

AN APPROACH TO BEHAVIORAL SOFTWARE MODELS ANALYTICAL REPRESENTATION

Elena Chebanyuk

Abstract: According to standard UML 2.5 Behavioral Software Models (BSMs) are UML diagrams that represent software behavior [UML 2.5, 2012]. An approach of BSM analytical representation is proposed in this paper. Such BSMs as Use Case Diagrams (UCDs), Collaboration Diagrams (CDs) and Sequence Diagrams (SDs) are considered at this paper. In order to design this approach objects and operations that are both common and specific for every type of considering UML diagrams are defined. In order to prepare general analytical representation of BSM it is proposed to form two tuples containing objects and operations. Thus, BSM is represented as a Cartesian product of these two tuples. Then analytical representation of conditional operation is proposed.

The peculiarity of the represented approach is that unified analytical representation of some operation for all types of considering BSMs is used. Such representation facilitates performing many operations in Model-Driven Architecture (MDA) area.

Literature review shows that researches of BSMs mostly focused on preparation of an analytical representation for parallel processes [Hu, 2014], [Dove, 2014] or designing transformation technics [Rhazali, 2014], [Romera, 2014], [FUML, 2013], [Wang, 2014], [Bonaris, 2014] and ect.. But conditional operations are important constituents of any complex algorithms of processes. Existing technics of conditional operations description, for example Object Constraint Language (OCL) [OCL 2.4, 2014], do not contain detailed notation for representation of any condition with given precision level. The notation for formal analytical representation of conditional operation is represented in this paper. It is shown that any complex conditions can be described in terms of the proposed notation. In order to prove this thesis different conditional blocks are designed from Collaboration Diagrams. Also the analytical representation of conditional block is obtained from other types of considering BSM.

Also in the point further research the general idea of framework for the analytical description of BSM is proposed.

In conclusion advantages of implementing of the proposed approach to perform different activities in software development lifecycle are formulated.

Keywords: Analytical representation, UML diagrams, Set-theory, Object Constraint Languages (OCL), Model-Driven Architecture (MDA).

ACM Classification Keywords: D.2.2 - Design Tools and Techniques, D.2.11 Software Architecture

Introduction

Today software development lifecycles that correspond to Agile methodology can better meet customer requirements than Rational Unified Approach (RUP). Involving Model-Driven Development (MDD) approach into software development lifecycles helps to raise an effectiveness of different processes [Swithinbank, 2005]. Software models, which are represented as UML diagrams, are central artifacts of MDD approach [UML 2.5, 2012]. The classification of UML models is represented in the paper [Chebanyuk, 2014] In MDD and MDA approaches BSM are Computation Independent Models (CIMs). Computation Independent Models are metamodels which contain basic information about software requirements and functionality [Chebanyuk, 2014]. Effective processing of CIM models allows obtaining qualitative initial information to design the rest of software artifacts. Preparing an analytical representation of BSM allows implementing a background for many MDA operation. These operations are model comparing, refactoring, reusing, merging and so on. Such operations are performed in different software development lifecycle activities, for example requirement verification and validation, algorithms analyzing and refinement, interface designing, testing, creating and modifying documentation and other.

Related papers

Analytical representation of BSM is an initial step for different model processing operations. These operations can be divided on two categories, namely software model transformations and processing.

Consider papers that are devoted to BSMs transformation in MDA sphere

Paper [Rhazali, 2014] proposes a method for transformation CIMs to Platform Independent Models (PIMs). The first step of this method is refining BSM, namely UC or CD, according to specific rules. Every requirement is associated with some business process. Then some use cases are mapped into classes. However this method doesn't contain defined rules for matching description of requirements and model elements to some business processes. This fact makes difficult usage both mathematical apparatus and formal languages for analyzing requirement specification. Also formal modeling requires a formal language. A formal language, unlike a natural language, is one that operates with explicitly defined rules.

In the paper [Wang, 2014] approaches that are used for software models transformation are systematized. The next approaches are considered: Template-Based Approaches, Target-Structure-Driven Approaches, Graph-Transformation-Based Approach, Relational Approaches and Approach with models serializing. The short description of these approaches is represented below.

Template-Based Approaches In this approaches, templates consisting of text in the target language include meta-code tags to access information from the source model. In the transformation process,

these tags will be interpreted and eventually replaced by code representing the corresponding parts of the source. These approaches usually cover model to code generation operations.

Target-Structure-Driven Approaches The basic metaphor is the idea of copying model elements from the source to the target. This kind of approaches was developed in the context of certain kinds of applications such as generating Enterprise Java Beans (EJB) implementations and database schemas from UML models.

