A FRAMEWORK FOR THE DEVELOPMENT AND DEPLOYMENT OF EVOLVING APPLICATIONS

The Domain Model

Georgios Voulalas and Georgios Evangelidis
Department of Applied Informatics, University of Macedonia, 156 Egnatia St., Thessaloniki, Greece

Keywords: Model-driven Development, Meta-Models, Evolving Business Applications, Application Generators, Application Deployment Platforms, Reflectional Programming.

Abstract: Software development is an R&D intensive activity, dominated by human creativity and diseconomies of scale. Model-driven architecture improves productivity, portability, interoperability, maintenance, and documentation by introducing formal models that can be understood by computers. However, the problem of evolving requirements, which is more prevalent within the context of business applications, additionally calls for efficient mechanisms that ensure consistency between models and code and enable seamless and rapid accommodation of changes, without interrupting severely the operation of the deployed application.

Having presented a framework that supports rapid development and deployment of evolving web-based applications, this paper elaborates on the Domain Model that is the cornerstone of the overall infrastructure.

1 INTRODUCTION

Software development has to deal with many important problems (Kleppe & Warmer & Bast, 2003): (a) the productivity, documentation and maintenance problem, (b) the portability problem, (c) the interoperability problem, and, (d) the evolution problem.

Model Driven Architecture (MDA) has come to cope with all these difficulties through the introduction of formal models that can be understood and processed by computers. Transformations between the models are executed by tools. However, MDA cannot ensure consistency between the produced code and the preceding models, while also fails to manage efficiently the problem of evolving requirements.

Motivated by the above-mentioned deficiencies, we introduced (Voulalas & Evangelidis, 2006) an innovative extension of MDA for the realization of a development and deployment framework that targets web-based business applications. The framework is structured on the basis of a universal database schema (meta-model). Development is supported by modelling tools that elicit functional specifications from users and transform them in formal definitions, and by data structures (part of the meta-model) that are utilized for the storage of the definitions. Deployment is supported by generic components (meta-components) that are dynamically configured at run-time according to the functional specifications provided during development, and by application-independent data structures (part of the meta-model) that hold all application-specific data. No code (SQL, Java, C++, JSP, ASP, etc.) is generated for the produced applications, and there always exists one deployed application, independently of the actual number of running applications.

The proposed framework includes three models:
- the Domain model that maps to the MDA Platform Independent Model and defines the structure of the data that the application is working on (objects, attributes, and associations), along with their behavioural aspect (methods) and business rules,
- the Application model that maps to the MDA Platform Specific Model and focuses on the targeted platform, and,
- the Operation model that maps to the MDA code layer, and consists of run-time instances of the meta-model and the meta-components.

The proposed framework copes with the weaknesses of the MDA model as follows:
(a) We can easily achieve evolution management by applying standard data versioning techniques. In case the
definition of a business object is modified this results in modifications to the underlying data instances, i.e., we can deal with changes at deployment time without recompiling and redeploying the application. Additionally, we can refer to a previous version of an application at anytime and examine old data in its real context by retrieving the corresponding data instances from the database, without the need of maintaining multiple installations.

(b) Since no code is generated and the middle model is generated automatically, all changes are realized through the Domain Model. Thus, there is consistency between the produced code and the preceding models.

This paper elaborates on the Domain Model that is the heart of the overall infrastructure. In Section 2, we present an example of a web-based business application that we will use as a case-study throughout the paper. In Section 3 we present a conceptual view of the Domain Model. In Section 4 we present a view of the Domain Model that realizes our example application. The last section concludes the paper and identifies our future steps.

2 CASE STUDY

Suppose that we have to develop a real estate portal, through which agents will be able to post property ads and potential buyers / renters to search for properties. The following statements outline the business operations of the proposed system:

- A realtor is able to post an unlimited number of property ads.
- Three types of properties ads are identified: residencies, business properties, and development land.
- A property can be available for sale or rent, or for both sale and rent.
- A potential buyer / renter that is interested in a specific property can submit a Viewing Request in order to arrange a viewing.
- A property owner can assign a property to a realtor in order to have it listed in the site, by submitting an Assignment Request.

In Figure 1, a conceptual diagram of the proposed system is illustrated.

