Project Number 260041
SUPPORTING ACTION

EnRiMa
Energy Efficiency and Risk Management in Public Buildings

DSS Generic Kernel final implementation

Technical report supplementing deliverable D4.7

Start date of the project: October 1, 2010

Duration: 42 months

Organisation name of lead contractor for this report: IIASA

Revision: final, March 31, 2014

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)	

Executive summary

The goal of this report is to describe the DSS Generic-Kernel final implementation that was designed and implemented to be reusable without substantial software modifications as a backbone of the EnRiMa Decision Support Systems (DSS) for a wide class of buildings and the corresponding mathematical models. This implementation is complementary to the dedicated Kernel implementation focused on the EnRiMa DSS currently being developed, and documented in deliverable D4.7. The Generic-Kernel prototype was developed according to the requirement analysis and the architecture specification, both described in detail in Deliverable 4.1a. Therefore, this report is complementary to three deliverables, namely D4.1a, D4.4, and D4.4.

The Generic-Kernel, in order to fulfil the requirements specified in the DoW and in deliverable D4.1a, has been designed and implemented by exploiting the synergies of three proven and complementary methodologies: Service Oriented Architecture (SOA), Web-enabled Structured Modeling Technology (SMT), and model-based DSS. All functionality of the Generic-Kernel is provided through the collection of Web-Services (WSs) to be used by other components of the EnRiMa DSS that shall share information (model specification, data, and model results).

This report consists of descriptions and (whenever appropriate) documentation of the following main components:

- Overview of the Generic-Kernel, composed of background, functional view, key features, and software developer's view.
- Overview of access control to the EnRiMa DSS resources managed by the Generic-Kernel.
- The Generic-Kernel architecture.
- The data types and operations handled by the Generic-Kernel.
- Web-services provided by the Generic-Kernel.
- Data-warehouse handling all needed data, including model specification, parameters, results of analysis, as well as data of authorized DSS users, and automatically gathered information about the modeling processes.
- Interface between the Generic-Kernel and other DSS components.
- Dedicated software environment for the Generic-Kernel validation.
- Web-based repository for software and documentation.

Contents

List of Figures

List of Tables

List of acronyms

ACL	Access Control List
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

Table 1: List of acronyms

1 Introduction

This report presents an overview of the implementation of the EnRiMa DSS Generic-Kernel.[1] The purpose of the Kernel is to explore ways to make the EnRiMa DSS implementation more generic, so that it is applicable to a wider range of problems, including diverse models that will be applicable to different buildings and public facilities. The Kernel is developed according to the requirement analysis and the architecture specification, both described in detail in Deliverable 4.1a [4]. Therefore, this report is complementary to [4].

The Kernel is complementary to the Kernel described in [6]. The latter is a dedicated implementation focused on the EnRiMa DSS, while the Kernel was designed and implemented to be reusable without substantial software modifications for a wide class of buildings and the corresponding mathematical models.

The Kernel is a rather complex software component. It not only meets the requirements specified in [4]; additionally, its implementation includes a comprehensive testing environment developed for not only validating the Kernel, but also for demonstrating use of the Kernel services by three main types of applications. Thus, a full Kernel documentation would result in an excessively long report. Therefore, a detailed documentation has been posted in a public GitHub[2] repository. The content of the posted documentation is summarized in Section 10. Moreover, descriptions of several issues included in the Requirement Analysis report [4] are not duplicated here. 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 report.

The structure and content of the remaining part of this appendix is as follows: Section 2 provides an overview of the Kernel, including characteristics of its key features. Next, Section 3 describes the services provided for controlling access to modeling resources (data, model specification, results of analysis) managed by the Kernel; it is followed by Section 4 documenting the Kernel architecture. Section 5 presents the XML schema, i.e., the formal specification of the common (for the Kernel and all DSS components using its services) data structures and operations on the data. This is followed by the presentation of the Web-Services (WSs) Section 6 composed of an overview and simple characteristics of each implemented WS. Section 7 summarizes the basic information about the Data-Warehouse (DW) Brief summary of the interface between the Kernel and other DSS components is provided in Section 8. Section 9 describes in detail the environment developed for validating the Kernel; it is composed of a summary of validation 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 section contains also a description of use a modern issue- and bug-tracking environments. The GitHub Kernel repository is presented in Section 10. Section 11 concludes the main part of the report. The Appendix contains two supplementary pieces: (1) an example of the XML-format representation of the WS input and output; (2) seven views on data entities and their relations, each view focused on a specific aspect.

[1] Thereafter, for brevity, referred to as *the Kernel*.

[2] Web-based hosting service for software development.

2 Overview of the DSS Kernel

2.1 Background

Modern model-based Decision-making Support Systems (DSS) consist of heterogeneous components, typically developed by diverse teams, and often run at remote locations. The DSSs developed and used for complex decision situations, like the EnRiMa, are actually also complex software. Therefore, a modular architecture supporting interoperability is a must for effective software development and maintenance; in particular, the DSS components need to be encapsulated in order to minimize their interdependencies. Moreover, assuring semantic consistency of objects used in different DSS components is a key condition for the DSS reliability and accountability. Any actually used DSS requires a robust and flexible control of access to its resources (models, data, solvers, results of analysis); this is necessary for assuring security, safety, privacy, and accountability for the DSS that is developed, maintained, and used by diverse developers and users, each having specified roles and responsibilities. Yet another issue is reusability and adaptability. Development of complex DSS requires large resources; their efficient use calls for software that is possibly easy to adapt for inevitable changes in specifications and models, sources of data, etc., as well as for reusing the software components for other applications, and/or replacing some components (especially solvers) by more efficient ones.

The Kernel has been designed and implemented to be a backbone of the EnRiMa DSS, meeting the above summarized requirements, and therefore being easy to be reused for other buildings. The Kernel provides its users with state-less services that support all functionality needed for integrating heterogeneous software components into a DSS supporting managers and operators of energy efficient buildings. The general assumption of the Kernel architecture is that all services used by end-users will be accessed through the User Interface (UI), and provided through Web-Services (WSs). Details on the requirements for the Kernel architecture are provided in Section 3 of [4]; therefore it is not discussed in this report.

The 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 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 (DSS Kernel) 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 Kernel provides Web-services through which all needed modeling information is exchanged. Thus, the Kernel is model-independent, and therefore can be re-used for other models.

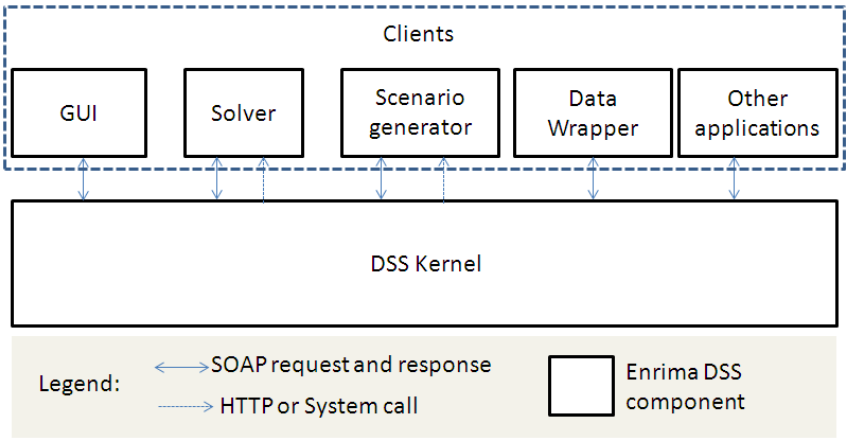

Figure 1: Functional view on the Kernel Web Services consumed by other EnRiMa DSS components.

2.2 Functional view

Figure 1 illustrates the functional view on the Web Services (WSs) provided by the Kernel; these services are used (consumed) by other EnRiMa DSS components, called clients. The Kernel provides state-less services handling the requests coming from the other DSS components. 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.

2.3 Key features

The Kernel architecture described in detail in Section 4 has several important properties discussed there. We summarize here only those related to the Kernel architecture and implementation, and present them by outlining its 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. Moreover, each module can be modified whenever desired because such modifications do not affect any other modules.
- 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 and maintenance.
- 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, i.e., heterogeneous applications supporting modeling processes, and elements of the underlying model.[3]

[3] More details about the SMS role are provided in Section 2.1 of [4].

- The EnRiMa Data-Warehouse (DW) that is integrated with the Kernel handles all DSS data (SMSs of all models, parameters of all models, sets and subsets of indices, results of diverse model analyses), as well as all other data needed for processes of the model-based decision-making support (including data about the DSS users and their access rights).
- The DW structure is transparent for the clients, which communicate through WSs that use the relevant elements of the common SMS. Therefore the DW schema is invariant, i.e., it is not modified when the DSS components are changed (e.g., for diverse model specifications or solvers). This greatly increases efficiency of the software development cycle.
- 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 always will be available) combined with an effective and efficient mechanism for data modification. Reliable reuse of diverse data versions 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 all DSS components. The data is provided in structures corresponding to the SMS, and can be tailored to fit the data structures used by the clients; 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/use and write permissions) 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 for supporting rational software development process, such as Jira[4], as well as facilitating efficient reporting and handling of problems that the DSS users want to share with the DSS developers.

In summary, the 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 (1) adaptation of the SMS to other buildings and the corresponding models, and (2) provision of the corresponding data. 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 Section 9.2) it also may not require any changes in the UI. Thus, the Kernel design and implementation facilitate reusability of software, data, and results of the model analyses.

2.4 Software developer's view

Figure 2 illustrates the client developer's view on the 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. All Kernel schemas are transparent for the clients. The WSs and XML schemas are described in Sections 6 and 5, respectively; they are currently composed of four elements defining the structures of the SMS, data, analysis, and WSs, respectively. Additionally, there are three schemas documenting the data structures for handling the ACL. The WSDL representation is used by the Kernel for validation of the XML-based requests. The marshaling and unmarshaling functions shown in Figure 2 are used by both the Kernel and the clients for serialization the WS-domain Data Transfer Objects (DTOs) to XML, and for the reverse process (also called demarshaling), respectively.

[4] See Section 9.6

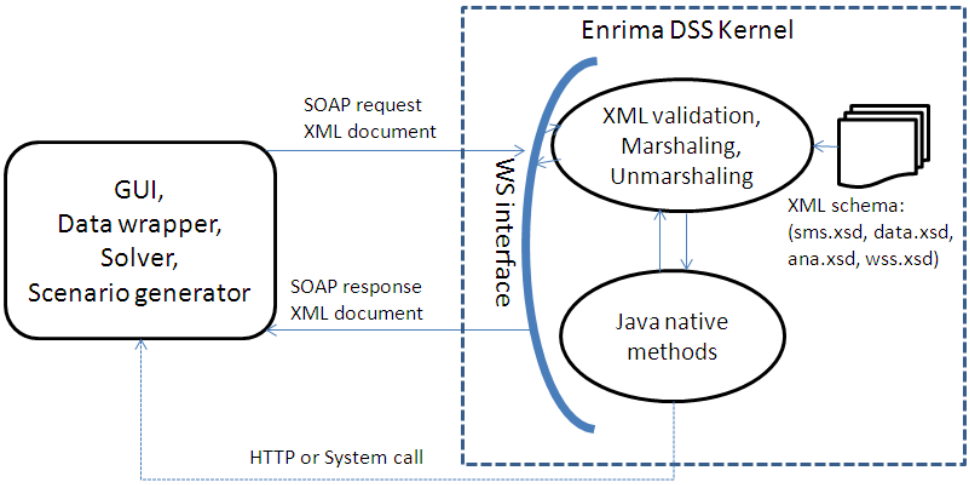

Figure 2: The Kernel Web Services consumed by other EnRiMa DSS components: software developer's view.

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 Section 8.

3 Access control to the DSS resources

Good practice requires a reliable and flexible access control to modeling resources (data, model specification, results of analysis), as well as protection of privacy of the users of a DSS. The EnRiMa DSS is a complex system consisting of heterogeneous components accessed by diverse users having different roles and authorities; the latter typically change during the DSS development and use. Therefore, the Kernel needs to support effective and flexible mechanism for access control to the DSS resources managed by the Kernel. The access rights to each object (defined here as an element of a resource) are of three types:

- Read, i.e., use its content (this also includes use of the DSS tools);
- Write, i.e., provide a new content of the object.
- Admin, i.e., administrate access rights the object.

