A FRAMEWORK FOR THE DEVELOPMENT AND DEPLOYMENT OF EVOLVING APPLICATIONS
The Domain Model

Georgios Voulalas, Georgios Evangelidis
Department of Applied Informatics, University of Macedonia, 156 Egnatia St., Thessaloniki, Greece
voulalas@uom.gr, gevan@uom.gr

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.

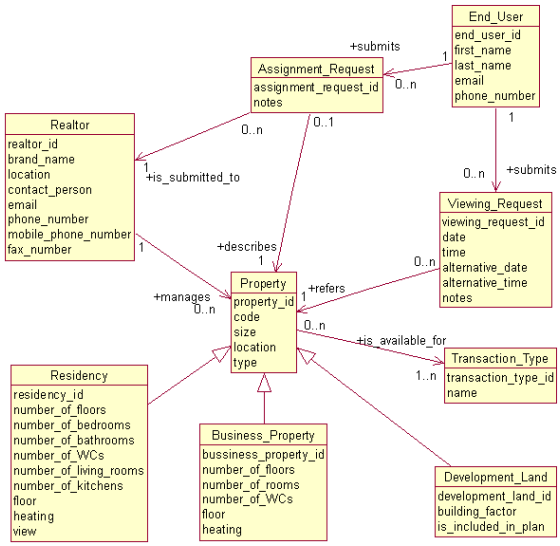

Figure 1: Conceptual Diagram of the Real Estate Portal

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 relationship 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 Viewing request refers to a specific Property; a Realtor manages zero to many Propertiesr.
- **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.
    - *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**.
    - *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 constraints the value of an **ATTRIBUTE**. Its main attributes are:
    - *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 run-time.
- **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 **PRE-CONDITIONS** 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**.

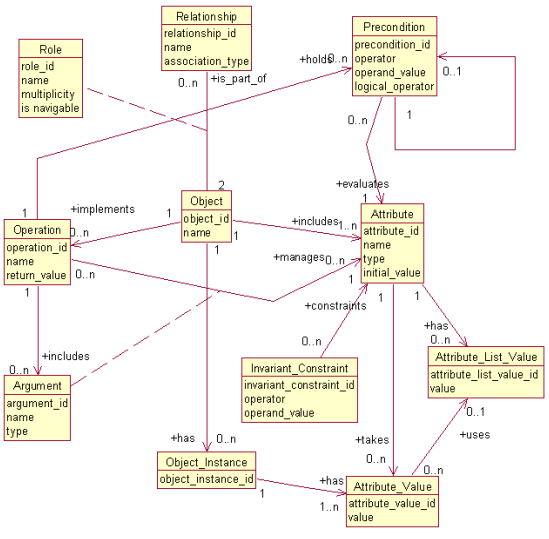

Figure 2: Conceptual Diagram of the Domain Model

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.

OBJECTS	
id	*name*
1	Property
2	Residency
3	Realtor
4	Transaction Type
5	Assignment Request
6	Viewing Request

ATTRIBUTES				
id	*name*	*type*	*initial_ value*	*object_id*
1	property_id	Integer		1
2	code	String		1
3	size	Real		1
4	location	String		1
5	number_of_floors	Integer	1	2
6	number_of_bedrooms	Integer		2
7	heating	String		2
8	view	String		2
9	brand_name	String		3
10	status	String		5

ATTRIBUTE LIST VALUES			
id	*value*	*order*	*attribute_id*
1	Central	1	7
2	Individual	2	7
3	Sea	1	8
4	Mountain	2	8
5	Panoramic	3	8

OPERATIONS			
id	*Name*	*return_value*	*object_id*
1	reject		5
2	process		6
3	submit		5
4	submit		6

ARGUMENTS			
id	*name*	*type*	*operation_id*
1	notes	String	2
2	agreed_viewing_date	Date	2

3	agreed_contact_date	Date	2

RELATIONSHIPS		
id	*name*	*association_type*
1	Property-Residency	Generalization
2	Property-Realtor	Association

ROLES					
id	*name*	*multipl*	*is_navigable*	*rel_id*	*obj_id*
1				1	1
2				1	2
3	manages	0..n	TRUE	2	1
4	is_managed_by	1	FALSE	2	3

OBJECT INSTANCES	
id	*object_id*
1	2
2	3
3	4
4	6

ATTRIBUTE VALUES				
id	*value*	*attr_id*	*object_instance_id*	*list_value_id*
1	Athens, Glyfada	4	1	
2	Athens Properties	9	2	
3		7	1	1
4		8	1	3

PRECONDITIONS					
id	*operator*	*op_value*	*log_operator*	*attr_id*	*op_id*
1	=	*pending*		10	1

INVARIANT CONSTRAINT			
id	*operator*	*operand_value*	*attribute_id*
1	>=	1	5
2	>=	0	6

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.
- **Introduce elements from an acceptable business rules classification scheme** (Business Rules Forum 2004 Practitioners' Panel, 2005; Butleris & Kapocius, 2002; Herbst, 2002), with the Ross method (Business Rules Forum 2004 Practitioners' Panel, 2005) being the prevalent.
- **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

Business Rules Forum 2004 Practitioners' Panel, 2005. The DOs and DON'Ts of Business Rules. Business Rules Journal, Vol. 6, No. 4, http://www.BRCommunity.com/a2005/b230.html

Butleris, R., Kapocius, K., 2002. The Business Rules Repository for Information Systems Design. ADBIS Research Communications: 64-77

Coronato, A., Cinquegrani, M., Giuseppe, D.P., 2002. Adding Business Rules and Constraints in Component Based Applications. CoopIS/DOA/ODBASE: 948-964

Herbst, H., 1996. Business Rules in Systems Analysis: a Meta-Model and Repository System. Inf. Syst. 21(2) 147-166

Kleppe, A., Warmer, S., Bast, W., 1996. MDA Explained. The Model Driven Architecture: Practice and Promise (Chapter One). Addison-Wesley.

OMG, 2003. Object Constraint Language Specification. http://www.omg.org/cgi-bin/doc?ptc/2003-10-14

Voulalas, G., Evangelidis, G., 2006. A framewor k for the development and deployment of evolving applications: Elaborating on the Model Driven Architecture towards a change-resistant development framework, ICSOFT 2006, 22-29