Meta-model Based and Component Based Approach for Information Systems Design

Baiba Apine, Ilgvars Apinis, Ojars Krasts, Uldis Sukovskis

Riga Institute of Information Technology
Kuldigas 45, LV-1083 Riga, Latvia
uldis.sukovskis@dati.lv

Abstract

The paper introduces an object-oriented and component based approach for software design. This approach allows to design software in convenient and easy to understand manner using metamodel of real world system. The first step - building of the entity-relationship diagram, is discussed using project management support information system as an example. Universal repository as a storage of model objects is proposed to build using meta-meta-model based three-layer architecture. And the final step - implementation of the designed model using COM objects is described with illustrations from the same example of project manager's information system.

Keywords: software development, object-oriented, meta-model.

1. Meta-model Based Tool for Registering of Objects

One particular implementation of meta-model based universal repository will be used to explain an approach applicable to design of different types of information systems. PARK is a tool for registering various kinds of objects and their attributes into universal repository. We created the tool for registering information about software development projects. This includes information about project itself, staff involved, activities and documents. Documents could be generated using information in the repository and preliminary designed document templates. Tool accepts MS Word, MS Excel and MS Project documents. The PARK tool is considered as sample of usage of the repository only.

The main features of this tool are:

- a relatively small number of concepts used: only eight object classes and 14 types of relations between objects, with the terms defined intuitive and easily interpreted even by a novice;
- new types of objects and relationships could be added easily;
- information from the repository could be retrieved in various ways;
- project documentation could be generated using information from the repository.

The entity-relationship diagram shown in Figure 1 is the formal meta-model for PARK. All objects have their own attributes depending on object type. Besides individual attributes there are some aplicable to all objects: for example, name, type and who and when created or modified this object in the repository.

Project is an object containing information about software development project. Usually there is one Project object in the repository only. Individual attributes for the Project object are: identifier, development environment, start date, end date etc.

Performer is an object containing information about a man involved in the project development process. This could be project manager, programmer, customer etc. Name, position, phone, fax and e-mail address are attributes of the Performer object.