Graph-Transformation-Based Approaches This category of model transformation approaches draws on the theoretical work on graph transformation. In particular, these approaches operate on typed, attributed, labeled graphs, which is a kind of graphs specifically designed to represent UML-like models. This kind of approaches is inspired by heavily theoretical work in graph transformations, and it is powerful and declarative, but also the most complex ones.

Relational Approaches This kind of approaches uses the mathematical concept of relations to specify how source and target models are linked. Relations are declarative but may be given execution semantics. It seems to strike a well balance between flexibility and declarative expression. Transformation Implemented using Extensible Stylesheet Language Transformations (XSLT). Models can be serialized as Extensible Markup Language (XML) using the XML Metadata Interchange (XMI), and implementing model transformations using XSLT. Most of the approaches given above focus on providing a concrete solution for the transformation from PIMs to Platform Specific Models (PSMs), and there is little research on the definition principles for mapping rules as well as a basic theory to validate the mapping rules between such models. The research about natural language translation by machine shows that the prerequisite of correct transformation between different languages is the same or similar characteristics of semantics expression within the source and the target.

Semantic-element-based Defining Approach for Model A model mapping approach based on semantic consistency was proposed by abstractly analyzing the characteristic of syntax and semantics of modeling languages. Abstract target semantic model must be firstly constructed through an in-depth analysis of target platform. Then, based on the idea of elements in source semantic domain being reconstructed in the target semantic domain, mapping relations from source model to target model are created via abstract target semantic model. The formalization of model transformation approach which uses patterns is proposed and is represented by the following: A pattern can be defined as a 3-tuple:

$$P = \langle C, A, SR \rangle \quad (1)$$

where $C = \{c \mid c \text{ is a conceptual element in PIM or PSM}\}$, and $A = \{a \mid a \text{ is a relevant attributes of the conation operations are performed}\}$.

The approaches, proposed in the paper [Wang, 2014], allow estimating the effectiveness of transformation operations from the different points of view. They can provide a general framework for

modifying and designing new model mapping technics when methods of software models transformation are designed.

Solving the task of formal processes description helps to prepare initial information for different transformation tasks. Examples of these task are: transformation business rules to class diagram [Bolaris, 2014], code generation from PSM models [FUML, 2013] and evaluation of system design [Romero, 2014].

Complex description of transformation approach from business rules to class diagrams is represented in the paper [Bonais, 2014]. Authors proposed description of all steps to perform this task. Formal models for entire and resulting information are also represented. Also constrains for initial and resulting models are proposed. Further rules for mapping process to class diagram constituents are formulated. This approach can be used for software requirement elicitation. One can use business rules of problem domain to define whether requirement specification corresponds to business needs. Also this approach can be used both for UML profile elicitation and class diagram refinement. Formal model for business rules representation contains vocabulary. Vocabulary matches facts to some objects or entities of problem domain. Second component of the proposed model is a set of rules. Every business rule is represented as the tuple

$$br = (statement, factType, modal, quatifier1, quatifier2) \quad (2)$$

where: *statement* - is the statement of the business rule. It is built by combining fact types and keywords;

factType - is the fact type that is used to build the business rule statement;

modal - is the modal keyword that is included in the business rule statement;

quatifier1 - is the first quantification keyword that is included in the business rule statement. It always

belongs to the first noun concept of the fact type;

quatifier2 - is the second quantification keyword that is employed in the business rule statement. It

always belongs to the second noun concept of the fact type.

Representation of business rules (2) can be used for formal analysis of functional requirements, algorithms, processes, comparing and refinement requirements specifications. Using this notation one can perform different operations of analyzing text information by means of formal models processing.

Standard FUML describes an intermediated framework between PSM model and code [FUML, 2013]. Target platform language is Java. Elements of a class are described as a tuple of properties, relationships and operations. Other tuples are association classes, interfaces and dependencies. Then

notation for description of class behavior is proposed. Also templates of code for every variant of classes are represented.

Also the collection of code templates is prepared. Classes are connected as constituents of design patterns. Thus, FUML activities are the next:

- Define behavior features of component.
- Represent the formal description of class.
- Compare this description with analytical representation of classes.
- Define the closest analytical representation of class and represent the corresponding template of code.

Authors [Romero, 2014] proposed a FUML interpreter. This interpreter verifies BSM. In order to perform this task authors presented the relationships between four components, namely abstract syntax, execution model, semantic domain, and base semantics.

