



Project Number 260041
SUPPORTING ACTION

EnRiMa

Energy Efficiency and Risk Management in Public Buildings

Deliverable 4.4: DSS Kernel Prototype Implementation

Start date of the project: October 1, 2010

Duration: 42 months

Organisation name of lead contractor for this deliverable: SU

Lead authors: SU, CET, TECNALIA, IIASA

Revision: 11, final public, March 29, 2013

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	5
List of Acronyms.....	6
Executive Summary	7
1 Introduction	8
1.1 Architecture of the Kernel Prototype.....	9
2 Database Structure.....	11
2.1 System Tables.....	11
2.2 Strategic Model Input Tables	12
2.2.1 Overview	12
2.2.2 Configuration	13
2.2.3 Building.....	14
2.2.4 Technologies	15
2.2.5 Pollution parameters.....	16
2.3 Operational Model Tables	17
2.4 Scenario Generator Tables.....	19
2.5 Solver Manager Tables.....	20
3 Data Services.....	22
3.1 Data Services – Example.....	23
4 Data Import/Export Services (CET).....	25
4.1 Implemented Functionality.....	25
4.2 Architecture Overview of the Web Service.....	25
4.3 Usage of the Web Service Client.....	26
4.3.1 Upload Operational DSS Result.....	27
4.3.2 Upload BMS Data	27
4.3.3 Upload Weather Forecast	27
4.3.4 Download BMS Set Points.....	28
4.4 Example Data	28
4.4.1 Energy Demand from BMS.....	28
4.4.2 Weather Forecasts	29
4.4.2.1 Weather Underground.....	29
4.4.3 Operational Optimization Result.....	32
4.5 Integration of MatLab® for Operational Optimization	33
4.6 Integration of KUBIK's BMS	36
4.6.1 BMS Client Architecture.....	36
4.6.2 BMS Client Database Structure	37

4.6.2.1	Measurement Database	37
4.6.2.2	Forecast Database	38
4.6.3	Database Structure used by the Data Transfer	39
4.6.4	Transfer processes	40
4.6.4.1	Weather forecast: From Tecnalia Meteo to the FTP server	40
4.6.4.2	Weather forecast: from the FTP server to Tecnalia database	41
4.6.4.3	KUBIK Measurements: from KUBIK Database to Tecnalia Database	41
4.6.4.4	KUBIK Measurements: from Tecnalia Database to DSS	42
4.6.4.5	Weather Forecast: from Tecnalia Database to DSS	43
4.6.4.6	DSS Output: from DSS to Tecnalia Database	43
4.7	Integration of Pinkafeld's & ENERGYbase's BMS	43
5	Scenario Generator Implementation Overview	45
6	Solver Manager Implementation Overview	46
7	Conclusion.....	47
	Acknowledgements	48
	References	49
	Appendix I - Web Service Server & Client Source Code	50
	Package eu.enrima.ws.processfile	50
	Package eu.enrima.ws.processfile.retrieve	58
	Package eu.enrima.ws.processfile.save	60
	Appendix II - Weather Forecast Quality Analysis	63
	Weather Forecast Analysis	63
	Access Weather Underground.....	63
	Weather Forecast Data	66
	Historical Weather Data	66
	Comparison and Conclusion	66
	References for Appendix II	67
	Appendix III - BACnet™	68
	Building Automation System (BAS).....	68
	BAS within EnRiMa	69
	BAS at Pinkafeld Campus.....	69
	Building Automation and Controls NETwork (BACnet)	69
	Required IT components and expertise	69
	FAQ.....	70
	Where is BACnet™ located?	70
	Can BACnet™ be accessed behind a router/firewall?	71

Where is DESIGO™ located within the BAS?.....	72
How can we predefine set-points in general/in Pinkafeld?	72
Which BACnet™ objects are available at Campus Pinkafeld?	72
BACnet™ security	72
BACnet™ Configuration	72
Available BACnet Tools	73
BACnet™ Tool	73
BACnet4J	74
Are the BACnet™ Objects at Campus Pinkafeld Accessible?	80
References for Appendix III.....	80
Appendix IV - Generic Kernel Prototype Implementation	82
Overview of the Generic Kernel	83
Functional View	83
Key Features.....	84
Software Developer's View	85
Generic Kernel Architecture	86
GitHub Repository	88
Conclusions	89
References for Appendix IV	89
List of Acronyms for Appendix IV	90

List of Figures

Figure 1-1: Overview of the prototype architecture	8
Figure 1-2: Overview of the kernel prototype architecture	10
Figure 2-1: System administrative tables	12
Figure 2-2: Strategic model input tables overview	13
Figure 2-3: Strategic model configuration tables	14
Figure 2-4: Strategic model building parameter tables	15
Figure 2-5: Strategic model technology tables	16
Figure 2-6: Strategic model pollution tables	17
Figure 2-7: Operational model input tables overview	18
Figure 2-8: Scenario generator tables	20
Figure 2-9: Solver manager tables	21
Figure 3-1: Example code for the Building data service	24
Figure 4-1: The data import/export web service architecture	26
Figure 4-2: Example output by using the data import/export web service client	27
Figure 4-3: KUBIK BMS client architecture	37
Figure 4-4: Measurement database schema.	38
Figure 4-5: Forecast database schema	39
Figure 4-6: IO_Weather table, data example.	39
Figure 4-7: IO_ZoneTemp table, data example	40
Figure 4-8: BuildingProp table, data example	40
Figure 4-9: Kubik server. Database schema	42
Figure 4-10: Possible Campus Pinkafeld data processing environment	44

List of Acronyms

API	Application Programing Interface
BACnet	Building Automation and Controls network
BMS	Building Management System
DSS	Decision Support System
EnRiMa	Energy Efficiency and Risk Management in Public Buildings
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
JPA	Java Persistence API
JSON	JavaScript Object Notation
kB	kilo Byte
SMS	Symbolic Model Specification
TCP/IP	Transmission Control Protocol / Internet Protocol
WS	Web Service
WSDL	Web Service Description Language
XML	eXtensible Markup Language

Executive Summary

The goal of this deliverable is to describe the DSS kernel prototype implementation of the EnRiMa decision support system. The kernel is a typical server-side software that does not have a graphical user interface. Its functionality is rather to provide the other modules in the system with core functionality. To offer this functionality, the kernel prototype consists of several software modules that provides application programming interfaces (API:s) for the other software modules in the system. The following functionality is provided by the implemented prototype kernel:

- Structured data access that enables the user interface module to retrieve and store information is the DSS database. This is provided via a set of data services.
- Interfaces for running the scenario generator tool and solver. This is provided via proxies.
- Functionally to handle users, their authentication and preferences. This is provided via the data services.
- Functionality to store and retrieve so-called model-instances, which are a coherent set of data that are used for a single optimization.
- Interfaces for interconnecting with external systems. This is provided via an import/export web service.

The prototype implementation described in this deliverable is the first prototype version of the kernel. It will be further improved during the project.

1 Introduction

This deliverable describes the prototype implementation of the Decision Support System (DSS) kernel. The DSS kernel is a central part of the DSS prototype software since it acts as a coordinator and service provider for the other modules in the DSS. In essence, the kernel is responsible for managing the overall system data, including input from the graphical user interface and outputs from the solvers, as well as communication with the external systems, such as Building Management Systems (BMS) through web-services. The kernel has been developed as part of task 4.6 (Implementation of the DSS Kernel) and task 4.4 (Data warehouse and data services). The design of the integration with the KUBIK laboratory test site as reported in this deliverable has been performed as a part of task 5.3 (Testing the GUI-enabled DSS at the laboratory facility). The design of the kernel has been influenced by the work in task 4.3 (Architecture) and task 5.1 (Enterprise services requirements analysis and design) as well as the ongoing work in work package 6. The kernel prototype also draws on the architecture design as presented in D5.1, Draft specifications for services and tools (SU et. al, 2012).

This deliverable describes the first version of the kernel prototype and a second version of the prototype will be described at the end of the project in deliverable D4.7, DSS Kernel implementation.

The general structure of the prototype software is illustrated in Figure 1-1. The boxes illustrate the main software modules, while the arrows denote control and information flow. As can be seen in the figure the *kernel* sits between the GUI and the modules performing the pre-processing (*Scenario generator tool*) and optimization (*Solver manager*). Together with the database and surrounding modules the kernel forms the DSS Engine.

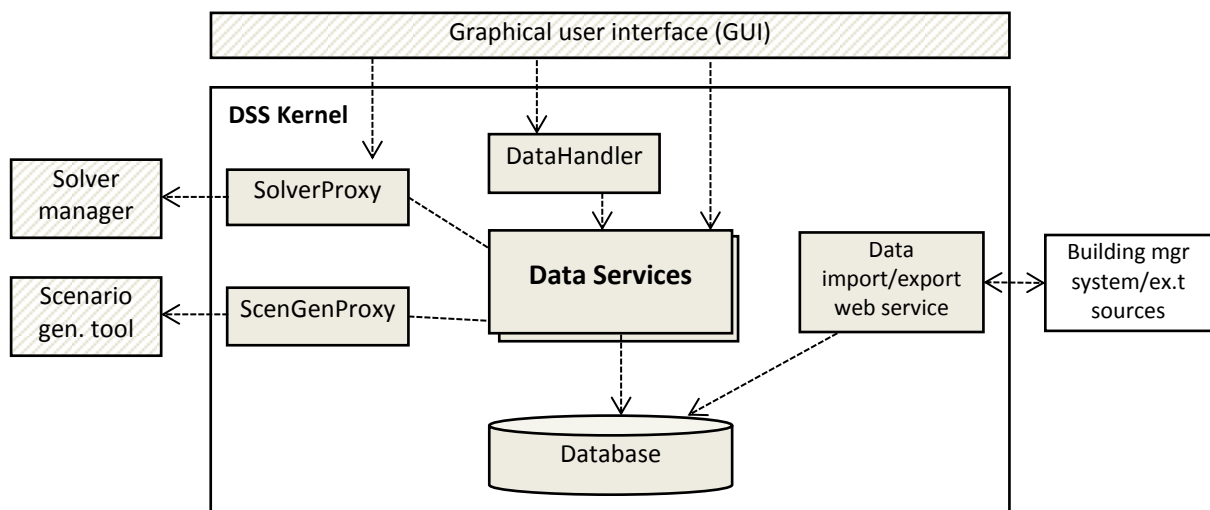


Figure 1-1: Overview of the prototype architecture

Before describing the implementation of the modules within the kernel, the role and contents of each part in the architecture will be briefly summarized below. A description of the modules can also be found in deliverable D5.1, Draft specification for services and tools (SU et. al, 2012). Note that we first focus on the overall architecture, that is, those modules that are surrounding the DSS kernel (marked with a dashed background in Figure 1-1).

The graphical ***user interface*** consists of the modules responsible for the interaction with the user. The user accesses the DSS via a *Web browser*. The user interface access the *kernel* functionality via the *data access services* described in this deliverable. For example, the user interface can retrieve a list of buildings to display for the user via the kernel *data services*. The access protocol is in-process Java calls. The user interface prototype is further described in deliverable D5.2, GUI prototype and evaluation (SU et. al, 2013).

The ***DSS kernel*** is responsible for providing functionality needed to manage the system data and to run the *scenario generator tool* and solver. The *kernel* also supports user handling, access control, and users performing queries of the result data. The kernel is interfacing the database via a JPA compliant object-relational mapper (ORM). The kernel inner structure is further described in this deliverable.

The ***solver manager*** is responsible for managing the optimizations. Essentially, it extracts the needed information from the *database*, runs an optimization and puts the result back. The *solver manager* is started by the *kernel* by creating a new process on the server. The *solver manager* is briefly described in this deliverable, and further described in deliverable D4.3, Stochastic optimization prototype (URJC, 2013).

The ***scenario generator*** tool creates a tree of scenarios to be used by the solver. It is started by the *kernel* by creating a new process on the server. Like the *solver manager* the *scenario generator tool* fetches input parameters from the database and stores the result in the database as well. The *scenario generator* is briefly described in this deliverable, and further described in deliverable D3.2, Scenario generation software tool (SINTEF, 2012).

The *GUI*, *kernel*, *solver manager* and *scenario generator tool* will be deployed on the same server, while the *database* due to administrative reasons will be deployed on a dedicated database server.

Overall, the architecture of the prototype is aligned with the architecture as outlined in D5.2. However, during the development work several changes were made. Firstly, modules in need of large data structures for their input and output are now getting these data from the database interface, rather than via the kernel. This includes the modules solver manager, scenario generator tool and the import/export module. The change was prompted by the need to simplify the data access, and letting the modules make use of the standardized database access mechanisms rather than a custom-made kernel API. Secondly, the GUI interface towards the kernel is also simplified to in-process java calls. This change was made to reach improved performance, but mostly to speed up the development by use of standardized tools. These changes affected the inner structure of the kernel, however the kernel retains its main functionality as stated in D5.1. An approach for making the kernel implementation more generic can be found in Appendix IV.

1.1 Architecture of the Kernel Prototype

Figure 1-2 shows an overview of the kernel prototype architecture. Note that the figure contains only the inner kernel module of Figure 1-1. The largest part of the functionality of the *kernel* is located in the *data services*. The data services ensure a structured access to the database and provide an easy-to use object layer for the *solverproxy* and *scengenproxy* modules. Note that the *data import/export web service* is inserting sets of data to the database and does thus not use the *data services* finer grained access.

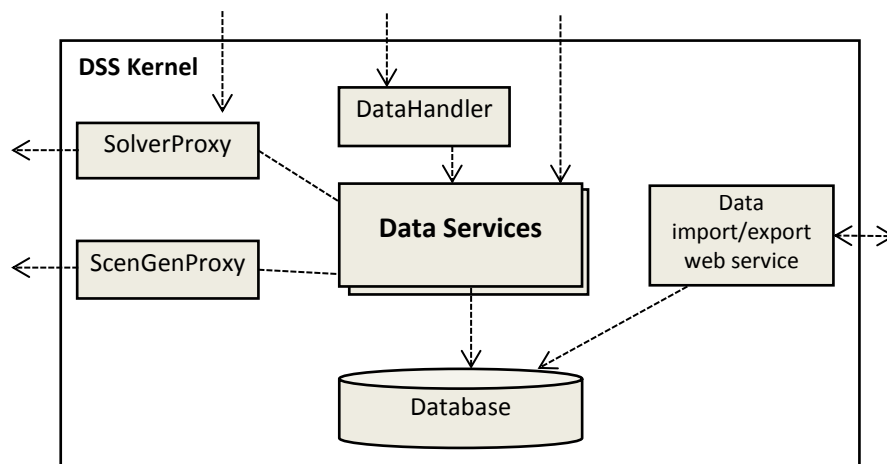


Figure 1-2: Overview of the kernel prototype architecture

The *SolverProxy* and *ScenGenProxy* will act as proxies for the *solver manager* and the *scenario generator tool*, respectively. Both proxies will prepare the inputs to respective module, and then execute the module. Currently, a *solverproxy* for the operational model is developed, the current implementation uses a MatLab® installation as the solver.

The *data handler* manages complex queries and updates to the database. Typically updates that spans multiple data services in a complex way are located in the *data handler*. One example of use in the current prototype implementation is the copying of input data (called case instances) in which a single copy request to the *data handler* can result in hundreds of objects to be copied. For simpler data access needs the *data services* are used directly.

The *data services* represent a structured access path to the *database*. Essentially, each table in the *database* got its own *data service*. The *data services* are used by the user interface and proxies to update and retrieve information from the *database*. The *data services* are described further in the “Data Services” section of this deliverable.

As stated earlier in this deliverable, the data *import/export* module manages the import and export of information to external systems. This module is implemented as a web service, making it remotely accessible for external systems. This module is further described in the “Data Import/Export Services” section of this deliverable.

2 Database Structure

The database is a central part of the DSS, since it stores both the values as entered by the user and the optimization results. The kernel prototype implementation makes use of a MySQL database. To have a structured approach for data handling, the prototype database is divided into five parts:

System – stores information about the user and the basic information of the building

Strategic model – stores the configuration for the strategic model. This includes storing the technologies that should be considered during optimization, energy prices used in the strategic model and so forth.

Operational model – stores the configuration of the operational model. For example, this includes the desired room temperature, per hour.

Scenario generator – this includes the information the scenario generator needs to be executed, as well as its output.

Solver – this part stores the solver-internal configuration of the parameters, as well as the result from the optimization.

Even though there are five parts in the database, they are all integrated. Central to this integration is the use of case instances. Essentially, a case instance (sometimes referred to as a model instance), contains the information needed to perform one optimization. When a user starts an optimization, via the GUI, a new case instance is created. This case instance is used by the modules of the DSS to process the optimization.

In subsequent sections, we include a database schema for each part of the database. The database schema graphically gives an overview of the tables used and their respective columns (for example, see Figure 2-1). While the lines between the tables denote the relationships between data stored in the tables, we also include the names of the columns in each table. To define the possible values of the columns in the database schema we use a set of standard data types. Most common is the INT, DOUBLE and VARCHAR data types. Here we use INT to store identifiers as well as integer numbers, the INT(11) datatype refers to the use of integers that are displayed using 11 positions. VARCHAR is used to store text, the VARCHAR(30) is referring to a 30-character text. DOUBLE is used for most columns that are included in the optimization calculations. DOUBLE refers to a double-precision decimal value.

2.1 System Tables

The system tables (Figure 2-1) handles information that is important for the system as a whole. This includes the access to building information (table Building), the users and their selected buildings (tables UserBuilding and EnrUser). Furthermore, a description of the parameters used in the strategic and operational model is stored in the SymbolDesc table. The system log containing errors are stored in the EnrLog table.

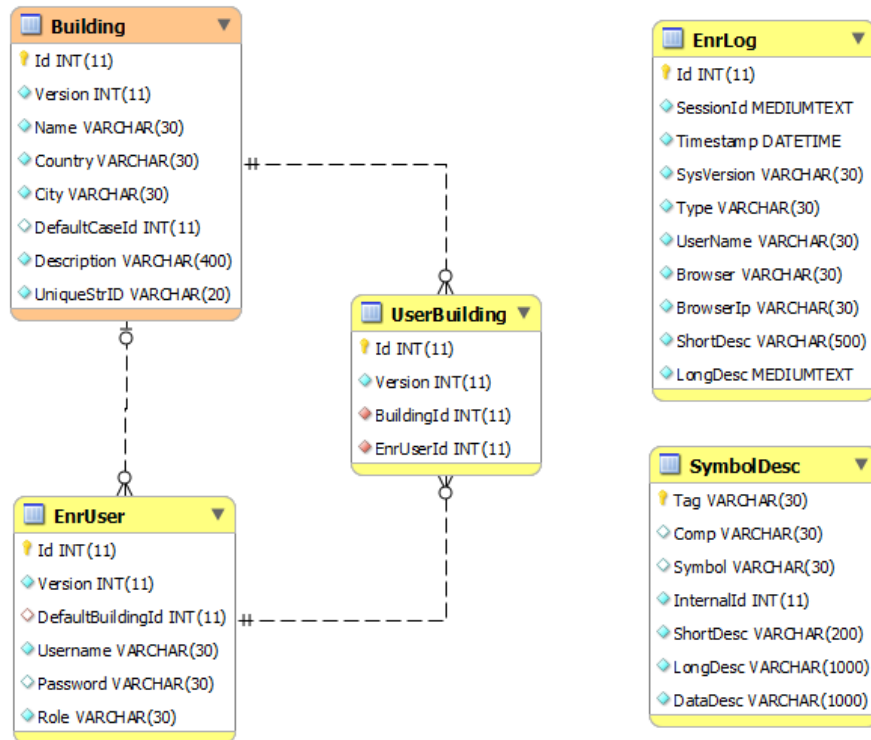


Figure 2-1: System administrative tables

2.2 Strategic Model Tables

2.2.1 Overview

The strategic model tables (Figure 2-2) are used for storing and providing access to data needed for the strategic optimization. These tables are divided into four basic groups: Configuration, Building, Technologies, and Pollution. These groups are described in more detail in the following sections.

Central to the handling of optimization data is the use of a case instance (CaseInstance table). The case instance binds together all parameter data that belongs to a certain optimization. A case instance is associated to a specific building (Building table). All tables containing parameter data are associated directly or indirectly with a specific case instance. For each optimization a new case instance is created to store the data used for that specific optimization.

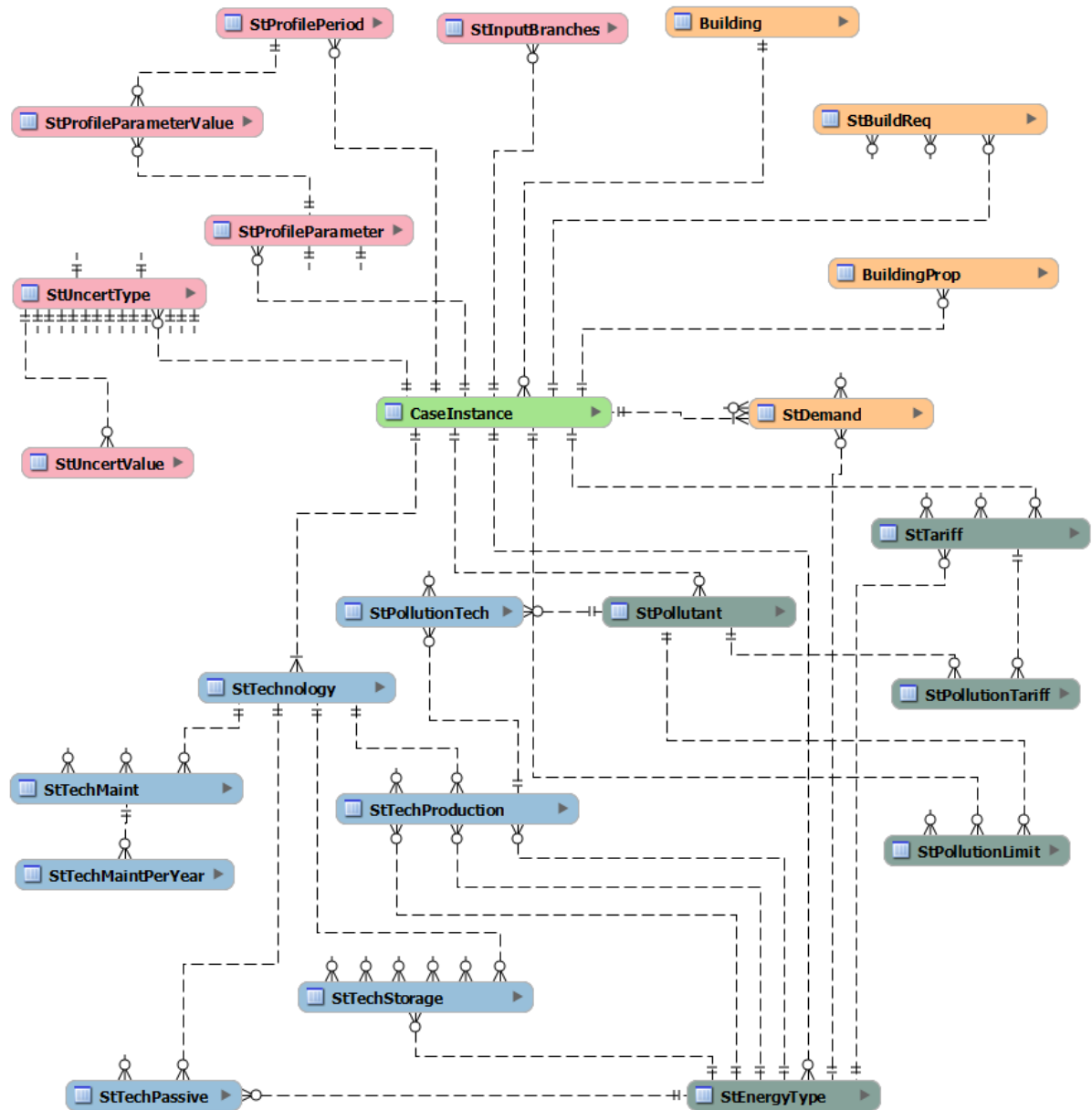


Figure 2-2: Strategic model input tables overview

2.2.2 Configuration

The configuration group of tables (Figure 2-3) contains information that is used to regulate the scope of an optimization and provide repeatable distributions sets for the various stochastic parameters.