For illustrating this issue, let us consider objects of diverse types, such as, SMS, various data sets (e.g., versions of model parameters, results of analysis), model instances, specification of model analysis, tools (e.g., scenario generator, optimization solver), groups of users, etc.

Each DSS user may have different access rights to a particular object or a set of objects. Users are identified by the corresponding login-names; for illustration, we use here the names *user1, user2, . . . , user5*. For example, *user1* may have read permission for all parameters of the operational model, and write permission for storing new versions of a subset of the parameters; *user2* may have read permission for the optimization tool and the corresponding results; *user3*

may have write permission for data sets defining the access rights.

It is practical (especially for administrating the DSS) to associate a set of access rights with a role. For example, the role USER may imply a set of access rights granted to all users, the role ADMIN provides the rights needed for the administration of the DSS, and the role SYSTEM grants the privileges needed for administrating the computing infrastructure. Typically, all users have the role USER, thus have a standard set of access rights. Moreover, it is also useful to define groups of users, and associate the roles with groups.

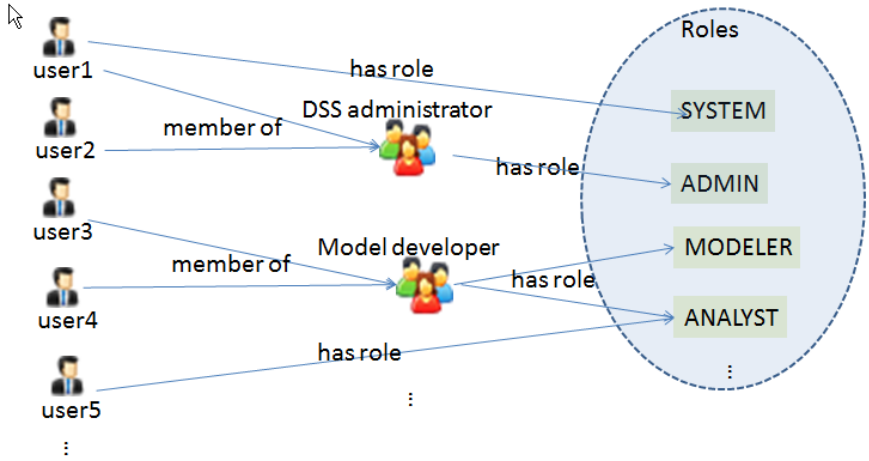

Figure 3: The users, groups, and roles example.

The relations between users, groups, and roles are illustrated by a simple example shown in Figure 3. Five roles are specified (the role USER is not shown because it is assigned by default to all users). In addition to the SYSTEM and ADMIN role outlined above, two roles typical for model development and use are defined: MODELER (a person who develops the DSS's models) and ANALYST (a person who uses the models) are defined. Depending on the needs, in particular the composition of the teams developing the models, or providing the data during the actual model use, more roles can easily be defined. Figure 3 illustrates also use of the groups: two groups (DSS administrator, and Model developer) are defined. Finally, let us point out that the roles can be assigned to either individual users or groups.

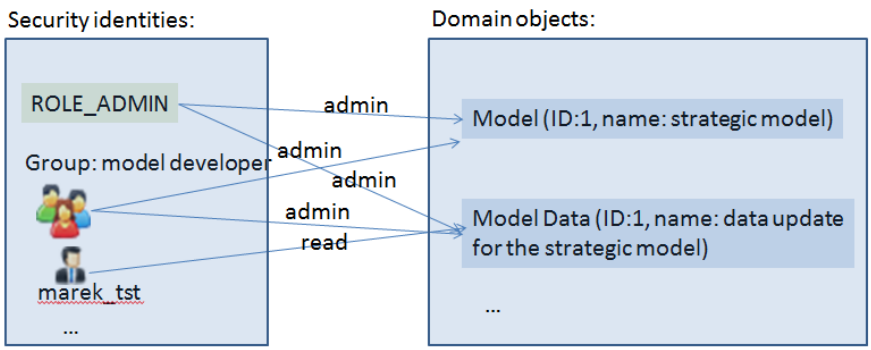

Figure 4: Illustration of the ACL use.

Figure 4 illustrates the use of the Access Control List (ACL) paradigm, which uses two concepts: security identities (which are assigned to roles, groups, and individual users) and domain objects (in our case the elements of the DSS resources). We have shown three security identities: role ADMIN, group of model developers, and a user having login-name *marek_tst*. Two objects are shown: a model (identified by the model id:1, and the corresponding name: strategic model), and a model data (identified by the data id:1, and the corresponding description specified by the person who created the data instance). The role ADMIN and the model developer group have granted the admin rights to both objects, while the user *marek_tst* has permission to read (i.e., to use it) the model data object.

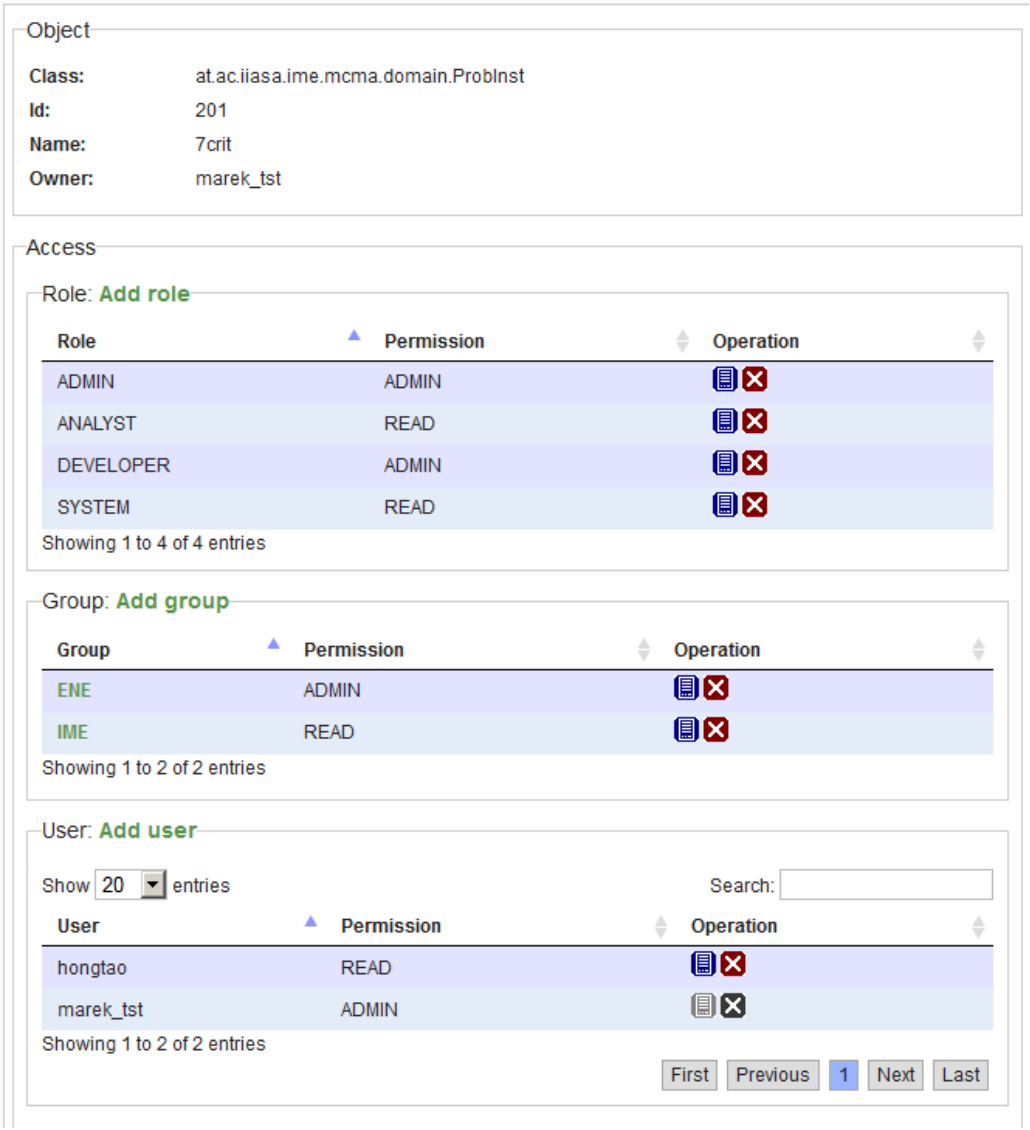

Figure 5: Illustration of GUI for administrating the ACL.

Figures 3 and 4 show very simple examples that illustrate the proven, effective, and powerful mechanism provided by the ACL paradigm. Configuration of the access rights is easy. It is typically done the system administrator who is experienced also in using DBMS; therefore a configuration is usually done by entering the corresponding data directly to the DBMS using one of many commonly used tools. Granting the rights to the users, and managing the user groups is typically done by the DSS administrator(s) who shall be provided by a simple interface. Figure 5

illustrates such a simple GUI applied to administrating access rights to the object identified by the id:201, and owned by *marek_tst*. By default, an object is owned by the user who created it, and the owner has by default the administration rights for the objects she/he owns. Figure 5 shows the situation with the access rights assigned to each of three access classes: (1) the roles (ADMIN, ANALYSTS, DEVELOPER, SYSTEM), (2) the groups (ENE, IME), and (3) the users (*hongtao, marek_tst*). Each of these entities has assigned one of the two permissions (admin or read). Adding a new entity to each class can be done by the corresponding drop list (accessed through the green buttons labelled: *Add role*, *Add group*, *Add user*, respectively). For each entity class there are two *Operation* buttons: the first supports modification of the permission type, the second removes the corresponding entity from the list.

4 Kernel architecture

The actual implementation of the Kernel 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 [7]. The description of the Kernel architecture is based on Section 3 of [4].

The Kernel architecture follows the SOA (Service Oriented Architecture) methodology of software design based on services, i.e., collections of software modules, each providing a tightly defined functionality. The services are state-less (i.e., each functions independently of the others) 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, inter-operations of tightly defined services provide collectively the desired functionality of each application; reuse of well-tested services also contributes to robustness of complex applications composed of possibly many software modules developed by several teams. This methodology has been applied to the design and implementation of the Kernel and its WSs.

As already explained, the other DSS components (called 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), and the clients' modifications can be done without requiring any changes of the Kernel and/or other clients. 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 6. The Kernel components developed in the Java programming language communicate through Java methods. The function and characteristics of each Kernel component are as follows:

Web services: publish the contract (as the WSDL file) through lightweight application service (Tomcat) and the endpoints implement the Web-services through calling the service adapter.

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

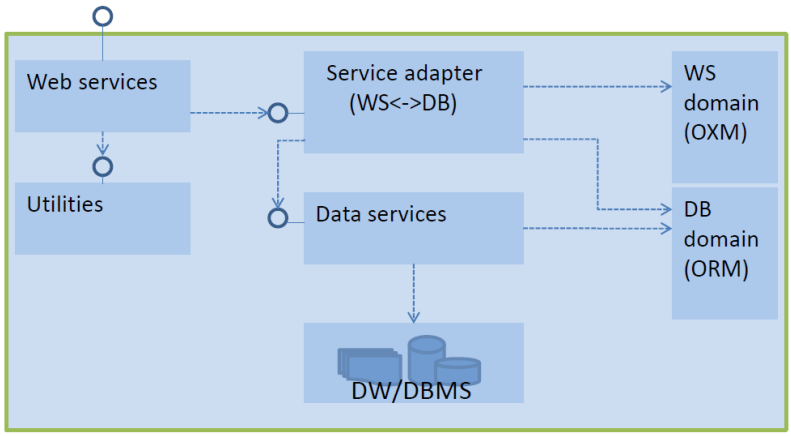

Figure 6: The Kernel components.

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 Section 9.3.

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 is 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 as the DBMS integrated with the Kernel. More details are provided on DW are provided in Section 7.

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 7. 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].