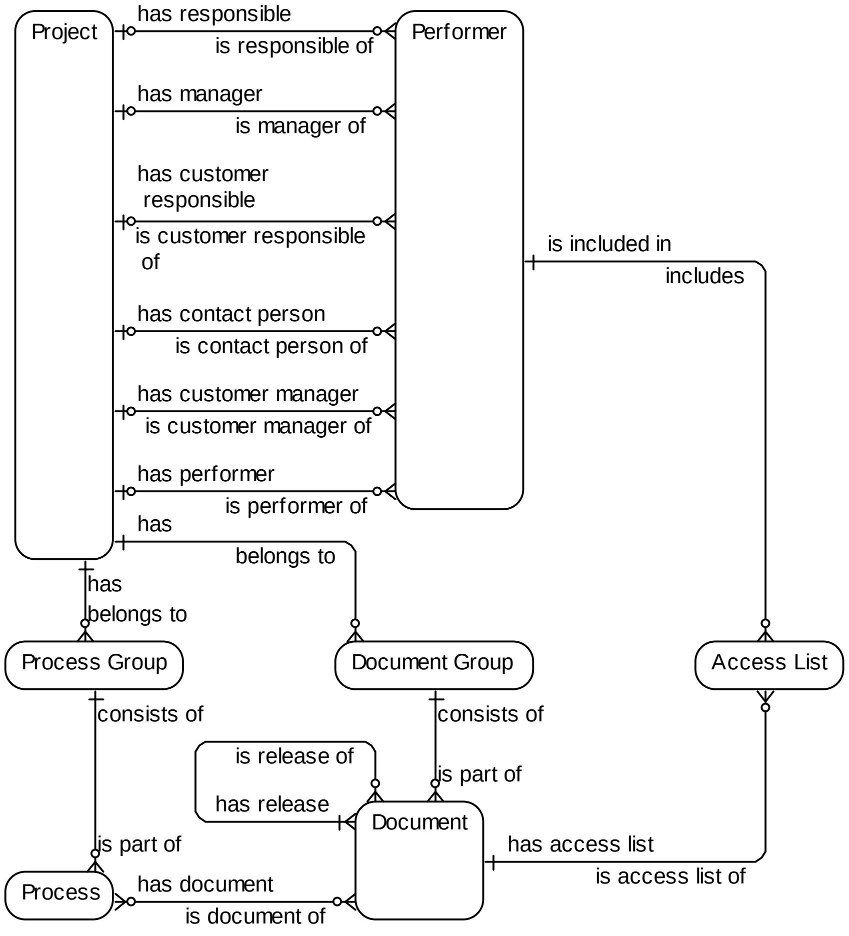


Figure 1. Object types and relationships

Process group is used for grouping of some processes. Process group samples are primary software life cycle processes, supporting processes etc.

Process is an object describing any activity within software development process. Start date, end date, description are attributes of the Process object. Samples of the Process object are coding, testing, auditing etc.

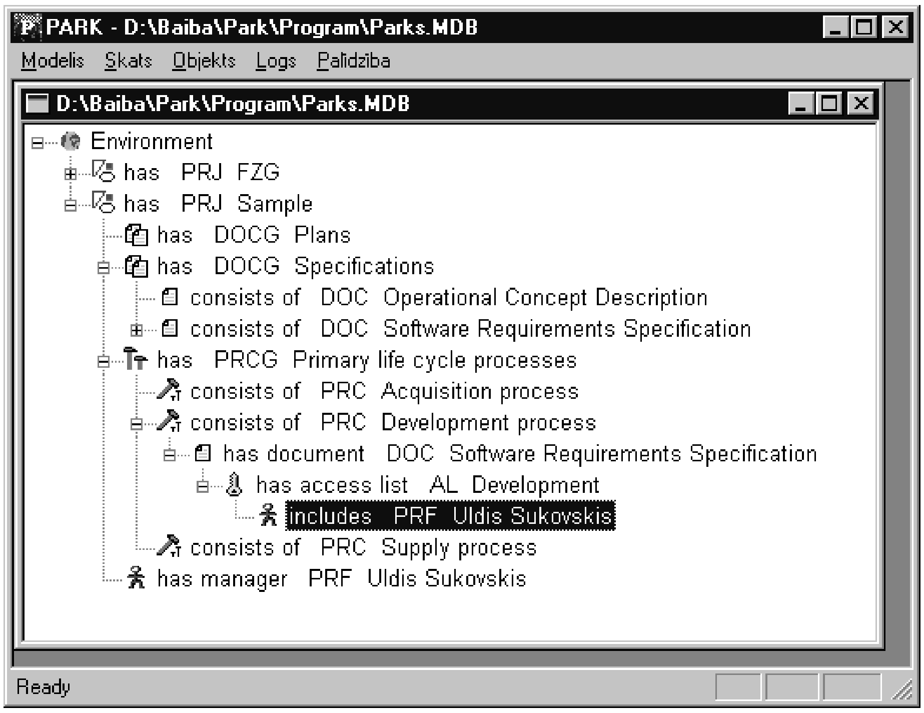
Document group is a head object for some group of documents. Document is an object describing a peace of project documentation. User manual, software requirements specification are samples of the Document object.

Access list is an object describing whether the particular Performer object has read, read/write or write access to the particular Document object.

PARK presents a model in the window which shows the objects and their relationships in a tree-like structure (see Figure 2). The way in which objects are displayed on the object tree is user defined. The branches of the tree are objects interconnected by relationships.

2. Repository of Models

The data describing a model are interrelated in a complex manner. Several versions of model need to be saved. Each model consists of various objects and relationships between them described by meta-model. Also a model itself may be considered as a complex object which refers to other models.