The execution model is an interpreter written in FUML objects and object nodes, which are interconnected by unary and binary relations. However, the base semantics for designing an executable code is represented. However number of building blocks that are designed for generation of code fragments is limited. And even small differences in processes can be causes of big differences in codes. Thus, in order to obtain executable code, reflecting the peculiarities of processes it is necessary to do the following:

- propose more precise notation of the abstract syntax;
- increase number of basic semantic elements that reflect differences in code;
- add to the base of code templates by new fragments of code;
- propose a technique of design patterns reusing when their number will be increased;
- include specific attributes to the description of processes behavioral features. The aim of these attributes is to provide more precise mapping of model processes to code templates.

Authors [Hu, 2014] says that many efforts were made to make UML executable by transforming single diagram to executable model such as Colored Petri Nets (CPN), however, approach like this could not provide more systematic and intuitive simulation of the entire system. Therefore it was proposed to use the Discrete Event System Specification (DEVS) as the target formalism and transform UML Diagrams to executable models.

Firstly two types of models for considering parallel execution of processes is proposed. The atomic model is the irreducible model definition that specifies the behavior for any modeled entity. The coupled model represents the composition of several atomic and coupled models connected by explicit couplings.

An atomic model M and a coupled model N are defined by the following equations:

$$M = \langle IP, OP, X, S, Y, \vartheta_{int}, \vartheta_{ext}, \vartheta_{con}, \lambda, ta \rangle \quad (3)$$

$$N = \langle IP, OP, X, Y, D, EIC, EOC, IC \rangle \quad (4)$$

where:

- S- is the state space;
- IP, OP - are the set of input and output ports;
- X, Y - are the set of Inputs/Outputs, which are basically lists of port-value pairs;
- $X = \{(p,v) \mid p \in IP, v \in X_p\}$, $Y = \{(p,v) \mid p \in OP, v \in Y_p\}$, where X_p and Y_p are input/output values on port p ;
- $\vartheta_{int} : S \rightarrow S$ is the internal transition function;
- $\vartheta_{ext} : Q \times X_b \rightarrow S$ is the external transition function, where $Q = \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\}$ is the total state set,
- $\vartheta_{con} : S \times X_b \rightarrow S$ is the confluent transition function, which decides the order between ϑ_{int} and ϑ_{ext}
- in cases of collision between simultaneous external and internal events;
- $\lambda : S \rightarrow Y_b$ is the output function, where Y_b is a set of bags composed of elements in Y ;
- $ta(s) : S \rightarrow R_0^+ \cup \infty$ is the time advance function which decide how much time the system stays at the current state in the absence of external events.

Extended DEVS modeling language has the following components: Models for Separation of Message Processing From State Transition and the abstract syntax. In terms of abstract syntax software models are divided on three elements, namely the Entity, the Atomic and the Coupled. Also functions for more precise description of atomic models are presented. For code generation operations special language Xpand is used. Then executable codes are obtained through Xtext framework because Xtext is seamlessly integrated with the Eclipse Java framework by code generator with Xtend. Also the proposed approach can be used to refine class and component diagrams. Entity "component" of model can be mapped to class in atomic model representation. And Component Diagram provides structural information about a coupled model including components in coupled model and couplings between them.

However extended DEVS language covers just a sphere of parallel execution of some processes. Such operations as conditional statements, cycle constructions, collection processing and other are not described by this language.

Also set-theory serves for representation of different constrains of software models. For example using OCL one can define a set of model constrains that reflects specific needs of a problem domain.

A UML diagram, such as a class diagram, is typically not refined enough to provide all the relevant aspects of a specification. There is, among other things, a need to describe additional constraints about

the objects in the model. Such constraints are often described in natural language. Practice has shown that this will always result in ambiguities. In order to write unambiguous constraints, so-called formal languages have been developed. The disadvantage of traditional formal languages is that they are usable to persons with a strong mathematical background, but difficult for the average business or system modeler to use OCL [OCL 2.4, 2014]. .

OCL has been developed to fill this gap. It is a formal language that remains easy to read and write. It has been developed as a business modeling language within the IBM Insurance division, and has its roots in the Syntropy method [OCL 2.4, 2014].

OCL has very flexible semantic. Using text notation one can represent some meaning of variables and set of the restrictions. These restrictions are invariants, pre and post conditions, initial and derived values and others. Flexibility of OCL notation allows including new elements to meet specific requirements of representation of peculiarities of business processes.

One of direction of these researches to precise the types of object interaction by means of an association link between object. Authors [Dove, 2014] proposed to extend the OCL notation to better express UML qualified association. A qualified association is an association that allows one to restrict the objects relations using a key

called a qualifier. Additional constrains are added. They are formulated by means of set theory and predicate logic and supplements UML diagrams by providing expressions.