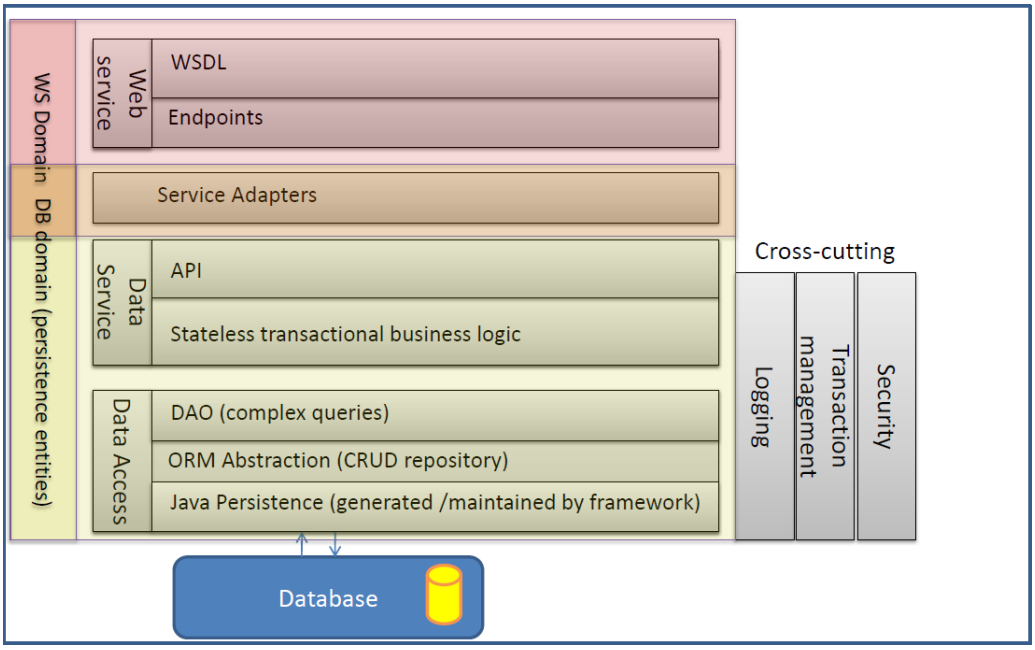

Figure 7: The Kernel multi-layered representation.

5 Specification of data types and operations

As already discussed, the main basis for assuring the consistency between the heterogeneous DSS components is the common SMS, and the common data structures as well as shared specification of the operations on the data. The role of SMS is described in detail in Section 2.1 of [4]; therefore, it is not discussed here.

We now comment on the specifications of the data structures and operations. Such a specification is done in the form of the XML schema. The full schema documentation is over 60 pages long; therefore, instead of presenting in the report the full specification, we provide only its short overview. However, the full documentation is posted at the Kernel GitHub repository.[5]

In order to avoid error-prone repetitions, the schema specification is organized into a set of mutually dependent files, each documenting the corresponding subset of the schema elements. These files have been generated automatically from development tools, and are posted in the `docs/ws_schema` directory of the Kernel GitHub repository. Machine-readable files are generated in the XML-format, the corresponding human-readable files in the PDF-format.

The specification of WSs is split into three files, each documenting the corresponding subset of WSs:

- aclWSS.pdf - WSs for access control,
- usrWSS.pdf - WSs for users' management,
- wss.pdf - all other WSs provided by the Kernel (in particular, SMS, model data, model analysis). The full resource hierarchy for these WSs is defined on page 2 of wss.pdf.

The data structures supported by the implemented WSs is specified in the corresponding schema documentation organized into the following files:

- acl.pdf - data structure for access control lists,
- ana.pdf - data structure for model analysis,
- data.pdf - all other (i.e., not specific for ACL, model analysis, SMS, user management) data structures,

[5] See Section 10.

• sms.pdf - data structure for the SMS,
• usr.pdf - data structure for user management (users, groups and roles).

These data structures are used for specification of DTOs of the corresponding WSs, and support all required functionality of the Kernel. New data structures can easily be defined whenever desired for supporting other DTOs that could better fit the internal data structures of the clients.

The PDF files provide printable documentation. However, hard-copies are used in only exceptional cases; for an actual development, one uses an XML editor, which supports interactive analysis of the corresponding schema. Each data type is documented by a diagram, which is especially helpful in analysis of complex types, as well as by the XML code.

To illustrate the content of the documentation we comment on the model analysis schema (ana.xsd) contained in ana.pdf. This schema is composed of the following elements:

• Specification of resource hierarchy (dependencies on the other schemas). It is followed by several (in this case four) descriptions of schemas, each description documenting the schema(s) defined in the corresponding namespace.
• For each namespace three sets of definitions are documented:
  – Schema(s); it can be either the main schema (in our example ana.xsd) or an imported schema (in our example is either data.xsd or sms.xsd).
  – Simple types (if any) defined for the corresponding schema.
  – Complex types (if any) defined for the corresponding schema.

The following five entity-relationship views on the DBMS schema implementation are presented in Appendix A:

• data.pdf - data structure schema,
• dataview.pdf - data structure view focused on fast access to large data-sets,
• instance.pdf - model instance schema,
• spec.pdf - SMS schema,
• usr_acl.pdf - users and ACL schema.

These views just illustrate fractions of the DW implementation developed with DB modeling tools. The views show only some of the tables and foreign keys; they does not show any of many constraints implemented for assuring consistency of the DB.

6 Web-Services

The Kernel functionality is provided through the WSs, therefore we present them in more detail than the other elements of the Kernel description. For the reader's convenience, we start with an overview of the WSs adapted from Section 3.2 of [4]. It is followed by presentation of the WSs implemented in the Kernel; this presentation is composed of simple characteristics of each implemented WS because such a form provides an easy overview. The formal specification of the WSs is organized into a number of files (see Section 5 for details) and posted at the Kernel GitHub repository (described in Section 10).

6.1 Overview

A WS is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with WSs in a manner prescribed by its description, i.e., using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. The main advantages of using WSs are as follows:

- Flexibility: each WS is only concerned with the transfer of semi-structured data between the corresponding software components. This gives flexibility of implementation, allowing systems to adapt to changes of requirements, technology etc. without directly affecting users.
- Interoperability: WSs support exchange of data between heterogeneous (i.e., developed and run on diverse hardware and software platforms) software components that can run at distant locations.
- State-less: WSs are independent, self-contained requests; they do not require information from, or state of, other requests; they are also independent of the content or state of other WSs.

Thus the DSS clients (applications that consume WSs) can build various processes that typically need exchange of data available from different sources and in diverse formats.

The Kernel provides WSs supporting the following top-level functionality:

- Interfacing the DSSE with the UI developed in WP5.
- Interfacing the Kernel with the scenario generator and solver manager.
- Other functionality needed for a DSS such as user handling, access control, users on-line problem reporting and comments, providing data for diverse reports for users and developers, backups, etc.

In order to meet these requirements, the WSs provided by the Kernel support:

- Storing various types and sets of the provided data (parameters of the model, specification of model analysis tasks, results of model analysis, data needed for administration of the DSS).
- Checking consistency of the provided data with the SMS, and with the specified sets of indices.
- Management of data versioning and acceptance.
- Definitions of model instances, analysis tasks.
- Providing information about the status of the solver tasks, and availability of solutions.
- Providing the stored data to the DSS components in diverse (corresponding the needs of the client applications) DTOs.
- Management of the DSS users, including authentication, specification of roles, access rights.
- Triggering execution of the scenario generator and solver manager.

From the client (applications that use Web-Services provided by the Kernel) perspective, a WS consists of four components:

- Envelope containing data needed for identification of the context within which the WS is used. The envelope includes elements such as: id of the user and his/her credentials, model id, data set and version, etc.
- Request (specification of the service). It identifies a specific WS, and provides the corresponding parameters.
- Response (specification of the response to the service). The message sent by WS server that may include one or more of the following: HTTP status, SOAP body contains the data of the response, error messages, and warnings.
- DTO (Data Transfer Object) containing structured data needed for processing the request or the response. DTO optionally contains specification of objects transferred as attachments for requests of storing/retrieving objects (files) that are not processed by the Kernel.

6.2 Implemented WSs

The Kernel provides all functionality described in Section 6.1. The formal documentation of the WSs is commented in Section 5. Here we provide an annotated list of the WSs implemented in the Kernel. As already mentioned, 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.

Therefore, below we list the services organized in such pairs. The following notation is used:
- A root-part of the WS name is used for both WSs, e.g., *getSMS*
- The name of the request WS is defined as a concatenation of the root-part and *Request*, e.g., *getSMSRequest*.
- Analogously, the response-WS name is refined as *getSMSResponse*.

Therefore, in the list below, the pairs of WSs are identified by their root-part name.

We present the WSs in an informal, descriptive way to provide as easy to follow overview of the implemented functionality. As already mentioned, the complete formal documentation has been posted, and is therefore available for the developers of applications consuming the WSs. Moreover, Section 9 presents examples of applications actually using the implemented WSs.

In the list of WSs presented below, functionality of each service is briefly stated, and followed by a summary of the input (content of the request) and of the output (content of the response). The data types used in the WSs' list are documented in the data.pdf described in Section 5. The common (for all WSs) elements of WSs, described in Section 3.2.3 of [4] as the WS envelope is not repeated here. An XML example of the *getSMS* services is presented in Appendix A.2. One should recall that the corresponding XML stream is generated by an application and therefore is rarely analysed by the developers. The full documentation of the WSs is available at the EnRiMa GitHub described in Section 10.

Formal specification of input and output parameters used by the WSs is available in the data.xsd schema. The meaning of the frequently used parameters is as follows:
- idModelSpec: automatically generated ID of the SMS version
- idModelData:automatically generated ID of the data version
- entity short-name: a short string that identifies the entity
- entitySpec: the data structure (defined in the data.xsd schema) containing the specification of the entity
- set members: indices that belong to a given (sub)set
- tupleValue list: list of vectors composed of indices related to a given entity or indexed set (also called: iterator)
- entityValues: composed of the entity short-name, and list of tupleValues
- status of processed request: 0 indicate a successful processing of the request, non-zero values are used as indicators of errors
- message: optional short text with additional information about errors and warnings.

The following 54 WSs (presented below as 27 pairs of WSs) have been developed, and are therefore provided by the Kernel; they have been published,[6] and thus are available for use by the client applications:

1. getSMS:
   - Function: provide the Symbolic Model Specification
   - Input: idModelSpec, see Appendix A.2
   - Output: SMS, see Appendix A.2
2. getEntityValues
   - Function: provide the stored values of an entity (either model parameter or model variable) together with the corresponding indices.
   - Input: idModelSpec, idModelData, and the entity short-name
   - Output: entitySpec, tupleValue list
3. getSetMembers
   - Function: provide the stored set members (i.e., indices belonging to a given set).
   - Input: idModelData, set short name, and (only for indexed subsets) tuple members

[6] URL: `http://www.ime.iiasa.ac.at/enrima_ws_tst1/enrima.wsdl`.