PARK - D:\Baiba\Park\Program\Parks.MDB

Modelis Skats Objekts Logs Palīdzība

D:\Baiba\Park\Program\Parks.MDB

```
Environment
  has PRJ FZG
  has PRJ Sample
    has DOCG Plans
    has DOCG Specifications
      consists of DOC Operational Concept Description
      consists of DOC Software Requirements Specification
    has PRCG Primary life cycle processes
      consists of PRC Acquisition process
      consists of PRC Development process
        has document DOC Software Requirements Specification
          has access list AL Development
            includes PRF Uldis Sukovskis
      consists of PRC Supply process
    has manager PRF Uldis Sukovskis
```

Ready

Figure 2. PARK window presenting information about software development project

Taking this into account we designed “universal” Repository. Structure of the Repository could be described with meta-meta-model (see Figure 3).

Meta_Object stores information what types of objects contain the meta-model. Long and short names of the object type are essential attributes for this object. For instance, objects of type named Performer and short name PRF (see Figure 2) could be stored in the Repository for the PARK meta-model.

Meta_Link stores information about relationships in the meta-model. Name of the relationship, two role names and cardinalities are the attributes of this object. For instance, PARK meta-model contains relationship consists_of / is_part_of with cardinalities 1..1 and 0..N (see Figure 2).

Meta_Object_Link stores information what objects are connected with what relationships.

Object stores specific information about objects of all types defined by meta-model (see Figure 2). This entity stores information about project as well as about performers. Values of attributes specific for every object type are coded as string. Role stores information about relationships between Objects.

There are three layers, related to design and usage of the Repository:

1) Physical layer: the lowest level of abstraction, at which it is described HOW the data are actually stored.

2) Conceptual layer: the next level of abstraction at which it is described WHAT data are actually stored in the data base and the relationships that exist among data. At this level it is possible to manipulate with generalized objects in a standard fashion, of course using methods which are supplied by a Physical layer.

3) Logical view layer: this is the highest level of abstraction at which only a part of the entire data base is described. This level is for particular tool. Many logical views may be provided for the same Conceptual level. The Logical view layer can’t use functions provided by the Physical layer directly.

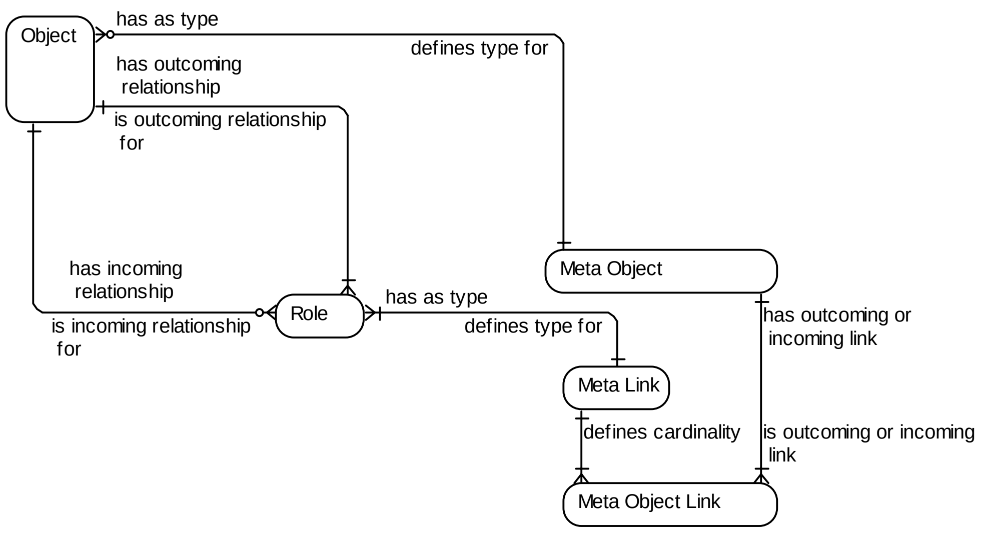

Object
has as type
defines type for
has outcoming relationship
is outcoming relationship for
Meta Object
has incoming relationship
is incoming relationship for
Role
has as type
defines type for
has outcoming or incoming link
Meta Link
defines cardinality
is outcoming or incoming link
Meta Object Link

Figure 3. Meta-meta-model of the Repository

Such the layered architecture allows relatively easy changing of the underlying physical level database (for the case if it is necessary to change a platforms, or in the case of performance problems). If a new tool is created, then corresponding Logical view can be added. We use PARK meta-model as the base of such Logical view.