3 THE DOMAIN MODEL

The Domain Model is structured on the basis of the O-O paradigm, augmented with the extensions introduced by the Object Constraint Language (OMG, 2003; Coronato & Cinquegrani & Giuseppe, 2002) for the description of constraints that govern the objects. It maps to the MDA Platform Independent Model, and adds functionality related to the deployment of the applications. More specifically, it models the information aspect (i.e. object attributes and associations between objects) and the behaviour aspect of the developed application (i.e. business rules and operations implemented by the objects). Finally, it stores the run-time instances of the deployed application, (i.e. the values that the object attributes take at run-time).

The main entities of the domain model are:

- **OBJECT**: Concept in the problem domain that constitutes a software entity. **OBJECTS** carry the information that is necessary for the execution of a process and implement operations that are executed at the different process steps. Its main attribute is the **Name**, i.e. a short, meaningful title.

  Examples: Property, Residency, Business Property, Transaction Type, Realtor, Viewing Request, and End User.

- **ATTRIBUTE**: Defines the static aspect (information) of an **OBJECT**. Its main attributes are:
  - **Name**: A short, meaningful title.
Type: Integer, real, string, date, or boolean.
- Initial_value: It can be an integer number, a real number, a string, a date or a boolean, depending on the type of the ATTRIBUTE. Null in case the ATTRIBUTE should not be initialized.

Examples: Brand name, phone number, and email for the Realtor OBJECT; size, location, and type for the Property OBJECT.

**ATTRIBUTE_LIST_VALUE**: A candidate value of an ATTRIBUTE. It is used for specifying a list of values that are used in insert / update operations. Its main attributes are:
- Value: It can be an integer number, a real number, a string, a date or a boolean, depending on the type of the associated ATTRIBUTE.
- Order: Specifies the order in which the values are displayed in the list for selection.

Examples: Central / Individual for the ATTRIBUTE 'heating' of the Business Property OBJECT; sea / mountain / panoramic for the ATTRIBUTE 'view' of the Residency OBJECT.

**OPERATION**: Operations define the dynamic aspect (behaviour) of an OBJECT. Its main attributes are:
- Name: A short, meaningful title.
- Return_value: It can be an integer, a real number, a string, a date or a boolean, depending on the return type. Null in case the OPERATION returns nothing.

Examples: Reject (OPERATION of an Assignment Request OBJECT), process (OPERATION of a Viewing Request OBJECT).

**ARGUMENT**: A parameter required for the execution of an operation. Its main attributes are:
- Name: A short, meaningful title.
- Type: Integer, real, string, date, or boolean.

Examples: notes (argument of the submit OPERATION), rejection reason (argument of the reject OPERATION).

**RELATIONSHIP**: Represents structural relationships between OBJECTS that exist for some duration (in contrast with transient links that, for example, exist only for the duration of an operation). Its main attributes are:
- Name: A short, meaningful title.
- Association_type: Association, aggregation or generalization.

Examples: Property – Transaction Type, Realtor – Property, End User – Assignment Request, Viewing Request – Property.

**ROLE**: Identifies a specific behaviour in a particular context at a specific time. Its main attributes are:
- Name: A short, meaningful title.
- Multiplicity: Specifies how many instances of the object may be associated with a single instance of the other object.
- Is_navigable: Indicates in what direction the role is navigating.

Examples: A Property is available for one or more Transaction Types; a Realtor manages zero to many Properties.

**OBJECT_INSTANCE**: A realization of an OBJECT. The main attribute of an OBJECT_INSTANCE is the Identifier, i.e. a string or number that uniquely identifies the OBJECT_INSTANCE.

Examples: A residency located at Athens, available for sale; the real estate agent that is responsible for the specific property; the end-user that is interested in buying the specific property; the viewing request that the end-user submitted in order to arrange a viewing of the property.

**ATTRIBUTE_VALUE**: The value of an ATTRIBUTE that a specific OBJECT_INSTANCE holds. Its main attribute is:
- Value: It can be an integer number, a real number, a string, a date or a boolean, depending on the type of the associated ATTRIBUTE.

Examples: 'Athens Properties' is the brand name of a real estate agency, 'Athens, Glyfada' is the location of the property, and '300.000' is the purchase cost of the property in euros.