- Output: set members

4. storeMainset
   - Function: store a main-set members (specifications [defined in the data.xsd schema] of indices),
   - Input: idModelData, set short-name, members (indices)
   - Output: status of processed request, message
5. storeSetMembers
   - Function: store a sub-set members.
   - Input: idModelData, set short-name, members
   - Output: status of processed request, message
6. removeSetMembers
   - Function: remove set members.
   - Input: idModelData, set short-name, (for indexed subset only) tuple members
   - Output: status of processed request, message
7. storeEntityValues
   - Function: store entity values.
   - Input: idModelSpec, idModelData, entityValues
   - Output: status of processed request, message
8. removeEntityValues
   - Function: remove entity values.
   - Input: idModelData, set short-name, tuple members
   - Output: status of processed request, message
9. defineModelData
   - Function: define model data
   - Input: modelData
   - Output: status of processed request, message
10. defineInstance
    - Function: define model instance
    - Input: modelInstance
    - Output: modelInstance, with generated idModelInstance
11. defineAnalysis
    - Function: define analysis
    - Input: analysis
    - Output: status of processed request, message
12. getAnalyses
    - Function: get a list of analyses
    - Input: idModelInstance
    - Output: a list of analysis
13. updateSolverStatus
    - Function: update solver status using information provided by the solver for users monitoring the computations
    - Input: idAnalysis, solver status
    - Output: status of processed request, message
14. getSolverStatus
    - Function: get solver status
    - Input: idAnalysis
    - Output: status of processed request, message
15. definePreference
    - Function: define user preferences for an analysis

- Input: idAnalysis, entity, entity type (MIN,MAX), value
- Output: status of processed request, message

16. getPreference
    - Function: get user preferences
    - Input: idAnalysis
    - Output: preference
17. storeSolution
    - Function: store solution (provided e.g., by scenario generator, or optimization solver)
    - Input: idAnalysis, entity values, tuple members
    - Output: status of processed request, message
18. login
    - Function: login to the kernel
    - Input: user credentials (login name and password)
    - Output: user-id, user's group, user's roles
19. createGroup
    - Function: create a group of users
    - Input: group-name
    - Output: status of the request, group-id
20. addGroupMembers
    - Function: add users to a group
    - Input: group-id, set of user-ids
    - Output: status of the request
21. assginUserRoles
    - Function: assign roles to a user
    - Input: user-id, set of role-ids
    - Output: status of the request
22. assginGroupRoles
    - Function: assign roles to a group
    - Input: group-id, set of role-ids
    - Output: status of the request
23. getGroupMembers
    - Function: get ids of a group members
    - Input: group-id
    - Output: a list of user-ids
24. getRoles
    - Function: get ids of all roles
    - Input: (none)
    - Output: ids of all defined roles
25. setPermission
    - Function: assign operation permissions to an object
    - Input: securityID, objectType, idObject, permissionID, permissionRequestType
    - Output: status of the request
26. getObjectPermissions
    - Function:get all securityID permissions for an object
    - Input: objectType, idObject
    - Output: list of securityIDPermissions
27. getObjectsWithUserPermissions
    - Function: get user's permissions for a list of objects
    - Input: userId, objectType, list of idObject

- Output: list of objectPermissions

7 Data warehouse

The Data-Warehouse (DW) has been designed and implemented in the PostgreSQL DBMS according to the requirements specified in [4]. It exploits the technology available from the SMT, thus it is also efficient for handling huge amounts of data with complex indexing structures. The corresponding DBMS schema is independent of the SMS (therefore also independent of specification of the model parameters, variables, and the corresponding indexing structures of these compound model entities), and is transparent for the client applications (therefore modifications of the model specification does not require modifications of these applications).

The data services used for accessing the DW assure consistency of all persistent elements of the modeling process, and support control of access to the data. However, the data services are hidden from the client applications; they manage (provide to the Kernel, and receive the previously provided) the data only through WSs using the DTOs suitable for their data structures.

The Kernel maintains (and thus the DW handles) the following data types:

- SMS.
- Model data, including sets, parameters, and results of analyses (all consistent with the SMS).
- Model instances.
- Model analysis tasks (specifications and results of the tasks corresponding to the requested runs of the scenario generator and solver manager).
- Data of authorised users of the DSS, together with their roles as users of the EnRiMa DSS.
- Data required for controlling the access to the DSS rsources.
- Model journal, i.e., automatically gathered information about modeling activities passing through the DSSE.

Although the DW structure is completely transparent for the clients, we present in Appendix A.1 four views on data entities and their relations, each view focused on:

- SMS, including the SMS version and optional submodels, specification of entities, selection of entities active in each model version, indexing structure (composed of sets, subsets (optionally indexed), containers of iterators (each composed of values of indices);
- data, including relations with the SMS, data versioning, indexing structure, handling of entity (parameters and variables) values, sets and their (optionally indexed) subsets, as well as members of all sets, and info on data uploaded from files.
- specific data view designed for fast access to large sets of data;
- model instance, including versions of the SMS and of data, specification of user preferences (for the instance analysis), computational task, results of analysis, state of the instance, solver of the optimization task, data about the users;
- data about users, their roles, groups of users, as well as data on access control to the DSS resources.

Note that each view presents the logical Entity-Relationship model (ER-model) of corresponding part of the DW, therefore many entities are included in more than one view. Moreover, the logical ER-model is independent of the DBMS used for actual implementation. ER-model is used for generating the physical model specific for a selected DBMS. A representation of the physical model generated for the PostgreSQL DBMS (used for the Kernel implementation) in the SQL-format used for generation of tables (including constraints for supporting the DB integrity) and sequences has posted[7] at the EnRiMa GitHub repository, see Section 10.

[7] In the `kernel/docs/dw_schema/sql/postgres` directory.

We stress that the independence of the DW structure from a particular data structure defined for a model provides qualitative advantages; in particular, it is more efficient (in terms of software development and maintenance) than traditional approaches in which the data-base tables and columns directly correspond to the model entities; the latter approach requires for each change of the model specification the corresponding modifications of both the data-base and all applications that are affected by the change.

8 Interface between Kernel and other DSS components

WSs serve as the only interface between clients and the Kernel; this greatly simplifies the DSS development, increases the reliability, and eases the maintenance and reusability of the DSS. Here we briefly comment on the two key issues related to the interface, because each of them is discussed in more detail in [4].

First issue is the interface between the Kernel and its clients. It is described in Section 6 of [4] which presents the general approach, and provides details on two topics:

- Triggering execution of the DSSE components.
- Illustrates use of WSs for downloading data and uploading results of optimization.

Moreover, detailed examples of the Kernel interface to diverse DSS components are provided by the testing applications documented in Section 9.

The philosophy of developing and using the WSs is summarized in Section 3.8 of [4]. It includes also an overview of public-domain tools supporting use of WSs within all programming languages and software tools commonly used for a wide-range of applications, including mathematical modeling and decision support. These tools support a generation of objects' definitions (e.g., classes) directly from the XML schema for all widely used programming languages and tools. Therefore, following the philosophy outlined there, and using the corresponding tools, substantially reduces the resources needed for the development of applications in diverse components, enables parallel development, supports consistency between these applications, and thus contributes to the effectiveness of the software development and to the reliability of the applications, as well as to the maintenance and reusability of the DSS components.

Practical examples of using such tools are given by the testing applications described in Section 9, and documented in the corresponding developer guides posted at the GitHub repository, see Section 10. This documentation guides the creation of the development environment, and shows how to replicate the application on a local computer; this helps in modifications of the test applications, and/or in the development of similar applications that consume the services provided by the Kernel.

9 Kernel validation

The Kernel has been validated using standard procedures for software development and testing; such procedures support verification of the quality and performance of the Kernel. The standard procedures applied are summarized in Section 9.1. Moreover, we have implemented the interface to the bug-tracking framework Jira; this is summarized in Section 9.6.

Additionally we have developed a testing environment composed of advanced[8] mock-ups of three EnRiMa DSS components:

- test-GUI, see Section 9.2;
- Data wrapper, see Section 9.3;

[8] We call them advanced because they provide some functionality of the corresponding software components.

• Scenario generator, see Section 9.4.

Finally we have implemented a complete integration test based on the strategic planning use case described in Section 9 of [3]. This test is described in Section 9.5.

9.1 Testing with SoapUI

We have selected the SoapUI[9] as a tool for testing the WSs because it is the world's leading functional testing tool for SOA and WS. With its easy-to-use graphical interface, SoapUI allow us to easily create and execute automated functional and load tests.

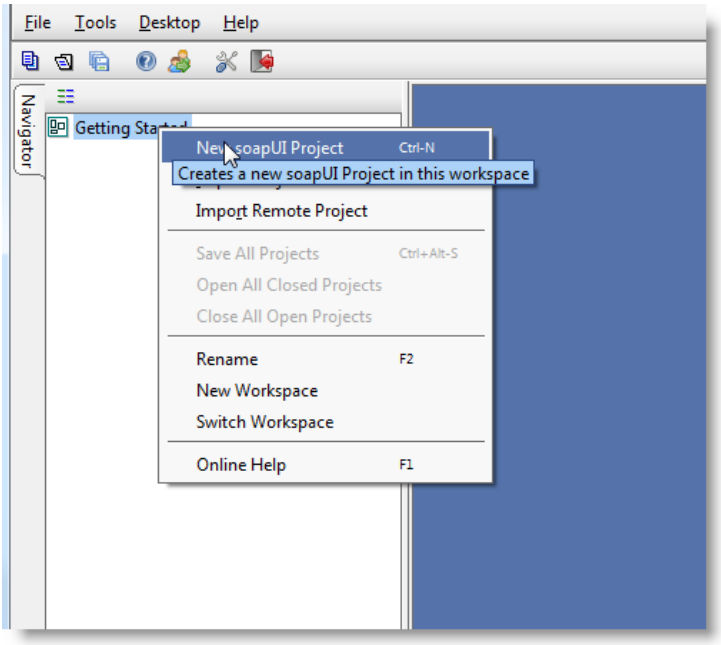

Figure 8: Creation of the test project with SoapUI.

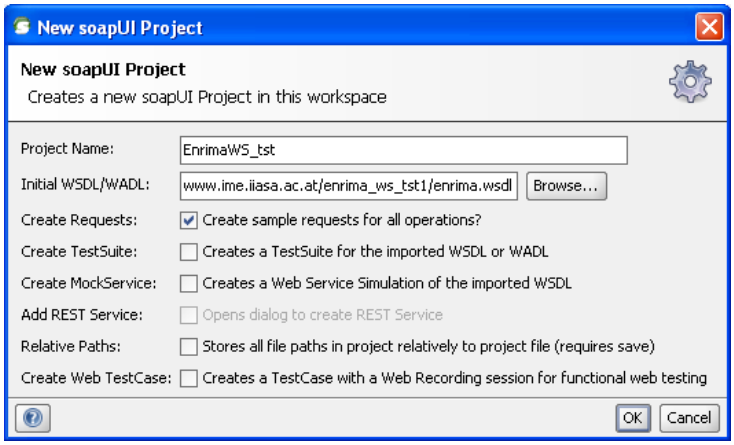

Figure 9: Adding the EnRiMa WSDL to the test project.

The testing procedure of WSs has followed the standard steps:

[9] `http://www.soapui.org/`.

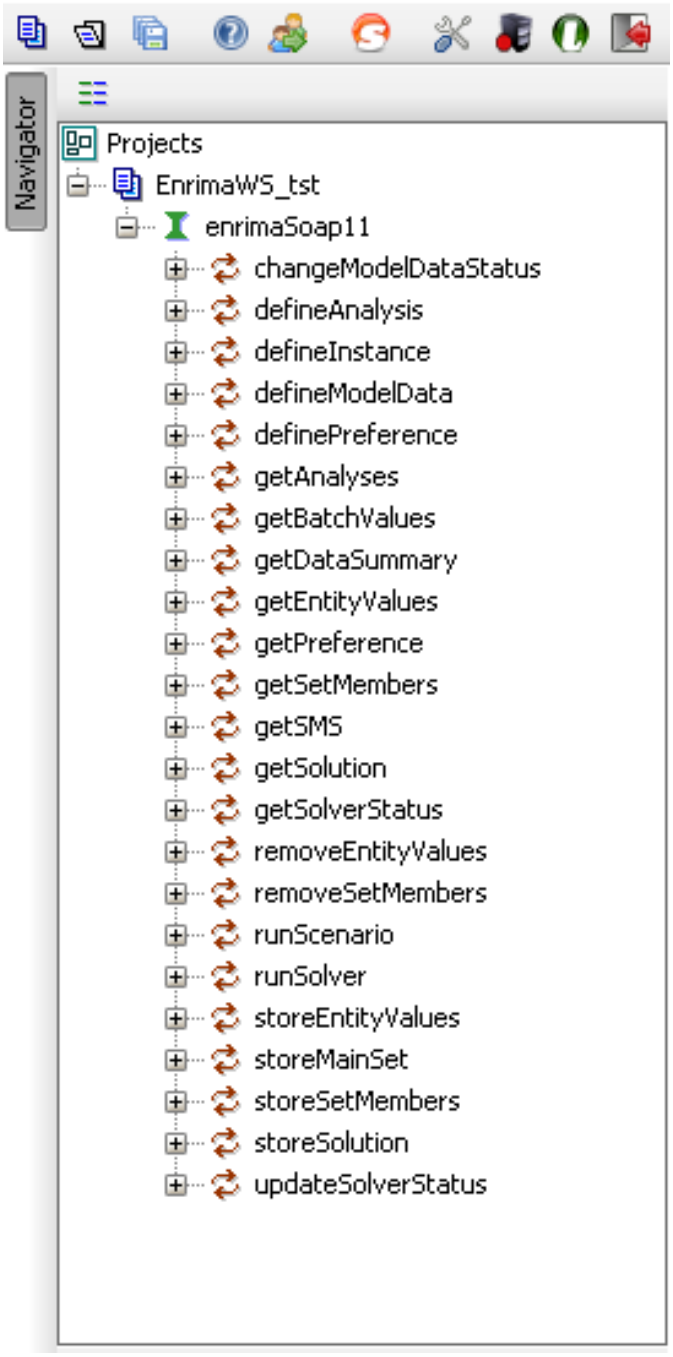

Figure 10: Operations provided by the WSs defined in the WSDL.