Each case instance has a diverse amount of input branches that are considered each year of the optimization, this information is in the StInputBranches table. One or more uncertainty distributions (StUncertType table) can be defined for a case instance. These uncertainty distributions have associated values (StUncertValue table) which provide the mean increase or decrease and the standard deviation to be applied to a stochastic parameter value, for each year of the optimization.

To provide for seasonal variations in hourly values the StProfileParameterValue table contains hourly values. A set of these values is based on a given profile season

(StProfilePeriod table), profile parameter (StProfileParameter table) and case instance combination. A season or period consists of a period and its relative weight. For example, data for a summer day might have a weight of “0.5” if the summer season is considered to be 50% of a year. A profile parameter consists of its name and the unit used for its measure.

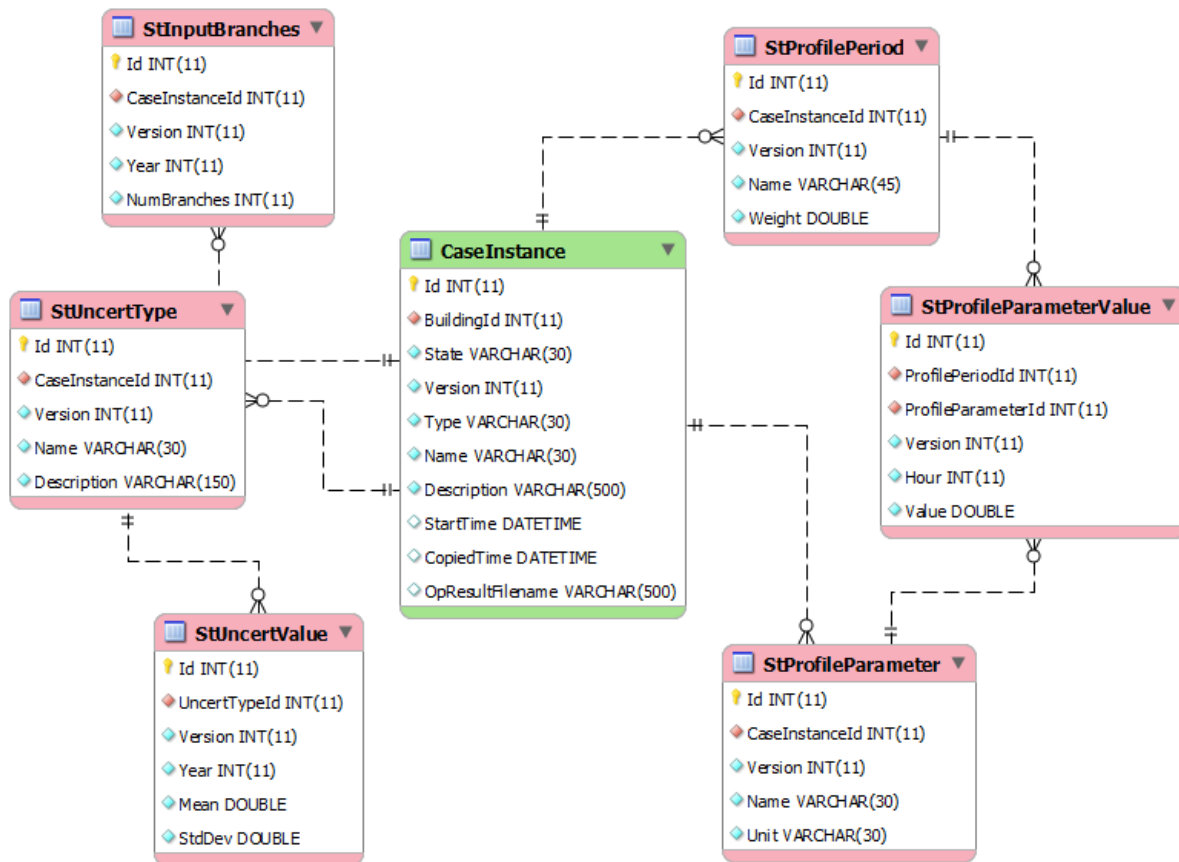


Figure 2-3: Strategic model configuration tables

2.2.3 Building

The building group of tables (Figure 2-4) contains information that is directly related to each building. Information which describes a building and which should not change between individual case instances is contained in the Building table. A building’s name and the country, in which it is located, are two examples of values stored in the Building table. Conversely, information contained in the BuildingProp table might change depending on the case it is associated to. These two tables (BuildingProp and Building) are common to both the strategic and the operational model.

The StDemand table stores information regarding the various energy type demands in a building and how the demand should be distributed. A building’s required energy efficiency and its investment limit are stored in the StBuildReq table. This table also contains information on how the values should be distributed throughout the period of an optimization.

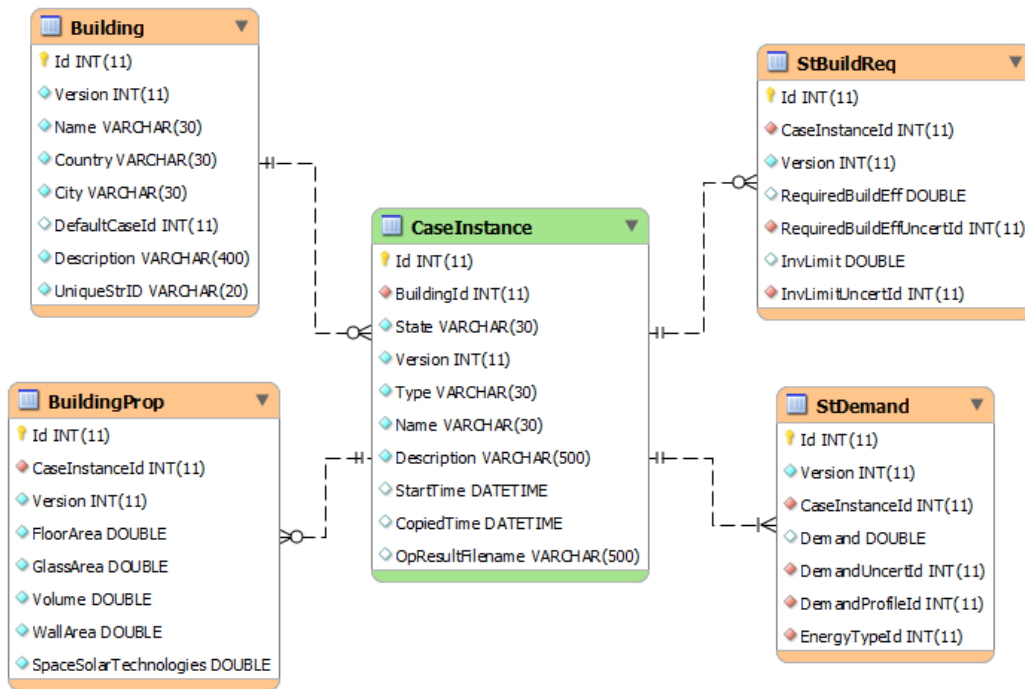


Figure 2-4: Strategic model building parameter tables

2.2.4 Technologies

The technology group tables (Figure 2-5) contain information regarding the various technologies a building might have installed or are considered for installation. Please note these are just the possible technologies, in principle, and the optimization decides which technologies to adopt. These technologies fall into the following categories: generation/production, storage and passive. While each of these technologies has its own unique attributes they also have many in common. The StTechnology table contains the parameters which are common to all types of technologies. A technology can have a maintenance cost that is distributed over the period of an optimization (StTechMaint table). The same technology can have costs that come at regular time intervals (StTechmaintPerYear table). A passive technology (StTechPassive table) can save energy of a given type; the amount of energy can be distributed over the period of an optimization. A storage tech (StTechStorage table) has many stochastic parameters regarding the charging or discharging of energy. A production technology (StTechProduction table) has various parameters regarding the production of energy. Each generation/production technology in turn can produce one or more associated types of pollutions (StTechPollutionTech table).

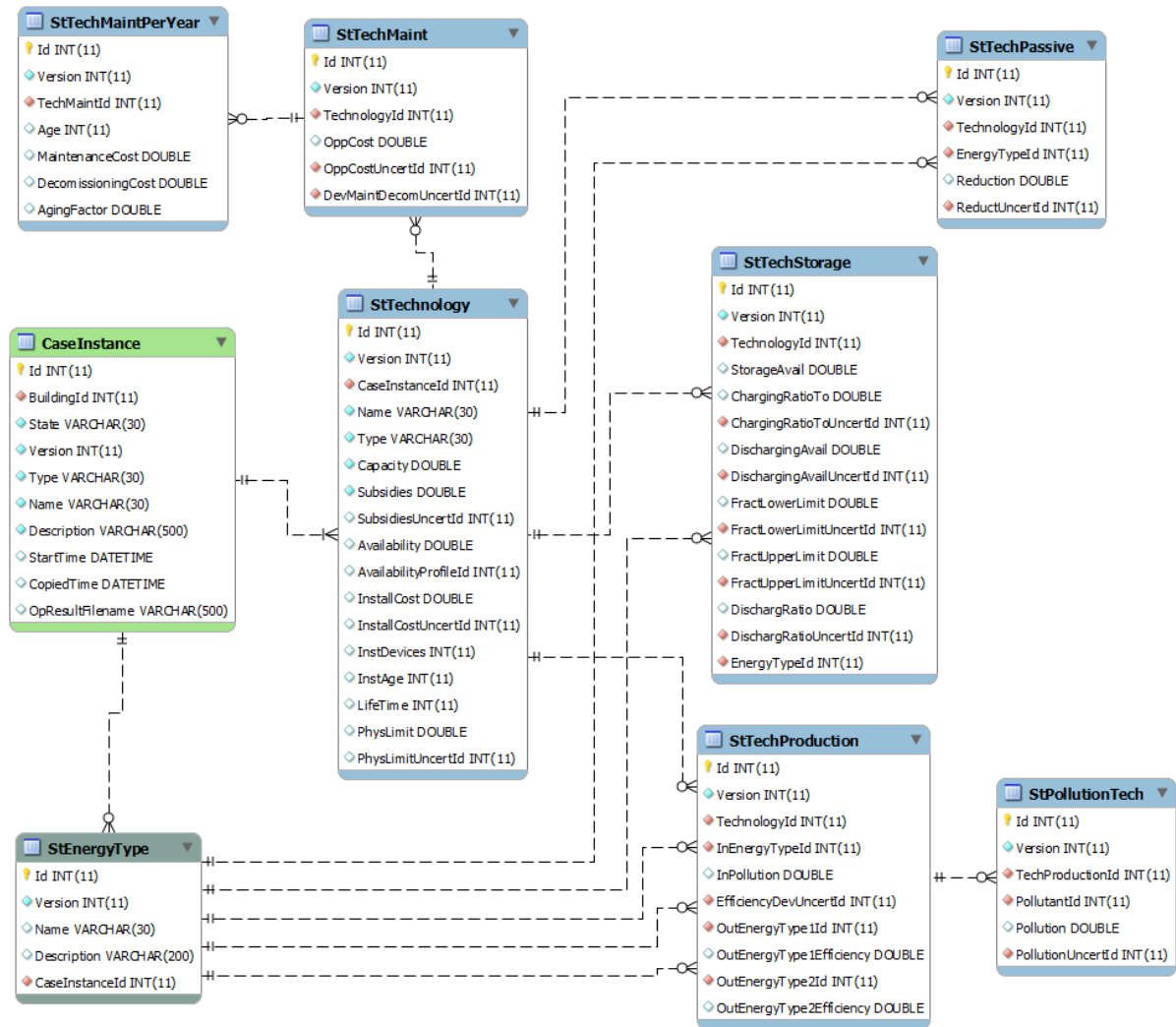


Figure 2-5: Strategic model technology tables

2.2.5 Pollution parameters

The pollution parameters group of tables (Figure 2-6) contain information regarding pollutants along with a table for energy types. The StEnergyType table is used to define the various types of energy that can be associated to energy parameters in other tables. The StPollutant table is used to define the different pollution types that can be associated to pollutant parameters in other tables. The parameters that define the limit allowed of the various pollutants in a given case instance are stored in the StPollutionLimit table. Information regarding the additional costs that can be incurred by the usage of the different energy types is found in the STariff table. These costs can be in the form of multiple pollutants (StPollutionTariff table). information.

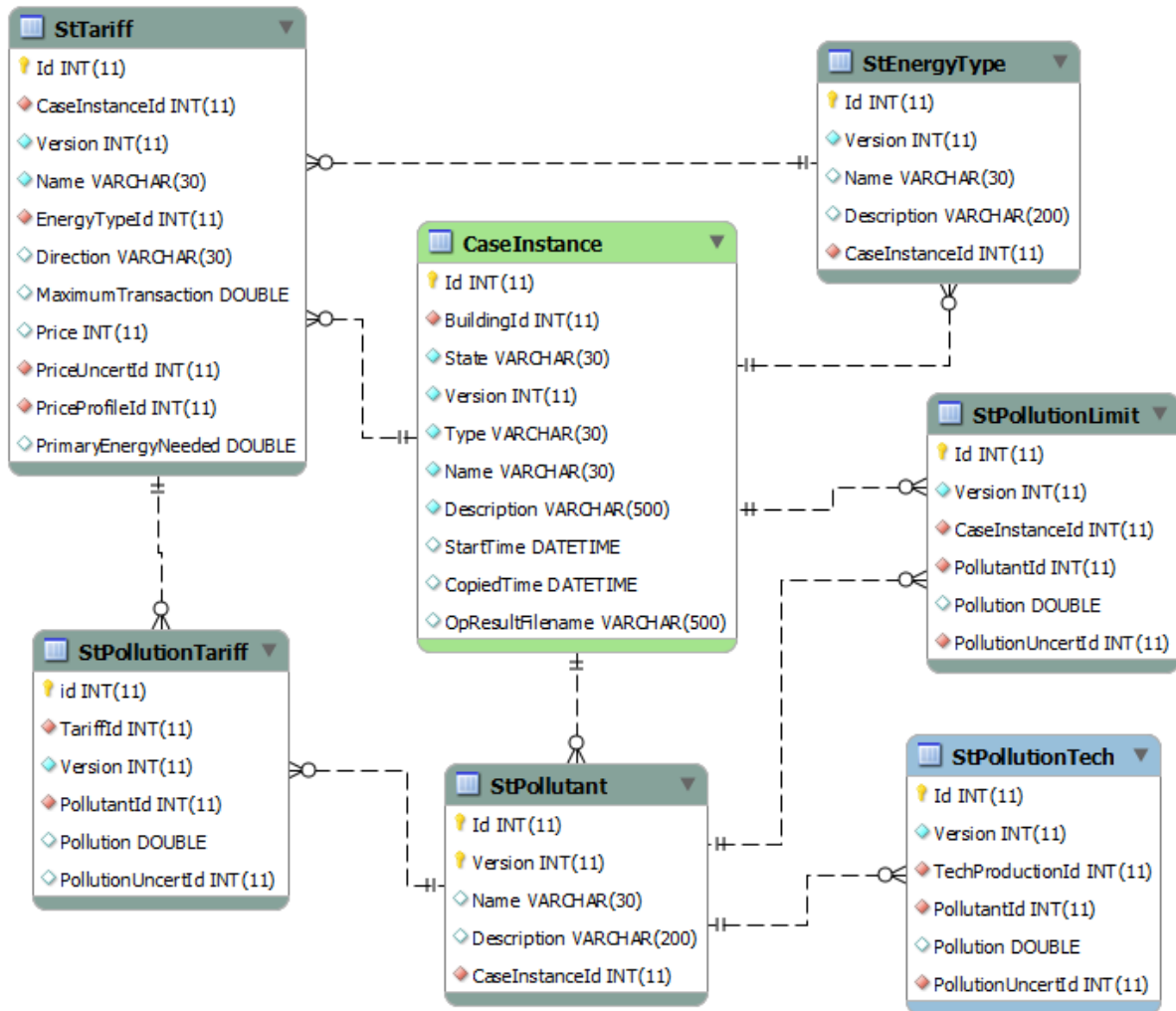


Figure 2-6: Strategic model pollution tables

2.3 Operational Model Tables

The operational model input tables (Figure 2-7) are used for inputting, storing and providing access to data needed for operational optimization. Just as for the strategic model tables, each case instance (CaseInstance table) is associated to a specific building (Building table), all other tables are associated directly or indirectly with a specific case instance. For each operational optimization a new case instance is created to store the data used for that specific case.

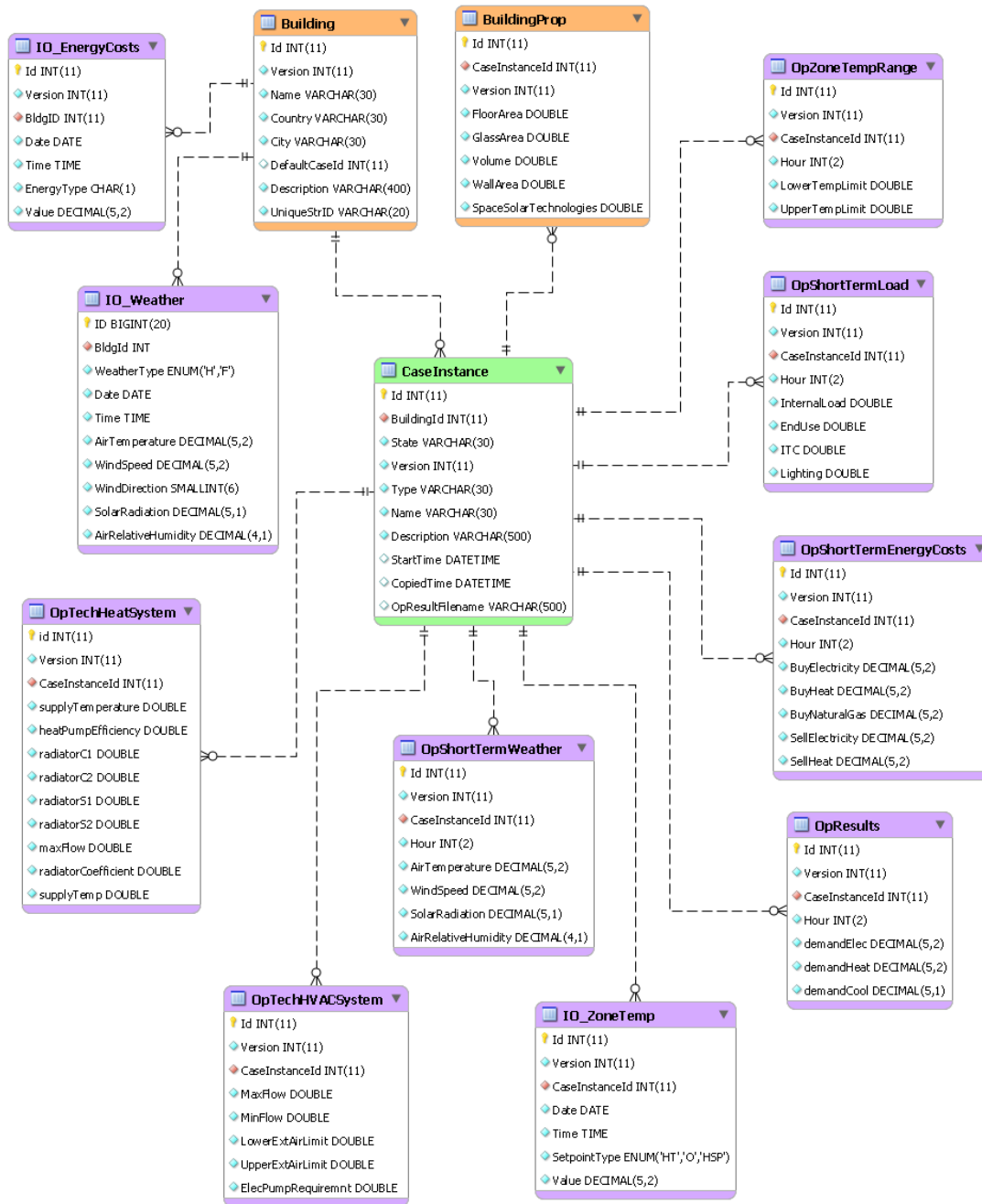


Figure 2-7: Operational model input tables overview

Information which describes a building and which should not change between individual case instances is contained in the Building table. More details about this in section 2.2.3.

The OpZoneTempRange table stores information regarding the temperature bandwidth which has to be considered during the operational optimization procedure. All already installed distributed energy technologies (e.g. CHP), HVAC systems, and additional information are stored in the tables OpAvailTechs, OpTechHVACSystem, and OpTechHeatSystem. The OpResults table contains the results regarding energy demand (electricity, heating, and cooling). All this tables starting with “OpAvail” store the already in place technologies for the sites. The OpShortTermEnergyCosts table contains all information regarding buying and selling energy (buy and sell electricity, buy and sell heat, buy natural gas) on an hourly base for all types of energy which are considered at the moment (electricity, natural gas, district

heating). The OpShortTermLoad table contains the values for internal load and the electricity consumption which is divided into electricity demand for lighting, information technology and communication demand, and other electrical end use.

Energy costs forecasts, weather forecasts, historical real weather information, historical real zone temperature and optimized zone temperatures the tables IO_EnergyCosts, IO_Weather respectively IO_ZoneTemp are used.

2.4 Scenario Generator Tables

The scenario generator tables (Figure 2-8) are used for storing the input, configuration and output of the scenario generator tool.

The input to the scenario generator (parameters and their statistical properties) is stored in the SG_StochParams table, and related tables. Information about the on-going scenario generation is store in the central SG_CaseInstances table, note that this table will be linked with the case instance as contained in the other database tables. This will ensure that all data belonging to a single optimization is retrievable.

Central to the input for the scenario generator in the handling of scenario tree branches (table SG_Input_Branches), and the set of tables handling the seasonal profiles, including the SG_EmbStochParamProfiles table. The output from the scenario generator is a set of tree nodes, stored in the SG_Output_ScenarioNodes table, and profile values stored in the SG_Output_EmbProfiles and associated tables.

Several of the scenario generator tables contains configurations that are used within the scenario generator. This includes, for example, the SG_InputCorrelations and SG_InputTransformations tables.