**PRECONDITION**: A condition that must hold before executing an OPERATION. It typically evaluates one or more ATTRIBUTES. Its main attributes are:
- Operator: Equal, not equal, less than or equal, less than, greater than or equal, greater than.
o **Operand_value**: It can be an integer number, a real number, a string, a date or a boolean, depending on the type of the associated **ATTRIBUTE**.

o **Logical_operator**: Logical NOT, logical AND, logical OR, logical XOR. Logical operators may be coupled with parentheses for preconditions sequencing and grouping.

Examples: The ‘reject’ **OPERATION** can only be executed on Assignment Request instances that have ‘pending’ value on the ‘status’ **ATTRIBUTE**; the ‘submit’ **OPERATION** can only be invoked on Viewing Request instances that have ‘un-submitted’ value on the ‘status’ **ATTRIBUTE**.

- **INVARIANT_CONSTRAINT**: A condition that must always hold as long as the system operates. It typically constrains the value of an **ATTRIBUTE**. Its main attributes are:
  
o **Operator**: Equal, not equal, less than or equal, less than, greater than or equal, greater than.

- **Operand_value**: It can be an integer number, a real number, a string, a date or a boolean, depending on the type of the associated **ATTRIBUTE**.

Examples: The value of the ‘price’ **ATTRIBUTE** should be always greater than zero; the value of the ‘number of floors’ **ATTRIBUTE** should be always greater than zero.

- **POST-CONDITION**: Defines either the return value of an **OPERATION** or modifications on the value of component **ATTRIBUTES** that must be performed. Examples of **POST-CONDITIONS** are: The value of the ‘status’ **ATTRIBUTE** changes to rejected, once the ‘reject’ **OPERATION** is executed on an Assignment Request instance; the value of the ‘status’ **ATTRIBUTE** changes to submitted, once the ‘submit’ **OPERATION** is executed on an Viewing Request instance.

- **GUARD**: Force the execution of **OPERATIONS** anytime triggers (i.e. all **ATTRIBUTES** involved in the guard condition) get a specific state.

Example: Once the value of the ‘status’ **ATTRIBUTE** of an Assignment Request instance changes to ’processed’, the value of the ‘status’ **ATTRIBUTE** of the associated Property instance, changes to ’published’.

Having presented the main entities and their attributes, let’s now examine the way those entities are associated.

- **OBJECT – ATTRIBUTE**: An **OBJECT** includes one or more **ATTRIBUTES**, while an **ATTRIBUTE** is included in exactly one **OBJECT**. Even **ATTRIBUTES** with exactly the same characteristics (e.g. notes) belonging to different **OBJECTS** (Viewing Request and Assignment Request), are distinct realizations of the **ATTRIBUTE** entity.

- **OBJECT – OPERATION**: An **OBJECT** implements optionally one or more **OPERATIONS**, while an **OPERATION** is implemented by exactly one **OBJECT**.

- **OBJECT – RELATIONSHIP**: An **OBJECT** participates optionally in one or more **RELATIONSHIPS**, and a **RELATIONSHIP** links exactly two **OBJECTS**.

- **OPERATION – ARGUMENT**: An **OPERATION** optionally takes as input one or more **ARGUMENTS**, while each **ARGUMENT** is used by exactly one **OPERATION**. Even **ARGUMENTS** with exactly the same characteristics (e.g. notes) used by different **OPERATIONS** (submit **OPERATION** of the Assignment Request **OBJECT**, submit **OPERATION** of the Viewing Request **OBJECT**), are distinct realizations of the **ARGUMENT** entity.

- **OBJECT – OBJECT_INSTANCE**: An **OBJECT** has optionally one or more **INSTANCES**, while an **OBJECT_INSTANCE** belongs to exactly one **OBJECT**. An **INSTANCE** is a run-time realization of an **OBJECT**.

- **ATTRIBUTE – ATTRIBUTE_VALUE**: An **ATTRIBUTE** takes optionally one or more **VALUES**, one for each **OBJECT_INSTANCE**. A **VALUE** is associated with exactly one **ATTRIBUTE**. An **ATTRIBUTE_VALUE** is the value that an **ATTRIBUTE** takes at runtime.