The key idea of the proposed approach is to make a qualified association a first class entity in OCL by exposing it as a map and allowing one to query and manipulate the map directly. For this a small extension to the OCL notation was proposed to denote the qualified association map and introduce a new collection library class to model a map. The collection of map specific operations was proposed. Pre and post conditions, rules for defining collection processing, equality and defining operation are defined. Listed operations can help to define relations between objects more precise.

Also researches are performed for solving the task of conceptualization the different facts of component model in Component Based Software Engineering (CBSE). This approach facilitates modeling of components using different schemas in software system. One of the proposed solutions is to use Z-semantic based conceptual model of component, called Z-Formal Specification of Component Model (ZFSCM). Is describes formal description of components and interconnections between them [Banerjee, 2014].

Z uses not only mathematical notation to describe properties of information system without constraining the way in which these properties are achieved, it also uses the schema to define these properties. They describe what the system must do instead of saying how it is to be done. This abstraction makes Z useful in the process of developing a system. Z-specification can serve as a single, reliable reference

point for those who investigate the customer's needs, test results and write instruction for the system. It makes easier to write mathematical description of complex dynamic software systems. The descriptions are usually smaller and simpler than any programming language can provide. They should contain a mixture of formal and informal parts. Z is based on the standard mathematical notation using mathematical data types to model the data in a system. It also used in axiomatic set theory, first-order predictive logic, lambda calculus. Another thing is that it has a way of decomposing a specification into small piece called schemas. After splitting the specification into schemas, it can present it piece by piece. Every piece is connected with formal mathematics. All expressions in Z notation are typed, thereby avoiding few of the paradoxes of set theory. Z contains a standard mathematical toolkit of commonly used mathematical functions and predicates, called catalog. Bur the task of designing flexible approach to describe BSM processes is remains actual.

Authors [Banerjee, 2014] proposed graphical notation for description of main elements os component diagram, namely service, interface, class and components, and relationships between them. Also they represented text-based description for type-checking. In order to design a component model it is proposed every requirement in specification match with service, then service join to components, after that, the model of components is designed. During design process two types of interfaces are defined, namely required and used. This model can be used for such model processing operations as components reuse and making changes to the components. This model can be complicated by other elements to describe different features of components and service that are represented in OMG standards for description of Service oriented modeling languages [SOAML, 2010] and UML components diagram [UML 2.5, 2012].

Task and challenges

The task is to design the approach for an analytical representation of BSM. In order to perform this task it is necessary to do the following:

- consider a general form of the analytical representation for three kinds of BSMs that are commonly used in software development activities, namely UCDs, CDs and SDs;
- define both common and specific objects and operations that are used in UML notation for every considering type of BSM;
- discover key features of operations for representing an analytical description of BSM;
- introduce the denotation for the analytical representation of conditional operation;
- prove that the representation of conditional operations is unified for different types of BSM.

Challenges are: the analytical representation of BSM should satisfy to the following conditions:

- contain such a notation that allows designing format of file for saving information about BSM. This notation should allow modifying a structure of file easily when BSMs are changed;

- be easy combined with technics [Hu, 2014], [Chebanyuk, 2014] and standards [FUML, 2013] that allow to store and visualize information about software model, including of sequence of its processes and sub process;
- consider all entire information for performing of operations processing models such as comparing, refactoring, reusing and others;
- allow to formalize representation of BSM for solving the MDA task of transformation between BSM of different types.

The proposed approach

The task of every BSM is to represent processes. Every BSM has its own notation. This notation allows representing processes with given degree of details.

For the analytical of BSM the Set-Theory tool is chosen. It is proposed to represent interconnection between constituents of BSM by means of multiplying two tuples of Cartesian product. One factor of the Cartesian product represents a BSM constituent and another one represents an operation defined in the notation of specific BSM.

Consider such BSMs as UCDs, CDs and SDs. According to UML standard these diagrams are designed using both common and specific elements. For example, the SD notation has more elements and operations to provide detailed representation of processes in comparison with AD or CD.

Forming tuples, containing constituents of different behavioral software models

Table 1 introduces the denotations of constituents in considering BSM according to UML standard.