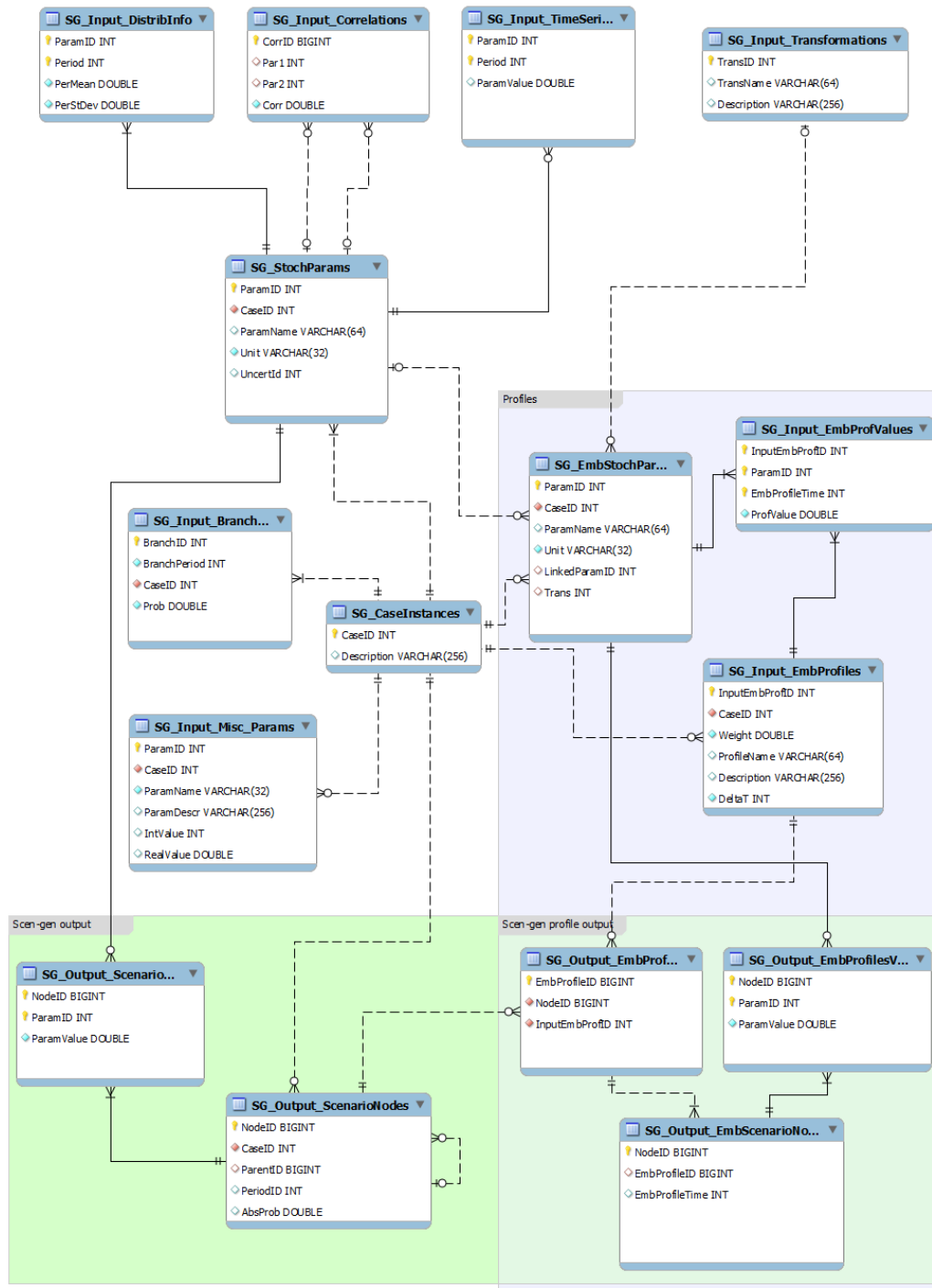


Figure 2-8: Scenario generator tables

2.5 Solver Manager Tables

The data model (Figure 2-9) supporting the operations of the Solver manager module provides the support for the following tasks:

- Storing information about the abstract model definitions, including sets, parameters and output variables, which will be the basis to create the specific instances associated to each case of analysis. The main tables involved in this task are "SMS_set", "SMS_par", "SMS_var" and "SMS_model"
- Storing the values corresponding to specific instances for each analysis case. The main tables linked to this feature are "SMS_mod_instances" as well as the three "SMS_instance_*" tables.
- Holding the results (output variables) of each analysis case, stored in table "SMS_output_vars". This permits other components such as the GUI to retrieve this information and present it to the end-user.

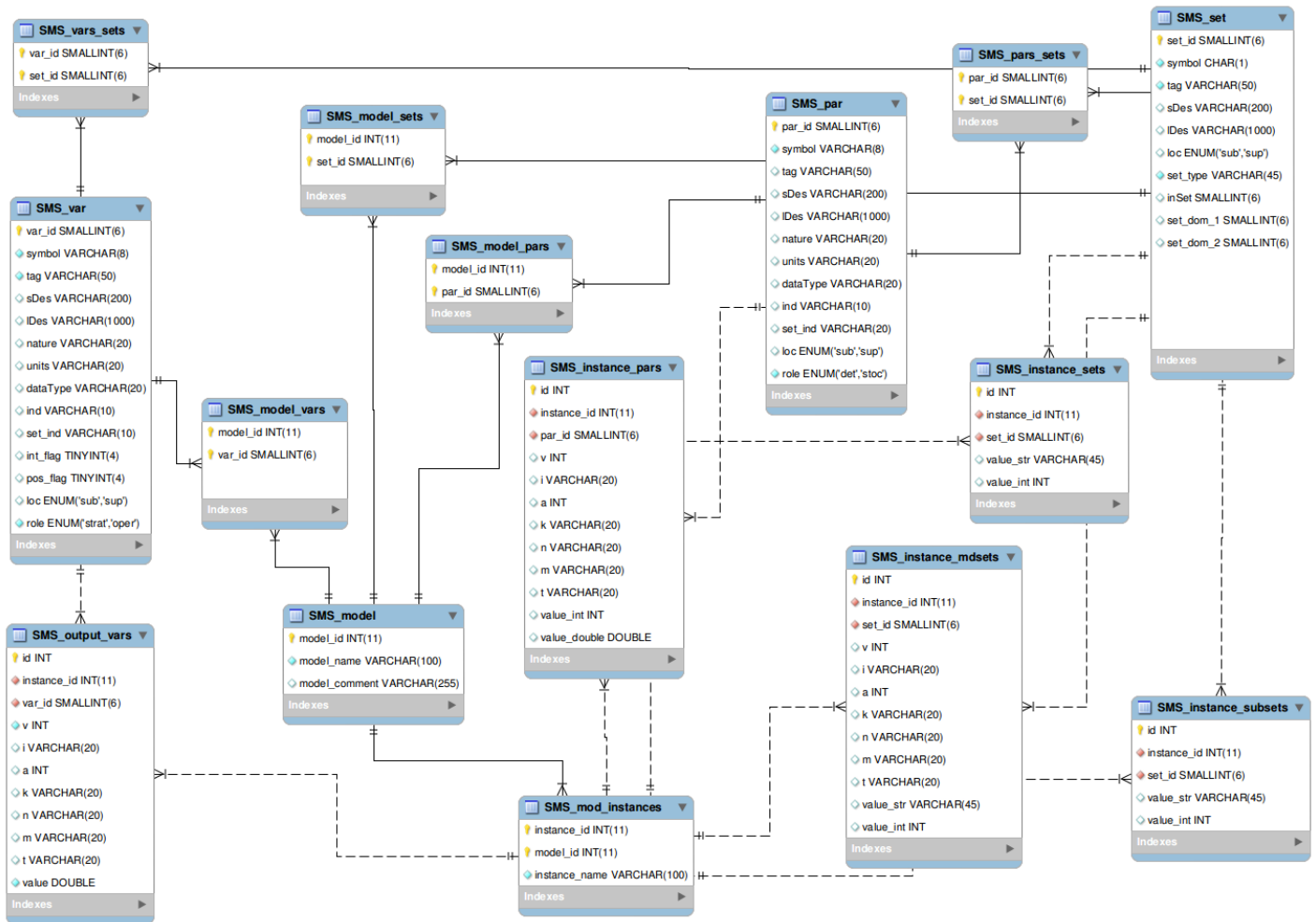


Figure 2-9: Solver manager tables

3 Data Services

The kernel data services consist of a set of classes performing data access to the database. By the use of these classes it is possible to query and update the database in a structured way. Currently the prototype consists of about 40 data services, each handling a table in the database. The implementation of the data services follows the JPA standard, and is making use of the open source persistence framework EclipseLink.

Besides handling the data used for the optimization, such as technology costs and energy demands, the data services also handle internal properties that are important for consistent data handling. This includes:

Creation of identifiers for new data. The data services and the underlying JPA framework are used to generate new unique identifiers when data is added to the database.

Version control. Each data service ensures that data is not becoming inconsistent if there are multiple users that are running at the same time. The version control is handled by the standardised JPA optimistic locking functionality.

The data service structure mirrors the tables in the database that is described in section 2 of this deliverable. Currently, we have implemented the following data services in the kernel prototype:

- | | | |
|---------------------|------------------------------|----------------------------|
| ▪ BuildingDat | ▪ OpResultsDat | ▪ StProfilePeriodDat |
| ▪ BuildingPropDat | ▪ OpShortTermEnergyCostsDat | ▪ StResultDat |
| ▪ CaseInstanceDat | ▪ OpShortTermLoadDat | ▪ StResultTechToInstallDat |
| ▪ EnrLogDat | ▪ OpShortTermWeatherDat | ▪ StTariffDat |
| ▪ EnrUserDat | ▪ OpTechHeatSystemDat | ▪ StTechMaintDat |
| ▪ IO_EnergyCostsDat | ▪ OpTechHVACSystemDat | ▪ StTechMaintPerYearDat |
| ▪ IO_WeatherDat | ▪ OpZoneTempRangeDat | ▪ StTechnologyDat |
| ▪ IO_ZoneTempDat | ▪ StBuildReqDat | ▪ StTechPassiveDat |
| ▪ OpResultsDat | ▪ StDemandDat | ▪ StTechProductionDat |
| ▪ BuildingDat | ▪ StEnergyTypeDat | ▪ StTechStorageDat |
| ▪ BuildingPropDat | ▪ StInputBranchesDat | ▪ StUncertTypeDat |
| ▪ CaseInstanceDat | ▪ StPollutantDat | ▪ StUncertValueDat |
| ▪ EnrLogDat | ▪ StPollutionLimitDat | ▪ SymbolDescDat |
| ▪ EnrUserDat | ▪ StPollutionTariffDat | ▪ UserBuildingDat |
| ▪ IO_EnergyCostsDat | ▪ StPollutionTechDat | |
| ▪ IO_WeatherDat | ▪ StProfileParameterDat | |
| ▪ IO_ZoneTempDat | ▪ StProfileParameterValueDat | |

3.1 Data Services – Example

Each data service contains a mapping to a database table, and a set of methods to retrieve and update information in the database. To illustrate how a data service is structured the following box (Figure 2-8) shows a simplified code extracted from the StBuildingDat data service.

The @Id and @Version annotations at the beginning of the code are used to indicate which attributes contain the generated identifier, and which are used for version control. The code continues with declarations of the building attributes, along with their minimum and maximum sizes. This enables the data service to validate the data and keep the database consistent.

The loadBuilding method is an example of how data retrieval is done via the data service. In this case a single building is retrieved based on its identifier. The copy method is an example of how the DataHandler module is used to copy building information. In this case the copying of building information is only used by administrators which desire to create a new building based on an already existing one.

```
@Entity
@Table(name = "Building")
public class BuildingDat implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Version
    private int version;

    @Size(min = 3, max = 30)
    private String name;

    @Size(min = 2, max = 30)
    private String city;

    @Size(min = 0, max = 100)
    private String description of the JPA classes.;

    private int defaultCaseId; // Default data/case set for the building

    /** Retrieve a single building */
    public static EntityItem<BuildingDat> LoadBuilding(int buildingId) {
        JPAContainer<BuildingDat> cont
        JPAContainerFactory.makeBatchable(BuildingDat.class,DB_PERSISTENCE_UNITNAME);
        cont.addContainerFilter(new Compare.Equal("id", buildingId));
    }
}
```

```
        EntityItem<BuildingDat> item = cont.getItem(buildingId);
        return item;
    }
    /** copies buildings, return a map of the newly created id:s */
    public static Map<Integer, Integer> copy(Map<Integer, Integer> buildIdMap) {
        return DataHandler.copyEntity(BuildingDat.class, "id", buildIdMap);
    }
    ... .
```

Figure 3-1: Example code for the Building data service

4 Data Import/Export Services (CET)

This section describes the prototype implementation of the data import/export web services within the EnRiMa project. Web service (WS) technology is used to simplify the communication between the test sites and the EnRiMa Server which is also known as EnRiMa decision support system (DSS).

One master web service has been designed to fulfill all project required needs regarding file/data transfers. The already available WS can be accessed via the following web address:

http://enrima.dsv.su.se/wsEnRima/

4.1 Implemented Functionality

An initial version of web services was developed to show the principle of uploading and downloading of building, weather, and optimization result files. The following functions are implemented in the prototype:

- collect and store building management system (BMS) data from the test sites on the EnRiMa server (as e.g. historical room temperatures, ambient air temperature, heating load, cooling load, electricity demand)
- collect and store weather forecast from e.g. Weather Underground where global weather data are available or from local weather data provider as e.g. at KUBIK. An initial weather forecast quality analysis has been done to check the quality of the Weather Underground service (see Appendix I - Web Service Server & Client Source Code and Appendix II - Weather Forecast Quality Analysis).
- store operational EnRiMa results in the EnRiMa database (see chapter 4.5 Integration of MatLab® for Operational Optimization) (used to test the operational DSS in the prototype)
- retrieve results of the operational optimization (temperature set points) from the server and tunnel the results to the test site and store it there for BMS integration.

The following activity is still on-going:

- finalize the BACnetTM communication which should exchange data with the existing building BMS (get historical values as e.g. room temperature, ambient air temperature, energy demand for heating/cooling/electrical uses; set temperature set points as an result of the operational DSS). This task is complicated and time consuming since it involves software and BMS vendors that are not involved in the EnRiMa project. Thus, retrieving this proprietary BMS communication information is crucial in EnRiMa and might create limitations for the EnRiMa functionalities. For more information about BACnetTM see chapter Appendix III - BACnet.

4.2 Architecture Overview of the Web Service

The figure below shows the overall architecture of the data import/export web service. One master web service (WS) is used to manage all data transfers (upload & download).

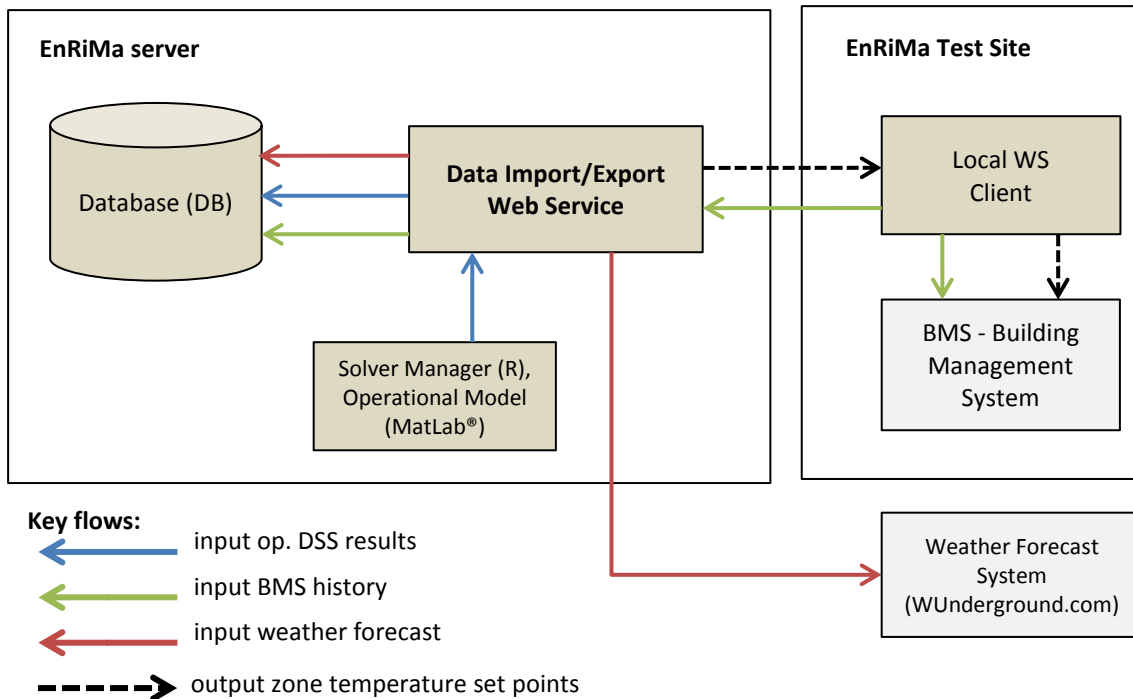


Figure 4-1: The data import/export web service architecture

The architecture has the following main constituents:

Data import/export Web Service: This is the main modules that are accessible via HTTP using standard Web Service protocols.

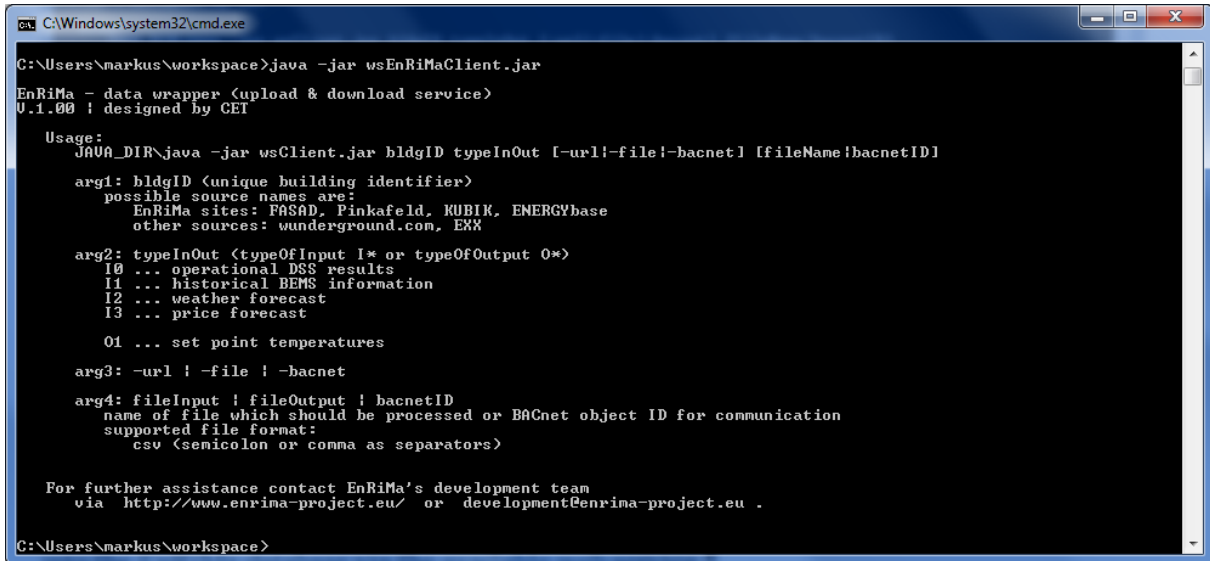
Local WS client: In order to import and export data from/to the building management system (BMS) at the test site there is a need to install Java communication client at the test site. This client communicates with the data import/export web service installed at the server. Note that the actual integration with the BMS is still under investigation.

More details about the web services server and client is available in Appendix I - Web Service Server & Client Source Code.

4.3 Usage of the Web Service Client

This section shows some examples on how the “Local WS Client” can be used to do required import and export tasks.

The following figure shows how the WS client can be used to do all required data transfers.



```

C:\Windows\system32\cmd.exe

C:\Users\markus\workspace>java -jar wsEnRiMaClient.jar

EnRiMa - data wrapper (upload & download service)
0.1.00 ! designed by CET

Usage:
JAVAA_DIR\java -jar wsClient.jar bldgID typeInOut [-url|-file|-bacnet] [fileName|bacnetID]

arg1: bldgID (unique building identifier)
possible source names are:
  EnRiMa sites: FASAD, Pinkafeld, KUBIK, ENERGYbase
  other sources: wunderground.com, EXX

arg2: typeInOut (typeOfInput I* or typeOfOutput O*)
I0 ... operational DSS results
I1 ... historical BEMS information
I2 ... weather forecast
I3 ... price forecast

O1 ... set point temperatures

arg3: -url | -file | -bacnet

arg4: fileInput | fileOutput | bacnetID
name of file which should be processed or BACnet object ID for communication
supported file format:
  csv (semicolon or comma as separators)

For further assistance contact EnRiMa's development team
via http://www.enrima-project.eu/ or development@enrima-project.eu .

C:\Users\markus\workspace>

```

Figure 4-2: Example output by using the data import/export web service client

4.3.1 Upload Operational DSS Result

The following command is used to upload the operational results (from operational EnRiMa optimization) into the EnRiMa database via the available web service:

```
C:\dev\enrima> java -jar wsClient.jar bldgID I0 -file fileName
```

Where:

bldgID reflects one of the test sites: FASAD, Pinkafeld, KUBIK, or ENERGYbase.

fileName has to be replaced by e.g. "103\1023\results.txt".

This command is used to store the operational optimization results into the EnRiMa database by use of a server-to-server communication. The whole script for the MatLab® implementation of the operational EnRiMa DSS is available in chapter 0.

4.3.2 Upload BMS Data

The following command is used to upload BMS data into the EnRiMa database via the available web service:

```
C:\dev\enrima> java -jar wsClient.jar bldgID I1 -file fileName
```

Where:

bldgID reflects one of the test sites: FASAD, Pinkafeld, KUBIK, or ENERGYbase.

fileName reflects the file which contains the historical BMS data

4.3.3 Upload Weather Forecast

The following command is used to upload weather forecast data from Weather Underground into the EnRiMa database via the available web service:

```
C:\dev\enrima> java -jar wsClient.jar bldgID I2 -url fileName
```

Where:

blDgID reflects one of the available data service providers: KUBIK, or wunderground.com.

fileName has to be replaced by the following URL if weather data should be retrieved from wunderground.com and stored within the EnRiMa database for Campus Pinkafeld: <http://api.wunderground.com/api/7b8bbd47a9bfa800/hourly/q/Austria/Pinkafeld.xml>.

4.3.4 Download BMS Set Points

The following command is used to get the optimization results back to the test site via the available web service:

```
C:\dev\enrima> java -jar wsClient.jar bldgID 01 -file fileName
```

Where:

blDgID reflects one of the test sites: FASAD, Pinkafeld, KUBIK, or ENERGYbase.

fileName has to be replaced by a filename where the optimisation results should be stored in.

At the moment the values are just shown on the screen.

4.4 Example Data

This section includes examples of the data that are currently used by the import/export web service.