1. Download SoapUI from `http://sourceforge.net/projects/soapui/files/`.
2. Install SoapUI
3. Create a SoapUI project:
   - In the navigator, i.e., the tree structure at the left in the SoapUI GUI, right click on the project node ”Projects” and select ”New SoapUI Project”, see Figure 8.
   - In the New SoapUI Project dialog enter a project name `EnrimaWS_tst`, and the Initial WSDL/WADL `http://www.ime.iiasa.ac.at/enrima_ws_tst1/enrima.wsdl`, see Figure 9. After clicking OK, followed by a successful adding of the WSDL to the project, the available operations of the provided WSs will be displayed in the navigator, as illustrated in Figure 10.

Figure 11: Testing the *getSMS* WS.

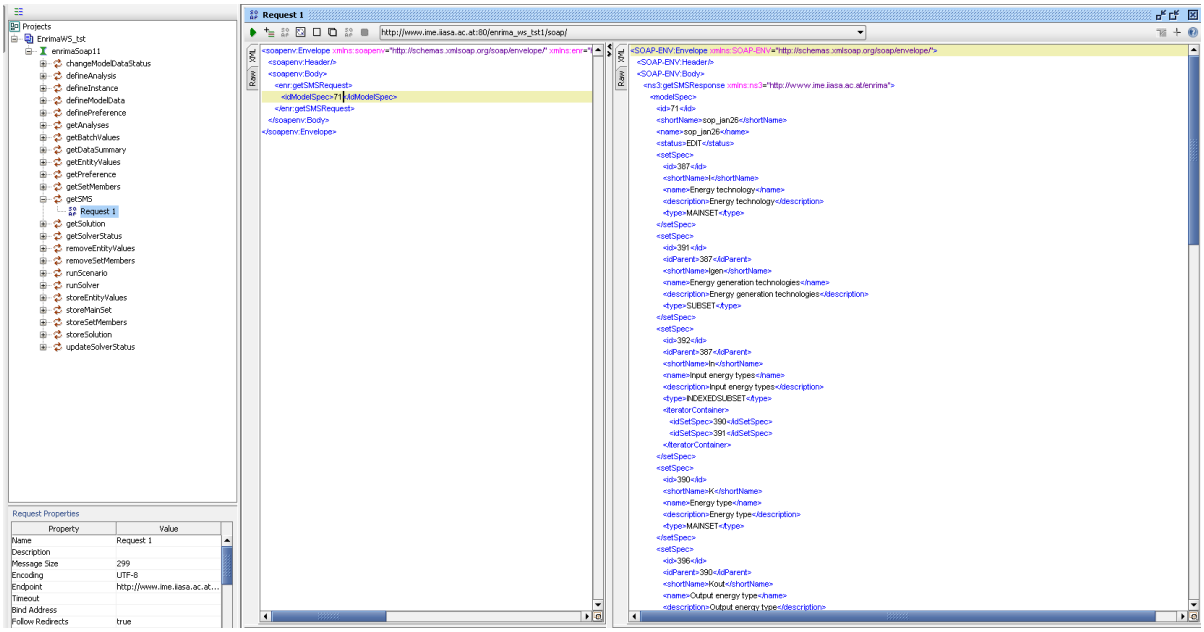

Figure 12: SMS of model 71 provided by the *getSMS* WS.

4. Functional testing of the WSs:
   - Select an operation from the services displayed in the navigator by double-clicking on its name; Figure 11 shows the result of selecting the *getSMS* WS. (for example: select ”getSMS”), then double click on the request.
   - Select a simple model prepared in the testing environment by replacing the `<idModelSpec>?</idModelSpec>` with `<idModelSpec>71</idModelSpec>` (this actually selects the model with ID equal to 71), and click the green triangle to run the functional test of the *getSMS* WS; the XML format of the results will be displayed in the right window, as illustrated in Figure 12.

   The above summarized tests have been successfully applied to all WSs described in Section 6. The described testing environment can easily be replicated, and thus used for independent testing the Kernel, as well as the forthcoming final implementation of the Kernel.

9.2 Test-GUI

We have developed a simple implementation of the GUI (called test-GUI to distinguish it from an actual GUI implementation) aimed at testing and verifying the functionality of the WSs typically used by a GUI, as well as providing an example of integrating the WSs with one of the most popular frameworks for the GUI development. The application can be installed at a local environment, and then also be used e.g., for a user-friendly display of the SMS. This application accompanied with the corresponding developer guide is posted at the Kernel GitHub repository.

9.3 Data wrapper

A simple data-wrapper has been developed to illustrate use of public domain tools for integration of WSs with applications developed in the Java programming language. This application can also be considered as a template for an actual data-wrapper. It is posted, together with the corresponding developer guide, at the Kernel GitHub repository.

9.4 Scenario generator

A simple mock-up of the scenario generator has been developed to illustrate use of public domain tools for integration of WSs with applications developed in the C++ programming language. It also is posted, together with the corresponding developer guide, at the Kernel GitHub repository.

9.5 Complete integration test

The complete integration test is based on the strategic planning use case described in Section 9 of [3]. This use case is composed of the following elements:

- Name: Data for strategic planning
- Actor/user: Building owner financial manager
- Goal/purpose: Prepare data for strategic planning
- Triggers: same as triggers for strategic
- Input: a base data set
- Results: an updated data set

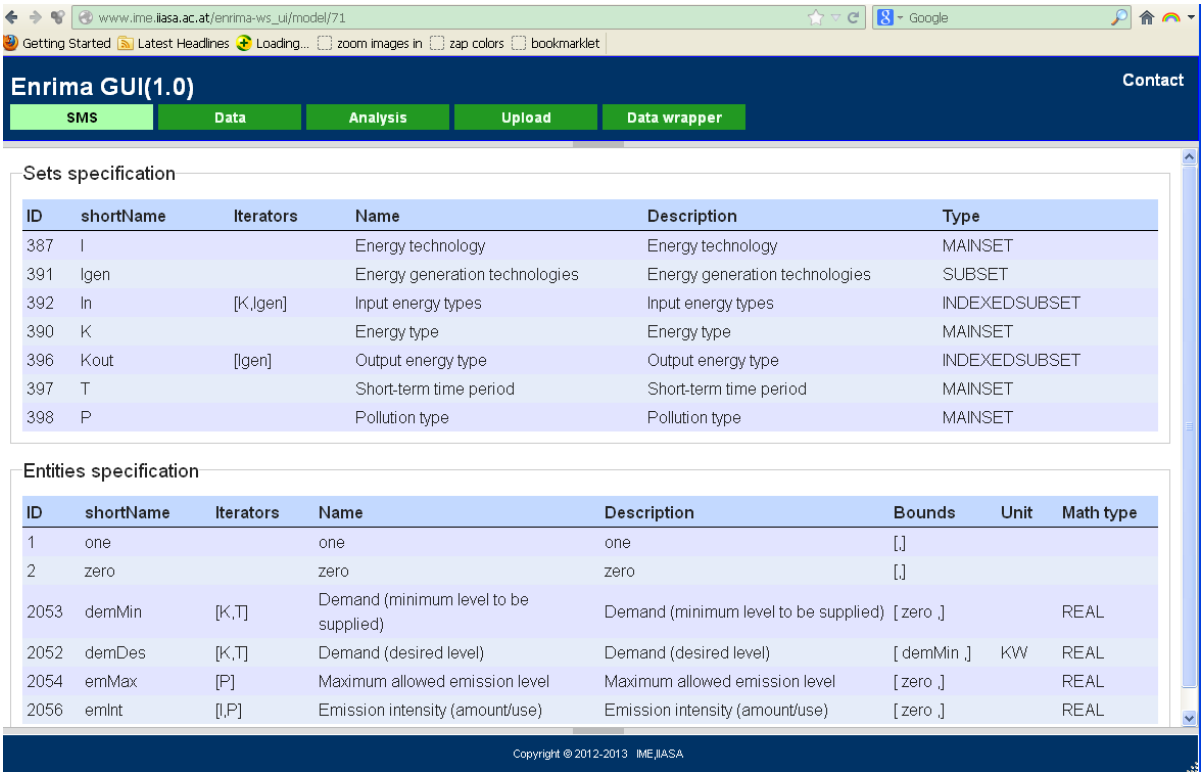

Figure 13: View on the SMS provided by the test-GUI application.

- Steps (workflow):[10]

[10]Each original step is followed by the comment on using the corresponding WS.

1. Select a version of symbolic model specification.
   Using a web browser one can access to the `test_gui` web application[11] This URL contains the SMS id=71; its sets and entities specification is shown in the XML format Figure 13. The test GUI application displays the SMS in a user-friendly way.
2. Receive all needed data updates from external sources.
   The user (or users responsible for specific sub-sets of data) shall upload the data, e.g., through a site-specific application similar to the data-wrapper described in Section 9.3, which also illustrates how a data-wrapper can use the WSs for uploading data to the Kernel. Data conversions (from the formats provided by the sources to the DTOs corresponding to the data objects used by the applications processing data) shall be done by the site-specific data wrappers. The provided data-wrapper application converts the data provided in the CSV-format file.
3. Receive all needed data updates from local sources.
   This step in handled in a way similar to that described for step 2.
4. Process/adapt data.
   Processing/adaptation of data should be done by a data-wrapper, or another dedicated application. Verification and approval of the data shall be done within the data management work-flow supported by the GUI.

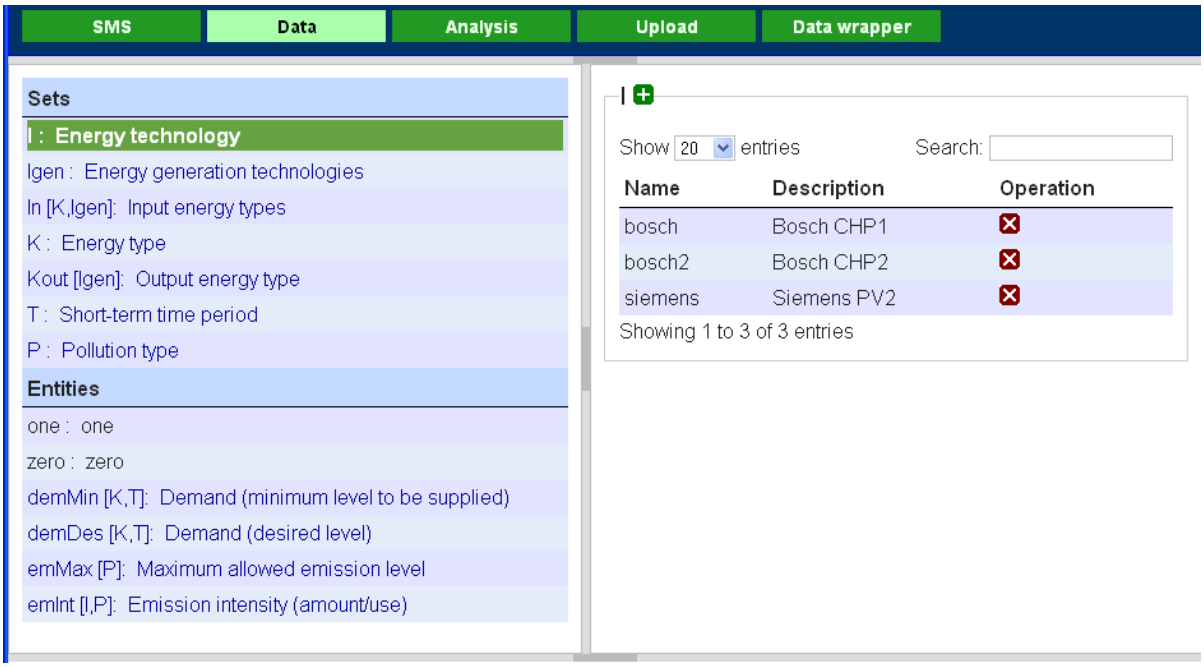

Figure 14: Interactive modification of data through the test-GUI application.