Taking into account the layered architecture of the Repository, we choose object - oriented approach in tool architecture. Object - oriented approach fits together with MS COM application architecture. Each object is implemented as COM object. There are attributes and functions common for all object types. Each object inherits those and has its own attributes and functions (see Figure 4). Objects expose their functions which are legal for the particular object type. These functions could be used by any software tool. We created one – Browser. Browser operates with objects on the Logical view layer. The main function of the Browser is to show the contents of the Repository according to the meta – model. Browser itself doesn't support any objects' function. It calls those implemented in corresponding object (see Figure 4).

This approach allows easy to change meta-model. For instance, add new object types. New Meta_Object and appropriate links to other object types must be registered in the Repository and corresponding COM object has to be implemented.

3. Functionality of Objects

Every object defined in the meta-model has functions applicable for this object. There is functionality corresponding to the conceptual layer and functionality corresponding to the logical view layer. Every object is implemented as COM object having its own set of functions. There are some functions equal for all objects defined in the meta-model. Those functions are defined in the conceptual layer. They don't depend on the type of object. Examples of functions defined in the conceptual layer are:

- functions Edit and View for entering or changing of the attributes of the object or viewing object attributes without ability to change them;
- functions Add object and Delete object for adding and deleting objects in the repository; function Add object allows add objects having some relationship with the particular object only;

- functions Add relationship and Delete relationship for adding and deleting relationship starting from the particular object;
- function Rename for global renaming of the object.

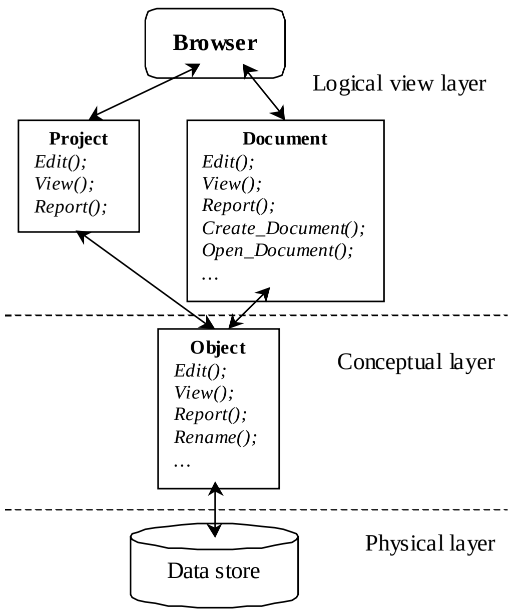


Figure 4. Object layers

Functions declared above have some implementation on the Conceptual layer. As all objects must be derived from an abstract Object defined on the Conceptual layer (see Figure 4), this set of functions is considered as default for objects defined in the meta-model and stored in the repository.

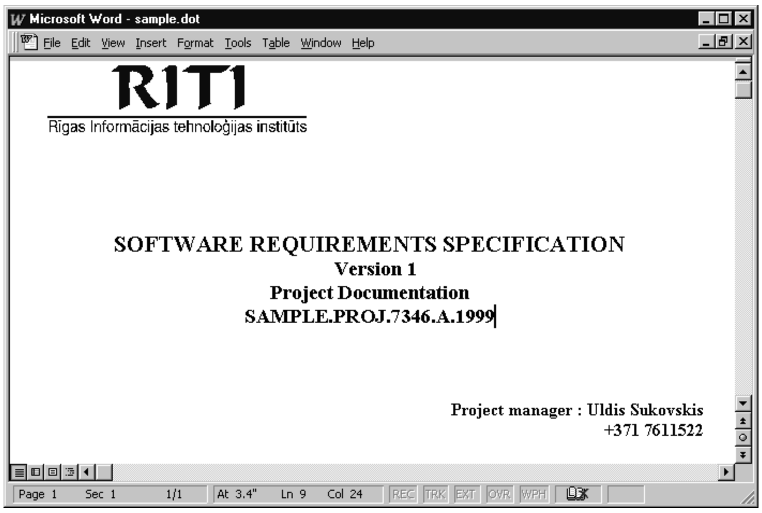
If the new object type is added to the meta-model, it by default has this set of functions with implementations on the Conceptual layer. If other implementation is necessary for some function, it is redefined on the Logical view layer.