4.4.1 Energy Demand from BMS

At ENERGYbase every day a file as shown below is created to reflect the hourly energy demand as well as the hourly temperatures:

[illegible]

,WS-01 BestrahlungUV-Sensor 30°,WS-02 Südfassade 60° Glas,WS-03 Südfassung 1 30°
24h_a,WS-03 Südfassung 30° global,WS-04 Schattenring Süd. 30° PV,WS-05
Vertikalausrichtung Nord (2),WS-06 Südfassung 30° PV,WS-06 Südfassung 30° PV_a,WS-
07 Aussenfeuchte,WS-08 Temp.Aussen (2),WS-09 Windgeschwindigkeit, WS-10
Aussentemperatur,Solar-35 primär Wärmemenge,Solar-30 sekundärWärmemenge,LA01-40
ErhitzerSorptionWärmemenge,LA02-40 ErhitzerSorptionWärmemenge,WP-42 Wärmepumpe-
WärmemengeVerbraucher,WP-39 Wärmepumpe-Wärmemenge-WMP2,WP-36 Wärmemenge-WMP1,LA01-
39 ErhitzerWärmemenge,LA01-41 ErhitzerPPWärmemenge,LA01-42
ErhitzerSüdWärmemenge,LA02-39 ErhitzerWärmemenge,LA02-41
ErhitzerPPWärmemenge,LA02-42 ErhitzerSüdWärmemenge,LA03-22
ErhitzerWärmemenge,LA03-23 Kühlung Wärmemenge,WP-49 KM-ZählerBrunnenpumpe 1,WP-47
KM-ZählerBrunnenpumpe 2,WP-48 KM-ZählerVerbraucher,WP-31 1-Stromverbrauch,WP-33 E-
Zähler Wärmepumpe 2,WP-34 Stromverbrauch WMP1+2,WP-32
StromverbrauchBrunnenpumpe,Solar-29 Stromverbrauchgesamt,KW-05 E-Zähler KW-
Verbraucher,WP-35 Heizungsverbraucher,BTA-21 StiegeWestNord-
ElektrischeEnergiePumpe,BTA-22 StiegeWestSüd-ElektrischeEnergiePumpe,BTA-23
StiegeOstNord-ElektrischeEnergiePumpe,BTA-24 StiegeOstSüd-
ElektrischeEnergiePumpe,LA01-37 Stromgesamt,"LA01-36
StromVentillatoren,Befeuchter,WRG","LA01-51 E-Zähler Vor-, Nachheizregister",LA01-
38 StromSorptionserhitzerElektrisch,LA02-37 Stromgesamt,LA03-19 E-Zähler,PV Summe

13/10/2012

00:00,0,0,197.91,0,0,140.5,0,213.82,95.56,11.08,0.02,11.16,330743,318997,80538,802

```

90,628170,273180,267280,19338,27691,5081,18076,27354,2989,89616,10055,288980,24424
0,557840,57203,64236,20603,54610,2335,3106,2814,1208,2833,1373,3664,4784,78275,290
,184,63626,35536,0
...
13/10/2012
23:00,0,0,197.91,0,0,371.2,0,213.82,75.5,11,3,10.8,331029,319274,80538,80290,62823
0,273180,267280,19338,27691,5081,18076,27354,2989,89616,10055,289370,244250,558220
,57203,64236,20609,54623,2336,3110,2815,1209,2834,1373,3666,4786,78277,290,184,636
31,35538,0

```

4.4.2 Weather Forecasts

4.4.2.1 Weather Underground

To enable the EnRiMa development team to develop an integrated weather forecast feature a free developer account has been created at <http://www.wunderground.com>. Details about the API are Available from: <http://www.wunderground.com/weather/api/d/docs>.

In the following sub-chapter it is shown that wunderground.com can provide the weather forecast either in an “Extensible Markup Language” (XML) or a “JavaScript Object Notation” (JSON) format. The main difference at this stage is that XML has bigger overhead within the file as JSON has. The file transfer of a weather forecast as XML file has 64 kB while a JSON file as only 44 kB. This is a reduction in transferred data by about 30%.

By using the weather underground weather forecast service we are able to get the following weather information:

- time for the forecast
- air temperature
- relative air humidity
- wind speed
- wind direction
- probability of precipitation

An information regards the solar irradiation is not available in this weather forecast service.

4.4.2.1.1 XML File from Weather Underground

The following table shows an “Extensible Markup Language” (XML) file for a 24 hour requested by the following API call (which is directly provided by wunderground.com): <http://api.wunderground.com/api/7b8bbd47a9bfa800/hourly/q/Austria/Pinkafeld.xml>:

```

<response>
  <version>0.1</version>
  <termsOfService>http://www.wunderground.com/weather/api/d/terms.html</termsOfService>
  <features>
    <feature>hourly</feature>
  </features>

```

```

<hourly_forecast>
  <forecast>
    <FCTTIME>
      <hour>21</hour><hour_padded>21</hour_padded><min>00</min><sec>0</sec>
      <year>2013</year><mon>2</mon><mon_padded>02</mon_padded>
      <mon_abbrev>Feb</mon_abbrev><mday>25</mday><mday_padded>25</mday_padded>
      <yday>55</yday><isdst>0</isdst><epoch>1361822400</epoch> <pretty>9:00 PM CET on
      February 25, 2013</pretty><civil>9:00 PM</civil><month_name>February</month_name>
      <month_name_abbrev>Feb</month_name_abbrev><weekday_name>Monday</weekday_name>
      <weekday_name_night>Monday Night</weekday_name_night>
      <weekday_name_abbrev>Mon</weekday_name_abbrev>
      <weekday_name_unlang>Monday</weekday_name_unlang>
      <weekday_name_night_unlang>Monday Night</weekday_name_night_unlang>
      <ampm>PM</ampm><tz/><age/>
    </FCTTIME>
    <temp>
      <english>31</english>
      <metric>0</metric>
    </temp>
    <dewpoint>
      <english>17</english>
      <metric>-8</metric>
    </dewpoint>
    <condition>Chance of Rain</condition>
    <icon>chancerain</icon>
    <icon_url>http://icons-ak.wxug.com/i/c/k/nt_chancerain.gif</icon_url>
    <fctcode>12</fctcode>
    <sky>73</sky>
    <wspd>
      <english>3</english>
      <metric>4</metric>
    </wspd>
    <wdir>
      <dir>WSW</dir>
      <degrees>252</degrees>
    </wdir>
    <wx/>
    <uvi>0</uvi>
    <humidity>55</humidity>
    <windchill>
      <english>-9998</english>
      <metric>-9998</metric>

```

```

</windchill>
<heatindex>
  <english>-9998</english>
  <metric>-9998</metric>
</heatindex>
<feelslike>
  <english>31</english>
  <metric>0</metric>
</feelslike>
<qpf>
  <english/>
  <metric/>
</qpf>
<snow>
  <english/>
  <metric/>
</snow>
<pop>20</pop>
<mslp>
  <english>30.19</english>
  <metric>1022</metric>
</mslp>
</forecast>
...
</hourly_forecast>
</response>

```

4.4.2.1.2 JSON File from Weather Underground

The following table shows the “JavaScript Object Notation” (JSON) file for a 24 hour requested by this API call (which is directly provided by wunderground.com): <http://api.wunderground.com/api/7b8bbd47a9bfa800/hourly/q/Austria/Pinkafeld.json>

```

{
  "response": {
    "version": "0.1"
    , "termsOfService": "http://www.wunderground.com/weather/api/d/terms.html"
    , "features": {
      "hourly": 1
    }
  }
  ,

```

```

"hourly_forecast": [
  {
    "FCTTIME": {
      "hour": "17", "hour_padded": "17", "min": "00", "sec": "0", "year": "2013", "mon":
      "2", "mon_padded": "02", "mon_abbrev": "Feb", "mday": "7", "mday_padded": "07", "yday":
      "37", "isdst": "0", "epoch": "1360252800", "pretty": "5:00 PM CET on February 07,
      2013", "civil": "5:00 PM", "month_name": "February", "month_name_abbrev":
      "Feb", "weekday_name": "Thursday", "weekday_name_night": "Thursday
      Night", "weekday_name_abbrev": "Thu", "weekday_name_unLang":
      "Thursday", "weekday_name_night_unLang": "Thursday Night", "ampm": "PM", "tz":
      "", "age": ""
    },
    },
    "temp": {"english": "32", "metric": "0"},
    "dewpoint": {"english": "26", "metric": "-3"},
    "condition": "Clear",
    "icon": "clear",
    "icon_url": "http://icons-ak.wxug.com/i/c/k/clear.gif",
    "fctcode": "1",
    "sky": "1",
    "wspd": {"english": "8", "metric": "12"},
    "wdir": {"dir": "NNW", "degrees": "347"},
    "wx": "",
    "uvi": "0",
    "humidity": "77",
    "windchill": {"english": "26", "metric": "-2"},
    "heatindex": {"english": "-9998", "metric": "-9998"},
    "feelslike": {"english": "26", "metric": "-2"},
    "qpf": {"english": "", "metric": ""},
    "snow": {"english": "", "metric": ""},
    "pop": "0",
    "mslp": {"english": "29.77", "metric": "1008"}
  }
  ,
  ...
]
}

```

4.4.3 Operational Optimization Result

The initial implementation of the operational DSS was done in MatLab®. Therefore also these results have to be saved within the database. The table below shows an exemplary result:

1	16	0	16	46.625	24.770	0.00020206	33.768	0	0	12	0.15	0.12
		0.06										
...												
24	16	0	16	33.076	19.891	0.0001317	33.076	0	0	12	0.17	
		0.15	0.08									

This table contains the following columns:

1. t: hour of day; all following parameter are related to this hour
2. zoneTemp: average air temperature within the zone according to the optimization
3. extAir: proportion of external air (HVAC system parameter)
4. saTemp: supply air temperature (HVAC system parameter)
5. heatRad: amount of heat from the radiator heating system
6. retWaterTemp: return water temperature (radiator system parameter)
7. flowWater: amount of water (radiator system parameter)
8. heatDemand: overall heating demand in the zone
9. lowAir: air flow (HVAC system parameter)
10. coolDemand: overall cooling demand in the zone
11. elecDemand: overall electricity demand in the zone
12. costElec: electricity costs
13. costHeat: heat purchase costs
14. costCool: purchase costs for cooling

4.5 Integration of MatLab® for Operational Optimization

At the moment the operational optimization engine is realized with MatLab®. In this section it is shown how the operational DSS is integrated in the overall EnRiMa DSS. If within the EnRiMa user interface the user decided to start an operational optimization the following command is started as a background job by the kernel:

```
C:\dev\enrima> matlab.cmd bldgNo caseNo bldgName
```

Where:

bldgNo: is the internal building number where an operational optimization should be performed (e.g. 1).

caseNo: is the internal case instance number to differ multiple optimization cases within the EnRiMa system (e.g. 1002).

bldgName: is the unique building name which is necessary to identify the building for the WS client to associate the according data to the according building (e.g. Pinkafeld).

By using the mentioned example values the matlab.cmd is started as follows:

```
C:\dev\enrima> matlab.cmd 1 1002 Pinkafeld
```

The last action within the matlab.cmd script is to import the optimization results by using the already available EnRiMa import/export web service (described in detail in chapter 4.2). The whole matlab.cmd script is shown in the listing below.

```
@echo off
rem *****
rem matlab.cmd
rem do an operational optimization (operative EnRiMa)
rem
rem designed be CET, 2013
rem *****

rem check parameter and available directories
IF "%1"==" " (set bldg=1) else (set bldg=%1)
IF "%2"==" " (set unique=1) else (set unique=%2)
IF "%3"==" " (set uniqueBldgName=unknownBldgId) else (set uniqueBldgName=%3)

rem define some paramters
set MATLAB_PATH=C:\Program Files (x86)\MATLAB\MATLAB Compiler
Runtime\v717\runtime\win32
set opEnRiMaStdOut=opEnRiMaStdOut.%bldg%.%unique%.txt
set ENRIMA_WS=wsEnRiMaClient.jar
set optDir=%bldg%\%unique%

if exist %optDir%\%opEnRiMaStdOut% (del %optDir%\%opEnRiMaStdOut%)
if not exist %bldg% (mkdir "%bldg%")
if not exist %bldg% (echo "problem by creating directory %bldg%" >>
%optDir%\%opEnRiMaStdOut%)
if not exist %bldg% (exit /b 1)
if not exist %bldg%\%unique% (mkdir "%bldg%\%unique%")
if not exist %bldg%\%unique% (echo "problem by creating directory %bldg%\%unique%"
>> %optDir%\%opEnRiMaStdOut%)
if not exist %bldg%\%unique% (exit /b 1)

SETLOCAL ENABLEEXTENSIONS
for /f "tokens=1-3 delims=-/ " %i in ('echo %date%') do (
set 'dd'=%i
set 'nn'=%j
set 'yy'=%k)
```

```

for /f "tokens=1-4 delims=:,./ " %%i in ('echo %time%') do (
    set 'hh'=%i
    set 'mm'=%j
    set 'ss'=%k
    set 'ff'=%l)
ENDLOCAL & SET v_Hour=%'hh'& SET v_Minute=%'mm'& SET v_Second=%'ss'& SET
v_Fraction=%'ff'& SET v_Day=%'dd'& SET v_Month=%'nn'& SET v_Year=%'yy'%
set timestring=%V_Hour%%V_Minute%%V_Second%.%v_Fraction%
set datestring=%V_Year%%V_Month%%v_Day%

rem switch to the working directory
c:
cd \dev\htdocs\enrima.eu\matlab

rem if result files exists delete them now
if exist results.txt (del results.txt)

rem start the log file
echo -----
-- >> %optDir%\%opEnRiMaStdOut%

rem copy the files required for the optimization
cd %bldg%
cd %unique%
del results.txt >> %opEnRiMaStdOut%
copy ..\..\opEnRiMa.exe . >> %opEnRiMaStdOut%
copy ..\..\dataLoad.txt . >> %opEnRiMaStdOut%
copy ..\..\results.fakeopt.txt . >> %opEnRiMaStdOut%
if not exist dataSystem.txt ( copy ..\..\dataSystem.txt . >> %opEnRiMaStdOut% )

rem if MATLAB Runtime Compiler is installed start the opEnRiMa.exe
rem otherwise copy the result template file
if exist %MATLAB_PATH% (
    set PATH=%MATLAB_PATH%;%PATH%
    echo execute opEnRiMa.exe >> %opEnRiMaStdOut%
    opEnRiMa.exe >> %opEnRiMaStdOut%
    del results.fakeopt.txt >> %opEnRiMaStdOut%
) else (
    echo copying fakeopt >> %opEnRiMaStdOut%
    copy results.fakeopt.txt results.txt >> %opEnRiMaStdOut%

```

```

del results.fakeopt.txt >> %opEnRiMaStdOut%
)

rem clean up after the optimization is done
del opEnRiMa.exe >> %opEnRiMaStdOut%
cd ..\..

rem import the result file into the EnRiMa DSS (DB) by use of the EnRiMa WS
if exist %ENRIMA_WS% (
    echo execute JAVA: java -jar %ENRIMA_WS% %uniqueBldgName% I0 -file
    %bldg%\%unique%\results.txt >> %optDir%\%opEnRiMaStdOut%
    java -jar %ENRIMA_WS% % uniqueBldgName % I0 -file %bldg%\%unique%\results.txt >>
    %optDir%\%opEnRiMaStdOut%
) else (
    echo %ENRIMA_WS% client is not available! >> %optDir%\%opEnRiMaStdOut%
)

echo matlab.cmd | FINISHED >> %optDir%\%opEnRiMaStdOut%
exit /b 0

```

4.6 Integration of KUBIK's BMS

This section describes the design of a BMS Client that can perform automatic data transfer between the local system of the KUBUIK laboratory site at Tecnalia and the DSS. The design will be implemented in forthcoming versions of the DSS. Main data transfers include:

- From KUBIK test site to the DSS: Measurements and weather forecast
- From DSS to Tecnalia server: Optimization results

The communication between KUBIK and the EnRiMa DSS will be based on the existing web service (WS) described in section 4.2 within this deliverable. Note thus that the intension with the web service as described earlier is that it should be used by all external systems that are in the need of communicating with the EnRiMa DSS.

4.6.1 BMS Client Architecture

In the premises of Tecnalia a server has been installed which acquires local information from two data sources:

- Measurements: The local measurements used in the EnRiMa project are executed by the KUBIK building management system (BMS) and stored in a KUBIK MySQL local database (kubik_db).
- Specific weather forecasts for the KUBIK location are daily generated by TecnaliaMeteo and uploaded to the Tecnalia FTP server (<ftp.tecnalia.com>). Tecnalia Meteo Area within Tecnalia Energy and Environment Division provides the weather

forecast to the Basque Government through Euskalmet trademark (<http://www.euskalmet.euskadi.net>). In fact, it is the regional Basque Government meteo public service.

Note that a common web service for transferring data from the test sites and KUBIK to the EnRima DSS is developed. The specification, structure and design of this web service is presented in the previous section.

The tecnia server pulls these two data sources periodically and stores the information in a local database (tecnalia_db). From here to the DSS data base it is transferred through the web service.

The following figure specifies the communication architecture, identifying the different services, servers and data transfer processes, in order to allow the data transfer automation from KUBIK site in Tecnia to EnRima's DSS currently hosted at parner SU.

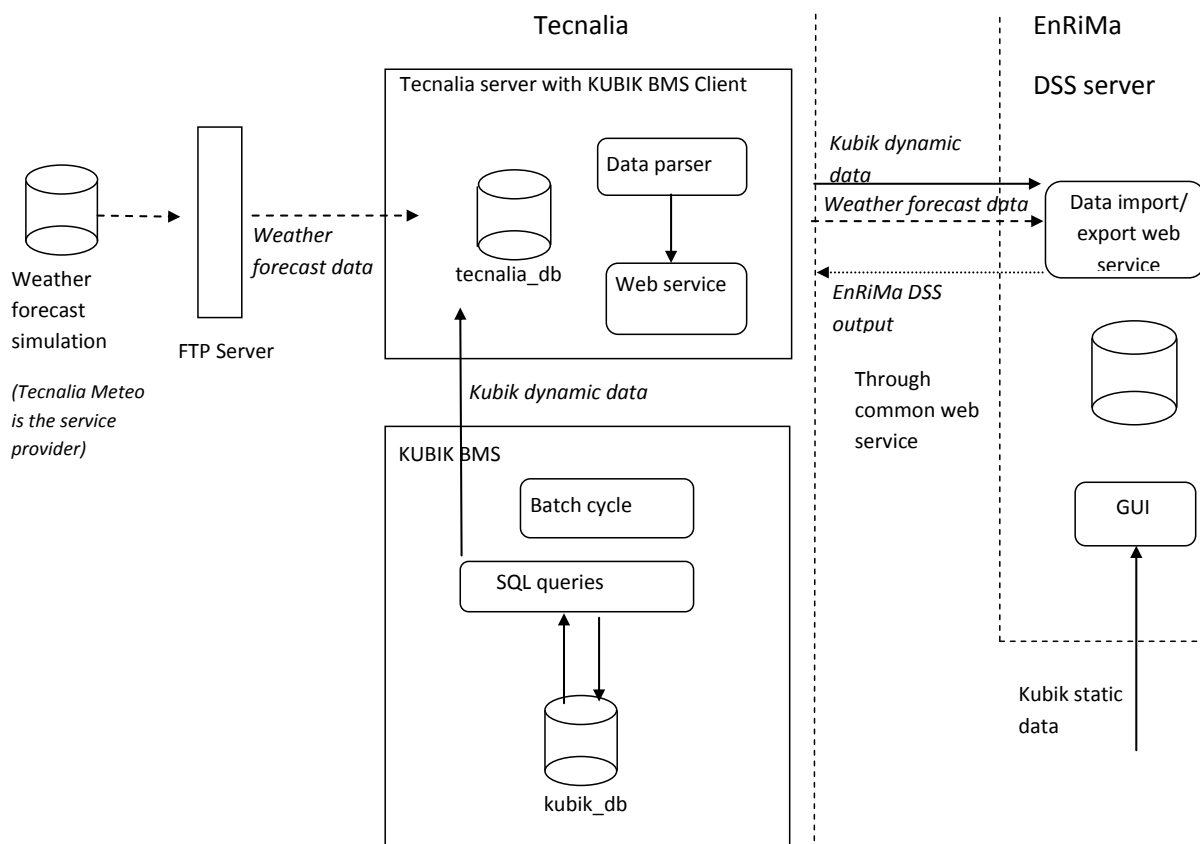


Figure 4-3: KUBIK BMS client architecture

4.6.2 BMS Client Database Structure

The local database tecnia_db is internally split in two databases, one for measurements (mangodb) and one for forecasts (GElocal).

4.6.2.1 Measurement Database

The measurements available in the KUBIK local database are periodically pulled by the Tecnia server into the measurement database, being the pulling period configurable (nowadays the pulling period is 1 minute). The information is acquired by means of a remote SQL connection to the kubik_db.

These measurements are stored in the table `pointvalues`. The time stamp of the measurements (`pointvalues.ts`) represents the instant in which the measured variable reflects a predefined threshold variation (in the case of temperature variables it is either 0.1 or 1°C). Therefore the measurements available in the table `pointvalues` are not evenly distributed in time and represent the instants when the variables change their values.

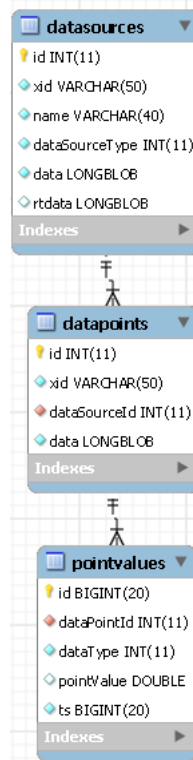


Figure 4-4: Measurement database schema.

4.6.2.2 Forecast Database

The information from the weather forecast is acquired through an FTP datasource and stored in the forecast database. This datasource manages a FTP client that connects periodically (now daily) to the Tecnia FTP server so as to download the CSV files that contain the weather forecasts provided by TecniaMeteo. The forecasted values are stored in the table `testimdata`.

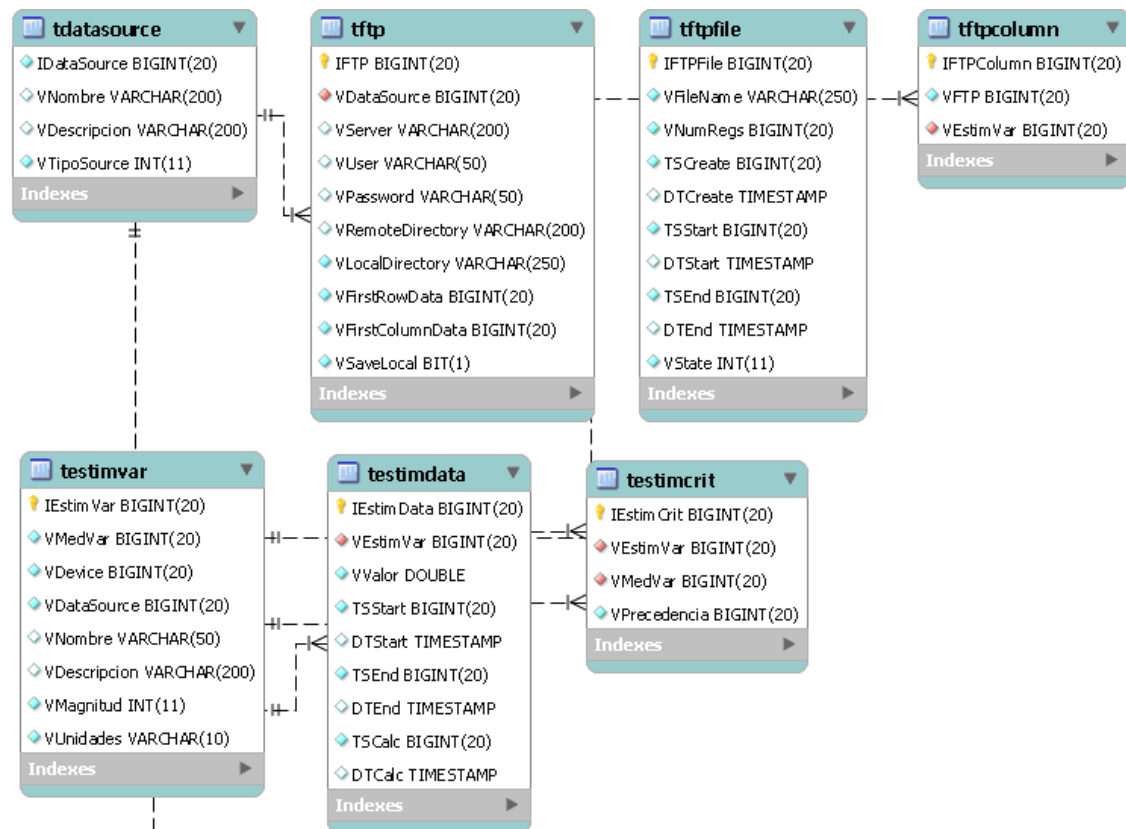


Figure 4-5: Forecast database schema.

4.6.3 Database Structure used by the Data Transfer

EnRiMa's DSS database contains tables which stores information and make it available to other modules: such as the solver of the strategic module or operational module.