Table 1. Denotations of BSM constituents

| Denotation from the UML standard | Denotation of the proposed approach | Use case | Collaboration | Sequence |
|----------------------------------|-------------------------------------|----------|---------------|----------|
| 1 | 2 | 3 | 4 | 5 |
| Object | O | - | + | + |
| 1 | 2 | 3 | 4 | 5 |

| Denotation from the UML standard | Denotation of the proposed approach | Use case | Collaboration | Sequence |
|--|-------------------------------------|---|---------------|----------|
| Complex object (object with properties defined) | O_c | - | + | - |
| Actor | Θ | + | + | + |
| Message | M | + (precedent is an equivalent of message) | + | + |
| Collection | Y | - | + | + |
| Subsystem | S | + | + | + |
| Multiplicity | N | + | + | - |
| Waiting for response | I | - | - | + |

Using Table 1 tuples of sets from constituents of different BSMs are formed.

Tuple for description of Use Case Diagram constituents is denoted as follows:

$$K = \langle \Theta, M, S, N \rangle \quad (5)$$

Tuple for description of Collaboration Diagram constituents is denoted as follows:

$$X = \langle O, O_c, \Theta, M, Y, S, N \rangle \quad (6)$$

Tuple for description of Sequence Diagram constituents is denoted as follows:

$$H = \langle O, \Theta, M, Y, S, I \rangle \quad (7)$$

Forming tuples, containing operations of different behavioral software models

Table 2 introduces the denotations of operations in considering BSM according to UML standard.

Table 2. Denotations of BSM operations

| Operation | Denotation of the proposed approach | Use case | Collaboration | Sequence |
|--|-------------------------------------|---|---------------|--|
| 1 | 2 | 3 | 4 | 5 |
| Object Creation | Δ | - | + | + |
| Element of collection processing | E | - | + | + |
| Waiting for response | Ψ | - | - | + |
| Representing different types of messages | Π | + (<code><<include>></code> <code><<extend>></code> <code><<generalize>></code>) | - | + (Asynchronous call Synchronous call Call of object to itself) |
| Conditional statement | Ω | + | + | + |
| 1 | 2 | 3 | 4 | 5 |
| Cycle | Σ | - | + | + |
| Parallel execution | Ξ | - | + | + |

Using Table 2 tuples of sets from constituents of different BSM notations are formed.

Tuple for description of Use Case Diagram operations is denoted as follows:

$$\Phi = \langle \Pi, \Omega \rangle \quad (8)$$

Tuple for description of Collaboration Diagram operations is denoted as follows:

$$\Gamma = \langle \Delta, E, \Omega, \Sigma, \Xi \rangle \quad (9)$$

Tuple for description of Sequence Diagram operations is denoted as follows:

$$\Lambda = \langle \Delta, E, \Psi, \Pi, \Omega, \Sigma, \Xi \rangle \quad (10)$$

Preparing an analytical representation of behavioral software models

General form of BSM representation by means of the Cartesian product is denoted as follows:

$$\left\{ \begin{array}{l} Z^* \subseteq Con \times Oper \\ Con = \begin{cases} K \\ X \\ H \end{cases} \\ Oper = \begin{cases} \Phi \text{ then } Con = K \text{ and } * = UC \\ \Gamma \text{ then } Con = X \text{ and } * = C \\ \Lambda \text{ then } Con = H \text{ and } * = S \end{cases} \end{array} \right. \quad (11)$$

where Z – is the representation of whole BSM. Its upper index $*$ shows the type of BSM, namely

- $*$ – UC – Use Case Diagram;
- $*$ – C – Collaboration Diagram;
- $*$ – S – Sequence Diagram.
- Oper – the set of BSM operations
- Con - the set of BSM constituents.

Consider an example of Use case Diagram analytical representation according to the proposed approach

$$Z^{UC} \subseteq K \times \Phi \quad (12)$$

In order to represent an analytical description of operations specific “building blocks” can be designed. These “building blocks” should contain key features of operation. The examples of analytical representation of UCD operations is introduced in the paper [Chebanyuk, 2014].

For example, consider an analytical representation of conditional statements for every type of BSM. Analyzing Table 2 one can see that conditional operations are met in three types of BSMs. From the other hand literature review shows that there is no technics for the analytical representation of complex conditional operation with given level of precision.

Consider a procedure of preparing the analytical representation of conditional operation. This representation should satisfy to the main feature of the proposed approach, namely, to be unified for all types of BSMs.

Designing the unified analytical representation of conditional statement for the all types of considering behavioral software models

The first step of this procedure is to define key features of this operation. Table 3 represents key features for the analytical description of conditional operations, parallel statements and also cycle operations.