The following three ways of uploading data are implemented in the test-GUI:

⋆ For a small set of data, we can use the forms designed for adding/removing data (set members, as well as the parameters with the accompanied indices) interactively. For example, one can add a new energy technology ”CHP1” through the following steps illustrated in Figure 14: (1) click the ”Data” button on the top toolbar, then you will see all the sets and entities listed in the left window; (2) click the ”I: Energy technology”, then you will see all the energy technologies listed in the right window. (3) click the green ”+” button, then you can type ”CHP1” and click ”Add” button to add the new energy technology.

[11] Posted at `http://www.ime.iiasa.ac.at/enrima-ws_ui/model/71`.

- ⋆ For site-specific data, one can use a data-wrapper to upload the data. Such a data-wrapper can easily and seamlessly be integrated into the provided test-GUI.
- ⋆ For large sets of data, one can upload data prepared in the XML-format file by the following steps. This functionality is provided by the test-GUI, and can be used by: (1) clicking the ”Upload” button on the top toolbar, (2) selecting the XML file, and (3) clicking the ”Upload” button.

5. Optionally, run the scenario generation (see the corresponding use-case).
   The corresponding WS has not been implemented in the testing environment.
6. Check data completeness for strategic planning.
   This functionality should be implemented through a dedicated GUI; this was not implemented in the test-GUI application.
7. Commit data.
   In the testing environment, the uploaded data is committed automatically. In the actual use of the Kernel, the data commitment shall be confirmed through the GUI.
8. Define the corresponding model instance.
   In the test-GUI, the version of SMS, model data, model instance are predefined. In the actual implementation of GUI, this functionality can be implemented by a simple UI form providing optional fields for the model-instance description and optional notes.
9. Send notification to the use case that requested the data is available.
   This functionality has not been implemented in the testing environment.

9.6 Jira

The test-GUI application described in Section 9.2 also shows an effective way of using the modern issue and bug-tracker environments like Jira.[12] Such environments provide two key functionalities very useful for management of the deployed software; namely, catching the exceptions in executions of the applications, and an effective handling of feedback from the users. We summarize below how these functions have been implemented in the test-GUI, and therefore are available for other EnRiMa DSS components.

The handling of exceptions occurring in execution of remotely run applications is rather difficult because information about such exceptions are typically difficult to obtain, especially from users not familiar with the software development technology. Therefore, exceptions catching should be implemented in each applications, and information about exceptions should be delivered automatically to the software developers. Such a functionality has been implemented as a dedicated module using the API of Jira, and is used also in the test-GUI.

The test-GUI shown in Figure 13 includes the API to the Jira application routinely used by the IIASA team for the management of software projects. We encourage users of our application to use *Contact* link, included also in the test-GUI (seen at the top-right side of the screen shown in Figure 13). After clicking on the *Contact* button a form shown in Figure 15 is generated, and the user can provide the developers with her/his comments or problem reports (the categories of the user input are customized for each application, and the user selects one of them from the list shown at the top of the form). After submitting the form a short confirmation message is displayed.

Note that the test-GUI is publicly available therefore no user authorization is included in this application. Therefore, for obvious reasons, the form includes the so-called captcha, and also asks for the email address of the user. In actual implementation the user authorization shall be included, therefore the user-email will be available from the Kernel, and the captcha protection

[12] `http://www.atlassian.com/JIRA`.

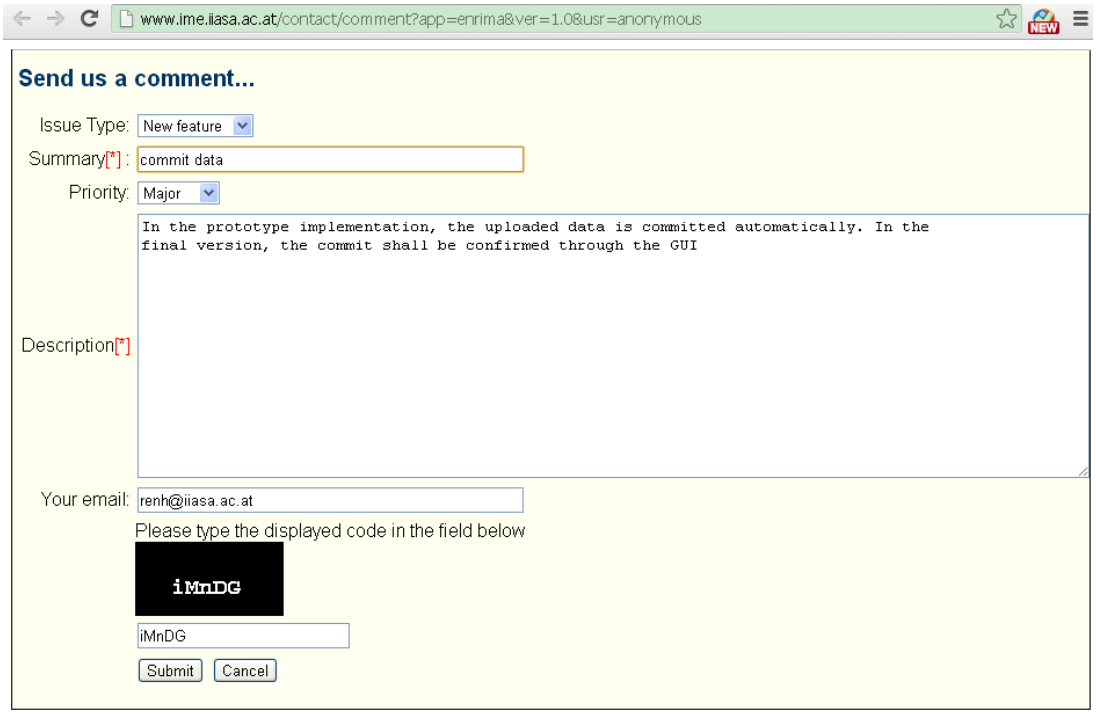

Figure 15: User interface to the Jira-based issue reporting.

will not be desired.

From the user point of view the interface is simple. However, the actual information passed to the software developers is reach; namely, Jira stores the user-comment in the DBMS together with information enabling an easy recreation of the situation when the comment was generated. The developers have access to the information shown in Figure 15; note that it includes also the link, which enables authorized developers a one-click recreation of the application state that was at the time the comment was sent, which obviously supports an efficient way of handling the user comments.

Notification of incoming comments can be enabled, as well as an automatic assignment of a software development team member who should start the workflow of handling the request. Jira also supports project management, thus the reported problems and suggestions are included into the set of issues related to the corresponding software project. The project management supports definitions of workflows and specifications of issues (e.g., type, deadlines, links with other issues, software components and versions) as well as linking the workflows with version control systems, such as Git (see Section 10).

10 Documentation and the GitHub Kernel repository

The Kernel is a complex software and therefore its documentation is by far too large to be included in any report. Moreover, the Kernel has been, and will continue to be modified in order to better address forthcoming requirements; therefore any traditional (printed) documentation would soon be outdated. In such situations a version control system combined with a Web repository appears to be the best way for providing the documentation.

Most probably, every software development team uses a version control system. The Kernel

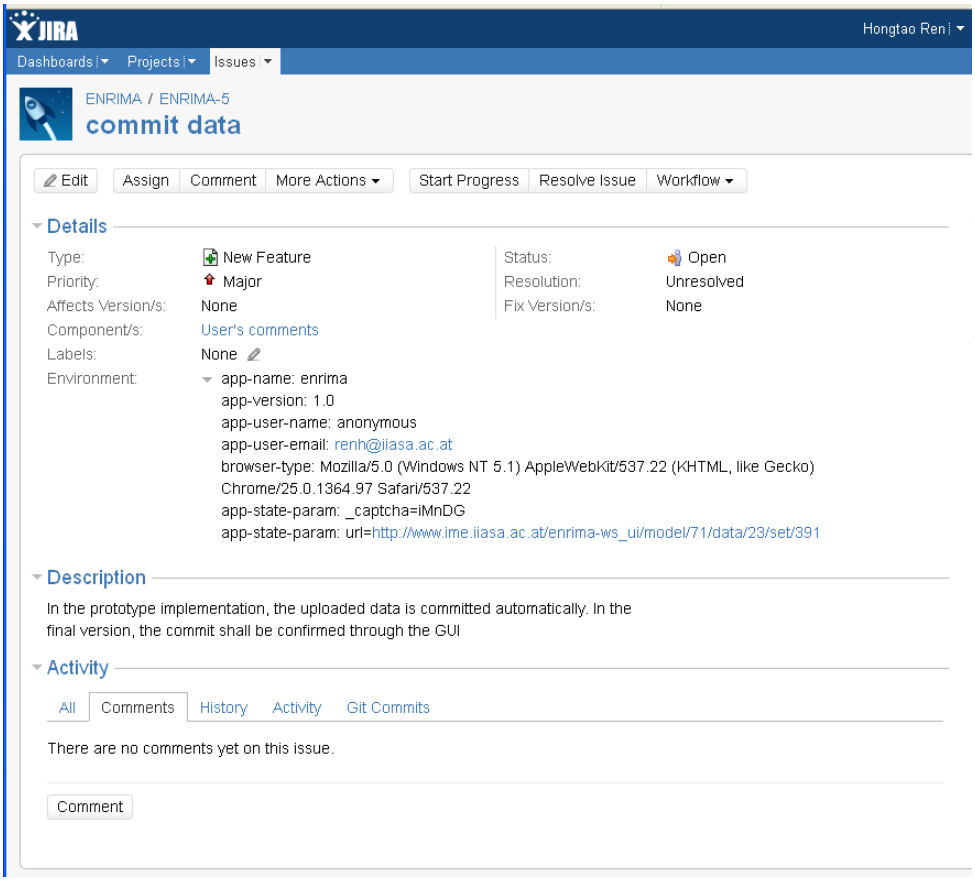

Figure 16: Information available to the developers through the the Jira-based issue reporting.

development team uses the Git[13], a free and open source distributed version control system. We use the 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 Kernel repository at the GitHub[14], the web-based hosting service that offers free accounts for open source projects.

The EnRiMa Kernel GitHub site `https://github.com/enrima-dev4`[15] has currently only one repository called *kernel* which contains the 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 report. These include the source-code of the client-applications developed with two purposes: first, to test the Kernel (additionally to the standard tests performed during the development phase); second, to show several simple ways 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

[13] `git-scm.com`.
[14] `http://github.com`.
[15] The repository name was derived from the *developers of EnRiMa WP4*.

documents its implementation at a local computing environment.

The repository is organized into a directory structure, and is 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.

11 Conclusions

The EnRiMa DSS consisting of several modular software components has been developed to support decision-making aimed at improving energy-efficiency of buildings; the DSS shall also be reusable, i.e., easily adaptable to other buildings or facilities. Therefore, a reliable and reusable Kernel supporting consistent integration of the DSS components is a key condition to achieve both goals. Any DSS has to meet specific needs of the users for which it is provided. In order to meet these needs efficiently, it is rational to design and implement the DSS in such a way that possibly many components can be reused without substantial modifications; and the needed modifications should be easy to implement. Moreover, since the building ICT infrastructures use diverse hardware and software platforms, interoperability of the DSS components is necessary for efficiency of reusability. The DSSE architecture described in [4] 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 Kernel implementation follows the corresponding requirements described in [4]. The WSs provided by the Kernel 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 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 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. The Kernel provides all services specified in the requirement analysis, and has been extensively and successfully tested through not only a standard testing tool, but also through the demonstration versions of three main types of its clients. Thus, the Kernel can be used by the other EnRiMa DSS components also for implementations of the DSS in other buildings that may require modifications of the underlying models and therefore also handling other data sets, as well as supporting diverse groups of the DSS users.

References

[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, SU, UCL, URJC, SINTEF, CET, AND TECNALIA. Requirement analysis. Deliverable D4.1, European Commission FP7 project number 260041, Brussels, Belgium, 2011.

[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, CET, TECNALIA, AND IIASA. DSS Kernel final implementation. Deliverable D4.7, European Commission FP7 project number 260041, Brussels, Belgium, 2014.

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

A Appendix

A.1 Data-warehouse schema

The DW schema is far too complex to be fully documented in this report. Therefore, it is documented at the Kernel GitHub repository. In order to illustrate the schema we present here five the entity-relation views, each focused on the corresponding functionality. These views are summarized in Section 7.