The tables identified below are the ones used for transferring the data from:

- The weather forecast service to the EnRiMa DSS.
- The KUBIK parameters: static ones through the user interface and dynamic ones from the data transfer automation.

Or from the DSS to the site.

	ID	BldgID	Weather Type: H=Historic value; F=Forecast Value	Date	Time	AirTemperature	WindSpeed	WindDirection	SolarRadiation	AirRelativeHumidity
<input type="checkbox"/>	1	4	H	2012-10-13	00:00:00	11.16	0.02	0	0.0	0.0
<input type="checkbox"/>	2	4	H	2012-10-13	01:00:00	11.14	0.02	0	0.0	0.0
<input type="checkbox"/>	3	4	H	2012-10-13	02:00:00	11.09	0.02	0	0.0	0.0
<input type="checkbox"/>	4	4	H	2012-10-13	03:00:00	11.84	0.02	0	0.0	0.0
<input type="checkbox"/>	5	4	H	2012-10-13	04:00:00	12.46	0.02	0	0.0	0.0

Figure 4-6: IO_Weather table, data example.

Weather (Figure 4-6): Parameters of the weather forecast ("F" value in "Weather type" column) or of the weather real measured values ("H" value in "Weather type" column).

			Id	Version	BldgId	Date	Time	SetpointType	Value
<input type="checkbox"/>			1	1	1	2013-01-24	00:00:00	O	10.00
<input type="checkbox"/>			2	1	1	2013-01-24	01:00:00	O	11.00
<input type="checkbox"/>			3	1	1	2013-01-24	02:00:00	O	10.00
<input type="checkbox"/>			4	1	1	2013-01-24	03:00:00	O	11.00
<input type="checkbox"/>			5	1	1	2013-01-24	04:00:00	O	10.00

Figure 4-7: IO_ZoneTemp table, data example

Zone temperature (Figure 4-7): Parameters of the real zone temperature set-point (“H” value in “Set-point type” column) or the result of the optimization given by the DSS (“O” value in the “Set-point type” column).

			Id	CaseInstanceId	Version	FloorArea	GlassArea	Volume	WallArea	SpaceSolarTechnologies
<input type="checkbox"/>			200	1000	16	2088.9	426	14580.4	6143	1980
<input type="checkbox"/>			201	1001	1	4	5	6	7	0
<input type="checkbox"/>			202	1002	2	1000	600	3000	400	2000
<input type="checkbox"/>			203	1003	1	2	3	4	5	6

Figure 4-8: BuildingProp table, data example

Buildings (Figure 4-8): Static parameters of the building to be introduced by the user through the interface. The data base is located in a server identified with the following DNS name: atlas.dsv.su.se.

4.6.4 Transfer processes

This section will specify the files or data transfer associated to four processes between the different servers in the previously specified architecture:

1. Transfer of weather forecast files containing weather forecast data provided by TecniaMeteo. These files are uploaded into the FTP server by TecniaMeteo and downloaded by the Tecnia EnRiMa server.
2. Query of values measured in KUBIK building and available in Kubik database to the Tecnia EnRiMa server.
3. Transfer of data (values of the weather forecast and measurements of Kubik) from the Tecnia EnRiMa server to the Stockholm University EnRiMa server via web services.
4. EnRiMa DSS output from Stockholm University EnRiMa server to Tecnia EnRiMa server via web services.

4.6.4.1 Weather forecast: From Tecnia Meteo to the FTP server

Tecnia’s Meteo Area owns Meteo models able to provide weather forecasts. This model should be run automatically once a day, at a certain hour (for instance 23:00), and the results stored in a CSV file. The name of the CSV file follows the format: “Derio_yyyyMMddhhmmss_yyyyMMddhhmmss_yyyyMMddhhmmss.txt”.

The 3 date fields in the filename represent:

- First field: Forecast date. The date when the forecast model has been executed.

- Second field: Start date. The start date of the first data period stored in the file.
- Third field: End date. The end date of the last data period stored in the file.

The file will contain data forecasts in Derio, where the KUBIK building is located:

- temperature (°C),
- relative humidity (%),
- wind speed (km/h)
- and solar irradiation (W/m²)

The values will be given in one hour time intervals for the following 24 hours.

Next lines show the format of the data contained in the file:

```
20130204000000 20130204010000 6.7 88 17.1 0
20130204010000 20130204020000 6.9 91 15.2 0
20130204020000 20130204030000 7.2 93 13.5 0
...
20130204130000 20130204140000 7.2 93 9.5 308
...
20130204230000 20130205000000 10.2 95 7.2 0
```

The first two fields represent the start and the end of each data period, and the following four fields represent the data itself.

The file containing the results of the forecasts will be transferred to the FTP server (<ftp.tecnalia.com>) at the folder (/Meteo) and secured by login and password.

4.6.4.2 Weather forecast: from the FTP server to Tecnalia database

A java application will be periodically launched to check if there are new forecast files available in the FTP server. In that case, those files are parsed and the contained information will be stored in the forecast database (table `testimdata`). The filename of the already processed files will be stored in the table `tftpfiles`, so that in subsequent connections the information is not parsed again. The format of the CSV files is defined in the table `tftpcolumns`, so that the Java application can parse the files correctly.

4.6.4.3 KUBIK Measurements: from KUBIK Database to Tecnalia Database

The KUBIK database is a MySQL database, where different values measured by KUBIK sensors are registered.

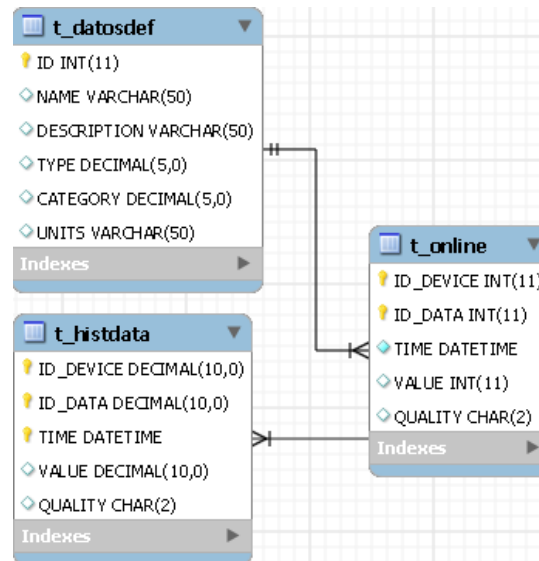


Figure 4-9: Kubik server. Database schema.

These sensors are identified by ID identifiers in the KUBIK database (table `t_datosdef`).

Concretely for the EnRiMa project, the agreed and needed values to be registered selected and transferred to the EnRiMa DSS database through the Tecnalia server and associated database are:

- External measured temperature (ID=36).
- Internal temperature set-point in the KUBIK cell under test (ID=280)
- Internal measured temperature in the KUBIK cell under test (ID=65).

Between brackets, it is specified the ID number of the magnitude in the KUBIK data base in table `t_datosdef`.

The KUBIK database registers these magnitudes once the value has changed from a given threshold. The reason for this is not to overload the database.

The table `t_histdata` holds all the measurements while the table `t_online` data just holds the last measured value of each variable.

A Java application running in the Tecnalia server periodically queries `t_online` so as to check if a new measurement is available for the three monitored variables in EnRiMa. Therefore, the measurement profile associated to each variable is exactly the same both in the KUBIK database and the Tecnalia database. The delay for having this information updated in the Tecnalia database depends on the pulling period of the application, which is now 1 minute.

4.6.4.4 KUBIK Measurements: from Tecnalia Database to DSS

However, the EnRiMa application, for simpler implementation and search in EnRiMa's database would need to receive from the Tecnalia server the information regarding measurements synchronized in a fixed measurement period, i.e.15 minutes.

So, an application is needed to:

- Query the measurement database for acquiring the value profile of all the EnRiMa variables within a certain communication period.
- Sample the profiles depending on the configured measurement period.

- Generate a common file with synchronized information for all the variables.
- Send that information to DSS by means of a web service.

The communication period will be equal to or multiple of the measurement period, depending on the compromise desired between data traffic load and update frequency in the DSS server. For instance, if the communication period is one hour the information sent to DSS will be parsed in four 15 minute period registers.

4.6.4.5 *Weather Forecast: from Tecnalia Database to DSS*

In this case, the information available in the forecast database is already synchronized in fixed hourly periods. Anyway, another application is needed to query the information and send it to DSS after it is downloaded from the FTP server. This application will use the same web service client as the application that sends the KUBIK measurements.

4.6.4.6 *DSS Output: from DSS to Tecnalia Database*

The output from the DSS, and concretely from the operational module is stored in table IO_ZoneTemp (see table 3 of this report).

The user executes the analysis via the user interface, once he/she introduces the static parameters of the building through the interface, and provided that the dynamic weather forecast data and KUBIK data have been automatically transferred.

The question is how the Tecnalia serves notices that a certain result has been stored in the table. With this purpose, a flag will be available in a specific table of the database . If the flag value is 24, then the DSS has stored certain set-points for the next 24 hours. If the flag value is 6, then the DSS has stored the optimum set-points for the following 6 hours. If the Flag value is 0, then no optimization result has been stored.

The time step of the DSS output is always one hour.

This table of the database would be composed by a flag index, the flag value and the time when the values have been stored in the database. The existence of a new flag index will be the indicator of having a new output result.

The server side of the web service will read the value of the highest index flag every time it receives the Tecnalia client web service request GET method. In case it is higher than the previously read index, the flag value and the related results will be retrieved. In case it is not, no information will be retrieved.

4.7 Integration of Pinkafeld's & ENERGYbase's BMS

This section shows an integration EnRiMa WS approach if the BMS environment is easier as mentioned in the chapter before. The data collection and distribution process at Campus Pinkafeld and ENERGYbase is quite easy. On a daily base the DESIGO building management system or more specific the DESIGO Insight Manager creates an ASCII file which contains all required values on an hourly base. The WS client presented in chapter 4.3 is executed on the BMS client to initiate the communication with the EnRiMa DSS via the central web service. The WS client can be used to upload this file daily and to download the optimization results back to the test site. All communication is initiated from the test site.

Therefore also firewalls are no problems. Figure 4-10 shows a simple layout of a future full automatic data processing environment for Campus Pinkafeld.

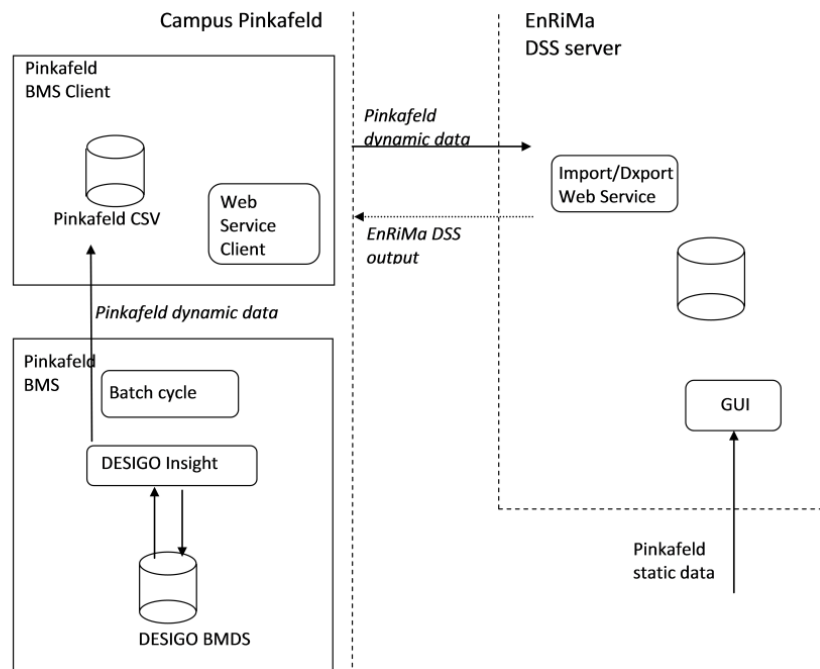


Figure 4-10: Possible Campus Pinkafeld data processing environment

5 Scenario Generator Implementation Overview

The scenario generation tool creates a scenario tree representing possible decisions that affect the building management. The scenario tree is based on the configuration values as set by the user via the GUI. The output from the scenario generator is fed into the solver manager that produces the solution that is shown to the user.

The input to the scenario generator is data about the parameters with their statistical properties and about the structure of the scenario tree. The input data is fetched by the tool from the Kernel database, where the tool stores the output as well. Essentially, the tool transforms the information the user has about parameters to information which can be used by the optimization model. In this sense, the tool does not generate new information; it merely processes all available information in the best possible way. The tool itself is embedded within the DSS Engine, making it (normally) invisible to the average user. For a more advanced user, input and output values may be visible as they are communicated to and from the tool, but beyond this, the tool will appear as a "black box" also to these users. The prototype of the tool is currently running at servers located at SINTEF, but will be moved to the DSS Engine server when it will be fully integrated.

A detailed description of the tool is given in deliverable D3.2, Scenario generation software tool (SINTEF, 2012).

6 Solver Manager Implementation Overview

The Solver manager is the module within the DSS Engine responsible for providing the solution for stochastic optimization problems. This module can work as an independent tool, though in the context of EnRiMa it works in coordination with the rest of the DSS Engine modules: the Scenario Generation tool and the Kernel. As a low-level tool (that is, running without direct interaction with end-users), it takes the models and parameters provided by other modules through the common DSS database. Then, the solver manager prepares these input data, applies the appropriate algorithms and computations, and it returns a solution. Once the solution is ready, it is delivered to the kernel database so that the GUI can retrieve it directly.

The Solver manager interacts with the Kernel through a data communication and preparation interface written in Python. The information exchange with the Kernel, and thereby the other DSS modules, is done via the Kernel database. In turn, the core of the Solver manager includes additional data preparation (both input and output) modules that interface with the optimizer, following suitable protocols. These data preparation modules in the core of the Solver manager have been created using the R statistical software and programming language. The interface for the optimizer is generic, in the sense that different abstract models can be defined to match specific scenarios and problems. Once the abstract model has been defined, multiple instances can be executed, each one running a specific stochastic optimization problem, receiving the new parameters defined by end-users on the GUI. The prototype of the solver manager is currently running at servers located at partner URJC, but will be moved to the DSS server when it will be fully integrated.

Please refer to deliverable D4.3, Stochastic optimization prototype, (URJC, 2013) for additional information and details about the internal structure of the Solver manager.

7 Conclusion

This deliverable describes the kernel prototype implementation. Currently, the prototype is able to handle the input and retrieval of information that is needed to configure and run both the operational model and the strategic model. The data services of the kernel are currently in use and serve the user interface with a structured and version controlled access to the information. The solver for the operational model is connected to the kernel and thus provides the ability to run the operational model (via the user interface). Moreover, as described in this deliverable, an import/export web service is implemented that can be used in the communication with external systems, including building management systems. A import/export web service client is developed that can easily be distributed and operated.

Next step of the implementation is to integrate the solver for the strategic model, as well as the scenario generator. The project will also implement the design of the KUBIK BMS Client as described in this document. This will enable the EnRiMa DSS to communicate with the BMS at the KUBIK laboratory site.

Acknowledgements

This deliverable is created in a joint effort of several authors. Martin Henkel (SU) and Janis Stirna (SU) have acted as editors and contributors, Wayne Westmoreland (SU) provided the overview of the database. Markus Groissböck (CET) contributed with the import/export web service description, the description of the database for the operational model, the integration of the operational solver, and also Appendix I and III, while David Berger (CET) wrote Appendix II. Moreover Eugenio Perea and Ana Mera (TECNALIA) contributed with the section on the design of the BMS Client at KUBIK. Marek Makowski, Hongtao Ren, and Janusz Granat (IIASA) contributed in particular to the description of the generic kernel design. Michal Kaut and Adrian Tobias Werner (SINTEF) wrote the description of the scenario generator tool, while Emilio Lopez Cano and Felipe Ortega (URJC) wrote the description of the solver manager.

In the course of the internal review of this document Michael Stadler (CET), Emilio Lopes Cano and Felipe Ortega (URJC) provided valuable comments.

We would like to thank the Austrian Federal Ministry for Transport, Innovation and Technology that also supports the Center for Energy and innovative Technologies (CET) through the “Building of Tomorrow” program. Additionally, the Theodor Kery Foundation of the province of Burgenland also supports the Center for Energy and innovative Technologies (CET) in course of EnRiMa. We also want to thank the University of Applied Science at Pinkafeld and University of Applied Science at Vienna (ENERGYbase) for their support.

References

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 (2013), Stochastic Optimization Prototype, EnRiMa Deliverable D5.3, European Commission FP7 Project Number 260041.

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

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

SU, CET, HCE (2013), GUI prototype and evaluation, EnRiMa Deliverable D5.2, European Commission FP7 Project Number 260041.

SU, IIASA, SINTEF, URJC, and CET (2012). Draft Specification for Services and Tools. EnRiMa Deliverable D5.1, European Commission FP7 Project Number 260041.

Appendix I - Web Service Server & Client Source Code

The web service is based on the “Axis2 Web Services” provided within the development environment Eclipse. Four packages have been created within this WS server and client software package:

- *eu.enrima.ws.processfile* contains all required files for the WS server and client interactions
- *eu.enrima.ws.processfile.retrieve* contains the code required for the retrieve data features
- *eu.enrima.ws.processfile.save* contains the code required for the save data features
- *eu.enrima.ws.processfile.tools* contains some generic tools and functions required in more than one package/class

Package *eu.enrima.ws.processfile*

The package *eu.enrima.ws.processfile* contains of four Java classes whereof two have been created completely by the Axis2 component within Eclipse:

- *ProcessFileWSCallbackHandler.java*
- *ProcessFileWSStub.java*

A main class for the WS server is the class *ProcessFileWS.java* (within the package *eu.enrima.ws.processfile*) which represents the main functions of the web services. It is shown in the list below.

```
// designed be CET, 2013
package eu.enrima.ws.processfile;
import eu.enrima.ws.processfile.retrieve.*;
import eu.enrima.ws.processfile.save.*;
import eu.enrima.ws.processfile.tools.Tools;

public class ProcessFileWS {

    public String retrieve (String bldgId, String importId, String date, String
time, String objectId) {
        RetrieveFile rfP = new RetrieveFile();
        rfP.doIt(bldgId, importId, date, time, objectId);
        return rfP.retCode;
    }
}
```

```

    public String save (String bldgId, String importId, String date, String time,
String objectId) {
        SaveFile sfP = new SaveFile();
        sfP.doIt(bldgId, importId, date, time, objectId);
        return sfP.retCode;
    }

    public int addTwoNumbers(int firstNumber, int secondNumber){
        return firstNumber + secondNumber;
    }
}

```

The main class for the WS client *ProcessFileWSclient.java* (within the package *eu.enrima.ws.processfile*) is shown below and is responsible for the interaction with the customer on the customer site (also known as Local WS client).

```

// designed be CET, 2013
package eu.enrima.ws.processfile;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import org.apache.axis2.client.Options;
import org.apache.commons.codec.binary.Base64;
import eu.enrima.ws.processfile.ProcessFileWSStub.Retrieve;
import eu.enrima.ws.processfile.ProcessFileWSStub.RetrieveResponse;
import eu.enrima.ws.processfile.ProcessFileWSStub.Save;
import eu.enrima.ws.processfile.ProcessFileWSStub.SaveResponse;
import eu.enrima.ws.processfile.retrieve.RetrieveFile;
import eu.enrima.ws.processfile.save.SaveFile;
import eu.enrima.ws.processfile.tools.Tools;

@SuppressWarnings("unused")

```

```

public class ProcessFileWSClient {

    /**
     * @param args
     * @throws Exception
     */
    public static void main(String[] args) throws FileNotFoundException,
        InterruptedException, MalformedURLException, IOException {
        String strReadFileName = null, strToTransfer = null, line = null;
        String strFileType = null, strBldgId = null, strTypeOfImport = null;
        int typeOfImport = 0, sizeString = 0;
        URL url = null;
        File file = null;
        InputStream is = null;
        BufferedReader inReader = null;
        boolean urlExists = true;
        ProcessFileWSStub stub = new ProcessFileWSStub();

        Tools.doLog("");
        Tools.doLog("EnRiMa - data wrapper (upload & download service)");
        Tools.doLog("V.1.00 | designed by CET");
        Tools.doLog("");
        if(args.length != 4) {
            Tools.doLog(Level(1) + "Usage:");
            Tools.doLog(Level(2) + "JAVA_DIR\\java -jar wsClient.jar bldgID typeInOut [-url|-file|-bacnet] [fileName|bacnetID]");
            Tools.doLog("");
            Tools.doLog(Level(2) + "arg1: bldgID (unique building identifier)");
            Tools.doLog(Level(3) + "possible source names are:");
            Tools.doLog(Level(4) + "EnRiMa sites: FASAD, Pinkafeld, KUBIK, ENERGYbase");
            Tools.doLog(Level(4) + "other sources: wunderground.com, EXX");
            Tools.doLog("");
            Tools.doLog(Level(2) + "arg2: typeInOut (typeOfInput I* or typeOfOutput O*)");
            Tools.doLog(Level(3) + "I0 ... operational DSS results");
            Tools.doLog(Level(3) + "I1 ... historical BEMS information");
            Tools.doLog(Level(3) + "I2 ... weather forecast");
            Tools.doLog(Level(3) + "I3 ... price forecast");
            Tools.doLog("");
            Tools.doLog(Level(3) + "O1 ... set point temperatures");
        }
    }
}

```

```

Tools.doLog("");
Tools.doLog(Level(2) + "arg3: -url | -file | -bacnet");
Tools.doLog("");
Tools.doLog(Level(2) + "arg4: fileInput | fileOutput | bacnetID");
Tools.doLog(Level(3) + "name of file which should be processed or BACnet
object ID for communication");
Tools.doLog(Level(3) + "supported file format:");
Tools.doLog(Level(4) + "csv (semicolon or comma as separators)");
Tools.doLog("");
Tools.doLog("");
Tools.doLog(Level(1) + "For further assistance contact EnRiMa's development
team");
Tools.doLog(Level(2) + "via http://www.enrima-project.eu/ or
development@enrima-project.eu .");
Tools.doLog("");
System.exit(0);
}
else {
    strBldgId = args[0];
    if (strBldgId.substring(0,5).equals("FASAD") //
strBldgId.substring(0,9).equals("Pinkafeld") ||
        strBldgId.substring(0,16).equals("wunderground.com") //
strBldgId.substring(0,3).equals("EEX") ||
        strBldgId.substring(0,5).equals("KUBIK") //
strBldgId.substring(0,9).equals("ENERGYbase") ) {
        Tools.doLog("origin of data: " + strBldgId);
    }
    else {
        Tools.doLog("Unknown data source: " + strBldgId);
        Tools.doLog("");
        Tools.doLog(Level(1) + "For further assistance contact EnRiMa's
development team");
        Tools.doLog(Level(2) + "via http://www.enrima-project.eu/ or
development@enrima-project.eu .");
        Tools.doLog("");
        System.exit(0);
    }

    // check if type of import is numeric and between 1 and 3
    strTypeOfImport = args[1];
    if(strTypeOfImport.equals("I0")) {
        Tools.doLog("input operational DSS results");
    }
}