- **OBJECT_INSTANCE – ATTRIBUTE_VALUE**: An **OBJECT_INSTANCE** has one or more **ATTRIBUTE_VALUES**, one for each **ATTRIBUTE**. An **ATTRIBUTE_VALUE** is associated with exactly one **OBJECT_INSTANCE**. The static aspect of an **OBJECT_INSTANCE** is at any time defined by the values of its **ATTRIBUTES**.
OPERATION – ATTRIBUTE – ARGUMENT: An OPERATION manages optionally one or more ATTRIBUTES using as input its ARGUMENTS. An ATTRIBUTE can be optionally managed by one or more OPERATIONS.

PRECONDITION – ATTRIBUTE: A PRECONDITION evaluates exactly one ATTRIBUTE. An ATTRIBUTE can be optionally evaluated by one or more PRECONDITIONS.

PRECONDITION – PRECONDITION: Two or more PRECONDITIONS can be optionally associated in order to form complex PRECONDITIONS, i.e. sequence of PRECONDITIONS each one evaluating different ATTRIBUTES or the same ATTRIBUTE in a different way.

INVARIANT_CONSTRAINT – ATTRIBUTE: An INVARIANT_CONSTRAINT restricts exactly one ATTRIBUTE. An ATTRIBUTE can be optionally constrained by one or more INVARIANTS_CONSTRAINTS.

In Figure 2, a conceptual diagram of the Domain Model is illustrated.

4 APPLYING THE DOMAIN MODEL

The following tables present an instance of the Domain Model that covers part of the functional specifications of the example application as prescribed in Chapter 2.

<table>
<thead>
<tr>
<th>OBJECTS</th>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Property</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Residency</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Realtor</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Transaction Type</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Assignment Request</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Viewing Request</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ATTRIBUTES</th>
<th>id</th>
<th>name</th>
<th>type</th>
<th>initial_value</th>
<th>object_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>property_id</td>
<td>Integer</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>code</td>
<td>String</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>size</td>
<td>Real</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>location</td>
<td>String</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>number_of_floors</td>
<td>Integer</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>number_of_bedrooms</td>
<td>Integer</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>heating</td>
<td>String</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>view</td>
<td>String</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>brand_name</td>
<td>String</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>status</td>
<td>String</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ATTRIBUTE LIST VALUES</th>
<th>id</th>
<th>value</th>
<th>order</th>
<th>attribute_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Central</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Individual</td>
<td>2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Sea</td>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Mountain</td>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Panoramic</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OPERATIONS</th>
<th>id</th>
<th>Name</th>
<th>return_value</th>
<th>object_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reject</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>process</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>submit</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>submit</td>
<td></td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ARGUMENTS</th>
<th>id</th>
<th>name</th>
<th>type</th>
<th>operation_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>notes</td>
<td>String</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>agreed_viewing_date</td>
<td>Date</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>agreed_contact_date</td>
<td>Date</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RELATIONSHIPS</th>
<th>id</th>
<th>name</th>
<th>association_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Property-Residency</td>
<td>Generalization</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Property-Realtor</td>
<td>Association</td>
<td></td>
</tr>
</tbody>
</table>
Modifications in the application data (e.g., insertion of a new property) result in modifications of the instances of the OBJECT_INSTANCES and ATTRIBUTE_VALUES entities, while modifications in the application logic require modifications of the instances of the other entities. Modifications of the structure of any entity are not required.

5 CONCLUSIONS

In this paper we elaborate on the Domain Model that is the cornerstone of our framework. The framework will facilitate the development and deployment of web-based business applications, while on parallel limit the side-effects that are induced by the continuously changing requirements. The framework conforms to the principles of MDA, however it is based on a different principle: developed applications will consist of run-time instances of generic components, and not of code packages.

Having identified the core elements of the Domain Model our next research steps will focus on:

- **Elaborating on the elements that specify the dynamic aspect of the modelled applications.**
  Specifically, the Post-condition and Guard entities should be further analyzed, while new entities that will model the body of operations should be specified.


- **Isolate all entities and mechanisms related to enterprise modelling**, business relationships establishment, role assignment, and personnel administration, and handle them through a separate model, called Enterprise Model.

REFERENCES