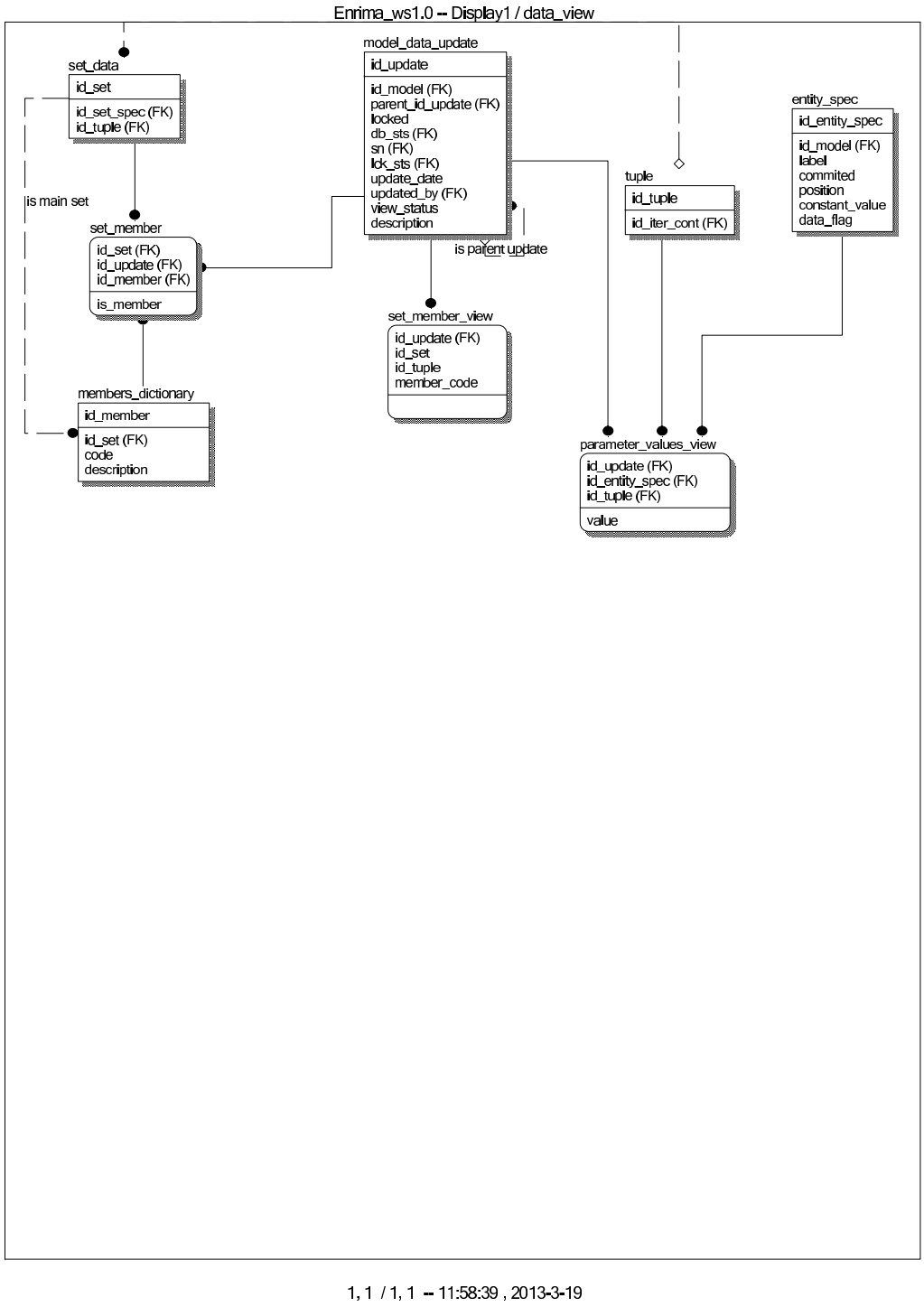

Figure 17: The entity-relationship view focused on fast access to large data-sets.

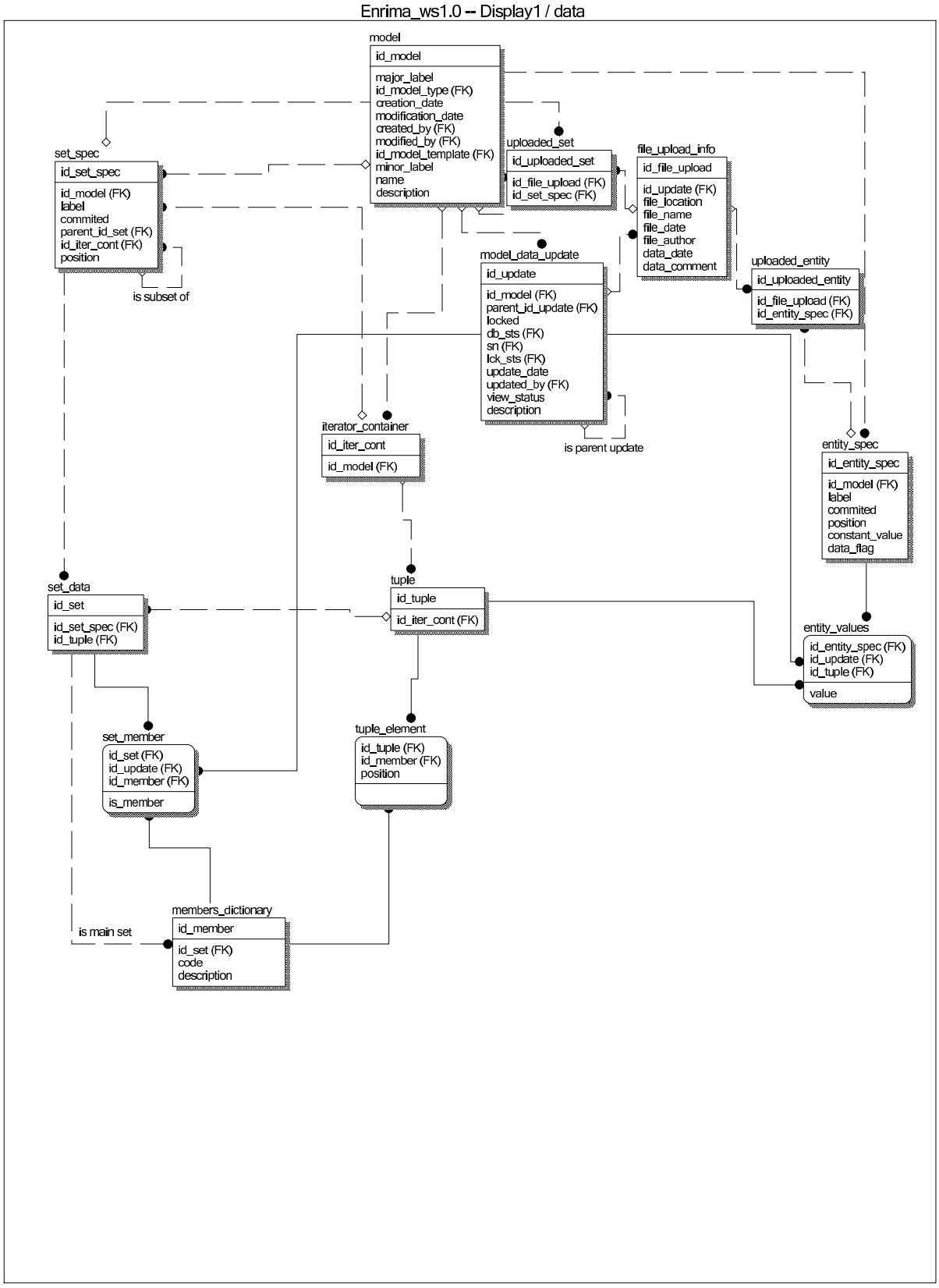

Figure 18: The entity-relationship view focused on model data.

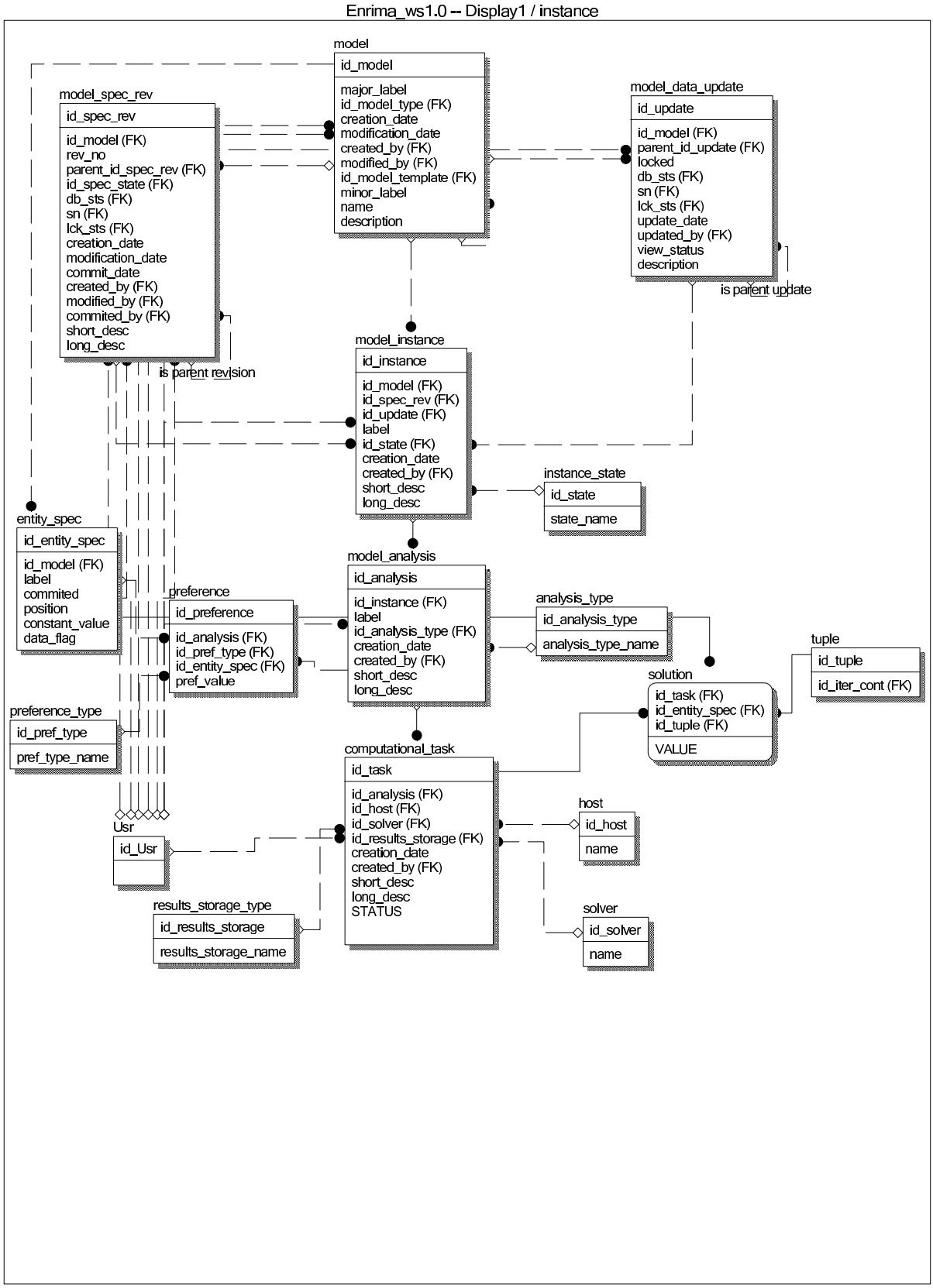

Figure 19: The entity-relationship view focused on the model instance.