```

```

    } else if(strTypeOfImport.equals("I1")) {
        Tools.doLog("input historical BEMS information");
    } else if(strTypeOfImport.equals("I2")) {
        Tools.doLog("input weather forecast");
    } else if(strTypeOfImport.equals("I3")) {
        Tools.doLog("input price forecast");
    } else if(strTypeOfImport.equals("O1")) {
        Tools.doLog("output optimized temperature set points");
    } else {
        Tools.doLog("Unknown data source: " + strTypeOfImport);
        Tools.doLog("");
        Tools.doLog(Level(1) + "For further assistance contact EnRiMa's
development team");
        Tools.doLog(Level(2) + "via http://www.enrima-project.eu/ or
development@enrima-project.eu .");
        Tools.doLog("");
        System.exit(0);
    }

    strFileType = args[2];
    strReadFileName = args[3];

    if ( strFileType.contentEquals("-url") ||
        strFileType.contentEquals("-file") ||
        strFileType.contentEquals("-bacnet") ) {
        if ( strFileType.equals("-url") ) {
            Tools.doLog("input remote file");
        } else if ( strFileType.equals("-file") ) {
            Tools.doLog("input local file");
        } else if ( strFileType.equals("-bacnet") ) {
            Tools.doLog("BACnet communication");
        }
    } else {
        Tools.doLog("Argument3 must be either '-file', '-url' or '-bacnet'. Is '"
+ strFileType + "'");
        System.exit(0);
    }
    Tools.doLog("");

```

```

        urlExists      =      checkIfWSEExists(SaveService.SAVESERVICE_WSDL_LOCATION.
toString());
        if ( !urlExists ) {
            Tools.doLog("Error: EnRiMa WS is NOT up and running or is NOT
accessible!");
            Tools.doLog("    Please conntact your EnRiMa partner.");
            Tools.doLog("");
            System.exit(0);
        }
        else {
            Tools.doLog("EnRiMa WS is up and running.",1);
        }
    }

    if ( strTypeOfImport.substring(0,1).equals("I") ) {
// INPUT procedure
        if ( ! strFileType.equals("-bacnet") ) {
            try {
                if ( strFileType.equals("-url") ) {
                    url = new URL(strReadFileName);
                    inReader      =      new      BufferedReader(new
InputStreamReader(url.openStream()));
                }
                else {
                    file = new File(strReadFileName);
                    inReader = new BufferedReader(new FileReader(file));
                }

                // Read the file
                strToTransfer = "";
                while ((line = inReader.readLine()) != null) {
                    if ( strToTransfer.length() == 0 )
                        strToTransfer = line;
                    else
                        strToTransfer = strToTransfer + "\n" + line;
                }

                // Close the output stream
                inReader.close();

```

```

        } catch (IOException e) {
            if ( strFileType.equals("-url") ) {
                Tools.doLog("Argument4 must be an valid / existing remote file name. Is '" + strReadFileName + "'!");
            }
            else {
                Tools.doLog("Argument4 must be an valid / existing local file name. Is '" + strReadFileName + "'!");
            }
            System.exit(1);
        }
    }

    // encoding byte array into base 64
    byte[] encoded = Base64.encodeBase64(strToTransfer.getBytes());

    // and convert it into a String
    String strEncoded = new String(encoded);

    // call the web service to transfer the file to the EnRiMa web server
    Tools.doLog("Initiate data transfer to EnRiMa's file transfer web service ... ",0);

    // define parameter for save WS
    Save sav = new Save();
    sav.setBldgId(strBldgId);
    sav.setImportId(strTypeOfImport.substring(1,2));
    sav.setDate(Tools.GetDate());
    sav.setTime(Tools.GetTimeS());
    sav.setObjectId(strEncoded);

    // call save WS
    SaveResponse saveRes = stub.save(sav);
    sizeString = saveRes.get_return();
    if ( sizeString > 0 ) {
        Tools.doLog("Successfully transfered " + sizeString + " bytes.",1);
        System.exit(0);
    }
    else {
        Tools.doLog("Return code: " + sizeString,1);
    }
}

```



```

        Tools.doLog("Either your file or the combination of buildingNumber and
typeOfImport is invalid.",1);
        System.exit(1);
    }
}
else if ( strTypeOfImport.substring(0,1).equals("0") ) {
// OUTPUT procedure
    Tools.doLog("Initiate EnRiMa's result transfer web service ... ",1);

    // define parameter for save WS
    Retrieve ret = new Retrieve();
    ret.setBldgId(strBldgId);
    ret.setImportId(strTypeOfImport.substring(1,2));
    ret.setDate(Tools.GetDate());
    ret.setTime(Tools.GetTimeS());
    ret.setObjectId(strToTransfer);

    for ( int hour=Integer.parseInt(Tools.GetHour()); hour<=24; hour++ ) {
        String strTime = Tools.GetHour() + ":00:00";

        // call retrieve WS
        RetrieveResponse retRes = stub.retrieve(ret);
        sizeString = retRes.get_return();
        if ( sizeString > 0 ) {
            Tools.doLog("Set point temperature for " + strTime.substring(0,2) + "
o'clock: " + Double.valueOf(sizeString)/100 + " grad Celsius",1);
            System.exit(0);
        }
        else {
            Tools.doLog("Return code: " + sizeString,1);
            Tools.doLog("Either your file or the combination of buildingNumber and
typeOfImport is invalid.",1);
            System.exit(1);
        }
    }
}
else {
    Tools.doLog("Invalid input/output type: " + strTypeOfImport);
    System.exit(0);
}
}

```

```

}

private static String Level(int level) {
    int spacesPerLevel = 3;
    String strLevel = "";

    for(int i=0; i<(level*spacesPerLevel); i++){
        strLevel += " ";
    }

    return strLevel;
}

public static boolean checkIfWSExists(String targetUrl) {
    HttpURLConnection httpUrlConn;
    try {
        httpUrlConn = (HttpURLConnection) new URL(targetUrl).openConnection();
        httpUrlConn.setRequestMethod("HEAD");
        httpUrlConn.setConnectTimeout(5000);
        httpUrlConn.setReadTimeout(5000);

        return (httpUrlConn.getResponseCode() == HttpURLConnection.HTTP_OK);
    } catch (Exception e) {
        return false;
    }
}
}

```

Package eu.enrima.ws.processfile.retrieve

The main file within the export procedure is the class *RetrieveFile.java* (within the package *eu.enrima.ws.processfile.retrieve*) is shown below.

```

// designed by CET, 2013
package eu.enrima.ws.processfile.retrieve;
import java.sql.Connection;
import eu.enrima.ws.processfile.tools.Tools;
import eu.enrima.ws.processfile.tools.DBtools;

public class RetrieveFile {

```

```
public String retCode;

public RetrieveFile () {
}

public void doIt (String bldgId, String fileTypeId, String date, String time,
String fileContentToProceed) {
    // check if bldgId is valid
    String tmpInstanceId = null;
    Connection connect = null;

    connect = DBtools.connectDB (connect);
    tmpInstanceId = DBtools.getUniqueId(connect, bldgId);
    if ( tmpInstanceId.equals("0") ) {
        Tools.doLog("Used unique building string is invalid: " + bldgId);
    }
    else {
        Tools.doLog("Case instance ID: " + bldgId + " => " + tmpInstanceId);
        DBtools.disconnectDB(connect);
        // continue work if unique building id is available
        int intImportId = 0;
        String retValue = "";
        intImportId = Integer.parseInt(fileTypeId);

        switch (intImportId) {
            case 1:
                retValue = retrieveBEMS(tmpInstanceId, fileTypeId, date, time, bldgId);
                break;
            default:
                System.out.println("'get' bldg <" + bldgId + ">, file type '" +
fileTypeId + "' is not defined!");
                break;
        }

        retCode = retValue;
    }
}
```

```

    private String retrieveBEMS(String bldgId, String bType, String date, String
hour, String objectId) {

    String retValue;
    Connection connect = null;

    connect = DBtools.connectDB (connect);
    retValue = DBtools.retrieveBEMS(connect, bldgId, date, hour, objectId );
    DBtools.disconnectDB(connect);

    return retValue;
}
}

```

Package eu.enrima.ws.processfile.save

The main file within the import procedure is the class *SaveFile.java* (within the package *eu.enrima.ws.processfile.save*) is shown below.

```

// designed be CET, 2013
package eu.enrima.ws.processfile.save;
import java.sql.Connection;
import org.apache.commons.codec.binary.Base64;
import eu.enrima.ws.processfile.tools.DBtools;
import eu.enrima.ws.processfile.tools.Tools;

public class SaveFile {

    public String retCode;
    public Tools Log = new Tools();

    public SaveFile () {

    }

    public void doIt (String bldgId, String fileTypeId, String date, String time,
String fileContentToProceed) {

        // check if bldgId is valid
        String tmpBldgId = null, tmpInstanceId = null;
        Connection connect = null;

```

```
connect = eu.enrima.ws.processfile.tools.DBtools.connectDB (connect);
tmpBldgId = DBtools.getBldgId(connect, bldgId);
if ( tmpBldgId.equals("0") ) {
    Tools.doLog("Used unique building string is invalid: " + bldgId);
}
else {
    Tools.doLog("Unique building ID: " + bldgId + " => " + tmpBldgId);
    tmpInstanceId = DBtools.getUniqueId(connect, bldgId);
    if ( tmpBldgId.equals("0") ) {
        Tools.doLog("Used unique building does not deliver a building ID: " +
bldgId);
    }
    else {
        Tools.doLog("Case instance ID: " + bldgId + " => " + tmpInstanceId);
        eu.enrima.ws.processfile.tools.DBtools.disconnectDB(connect);
        // continue work if unique building id is available
        // decoding byte array into base64
        byte[] strByte = fileContentToProceed.getBytes();
        byte[] decoded = Base64.decodeBase64(strByte);

        // and convert in into a String
        String strDecoded = new String(decoded);
        int intFileTypeId = 0;
        intFileTypeId = Integer.parseInt(fileTypeId);

        switch (intFileTypeId) {
            case 0:
                saveOpResults(tmpBldgId, tmpInstanceId, date, time, strDecoded);
                break;

            case 1:
                saveBEMS(tmpBldgId, date, time, strDecoded);
                break;

            case 2:
                saveWeather(tmpBldgId, date, time, strDecoded);
                break;
```

```
        case 3:
            savePrice(tmpBldgId, date, time, strDecoded);
            break;

        default:
            Tools.doLog("'save' for data source <" + bldgId + ">, file type '" +
fileTypeId + "' is not defined!");
            strDecoded = "";
            break;
    }

    retCode = Integer.toString(strDecoded.length());
}
}
}

private int saveOpResults(String bldgId, String instanceId, String date, String
time, String str) {
    return OpResults.saveIt(bldgId, instanceId, date, time, str);
}

private int saveBEMS(String bldgId, String date, String time, String str) {
    return BEMS.saveIt(bldgId, date, time, str);
}

private int saveWeather(String bldgId, String date, String time, String str) {
    return Weather.saveIt(bldgId, date, time, str);
}

private int savePrice(String bldgId, String date, String time, String str) {
    return Price.saveIt(bldgId, date, time, str);
}
}
```

Appendix II - Weather Forecast Quality Analysis

Assessment for Pinkafeld Campus

As EnRiMa is a project with the aim to improve energy efficiency at different places in Europe the aim was to find a weather data provider that is able to fulfil this project requirement. Also having only one weather data provider for all European (test) sites was intended. Therefore the weather forecast service from Weather Underground (wunderground, 2013a) has been investigated in this report and an initial analysis of the weather forecast quality for Pinkafeld Campus (one of our Austrian test site) has been performed by CET.

As Weather Underground is a service provider that is able to deliver worldwide weather forecasts it has been chosen to be EnRiMa's primary weather service provider. Beside this weather forecast service other services can be considered as well. For example for KUBIK there is a TECNALIA internal approach to do the weather forecast on a daily base and collect those forecasts with the web-services and store it on the EnRiMa server.

"Weather Underground is committed to delivering the most reliable, accurate weather information possible. Our state-of-the-art technology monitors conditions and forecasts for locations across the world, so you'll always find the weather information that you need" (wunderground, 2013b).



Weather Underground monitors conditions and forecasts for locations across the world. It's a network of personal weather stations with almost 23,000 stations in the US and over 13,000 across the rest of the world (wunderground, 2013b). Everyone who owns a personal weather station (PWS) can share his data with the rest of the world. The list below shows how it works to be integrated in the wunderground.com weather services (wunderground, 2013d):

1. Purchase Weather Station Hardware
2. Placing Your Weather Station to Report Accurate Readings
3. Installation and Configuration of Software
4. Upload Your Data to Weather Underground

Weather Forecast Analysis

An initial weather forecast quality analysis for Pinkafeld Campus will be shown in this chapter by considering the weather forecast and the historical weather data from Weather Underground.

Access Weather Underground

The weather forecast data will be obtained by a wunderground.com provided API requests which are made over HTTP. The returned data can be called as a JSON- or XML format depending on the used API call. By replacing the "xml" tag through the "json" tag the XML or JSON formatted weather forecast will be delivered by wunderground.com. This file is further processed and stored in the EnRiMa database by the use of the web service described in section 4.

(<http://api.wunderground.com/api/7b8bbd47a9bfa800/hourly/q/Austria/Pinkafeld.xml>). More details about that can be found on the Weather API Introduction (wunderground, 2013c) (a screenshot of the API description is available in the picture below).

The screenshot displays the 'Weather API: Introduction' page from wunderground.com. The page layout includes a top navigation bar, a left sidebar with a 'Weather API' table of contents, a main content area, and a bottom resources section.

Weather API: Introduction

Getting Started

Before you start using the Weather API, it is important to know that

1. Most of the API features require an API key. [Sign up for a key.](#)
2. API requests are made over HTTP. Data features return JSON or XML. WunderMap layer features return image files. This documentation is full of examples of how to use API features. See [code samples for several languages](#), and [user-generated code and libraries](#).
3. Multiple API features can be combined into a single HTTP request. This is an easy way to economize your requests. The [Data Features](#) page documents how features can be combined into a single request.
4. Per the [Terms of Service](#) the Weather Underground logo must be included with a credit line where Weather Underground data is displayed. Please see individual Logo variations for all acceptable combinations of layout in the [Logo Usage Guide](#).

Example

Current Conditions in US City

http://api.wunderground.com/api/Your_Key/conditions/q/CA/San_Francisco.json [Show Response](#)

Resources

Community

- Community Forum
- Twitter Feed

Code + Tools

- Try the API with Apigee
- Code Samples
- User-Generated Code and Libraries

Images

- Weather Conditions Icon Sets
- Logo Usage Guide

Figure A2-1: Weather forecast API introduction (wunderground.com)

To get the recent weather forecast for Pinkafeld the following link was used in a browser to obtain the required weather forecast data: <http://english.wunderground.com/cgi-bin/findweather/getForecast?query=Pinkafeld,at&hourly=1&yday=65&weekday=Donnerstag>. The figure below shows the output of this link. The figure shows all data which has been required for the weather forecast analysis (point in time, temperature at this time).

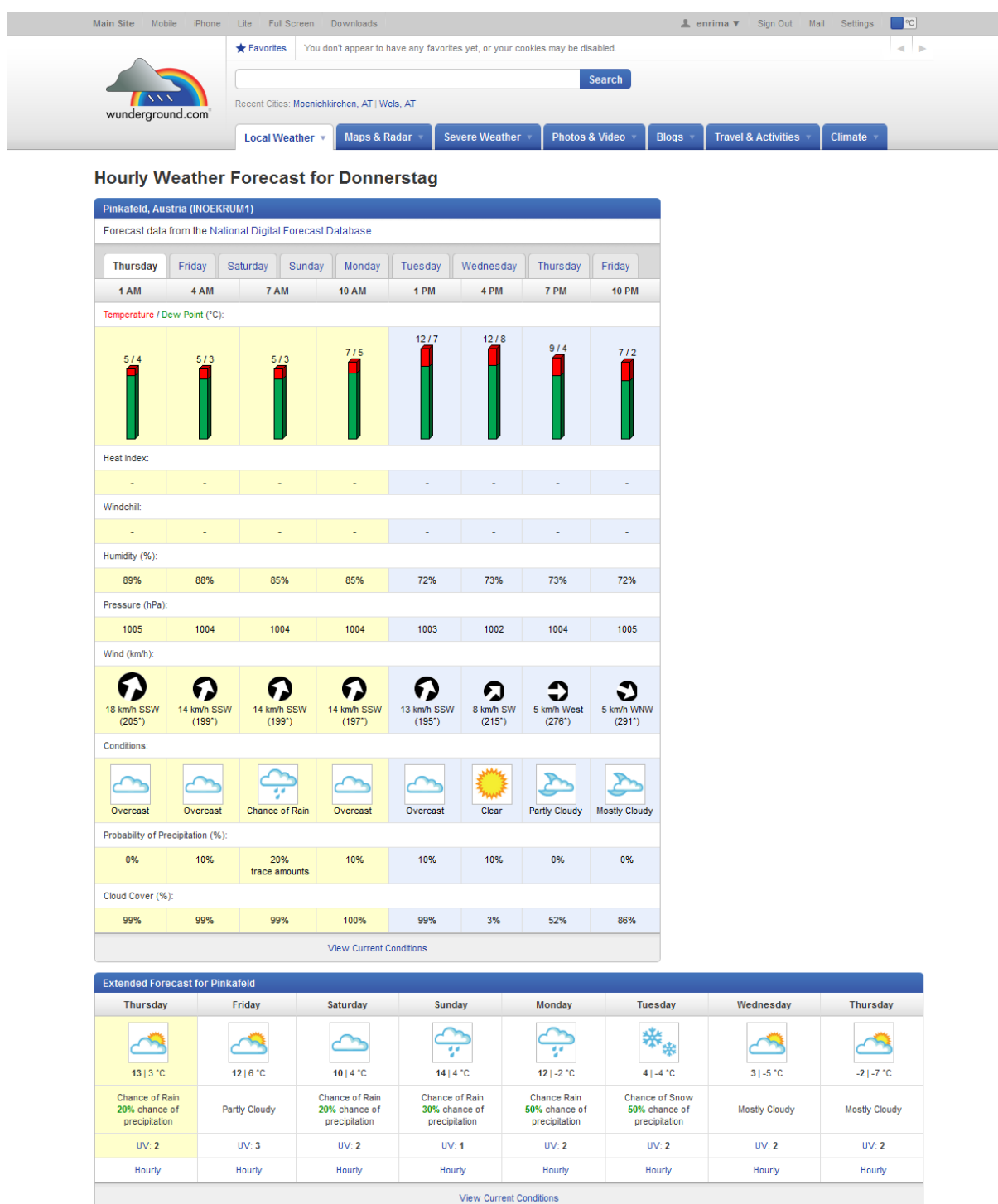


Figure A2-2: Weather forecast Pinkafeld for 7 March 2013 (wunderground.com)

For test purposes we created a developer account at Weather Underground with the ID 7b8bbd47a9bfa800. The username and the according password are available through CET.

The recent XML respectively JSON file for Pinkafeld can be accessed via the following links:

- <http://api.wunderground.com/api/7b8bbd47a9bfa800/hourly/q/Austria/Pinkafeld.xml>
- <http://api.wunderground.com/api/7b8bbd47a9bfa800/hourly/q/Austria/Pinkafeld.json>

Weather Forecast Data

For the initial weather forecast quality analysis the browser based approach (see Figure A2-2 above) has been used. The table below shows the weather forecast for 19th February to 21st February 2013 which has been evaluated every day at 8 AM. The table shows the ambient air temperature (°C) for each observed hour.

Table A2-1: Weather forecast (ambient air temperature, °C) (wunderground.com)

date / time	07:00	10:00	13:00	16:00	19:00	22:00
19.02.	-3	-1	0	-1	-3	-5
20.02.	-9	-5	-2	-5	-4	-6
21.02.	-6	-7	-4	-5	-4	-5

Historical Weather Data

The actual temperatures at Pinkafeld have been taken from the following wunderground.com link: <http://www.wunderground.com/q/zmw:00000.3.11185>. The actual weather data for Pinkafeld is shown in the table below.

Table A2-2: Actual Weather Data (ambient air temperature, °C) (wunderground.com)

date / time	07:00	10:00	13:00	16:00	19:00	22:00
19.02.	-3.6	-1.7	-0.8	-1.2	-5.6	-7.5
20.02.	-8.2	-5.3	-1.3	-3.0	-5.0	-5.8
21.02.	-7.2	-6.5	-4.1	-3.8	-5.2	n/a

Comparison and Conclusion

The following Figure A2-3 compares the weather forecast temperatures (blue line) with the historical weather temperatures (red line). Wunderground.com uses the Austrian “Zentralanstalt für Meteorologie und Geodynamik” (ZAMG, <http://www.zamg.ac.at>) to get the forecasts and the historical weather data. The green line shows the temperature difference of historical and forecasted temperatures.

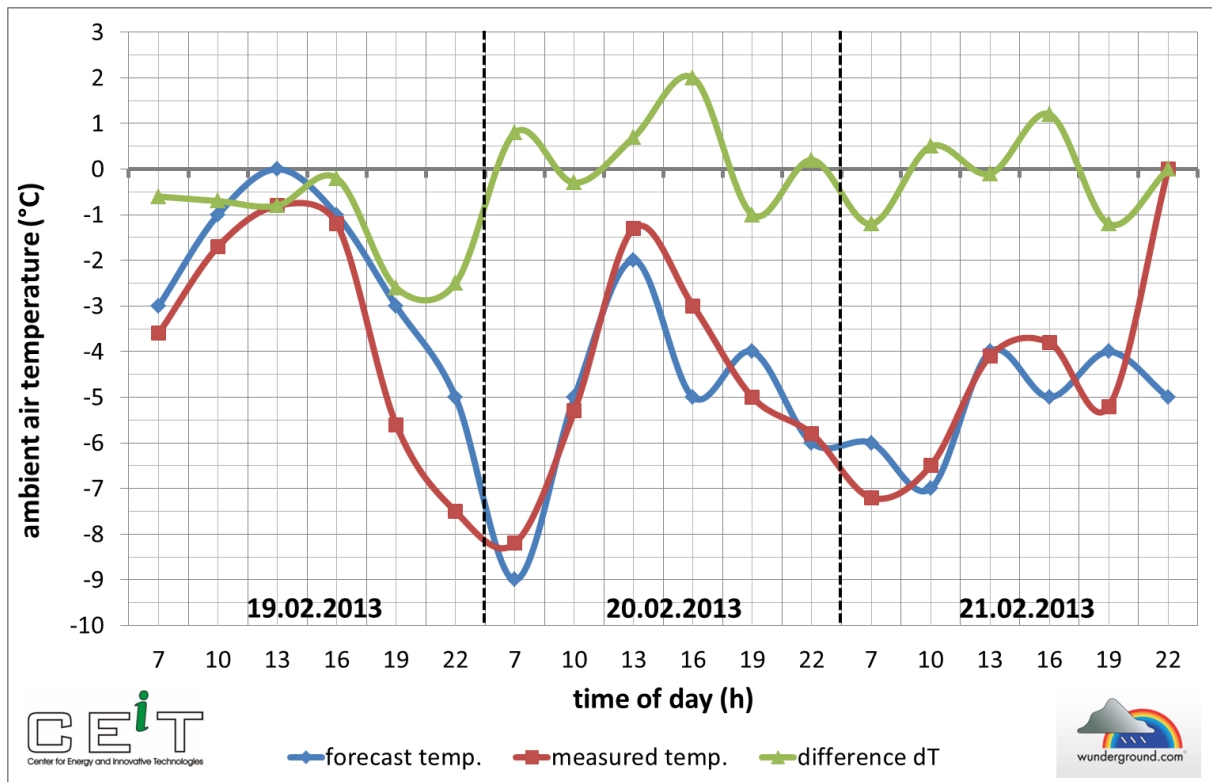


Figure A2-3: Comparison of forecast and historical weather temperature (based on wunderground.com)

The weather forecast quality from Weather Underground can be assessed with satisfactory as the difference between the weather forecast and the observed historical temperature is between +2 and -3 °C which should be sufficient.