Table 3. Denotations of some behavioral software models operations

| Conditional statement | Parallel execution of operation | Cycle |
|--|--|--|
| Object from which brunching is started (root object) Condition (can be both simple or complex) Action or actions that are executed when condition is true (Optional) Action or actions that are executed when condition is false (Optional) Action or actions that are executed when all conditions are false (Optional) object, which meets all conditional brunches | Condition when parallel processes are started. See description of conditional statements. Action or actions that are executed when every thread of the parallel process is executed A marker (may be process or an object) from which parallel execution is started A marker (may be process or an object) showing that parallel execution is ended | Cycle conditions Counter values (optional) Cycle should contain as many analytical blocks as many conditions are in the cycle. See description of conditional statement |

Table 3 shows that more complex operations contain conditions as a part of them.

Consider the process of designing building block for conditional statements representation using key features, of its operation that are represented in the Table 3.

Let us assume the following denotations:

c – conditional operation.

$root$ - object from which brunching is started (root object for the Table 3).

ω_i - is a condition with number i where $i=1, \dots, n$ and n is a number of conditions that are started from the root object.

$brunch$ – object that meets all condition brunches.

An analytical representation of actions which are related to condition ω_i is denoted as follows:

$$\langle \omega_i \rangle \left[\begin{array}{l} \omega_i = true(\text{actions that are preformed when } \omega_i = true) \\ \omega_i = false(\text{actions that are preformed when } \omega_i = false) \end{array} \right] \quad (13)$$

Using the represented denotations introduce a general form of condition operation analytical representation.

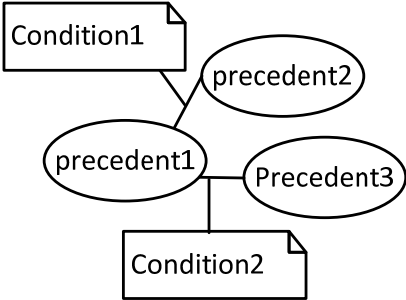
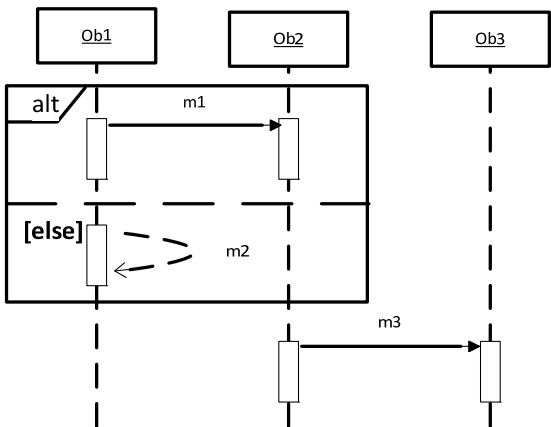
$$c = \left\{ \begin{array}{l} root \\ \langle \omega_1 \rangle \left[\begin{array}{l} \omega_1 = true(\text{actions that are preformed when } \omega_1 = true) \\ \omega_1 = false(\text{actions that are preformed when } \omega_1 = false) \end{array} \right] \\ \dots \\ \dots \\ \langle \omega_n \rangle \left[\begin{array}{l} \omega_n = true(\text{actions that are preformed when } \omega_n = true) \\ \omega_n = false(\text{actions that are preformed when } \omega_n = false) \end{array} \right] \\ else \left[\text{actions that are performed when all conditions are false} \right] \\ brunch \end{array} \right. \quad (14)$$

Analytical representation of conditional operations for different types of behavioral software models

Prove that statement (14) can be applied to three types of BSMs. For this consider different types of processes representation analyzing peculiarities of considered BSMs.