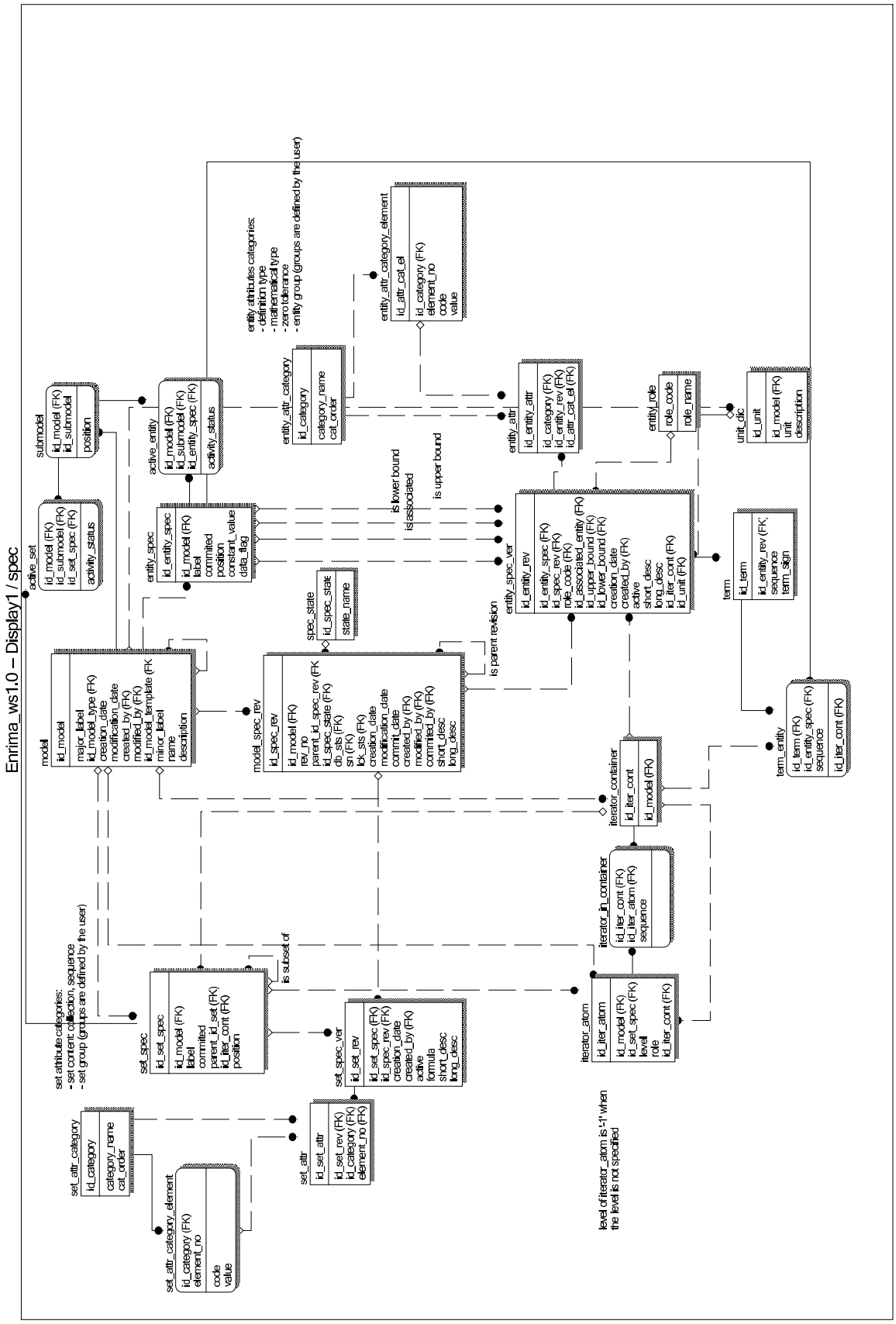

Figure 20: The entity-relationship view focused on SMS.

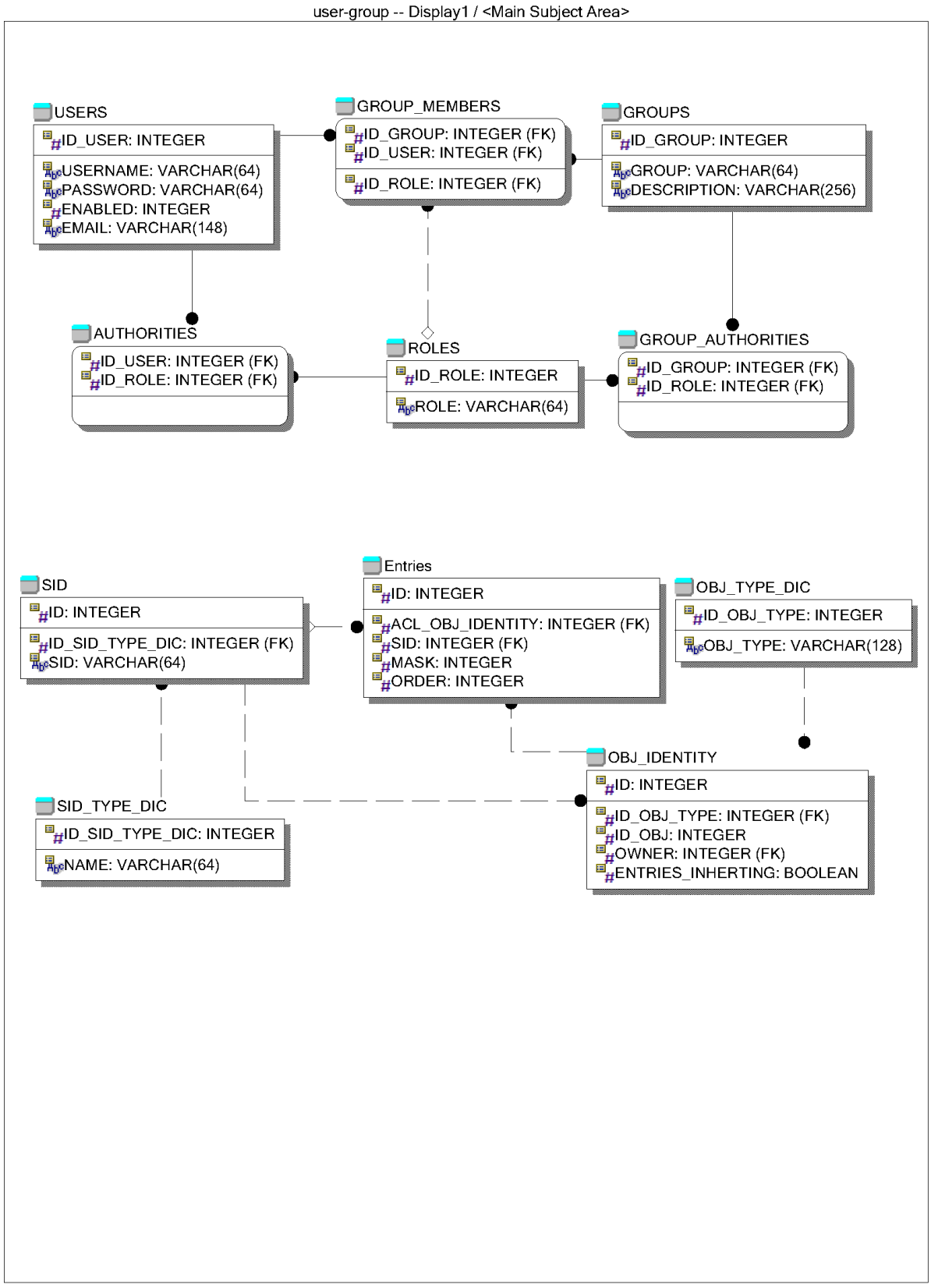

Figure 21: The entity-relationship view focused on the users and ACL.

A.2 Example of the XML-format representation of the WS input and output

As an illustration of XML-format request and response specifications we provide below such specifications for the *getSMS* WS.

Request specification

Request specification is composed of only the SMS id, defined in the sms.xsd schema as `idModelSpec`. The corresponding XML-format text is therefore very simple:

```
<enr:getSMSRequest>
   <idModelSpec>71</idModelSpec>
</enr:getSMSRequest>
```

Response specification

The response (output) of the *getSMS* WS is composed of the SMS provided in the format defined by the sms.xsd schema. The corresponding XML-format text is therefore rather complex; however, we recall that such text is processed by suitable applications, and therefore aimed to be directly seen by neither the DSS users nor developers of the client applications. Note that the corresponding name space is declared in the first list of the listing below.

The SMS used for the example shown below has been generated interactively using the GUI developed for the SMT [5] prototype; the SMS corresponds to an early testing version of the EnRiMa strategic planning model.

```
<ns3:getSMSResponse xmlns:ns3="http://www.ime.iiasa.ac.at/enrima">
<modelSpec>
   <id>71</id>
   <shortName>sop_jan26</shortName>
   <name>sop_jan26</name>
   <status>EDIT</status>
   <setSpec>
      <id>387</id>
      <shortName>I</shortName>
      <name>Energy technology</name>
      <description>Energy technology</description>
      <type>MAINSET</type>
   </setSpec>
   <setSpec>
      <id>391</id>
      <idParent>387</idParent>
      <shortName>Igen</shortName>
      <name>Energy generation technologies</name>
      <description>Energy generation technologies</description>
      <type>SUBSET</type>
   </setSpec>
   <setSpec>
      <id>392</id>
      <idParent>387</idParent>
```

```
      <shortName>In</shortName>
      <name>Input energy types</name>
      <description>Input energy types</description>
      <type>INDEXEDSUBSET</type>
      <iteratorContainer>
         <idSetSpec>390</idSetSpec>
         <idSetSpec>391</idSetSpec>
      </iteratorContainer>
   </setSpec>
   <setSpec>
      <id>390</id>
      <shortName>K</shortName>
      <name>Energy type</name>
      <description>Energy type</description>
      <type>MAINSET</type>
   </setSpec>
   <setSpec>
      <id>396</id>
      <idParent>390</idParent>
      <shortName>Kout</shortName>
      <name>Output energy type</name>
      <description>Output energy type</description>
      <type>INDEXEDSUBSET</type>
      <iteratorContainer>
         <idSetSpec>391</idSetSpec>
      </iteratorContainer>
   </setSpec>
   <setSpec>
      <id>397</id>
      <shortName>T</shortName>
      <name>Short-term time period</name>
      <description>Short-term time period</description>
      <type>MAINSET</type>
   </setSpec>
   <setSpec>
      <id>398</id>
      <shortName>P</shortName>
      <name>Pollution type</name>
      <description>Pollution type</description>
      <type>MAINSET</type>
   </setSpec>
   <entitySpec>
      <id>1</id>
      <shortName>one</shortName>
      <name>one</name>
      <description>one</description>
      <constantValue>1.0</constantValue>
      <role>CONSTANT</role>
   </entitySpec>
```

```
<entitySpec>
   <id>2</id>
   <shortName>zero</shortName>
   <name>zero</name>
   <description>zero</description>
   <constantValue>0.0</constantValue>
   <role>CONSTANT</role>
</entitySpec>
<entitySpec>
   <id>2053</id>
   <shortName>demMin</shortName>
   <name>Demand (minimum level to be supplied)</name>
   <description>Demand (minimum level to be supplied)</description>
   <idLowerBound>2</idLowerBound>
   <iteratorContainer>
      <idSetSpec>390</idSetSpec>
      <idSetSpec>397</idSetSpec>
   </iteratorContainer>
   <role>PARAMETER</role>
   <mathType>REAL</mathType>
</entitySpec>
<entitySpec>
   <id>2052</id>
   <shortName>demDes</shortName>
   <name>Demand (desired level)</name>
   <description>Demand (desired level)</description>
   <idLowerBound>2053</idLowerBound>
   <iteratorContainer>
      <idSetSpec>390</idSetSpec>
      <idSetSpec>397</idSetSpec>
   </iteratorContainer>
   <role>PARAMETER</role>
   <mathType>REAL</mathType>
   <unit>KW</unit>
</entitySpec>
<entitySpec>
   <id>2054</id>
   <shortName>emMax</shortName>
   <name>Maximum allowed emission level</name>
   <description>Maximum allowed emission level</description>
   <idLowerBound>2</idLowerBound>
   <iteratorContainer>
      <idSetSpec>398</idSetSpec>
   </iteratorContainer>
   <role>PARAMETER</role>
   <mathType>REAL</mathType>
</entitySpec>
<entitySpec>
   <id>2056</id>
```

```
      <shortName>emInt</shortName>
      <name>Emission intensity (amount/use)</name>
      <description>Emission intensity (amount/use)</description>
      <idLowerBound>2</idLowerBound>
      <iteratorContainer>
         <idSetSpec>387</idSetSpec>
         <idSetSpec>398</idSetSpec>
      </iteratorContainer>
      <role>PARAMETER</role>
      <mathType>REAL</mathType>
   </entitySpec>
</modelSpec>
</ns3:getSMSResponse>
```