References for Appendix II

wunderground (2013a): Weather Forecast & Reports, Long Range & Local [Online]. Available from: <http://www.wunderground.com> (Accessed: 30 January 2013).

wunderground (2013b): About Us | The First Internet Weather Forecast Service [Online]. Available from: <http://www.wunderground.com/about/background.asp> (Accessed: 7 March 2013).

wunderground (2013c): About Us | The First Internet Weather Forecast Service [Online]. Available from: <http://www.wunderground.com/weather/api/d/docs?d=index> (Accessed: 30 January 2013).

wunderground (2013d): Personal Weather Stations | Worldwide Weather Station Project [Online]. Available from: <http://english.wunderground.com/weatherstation/setup.asp> (Accessed: 19 March 2013).

Appendix III - BACnet™

This appendix describes the use of BACnet™ for the integration of the Pinkafeld site. In doing so the appendix also identifies a set of issues that need to be resolved in order to use BACnet™.

Several standards are in use for the integration of building management systems. Some of them are designed for wired (e.g. KNX - Konnex, LONWorks, BACnet™) devices while others are designed for wireless devices (e.g. Zigbee, Z-Wave, GPRS - general packet radio service, PLC – programmable logic controller). In the first sections of this appendix, we introduce the available building systems, before describing the specifics about the EnRiMa DSS and the BACnet™ protocol.

Building Automation System (BAS)

The building automation (BA) is the aggregate of monitoring, control, automatic, and optimization appliances/devices within a building. It is an important part of the facility management. The overall goal is a trade-spanning automatic process, which operates within given constraints (parameters) and simplifies the use and control of the whole system. All sensors, actors, controls, consumers and other technical equipment within a building are connected with each other via the automation controllers.

The main goal of building management and building automation is to increase efficiency, to reduce costs and emissions. Furthermore, safety could be an additional occasion to deal with building automation. The following topics are covered within building automation (Technologies, 2011):

- building management
- lighting control
- HVAC control
- energy production
- energy distribution and safety

Figure A3-1 shows an example structure of a building automation system (BAS).

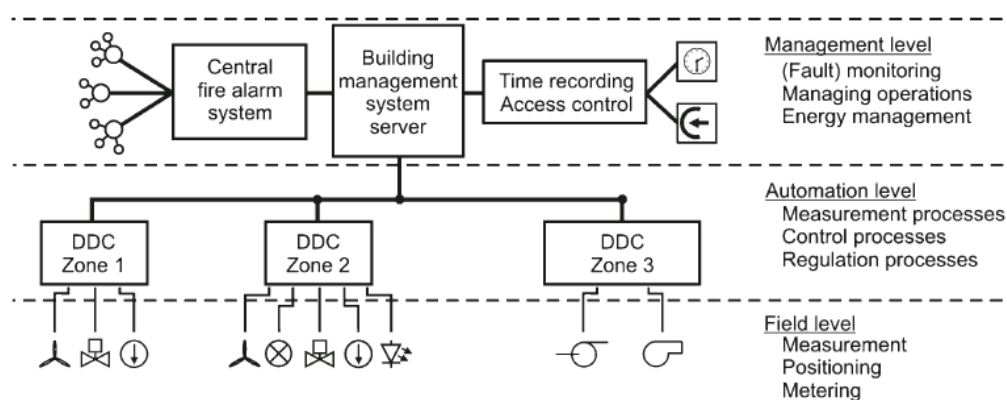


Figure A3-1: Three-layer model in building automation (Merz, Hansemann & Hübner, 2009)

Merz, Hansemann & Hübner (2009) shows three levels within the building automation system: the management level, where the human interface device is located; the automation level, where the small intelligent system – the so-called direct digital control (DSS) - is located; and the field level, where the sensors and actors are located.

BAS within EnRiMa

A major goal of EnRiMa is to optimize the short-term supply and demand like cooling, heating and electricity, as well as local distributed Energy Resources (DER). To minimize human interaction a direct communication with the test site and the building automation (BA) would be favourable. The main goal of this work is to explore this direct communication possibility and influence to set temperature set points directly.

Therefore, it is crucial to test the communication between the EnRiMa components and the building management system. In WP1 and WP2 we already demonstrated the communication from DESIGO™ to EnRiMa in one way and are now able to read data from DESIGO™, but overwriting set points and communication in the other direction seems to have a lot of barriers. Besides all these technical barriers it needs to be noted that in course of WP1 we got strong stakeholder feedback that this direct communication is not expected nor wished by the building owner due to liability issues. Thus, this BACnet communication work is just intended to demonstrate the communication for the exploitation plan in WP7 and maybe to diminish the stakeholders concerns of direct communication for the operations version of EnRiMa.

BAS at Pinkafeld Campus

A new version of DESIGO™ was implemented during autumn 2012. Some enhancements are important for the EnRiMa project.

Building Automation and Controls NETWORK (BACnet)

BACnet is an abbreviation for “building automation and controls network”. “A key design criterion (enumerated in some detail at the kick-off meeting in Nashville) was that the protocol had to be applicable to all building automation needs. To accomplish this, BACnet specifies nearly all of the most common functions: analog and binary input, output and values; control loops; schedules, etc., that clearly apply to almost any kind of monitoring or control application.” BACnet is a European standard protocol within CEN, the Committee for European Standardization (BACnet, 2012).

Required IT components and expertise

The following software tools and skills are required to fulfill the overall target of EnRiMa.

- EnRiMa server (database, web-server, solver, scenario generator)
- data wrapper (retrieve BACnet™ object details from BMS, weather data provider and so on)

- set point reader (read the optimization results from the EnRiMa server and write them into the BMS; periodical task to define the set-points). This is basically the focus of this study.
- BACnet™ enabled BMS
- BACnet™ communication software (e.g. BACnet4J, 2012) and BACnet™ software experts (e.g. Siemens)
- BACnet experts (e.g. Siemens)

As this lists contains items as “BACnet™ experts”, “BACnet™ software experts “, and “BACnet™ enabled BMS” also vendors which are not part of the EnRiMa consortium are required to fulfill this building automation tasks. Therefore it is not known at the moment of writing this deliverable if the project team is able to get in contact with the required experts (which are not part of the EnRiMa project consortium) to achieve the functionality to extend EnRiMa with this mentioned BACnet™ communication.

FAQ

Where is BACnet™ located?

It can operate within all three levels of the building automation system: the management, the automation and the field level. If a proposed Java tool should be able to interact with the management and/or automation level it is required to understand the BACnet™ protocol stack. A Java implementation of the BACnet™ protocol stack is available (BACnet4J, 2012). The protocol stack is an implementation of a computer networking protocol suite. The terms are often used interchangeably. Strictly speaking, the suite is the definition of the protocols, and the stack is the software implementation of them.

The BACnet™ implementation at Campus Pinkafeld is given by Figure A3-1 (based on information from Campus Pinkafeld) and consists of a BMS server and controller as well as several sensors and actors.

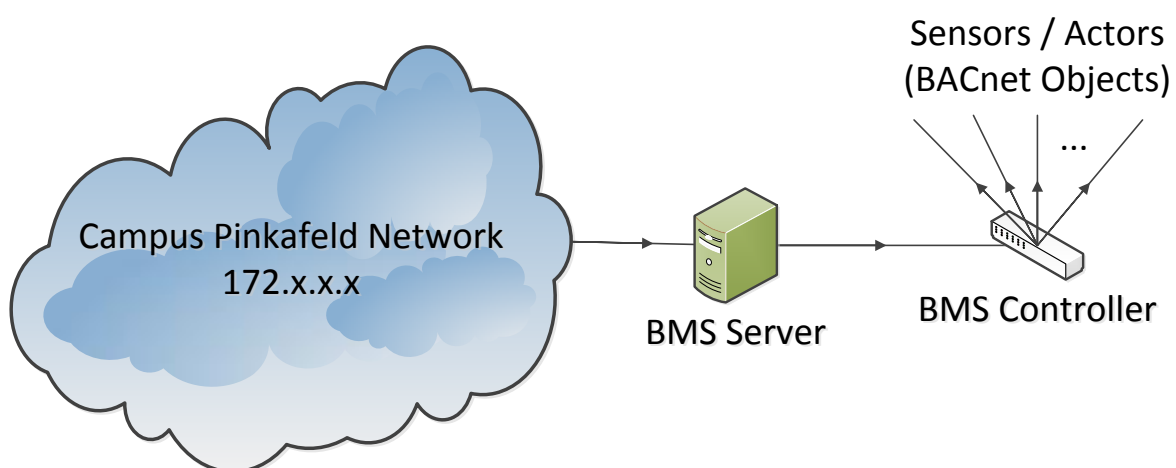


Figure A3-1: BACnet™ infrastructure at Campus Pinkafeld

Can BACnet™ be accessed behind a router/firewall?

Yes, it is possible. Figure A3-2 shows TCP/IP or more correctly UDP/IP based BACnet™ network could look like.

As the security of most of the building owners does not allow to access their building management system from outside of the building it is necessary to initiate the communication to the EnRiMa server from inside of the building.

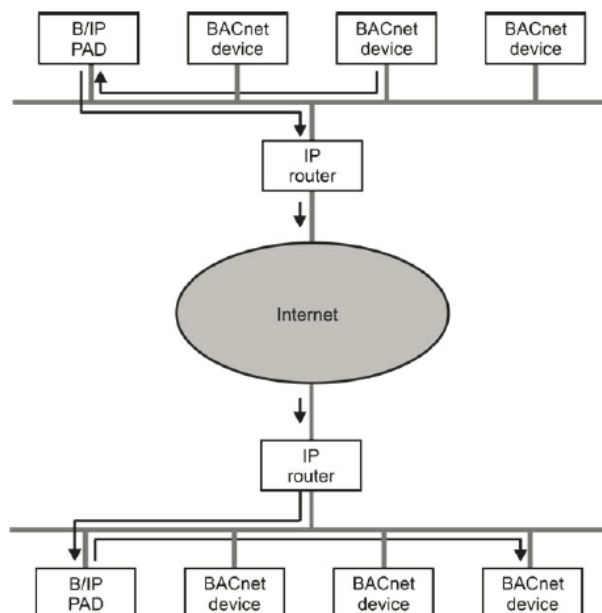


Figure A3-2: Example BACnet™/IP Network (Merz, Hanseemann & Hübner, 2009)

Figure A3-3 shows the ISO layer concept and where the BACnet stack is placed within this model.

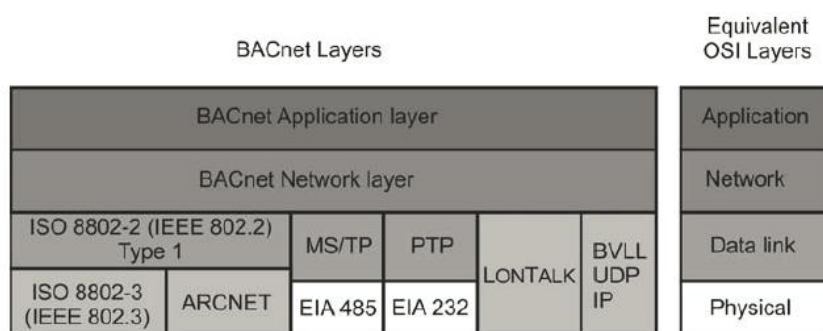


Figure A3-3: BACnet™ layers compared with OSI layers (Merz, Hanseemann & Hübner, 2009)

UDP and TCP are within layer 4 of the 7-layer ISO model. The ‘transmission control protocol’ (TCP) is a connection oriented protocol while the ‘user datagram protocol’ (UDP) is a connectionless protocol. Therefore, for UDP there is no handshake sequence necessary to exchange information. UDP is less reliable as TCP but was designed with less communication overhead.

As BACnet/IP is based on TCP/IP it can be routed via router and firewalls. Of course the BACnet port 47808 (= 0xBAC0) (for default configurations) respectively 47816 (= 0xBAC8) for the installation at Campus Pinkafeld has to be allowed.

Where is DESIGO™ located within the BAS?

DESIGO™ is part of the management level within the building automation system (BAS). But it has access to the automation level (controller) to do the required actions as e.g. set a threshold, read a recent value of a BACnet object.

How can we predefine set-points in general/in Pinkafeld?

If we want to set the values for the next e.g. 24 hours we need a tool which sets the temperature every hour. This means we need a tool at the test buildings to interact with the BMS directly. It is possible to overwrite the target value of e.g. the inlet temperature of the heating system or the inlet temperature of the air ventilation system. By doing this the required zone air temperature can be achieved. The change of a target value has to be done at least once an hour. Maybe every 15 minutes is possible as well. This depends on the definition of the time step within the operational module.

Which BACnet™ objects are available at Campus Pinkafeld?

We arranged that Mr. Michtner from Siemens Austria will deliver this list, which will allow us to test the BACnet™ communication via a java tool (see chapter 0). We are still waiting for an up-to-date BACnet™ object list for Campus Pinkafeld.

BACnet™ security

According to Mr. Michtner from Siemens Austria there is no possibility to add a security layer. If a client can access the BACnet™ network it is automatically possible to read and write all available BACnet™ objects.

It should make no difference if the BACnet™ server or the BACnet™ controller is used for the communication.

BACnet™ Configuration

Within Campus Pinkafeld the BACnet™ port 0xBAC8 (47816) is used while the default port is 0xBAC0 (47808). The BAS computer and the controller are located within the TCP/IP address field of 172.16.120.x/255.255.255.0. The TCP/IP address of the BAS computer and the controller are:

172.16.120.10 ... BMS server - BACnet-ID: 2100001, BACnet-Port: 0xBAC8 = 47816
 172.16.120.11 ... BMS controller
 172.16.120.12 ... BMS configuration laptop

The default BACnet™ port was changed from the default 0xBAC0 (47808) to 0xBAC8 (47816). The address for the BMS configuration laptop is available for Siemens during the DESIGO™ installation/update procedure. Each BACnet™ object value has to be unique and is defined by an object instance number, an object type, and an object instance. Therefore an example to address a BACnet™ object is shown below according to Mr. Michtner from Siemens Austria:

object-instance-no 2098177
 object-typ 0

<i>object-instance</i>	7
------------------------	---

Available BACnet Tools

The list below shows some JAVA-based BACnet tools:

- <http://sourceforge.net/projects/bacnet4j/> (<http://bacnet4j.sourceforge.net>)
- <http://bacnetstack.com> (OS independent)
- <http://sourceforge.net/projects/bacnet/?source=directory>
- <http://sourceforge.net/projects/bacrabbit/>
- <https://github.com/Frozenlock/Bacnet-scan>
- <https://github.com/HariYadav/JBACnet>
- <https://github.com/diekmann/BACnetSim>
- <https://github.com/snowwindwaverider/mango>
- <https://github.com/alclabs/bacnet>
- <https://github.com/mlab/mlep>

The list below shows some BACnet™ tools designed and compiled for Windows:

- <http://sourceforge.net/projects/vts/> (Windows)
- <http://www.bacnet.org/Developer/index.html>
- <http://www.cimetrics.com/index.php/product-solutions.html>
- <http://www.newron-system.com/BACnet-Software,26?lang=en>
- <http://www.polarsoft.biz/products.html>
- <http://www.scadaengine.com/index.html>

Within the following chapters some BACnet™ tools are described briefly.

BACnet™ Tool

The open source command line tool can be downloaded from sourceforge.net [13]. "This BACnet™ protocol stack library provides a BACnet™ application layer, network layer and media access (MAC) layer communications services. It is an open source, royalty-free library for an embedded system, Windows, Linux, or other operating system. Example BACnet™ client and server applications are included." [13]

This is a collection of several command line programs which can be used to read, write, and scan single and multiple BACnet™ object details. Our purposed BACnet™ environment setup si:

<code>c:\> set BACNET_IP_PORT=47816</code>	<code># port: 0xBAC8 = 47816</code>
<code>c:\> set BACNET_BBMD_PORT=47816</code>	
<code>c:\> set BACNET_BBMD_ADDRESS=172.16.120.10</code>	<code># BAS computer</code>

```
# search BACnet devices:
c:\> bacwi -1
# read all details from the BACnet object 2098177:
c:\> bacepics -v 2098177
```

BACnet4J

The open source Java package (classes) BACnet4J can be downloaded from sourceforge.net. BACnet4J is "a high-performance implementation of the BACnet™ I/P protocol written for Java by Serotonin Software. Supports all BACnet™ services and full message segregation. Can be used for field devices or for control platforms." (BACnet4J, 2012).

This collection of Java classes has been modified by CET to the needs within the project EnRiMa to be able to read, write, and scan single and multiple BACnet object details. It is planned to use the bnEnRiMa.jar file to be able to communicate with the BMS system. If everything works the command below can be used to do the BACnet™ communication:

```
C:\> java.exe -jar bnEnRiMa.jar localIPAddress broadcastIPAddress portNo
                                localDeviceID remoteDeviceID
```

Where:

localIPAddress: is the local TCP/IP address which should be used for the communication

broadcastIPAddress: is the TCP/IP broadcast address which should be used to do broadcasts in the local network

portNo: is the TCP/IP port which should be used for the BACnet™ communication

localDeviceID: is the BACnet™ object ID for the software client

remoteDeviceID: is the BACnet™ object ID which will be contacted

```
/*
 * Copyright (C) 2006-2009 Serotonin Software Technologies Inc.
 * @author Serotonin Software Technologies Inc., Matthew Lohbihler
 * @modifications Copyright (C) 2012 CET
 * CET team: Markus Groissböck, Michael Stadler, David Berger
 */
package com.serotonin.bacnet4j.test;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import com.serotonin.bacnet4j.*;

public class QuickTest {
```

```
private LoopDevice loopDevice;
private final LocalDevice localDevice;
public static String broadcastAddress = "255.255.255.255";
private static int portNo = 1234;
private static int localDeviceID = 1234;
private static int startDeviceID = 260001;

// remote devices found
final List<RemoteDevice> remoteDevices = new ArrayList<RemoteDevice>();

/* Note same Broadcast address, but different ports!!!
 * @param args
 * @throws java.Lang.Exception
 */
public static void main(String[] args) throws Exception {
    System.out.println();
    System.out.println("bnEnRiMa - read BACnet object details");
    System.out.println("(c) by CET");
    System.out.println("(based on BACnet4J)");
    System.out.println();

    if ( args.length != 4 ) {
        System.out.println("usage: /> java.exe -jar bnEnRiMa.jar broadcastAddress
portNo localDeviceID remoteDeviceID");
        System.out.println();
        System.exit(0);
    }

    broadcastAddress = args[0];
    portNo = Integer.parseInt(args[1]);
    localDeviceID = Integer.parseInt(args[2]);
    startDeviceID = Integer.parseInt(args[3]);

    System.out.println("broadcast address: " + broadcastAddress);
    System.out.println("Local port: " + portNo);
    System.out.println("Local object ID: " + localDeviceID);
    System.out.println("remote object ID: " + startDeviceID);
    System.out.println();
}
```

```
QuickTest dt = new QuickTest(broadcastAddress, portNo);

try {
    dt.setLoopDevice(new LoopDevice(QuickTest.startDeviceID, broadcastAddress,
portNo+1));
}
catch (RuntimeException e) {
    dt.localDevice.terminate();
    throw e;
}

try {
    dt.doDiscover();
    dt.printDevices();
}
finally {
    dt.localDevice.terminate();
    System.out.println("Cleanup LoopDevice");
    dt.getLoopDevice().doTerminate();
}
}

public QuickTest(String broadcastAddress, int port) throws IOException {
    localDevice = new LocalDevice(localDeviceID, broadcastAddress);
    localDevice.setPort(port);
    localDevice.getEventHandler().addListener(new DeviceEventListener() {

        public void listenerException(Throwable e) {
            System.out.println("DiscoveryTest listenerException");
        }

        public void iAmReceived(RemoteDevice d) {
            System.out.println("DiscoveryTest iAmReceived");
            remoteDevices.add(d);
            synchronized (QuickTest.this) {
                QuickTest.this.notifyAll();
            }
        }
    })
}
```

```

    public boolean allowPropertyWrite(BACnetObject obj, PropertyValue pv) {
        System.out.println("DiscoveryTest allowPropertyWrite");
        return true;
    }

    public void propertyWritten(BACnetObject obj, PropertyValue pv) {
        System.out.println("DiscoveryTest propertyWritten");
    }

    public void iHaveReceived(RemoteDevice d, RemoteObject o) {
        System.out.println("DiscoveryTest iHaveReceived");
    }

    public void covNotificationReceived(UnsignedInteger
subscriberProcessIdentifier,
        RemoteDevice initiatingDevice, ObjectIdentifier
monitoredObjectIdentifier,
        UnsignedInteger timeRemaining, SequenceOf<PropertyValue> listOfValues) {
        System.out.println("DiscoveryTest covNotificationReceived");
    }

    public void eventNotificationReceived(UnsignedInteger processIdentifier,
RemoteDevice initiatingDevice,
        ObjectIdentifier eventObjectIdentifier, TimeStamp timeStamp,
UnsignedInteger notificationClass,
        UnsignedInteger priority, EventType eventType, CharacterString
messageText, NotifyType notifyType,
        Boolean ackRequired, EventState fromState, EventState toState,
NotificationParameters eventValues) {
        System.out.println("DiscoveryTest eventNotificationReceived");
    }

    public void textMessageReceived(RemoteDevice textMessageSourceDevice, Choice
messageClass,
        MessagePriority messagePriority, CharacterString message) {
        System.out.println("DiscoveryTest textMessageReceived");
    }

    public void privateTransferReceived(UnsignedInteger vendorId,
UnsignedInteger serviceNumber,
        Encodable serviceParameters) {
        System.out.println("DiscoveryTest privateTransferReceived");
    }

```

```

    }

    public void reinitializeDevice(ReinitializedStateOfDevice
reinitializedStateOfDevice) {
        System.out.println("DiscoveryTest reinitializeDevice");
    }

    @Override
    public void synchronizeTime(DateTime dateTime, boolean utc) {
        System.out.println("DiscoveryTest synchronizeTime");
    }
});
LocalDevice.initialize();

}

/**
 * Send a WhoIs request and wait for the first to answer
 *
 * @throws java.lang.Exception
 */
public void doDiscover() throws Exception {
    // Who is
    System.out.println("Send Broadcast WhoIsRequest() ");
    // Send the broadcast to the correct port of the LoopDevice !!!
    LocalDevice.sendBroadcast(LoopDevice.getPort(), new WhoIsRequest(null, null));

    // wait for notification in iAmReceived() Timeout 2 sec
    synchronized (this) {
        final long start = System.currentTimeMillis();
        this.wait(2000);
        System.out.println(" waited for iAmReceived: " + (System.currentTimeMillis()
- start) + " ms");
    }

    // Another way to get to the list of devices
    // return LocalDevice.getRemoteDevices();
}

@SuppressWarnings("unchecked")

```

```

private void printDevices() throws BACnetException {
    for (RemoteDevice d : remoteDevices) {

        LocalDevice.getExtendedDeviceInformation(d);

        List<ObjectIdentifier> oids = ((SequenceOf<ObjectIdentifier>)
LocalDevice.sendReadPropertyAllowNull(d, d
        .getObjectIdentifier(), PropertyIdentifier.objectList)).getValues();

        PropertyReferences refs = new PropertyReferences();
        // add the property references of the "device object" to the list
        refs.add(d.getObjectIdentifier(), PropertyIdentifier.all);

        // and now from all objects under the device object >> ai0, ai1, bi0, bi1...
        for (ObjectIdentifier oid : oids) {
            refs.add(oid, PropertyIdentifier.all);
        }

        System.out.println("Start read properties");
        final long start = System.currentTimeMillis();

        PropertyValues pvs = LocalDevice.readProperties(d, refs);
        System.out.println(String.format("Properties read done in %d ms",
System.currentTimeMillis() - start));
        //      printObject(d.getObjectIdentifier(), pvs);
        for (ObjectIdentifier oid : oids) {
            printObject(oid, pvs);
        }

    }

    System.out.println("Remote devices done...");
}

private void printObject(ObjectIdentifier oid, PropertyValues pvs) {
    System.out.println(String.format("\t%s", oid));
    for (ObjectPropertyReference opr : pvs) {
        if (oid.equals(opr.getObjectIdentifier())) {
            System.out.println(String.format("\t\t%s",
opr.getPropertyIdentifier().toString()));
        }
    }
}

```

```

    }
}

/**
 * @return the LoopDevice
 */
public LoopDevice getLoopDevice() {
    return LoopDevice;
}

/**
 * @param LoopDevice
 *      the LoopDevice to set
 */
public void setLoopDevice(LoopDevice LoopDevice) {
    this.loopDevice = LoopDevice;
}
}

```

Are the BACnet™ Objects at Campus Pinkafeld Accessible?