Table 4. Examples of BSMs notation application

| BSM fragment including conditional operations | Analytical representation of BSM fragment |
|---|---|
| Collaboration diagram | |
| | $Z^c \subseteq \Gamma \times X$ $\Gamma = \langle \Omega \rangle \text{ where } \Omega = \{\omega_1, \omega_2 \mid \omega_1 = \text{cond}_1, \omega_2 = \text{cond}_2\}$ $X = \langle O, M \rangle \text{ where } O = \{ob_1, ob_2, ob_3, ob_4, ob_5\}$ $M = \{m_1, m_2, m_3, m_4, m_5\}$ $c = \begin{cases} ob_1 \\ \langle \text{cond}_1 \rangle \\ \langle \text{cond}_2 \rangle \\ \text{else} \end{cases} \begin{cases} [\text{cond}_1 = \text{true}(\langle m_1, ob_4 \rangle, \langle m_2, ob_5 \rangle)] \\ [\text{cond}_2 = \text{true}(\langle m_3, ob_2 \rangle, \langle m_4, ob_3 \rangle)] \\ [\langle m_5, ob_3 \rangle] \end{cases}$ $Z^c \subseteq \langle c \rangle$ |
| | $Z^c \subseteq \Gamma \times X$ $\Gamma = \langle \Omega \rangle \text{ where } \Omega = \{\omega_1 = \text{cond}_1, \omega_2 = \text{cond}_2\}$ $X = \langle O, M \rangle \text{ where } O = \{ob_1, ob_2, ob_3\}$ $M = \{m_1, m_2, m_3\}$ $c = \begin{cases} ob_1 \\ \langle \text{cond}_1 \rangle \\ \langle \text{cond}_2 \rangle \\ \text{else} \end{cases} \begin{cases} [\text{cond}_1 = \text{true}(\langle m_1, ob_1 \rangle)] \\ [\text{cond}_2 = \text{true}(\langle m_2, ob_2 \rangle)] \\ \langle m_3, ob_3 \rangle \end{cases}$ $Z^c \subseteq \langle c \rangle$ |
| | $Z^c \subseteq \Gamma \times X$ $\Gamma = \langle \Omega \rangle \text{ where } \Omega = \{\omega_1 \mid \omega_1 = \text{cond}_1\}$ $X = \langle O, M \rangle \text{ where } O = \{ob_1, ob_2, ob_3, ob_4\}$ $M = \{m_1, m_2, m_3, m_4\}$ $c = \begin{cases} ob_1 \\ \langle \text{cond}_1 \rangle \\ \langle (m_1 m_3), ob_3 \rangle \end{cases} \begin{cases} [\text{cond}_1 = \text{true}(\langle m_2, ob_2 \rangle, \langle m_3, ob_3 \rangle)] \end{cases}$ $Z^c \subseteq \langle c \rangle, \langle m_4, ob_4 \rangle$ |

| BSM fragment including conditional operations | Analytical representation of BSM fragment |
|---|--|
| Use Case diagram | |
|  | $Z^{UC} \subseteq K \times \Phi$ $K = \langle \Omega \rangle, \Omega = \{\omega_1, \omega_2 \mid \omega_1 = \text{Condition1}, \omega_2 = \text{Condition2}\}$ $\Phi = \langle M \rangle, M = \{m_1, m_2, m_3 \mid m_1 = \text{precedent1}, m_2 = \text{precedent2}, m_3 = \text{precedent3}\}$ $c = \begin{cases} m_1 \\ \langle \omega_1 \rangle [\omega_1 = \text{true}(\langle m_2 \rangle)] \\ \langle \omega_2 \rangle [\omega_2 = \text{true}(\langle m_3 \rangle)] \end{cases}$ $Z^{UC} \subseteq \langle c \rangle$ |
| Sequence diagram | |
|  | $Z^s \subseteq \Lambda \times H$ $\Lambda = \langle \Omega \rangle \text{ where } \Omega = \{\omega_1 \mid \omega_1 = \text{alt}\}$ $H = \langle O, M \rangle \text{ where } O = \{ob_1, ob_2, ob_3\}, M = \{m_1, m_2, m_3\}$ $c = \begin{cases} ob_1 \\ \langle \omega_1 \rangle \begin{cases} [\omega_1 = \text{true}(\langle m_1, ob_2 \rangle)] \\ [\omega_1 = \text{false}(\langle m_1, ob_1 \rangle)] \end{cases} \end{cases}$ $Z^s \subseteq \langle c \rangle, \langle m_3, ob_3 \rangle$ |

Using key features of conditional operation and (14) it is proved that the proposed approach allows to obtain unified analytical representation for the same operation in different types of BSM. Such representation facilitates many operations in MDA sphere and helps to archive an analytical apparatus for precise processing of software models. Thus, in turn, helps to realize many MDD activities more effectively.

Conclusion

The general idea of the approach for the analytical representation of BSM is proposed in this article. Firstly processes are decomposed on operations. Table 2 illustrates types of operations that are typically for different BSM according to UML standard. For the analytical representation of these operations the concept of "building blocks" is proposed. One "building block" corresponds to one type of operation. The detailed analytical representation of conditional operations is proposed. This approach proposes unified notation for an analytical representation of operations in different BSM.

Using this approach one can represent processes with given level of precision and number of details. It provides effective analysis of requirements, test scenarios generation, algorithms elicitation and performing other activities in software development lifecycle that require information about processes. Manipulations with "building blocks" lets to skip, combine or represent some operation with additional details allows focusing just on important features of processes for analysis. For example, when conditional block is used, some number of conditions can be added or removed.