Some functions are defined and implemented in the logical view layer. These are functions specific for document objects. For instance, function Create document for creating MS Word, MS Excel or MS Project document for the document object. Documents are generated on basis of the predefined document templates. Values from the repository are filled in the document.

3.1 Generation of Documents

Sample of implementation of the objects' functionality on the Logical view layer is generation of documents.

MS Word, MS Excel or MS Project documents could be generated using information from the repository. This is the essential function for the document object. Documents are generated on basis of predefined document templates. Function uses predefined document properties and user defined document properties. Function Generate documents fills values of the predefined and user defined properties from the information stored in the repository (see Figure 5).



Microsoft Word - sample.dot
File Edit View Insert Format Tools Table Window Help
RITI
Rigas Informācijas tehnoloģijas institūts

SOFTWARE REQUIREMENTS SPECIFICATION
Version 1
Project Documentation
SAMPLE.PROJ.7346.A.1999

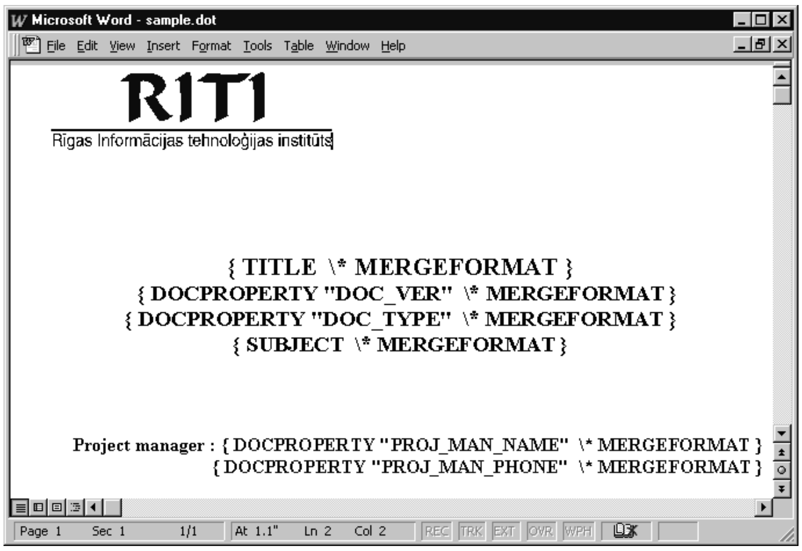
Project manager : Uldis Sukovskis
+371 7611522

Page 1 Sec 1 1/1 At 3.4" Ln 9 Col 24 REC TRK EXT OVR WPH

Figure 5. Document sample with automaticaly filled properties

There are two steps to define document template:

- define properties to the document template;
- add fields of previously defined properties in the document text in appropriate places (see Figure 6). Standard properties of the document could be also used.



Microsoft Word - sample.dot
File Edit View Insert Format Tools Table Window Help
RITI
Rigas Informācijas tehnoloģijas institūts

{ TITLE * MERGEFORMAT }
{ DOCPROPERTY "DOC_VER" * MERGEFORMAT }
{ DOCPROPERTY "DOC_TYPE" * MERGEFORMAT }
{ SUBJECT * MERGEFORMAT }

Project manager : { DOCPROPERTY "PROJ_MAN_NAME" * MERGEFORMAT }
{ DOCPROPERTY "PROJ_MAN_PHONE" * MERGEFORMAT }

Page 1 Sec 1 1/1 At 1.1" Ln 2 Col 2 REC TRK EXT OVR WPH

Figure 6. Document template