and JavaScript code.

3.4.4 Solver manager and scenario generation architectural prototypes

Prototypes of both the solver manager module and the scenario generation tool module have been developed. These implementations have been described in other deliverables:

- The *scenario generation tool* module implementation is a fully functional tool written in C++. This tool is described in deliverable D3.2 (SINTEF, 2012).
- A prototype *solver manager* module has been implemented using the open source language R. This is described in deliverable D4.2 (URJC et al., 2012).

Both of these modules are expected to evolve as the project progresses.

4 Conclusion

The EnRiMa project will provide a decision support system that enables building owners and operators of public buildings to improve the buildings energy efficiency. Previous deliverables produced by the project have described the functionality of the system, such as the main requirements on the system (D4.1, IIASA et al, 2011), the needed mathematical models for describing the buildings energy flows (D2.2, UCL et al., 2012), and the strategic and operational decision support models (D4.2, URJC, 2012).

In this deliverable we have specified the software architecture for services and tools to be used to build the EnRiMa decision support system. The software architecture describes the overall structure of the system. Thus while the previous deliverables describe the functionality and the models needed to calculate energy flows and consumption, this deliverable describes which software modules are needed to realize the functionality and calculations into a decision support system.

The software architecture for services and tools as presented in this deliverable defined the main software modules of the system, and how they interact in order to fulfil business requirements. Furthermore, the type of services provided by each module is specified as well as the tools to be used for their construction. The deliverable is divided into several views: module, information, workflow, communication, dynamic, and deployment views. Each view is describing a certain aspect of the system to be constructed. The validation of the architecture has been done by the use of a series of architectural prototypes described in the deliverable.

This deliverable will be used in the following project activities and deliverables:

- DSS Engine construction (WP4). This deliverable provide the main architecture for how the DSS Engine modules will interact, their types of information input and output flows and the main responsibilities of each module. The construction of the DSS Engine will lead to deliverable D4.4, the DSS Kernel prototype implementation.
- DSS User interface construction (WP5). This deliverable provides the design of the inbound and outbound communication with the DSS Engine as developed in WP4. Moreover, it provides a blueprint of the modules and tools needed to build the user interface. The construction of the user interface will lead to the deliverable D5.2, GUI Prototype and evaluation.

The architecture as defined in this deliverable provides a robust foundation for building the EnRiMa DSS. As the project and implementation progresses the architecture of the system will evolve.

Acknowledgements

This deliverable has been developed in an iterative process over almost a year. Lead authors during this process have been SU (Martin Henkel and Janis Stirna). Contributions in the form of module descriptions, scenarios and comments from IIASA (Marek Makowski, Hongtao Ren, Janusz Granat), SINTEF (Michal Kaut, Adrian Tobias Werner, Lars Hellemo), URJC (Emilio Lopez Cano), CET (Markus Groissböck, Michael Stadler) have been integrated into this deliverable.

Quality control was perfomed by Tecnalia (Eugenio Perea, Ana Mera) and SINTEF (Lars Hellemo). UCL (Afzal Siddiqui) also provided comments that improved the work.

References

Ambler S.W., Lines M., (2012), Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise, IBM Press, ISBN 978-0132810135.

Fowler, M. (2003), Patterns of Enterprise Application Architecture, Addison-Wesley.

Gamma E., Helm R., Johnson R., & Vlissides J. (1994), Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, ISBN 978-0201633610

Ghezzi, C., Jazayeri, M., Mandrioli, D. (2002), ”Fundamentals of software engineering”, Prentice Hall.

Gorton, I. (2006), Essential software architecture, Springer-verlag.

HCE, IIASA, SU, UCL, URJC, SINTEF, CET, and TECNALIA (2011). Requirement Assessment. EnRiMa Deliverable D1.1, European Commission FP7 Project Number 260041.

IIASA, SU, UCL, URJC, SINTEF, CET, HCE, and TECNALIA (2011), Requirement Analysis, EnRiMa Deliverable D4.1, European Commission FP7 Project Number 260041.

Krausner, G. E.; Pope S. T., (1988). A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System (Report). ParcPlace Systems, Inc., available http://www.itu.dk/courses/VOP/E2005/VOP2005E/8_mvc_krasner_and_pope.pdf, retrieved 2012-06-20

OMG (2011), OMG Unified Modeling Language (OMG UML), Superstructure, version 2.4.1, Document Number: formal/2011-08-06, available from www.omg.org

Shneiderman, B. and Plaisant, C. (2010), Designing the User Interface: Strategies for Effective Human-Computer Interaction: Fifth Edition, Addison-Wesley Publ. Co., Reading, MA.

SINTEF (2012), Scenario generation software tool – Documentation for the software tool, part of EnRiMa Deliverable D3.2, European Commission FP7 Project Number 260041.

UCL, IIASA, CET, SINTEF, TECNALIA, and HCE (2012), A Mathematical Formulation of Energy Balance and Flow Constraints, EnRiMa Deliverable D2.2, European Commission FP7 Project Number 260041.

URJC, UCL, IIASA (2012), Symbolic Model Specification, EnRiMa Deliverable D4.2, European Commission FP7 Project Number 260041.