Unified representation of operations allows implementing fast and precise transformations between BSM of considering types. Also integrating of three components, namely: approaches, proposed in paper [Bonais, 2014], methods of transformation CIM to PIM models [Rhazali, 2014] and analytical representation of BSM objects allows to design different transformation technics. Their purpose is to solve the important MDA task, namely, obtaining precise information when resulting BSM contain more detailed information about processes in comparison with initial. Also the advantage of involving unified analytical representation into transformation technics in comparison with approaches, proposed in the paper [Wang, 2014] is that there is no need to design transformation rules or patterns for representation of initial and resulting information.

Also proposed approach lets designing technics for comparing both the same and different types of considering BSMs. Unified representation of operations allows to find precise solution whether the content of different BSMs is matched.

The proposed approach provides a ground for performing such MDA operation as: joining, merging and refactoring software models of the same types. The notation of the proposed approach contains elements, allowing describing both static and behavioral software models (Table 1). It helps to improve notations and technics for different formal specifications. For example, add more complex and precise rules for representation of information systems properties in ZFSCM [Banerjee, 2014].

Using representation of conditional block, proposed in this paper with the OCL language [OCL, 2012] lets remove the limitations when complex conditions are described. Thus, in turn, forms a background for designing UML extensions and profiles considering all peculiarities of an application domain.

Further research

The area of further research is to design a framework for an analytical representation of BSMs. In order to archive this goal it is necessary to do the following:

- define key features of all operations that are specific for considering BSM;
- design analytical representations of these operations;
- propose a notation for storing information about BSMs that includes the sequences of processes;
- design a format of file for storing information about BSMs and technics of visualizing models from these files.

Bibliography

- [Banerjee, 2014] Banerjee, P., & Sarkar, A. Z–Specification of Component Based Software. International Journal of Software Engineering and Its Applications, Science & Engineering Research Support Society (SERSC), Australia, 8(1), 2014.
- [Bonais, 2014] M. Bonais, K. Nguyen, E. Pardede, W. Rahayu. A formalized transformation process for generating design models from business rules. Proceedings of PACIS 2014 Chengdu, China, 2014 Access mode http://www.sersc.org/journals/IJSH/vol8_no3_2014/8.pdf
- [Chebanyuk, 2014] E. Chebanyuk Method of behavioral software models synchronization. International journal "Information models & analyses" Volume 3, Number 2, 2014.
- [Dove, 2014] A. Dove, A. Barua, Y. Cheon. Extending OCL to Better Express UML Qualified Associations. In: SEKE'2014: 26th International Conference on Software Engineering and Knowledge Engineering, 2014
- [FUML, 2013] OMG standard. Semantic of a foundational subset for executed UML model, 2013 Access mode <http://www.omg.org/cgi-bin/doc?formal/2013-08-06.pdf>
- [Hu, 2014] J. Hu, L. Huang, B. Cao, X. Chang. Extended DEVSMML as a Model Transformation Intermediary to Make UML Diagrams Executable. In: Services Computing (SCC), 2014 IEEE International Conference on. IEEE, 2014.
- [OCL 2.4, 2014] OMG Standard Object Constraint Language 2.4 Access mode <http://www.omg.org/spec/OCL/2.4/>
- [Rhazali, 2014] Y. Rhazali, Y. Hadi, A. Mouloudi Transformation Method CIM to PIM: From Business Processes Models Defined in BPMN to Use Case and Class Models Defined in UML International Journal of Computer, Information, Systems and Control Engineering Vol:8 No:8, 2014
- [SOAML, 2010] Service oriented architectural modeling language Version 1.0.1 OMG standard. Access mode <http://www.omg.org/spec/SoaML/1.0.1/PDF>

[Swithinbank, 2005] IBM RedBook. Patterns: Model-Driven Development Using IBM Rational Software Architect
Access mode <http://students.mimuw.edu.pl/~zbyszek/posi/sg247105.pdf>

[UML 2.5, 2012] OMG standard. Unified Modeling Language 2.5, 2012 Access mode
<http://www.omg.org/spec/UML/2.5/Beta1/>

[Wang, 2014] L. Wang, Y. Zhang. Semantic-element-based Defining Approach for Model Transformation Rules.
International Journal of Smart Home Vol.8, No.3, 2014. Access mode
<http://dx.doi.org/10.14257/ijsh.2014.8.3.08>

Authors' Information



Elena Chebanyuk – lecturer in National Aviation University, associate professor of software engineering department, Ukraine;

e-mail: chebanyuk.elena@gmail.com

Major Fields of Scientific Research: Model-Driven Architecture. Domain engineering, Code reuse.