No. The problem we have is that up to now no BACnet™ object has responded any of our requests. We hope that we can fix this issue in the next months. However, this depends on the support of Siemens, the DESIGO™ vendor at Pinkafeld, which is not an official partner in the EnRiMa project. So far the support of Siemens was limited and no specific information regarding the communication (i.e. are our IP addresses and BACnet™ object references correct) could be obtained from Siemens or Campus Pinkafeld. Furthermore, as pointed out in WP1 this is also a liability issue and Campus Pinkafeld might object such a fully automated procedure.

References for Appendix III

BACnet (2012) BACnet Tutorials [Online]. Available from:
<http://www.bacnet.org/Tutorial/index.html> (Accessed: 28 November 2012).

BACnet4J (2012) BACnet4J [Online]. Available from:
<http://sourceforge.net/projects/bacnet4j/> (Accessed: 04 December 2012).

Merz, H., Hansemann, T. & Hübner, C. (2009), *Building Automation: Communication Systems With EIB/KNX, LON und Bacnet*. Berlin: Springer.

Technologies (2011) Building Automation [Online]. Available from:
<http://www.technologies.co.il/beta/Resources/Pages/1968/5a.pdf> (Accessed: 28 November 2012).

Appendix IV - Generic Kernel Prototype Implementation

This appendix presents an overview of the implementation of the EnRiMa DSS generic kernel prototype. The purpose of the generic kernel is to explore ways to make the kernel implementation more generic, so that it is applicable to a wider range of problems, including a wider range of buildings and corresponding mathematical models. The generic kernel is developed according to the requirement analysis and the architecture specification, both described in detail in Deliverable 4.1a [4]. The full description of the generic kernel prototype implementation is provided in [3].

The generic kernel development is complementary to the kernel described in the main part of this report. The latter is a dedicated implementation focused on the EnRiMa DSS prototype currently being developed, while the generic one was designed and implemented to be reusable without substantial software modifications for a wide class of buildings and the corresponding mathematical models.

The generic kernel is a rather complex software component, thus its full documentation would result in an excessively long report. Therefore, a large part of the documentation has been posted in the GitHub repository web hosting service; these elements are only shortly commented on in this report. Moreover, descriptions of several issues included in [4] are not duplicated in this report. In particular, a key issue of the role of the Symbolic Model Specification (SMS) discussed in detail in Section 2.1 of [4] is only briefly mentioned in this appendix.

Below we provide an overview of the generic kernel, including characteristics of the kernel's key features. Next, the kernel architecture is described, followed by the short presentation of the GitHub kernel repository, and conclusions.

The full description [3] covers the following topics:

- The XML schema, i.e., the formal specification of the common (for the generic kernel and all DSS components using its services) data structures and operations on the data.
- The presentation of the Web-Services (WSs) composed of an overview and simple characteristics of each implemented WS.
- The summary of the basic information about the Data-Warehouse (DW).
- Overview of the interface between the generic kernel and other DSS components.
- Detailed description of the environment developed for testing the generic kernel prototype; it is composed of a summary of testing done with a dedicated testing tool, followed by the presentation of three applications (test-GUI, data-wrapper, scenario generator) developed for both testing the kernel, and demonstrating the use of kernel services by three different types of applications; moreover, a complete integrity test is presented; this part contains also a description of use a modern issue- and bug-tracking environments. The testing environment is posted at the GitHub site described below.
- Supplementary documentation:
 - An example of the XML-format representation of the WS input and output;
 - Four views on data entities and their relations, each view focused on a specific aspect.

Overview of the Generic Kernel

The generic kernel is one of the three components of the EnRiMa Decision Support System Engine (DSSE) that is the backbone of the EnRiMa DSS providing to its users three types of stateless services, each through one of the DSS components, namely the DSS Kernel, Solver Manager, and Scenario Generator. The general assumption of the generic kernel architecture is that all DSSE services used by end-users will be accessed through the User Interface (UI), and provided through Web-Services (WSs). The overview of the generic kernel architecture is provided in Section 2 of [4]; therefore it is not discussed in this report, which focuses on only the generic kernel.

The generic kernel is built on the Structured Modeling (SM) methodology, originally proposed by Geoffrion [1] and implemented for the integrated modeling systems, see e.g., [2]. The SM provides a proven, generic, framework supporting the whole modeling cycle, which is especially effective for development of non-trivial model-based DSS, which is typically done by interdisciplinary teams working at distant locations. Separation of model specification from management of data used for model instances was a break-through in modeling technology, and it is now considered to be a key element of good modeling practice. The SMT, see e.g., [5], extends the original SM framework by offering additional functionality, in particular optional attributes of the SMS, which facilitate development and maintenance of the UI, as well by providing a DW infrastructure accessible through WSs. This enables on the one hand efficient handling of also huge amounts of data having complex indexing structures, on the other hand separating the modeling activities into two parts. First, handled by users through a UI; second, provided by a server that maintains all persistent elements of the model and the corresponding modeling process. SMT also extends the SM approach by providing the Data Warehouse (DW) that handles all data of the whole modeling process in a way transparent to the model users. The SMS plays a key role in assuring consistency of these data. The generic kernel provides Web-services through which all needed modeling information is exchanged. Thus, the generic kernel is model-independent, and therefore can be re-used for other models.

Functional View

Figure A4-1 illustrates the functional view on the Web Services (WSs) provided by the generic kernel; these services are used (consumed) by other EnRiMa DSS components, called clients. The DSSE provides state-less services therefore the requests are coming from the other DSS components, and the corresponding services are provided by the kernel. The communication between clients and the kernel is provided by the XML-based SOAP (Simple Object Access Protocol). Each service is composed of a pair of related WSs, first used for the service request (by the client), and second for the response provided by the kernel. Additionally, the kernel provides services used for triggering the execution of some clients; this is done through an HTTP or a system call.

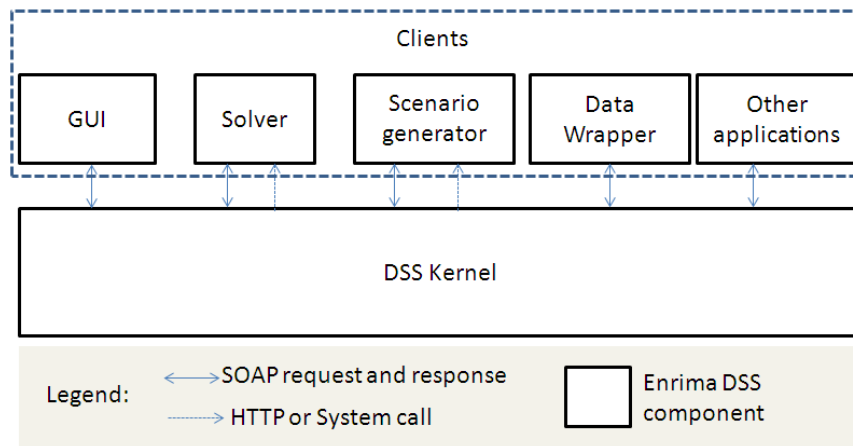


Figure A4-1: The Generic Kernel Web Services.

Key Features

The generic DSSE architecture described in detail in [3] has several important properties discussed there. We summarize here only those related to the kernel architecture and implementation, and present them by outlining the generic kernel's key features:

- The kernel enables robust integration of heterogeneous (i.e., developed and run on diverse hardware and software platforms, possibly at distant locations) components into a DSS fitting particular needs.
- The modular structure supports efficient development and modifications of the DSS components, with minimum interdependencies during the whole cycle of software development use, and maintenance. New functionality can be provided by new modules without causing problems to the already developed components.
- The WSs are defined automatically in the WSDL (Web-Service Description Language) using public domain tools, which also generate the documentation of the WSs. Each needed DSS functionality is provided in the same way to the DSS clients, which assures consistency and supports effectiveness of the software development.
- The WSs are based on a common (for all DSS components) Symbolic Model Specification (SMS), which enables easy assurance of consistency of use DSS components (both software and elements of the underlying model). More details about the SMS role are provided in Section 2.1 of [4].
- The EnRiMa Data-Warehouse (DW) handles not only all DSS data (model parameters, sets and subsets of indices, results of diverse model analysis) but also all other needed for processes of the model-based decision-making support. The DW structure is transparent for the clients, which communicate using the relevant elements of the common SMS.
- The DW supports data versioning in a very effective way through the mechanism of defining (in a way transparent to the users) data updates. This supports the required data persistency (after data is committed it will always be available.) with possibilities of modifying the data. Reliable reuse of data is supported through definitions of the model instances.

- The concerted use of the SMS, WSs, and the DW assures semantic consistency of the model entities across the DSS components. The data is provided in structures corresponding to the SMS; moreover, organization of indices into corresponding sets and (optionally indexed) subsets effectively supports a proper use of also complex indexing structures of the model entities.
- The kernel supports the control of access (read and write permission) to the data and applications triggered by the Kernel. It implements the ACL (Access Control Lists) mechanism which provides an easy to use way of granting permissions to either individual users, or groups of users defined by the DSS administrator.
- The kernel supports use of bug-tracking utilities (such as Jira, see [3]) for supporting rational software development process, as well as facilitating efficient reporting and handling of problems that the DSS users want to share with the DSS developers.
- The kernel design and implementation facilitate reusability of software, data, and results of the model analyses. In particular, modifications of the SMS require changes neither in the DW structure nor in the implementation of the WSs; depending on the technology used for the UI (see [3]) it also may not require any changes in the UI.

In summary, the generic kernel has been designed and implemented in such a way that it can be used for other buildings without substantial modifications. Actually, the modifications are limited to adaptation of the SMS to the new building, and provision of the corresponding data.

Software Developer's View

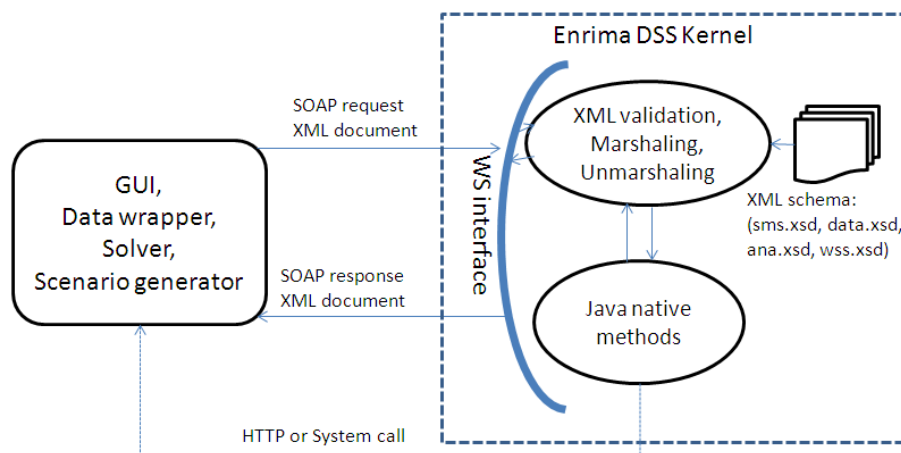


Figure A4-2: The Kernel Web Services consumed by other EnRiMa DSS components: software developer's view.

Figure A4-2 illustrates the client developer's view on the generic kernel. As already mentioned, each service is composed of the {request, response} pair of WSs; the interface between the kernel and the clients is provided through the SOAP based on the automatically generated and parsed XML-format documents. The WSs are published by the kernel in the automatically generated WSDL; they are specified using the XML schema (WSs and XML schema are described in [3]), currently composed of four elements defining the structures of the SMS, data, analysis, and WSs, respectively. The WSDL representation is used by the

Kernel for validation of the XML-based requests. The marshaling and unmarshaling functions shown in Figure A4-2 are used by both the generic kernel and the clients for serialization the WS-domain objects (called DTOs, Data Transfer Objects) to XML, and for the reverse process (also called unmarshaling), respectively.

Additionally, the kernel can support triggering execution of the clients; this is done by either the HTTP or the system calls. Such a call typically contains a small number of parameters to enable identification of the task the client is requested to perform; typically, such parameters are used in the subsequent WS-requests by the client for receiving from the kernel the needed data in WS-responses. There is an easy way to develop one version of an application that can be executed through either HTTP or call providing the same set of parameters. The HTTP calls are used for remote calls, and the system calls are used when the kernel has the authorization to access the client either through ssh, or rsh, or directly on a shared file-system. Moreover, the system call is very useful for the off-line software development and maintenance. More details on the WSs consuming by the DSS clients are provided in [3].

Generic Kernel Architecture

The actual implementation of the generic kernel prototype follows the architecture presented in Section 3.6 of [4], which in turn conforms to the requirement analysis of the DSSE also discussed in detail in [4], and earlier in [6]. The current description of the generic kernel architecture is based on Section 3 of [4].

The Kernel architecture follows the SOA methodology of software design based on services, i.e., collections of software modules, each providing a tightly defined functionality. The services are state-less (function independently) and loosely coupled with hardware and software technology. Thus, the services can be easily reused (both within an application, and with different applications); moreover an application can be composed of software modules run on different hardware and software platforms. In other words, interoperations of tightly defined services provide collectively the desired functionality of the application. This methodology has been applied to the design and implementation of the generic kernel and its web-services.

As already explained, the other DSS components (clients) access the kernel functionality only through WSs. Such a single interface type has obvious advantages for development and maintenance of the rather complex Kernel software component. This also implies that the clients are independent of changes in the Kernel implementation (as long as the WS specification will not be changed). Moreover, the Kernel may be moved to another hardware, or use another DBMS, or even its architecture can be changed; none of these would cause any changes to the clients.

Actually, the developers of the DSS clients do not need to be familiar with the kernel architecture or implementation of the data services or any other element of the kernel, because all these components are transparent for the clients. However, for documenting the Kernel implementation, we summarize here the kernel architecture. The Kernel consists of seven modules illustrated in Figure A4-3. The kernel components developed in the Java programming language communicate through Java methods.

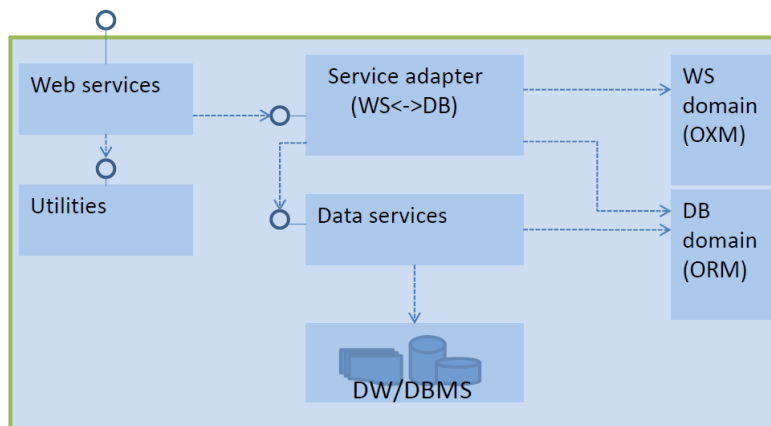


Figure A4-3: The Generic Kernel Components

The function and characteristics of each Kernel component are as follows:

Service adapter: provides link between WSs and the internal data services; transform formats between Web services domain objects and database domain objects; makes the data services transparent (a black box) for components that do not belong to the Kernel.

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). Such tools also generate classes for applications consuming WSs in other DSS components (UI, Solver Manager, Scenario Generator, data wrappers and other applications integrated with the ICT infrastructure of each building); more details are provided in [3].

Database domain: the Java Persistence API (JPA) entity objects generated using the database schema through object-relational mapping tools, such as Hibernate, Eclipse link, Toplink, etc.

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.

Data Warehouse: dedicated data structures implemented within a DBMS, accessible to external (to the Kernel) clients only through the WSs provided by the Kernel. Therefore neither a DBMS choice nor the corresponding DBMS schema influences other (than Kernel) DSS components. Currently, PostgreSQL is used for prototyping. More details on DW are provided in [3].

Utilities: a container of diverse back-office applications that support various functions needed by the Kernel.

The multi-layered representation of the Kernel is illustrated in Figure 4. The top layer contains the WSs published within the WS-domain in the WSDL format, and endpoints indicating a location for accessing the service. This domain is shared by all components of the EnRiMa DSS. The middle layer contains the service adapter which maps the requested operations into the domain data services. Data services layer provides stateless transactional business logic, 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. More details on the Kernel data services are provided in Section 3.4.1 of [4].

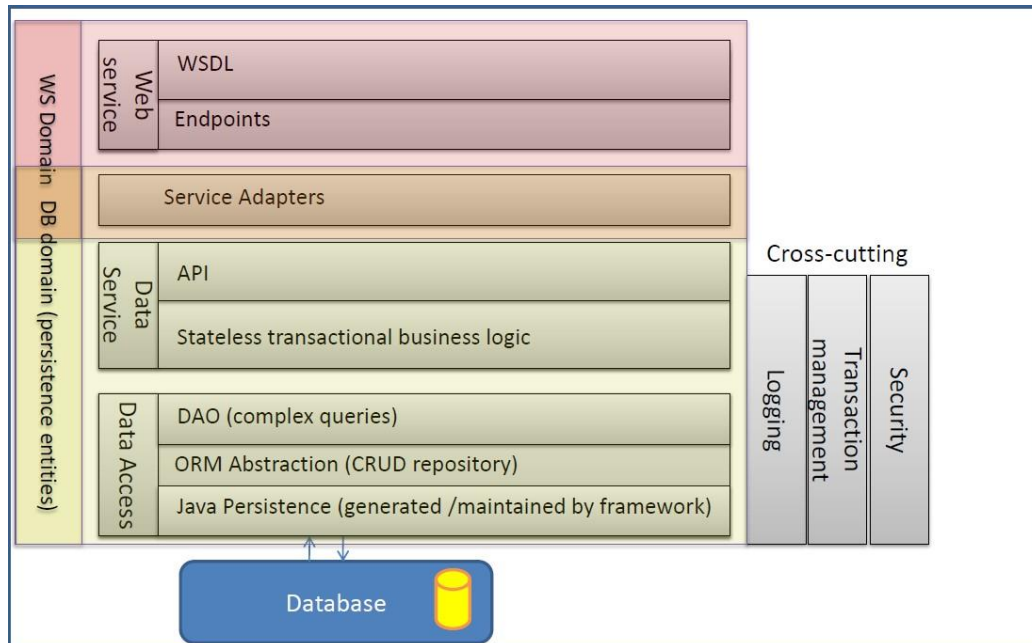


Figure A4-4: The generic kernel multi-layered representation.

GitHub Repository

Most probably, every software development team uses a version control system. The development team uses GitHub, a free and open source distributed version control system. We use Git for private repositories of software and documents organized into projects shared with collaborators working in remote locations; projects can also be openly shared, but to access the file it is necessary to install the Git. To overcome this requirement, we have created the EnRiMa repository at the GitHub, the web-based hosting service that offers free account for open source projects.

The GitHub site for the generic kernel, <https://github.com/enrima-dev410> has currently only one repository called kernel which contains the generic kernel-related software and documentation.

The content of the repository is managed through the Git, thus versioning of the files is supported. This is an open access repository; therefore everyone can upload files from it. Users who have the Git installed on a local computer can follow the Git Read-Only link to download the file. Users who have no Git installed can use the ZIP download option.

The enrima-dev4/kernel GitHub repository contains several elements that are essential parts of the Kernel implementation, but – due to either size or nature – have not been included in the deliverable. These include the source-code of the client-applications developed with two purposes: first, to additionally test the generic kernel prototype; second, to show a simple way of the WSs integration with three types of application (developed in Java and C++ programming languages, and with a framework for the GUI development). Each of these

clients is accompanying with the developer guide, which documents its implementation at a local computing environment.

The repository is organized into a directory structure, and documented on line. Therefore the structure description is not duplicated in this report; moreover, new items will most likely be added to the repository, therefore such a description would soon be outdated.

Conclusions

A generic kernel consisting of several modular software components has been developed to support decision-making aimed at improving energy-efficiency of buildings. The generic kernel is also reusable, i.e., easily adaptable to other buildings or facilities. The generic architecture based on the WSs and SMT enables effective development of DSS components that can be combined into a robust model-based DSS for operators of energy-efficient buildings.

The generic kernel prototype implementation follows the corresponding requirements described in [4]. The WSs provided by the kernel prototype enable efficient interface between components that consume WSs, and use of the SMS assures data consistency across all components accessing data through WSs. Moreover, the DTOs can be specified according to the needs of each component, and the public-domain tools support the automatic generation of the corresponding classes that can be directly embedded into the client applications. The DBMS schema used for implementation of the DW is independent of data models used by clients; moreover the use of the DBMS is transparent for the client applications. The client applications can be developed on heterogeneous hardware and software platforms, and run at distant locations. There are public-domain tools supporting use of WSs within all programming languages and software tools commonly used for modeling tasks. The generic kernel implementation is built on the SMT that has been successfully used for collaborative interdisciplinary research on the development of large and complex models.

Thus the generic kernel inherits efficient and robust modeling methodology and technology that supports efficient development and implementation of the DSS, as well as its use, maintenance, and reusability. Thus, the implemented generic kernel prototype provides a good basis for development of the remaining kernel functionality.

References for Appendix IV

- [1] GEOFFRION , A. An introduction to structured modeling. *Management Science* 33, 5 (1987), 547–588.
- [2] GEOFFRION , A. Integrated modeling systems. *Computer Science in Economics and Management* 2 (1989), 3–15.
- [3] IIASA. DSS generic-kernel prototype implementation. Technical report, European Commission FP7 project number 260041, Brussels, Belgium, 2013. Limited distribution: the report is available on request.
- [4] IIASA, URJC, SINTEF, AND TECNALIA. Requirement analysis of the decision support system engine. Deliverable D4.1a, European Commission FP7 project number 260041, Brussels, Belgium, 2012.

[5] MAKOWSKI , M. A structured modeling technology. European J. Oper. Res. 166, 3 (2005), 615–648. draft version available from <http://www.iiasa.ac.at/~marek/pubs/prepub.html>.

[6] SU, IIASA, SINTEF, AND CET. Draft specification for services and tools. Deliverable D5.1, European Commission FP7 project number 260041, Brussels, Belgium, 2012.

List of Acronyms for Appendix IV

AJAX	Asynchronous JavaScript and XML API Application Programing Interface DB Data-Base
DBMS	Data-Base Management System
DoW	Description of work, Annex 1 to the EnRiMa Grant Agreement
DTO	Data Transfer Object
DSS	Decision Support System
DW	Data Warehouse
GitHub	Web-based hosting service for software development
GUI	Graphical User Interface
HTTP	The Hypertext Transfer Protocol
ICT	Information and Communication Technology
JAXB	Java Architecture for XML Binding
JAX-WS	Java Architecture for Web Services
OOP	Object-Oriented Programing
OXM	Object–XML Mapping rsh remote shell
SM	Structured Modeling
SMS	Symbolic Model Specification
SMT	Structured Modeling Technology
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
ssh	Secure shell
UI	User Interface
WSDL	Web Service Definition Language
WS	Web-Service
XML	Extensible Markup Language