
METHOD OF DOMAIN MODELS DESIGNING

Elena Chebanyuk

Abstract: *Method of domain models designing is proposed in this paper. Domain models are represented as class diagrams. Initial information for domain models designing contains both analytical representation of domain data streams and domain entities.*

Domain data streams are defined using collaboration diagram. Analytical representation of domain data streams as a subset of Cartesian product of the following sets: objects and messages are proposed. Also an approach of decomposition collaboration diagram into fragments for defining data streams is represented.

Rules of defining relations between classes of domain model by means of analysis both analytical and graphical representation of domain data streams are formulated. Both description of domain entities and interconnections between them are represented in terms of algebra describing software static models.

Proposed method involves an iterative approach for domain model designing. Firstly it is necessary to obtain information about class diagram constituents. Then information about methods of classes is complimented using analytical representation of domain data streams. Using both analytical representation of domain data streams and analytical description of entities relations between class diagram constituents are defined and clarified.

An example of designing domain model for domain "designing cutting schemas for leather goods details" is represented. Also information about interconnections of domain entities and domain processes is represented.

Keywords: *class diagram; collaboration diagram; model transformation; model driven architecture; set-theory tool; transformation rules; cutting schemas designing.*

ACM Classification Keywords: *D.2.2 Design Tools and Techniques; D.2.11 Software Architectures*

Introduction

Using models in software development processes increases productivity of various development activities, such as domain analysis, automated code generation, designing domain specific languages, representation of a software system with necessary details, testing, requirement analysis, software documentation, code reuse and other tasks. It is a background for development of special technics and approaches for software models transformation.

Often software models are represented as UML diagrams. Most of the models that are used in software development process can be divided into static and dynamic (behavioral).

Static software models emphasize a structure of a software system using objects, attributes, operations, and relationships. Examples of static software models are class diagrams, component diagrams, packages diagrams and composite structure diagrams [Gupta 2012].

Dynamic of behavioral software models emphasize the dynamic behavior of a software system by showing collaborations among objects, processes and data flows and changes to the internal states of objects. It includes collaboration diagrams, sequence diagrams, activity diagrams and state machine diagrams [Gupta, 2012].

Important task of Model Driven Architecture (MDA) is designing of languages, technics, rules and other tools for transformation of behavioral (dynamic) models into static software models. Using of such technics allows applying an MDA approach for raising effectiveness of processes in software development life cycle according agile methodology.

It is a background of appearing series of papers that are devoted to different aspects of software development processes, creating of analytical tools, generating new artifacts from behavioral software models [Gupta, 2012], estimation of code reuse effectiveness, tools for an analytical description of software static models [Chebanyuk, 2013] and other aspects that are based on software models represented in a form of UML diagrams [Acretoaie, 2013; Whittle, 2009; Kappel, 2012].

This paper is a continuation of researches that are presented in paper [Chebanyuk, 2013].

Related works

Paper [Gupta, 2012] represents an approach of generating test cases based on use case models that are refined by state diagrams. State diagrams are transformed into usage graphs and then to usage models from which test cases are generated. Also an automated approach of using Adaptive Agent to automatically generate test scenarios from the UML Activity Diagrams is represented. Also the features of PETA tool for the solving of automatic generation of test cases are presented. PETA tool is Java/eclipse based platform for automated software testing.

But an operation [Gupta, 2012] of representation UML activity diagram as state table and writing it into in to some file is rather consuming when activity diagram is large. Also mechanical errors are possible when test cases are generated.

Recently, several approaches adopting the Model Transformation technics to software development processes have been proposed [Whittle, 2009]. These approaches use the concrete syntax of the source and target models to define transformation rules, and thus propose a change to the overall model transformation mechanism. Namely, the transformation definition directly references the source and target models.

Paper [Acretoaie, 2013] is devoted to definition and implementation of a model transformation language focused on usability. This language uses transformation templates and attributes for refactoring models. Before refactoring a model must satisfy to some preconditions. And after refactoring some postconditions must hold true too. But templates that are proposed in paper [Acretoaie, 2013] relate only to class diagram. Use of information from behavioral models allows clarifying patterns, designing new templates, and increasing an effectiveness of models refactoring procedure.

Also an approach originates in the Aspect Oriented Modeling (AOM) field. For example (MATA) [Kappel, 2012] is an aspect composition language based on graph transformation rules expressed in concrete syntax.

Task and challenges

To design transformation rules in order to convert software models of one type (for example static model of one type converts to static model of another type).

But more effective domain models can be designed using a complex approach considering initial information from both types of software models static and dynamic. Also it is necessary to consider information about peculiarities both data streams with and structural components of domain model.

Domain model that is designed according to the proposed method must meet the following requirements of completeness, information content, accuracy and not contradictory. Also the format that is proposed for saving

information about domain models should be compatible with formats of ontology saving (for example RDF or OWL), to allow transformation, refactoring, verification and other operations with models.

Task: to propose the method of domain models designing. Domain model must meet the following requirements: completeness, information content, accuracy and not contradictory. In order to meet this requirements domain model should be designed using information from both behavioral software models and structural components of domain. In order to analyze domain processes and data streams to propose the rules of collaboration diagrams analysis. To design rules for mapping fragments of collaboration and class diagrams.

Analytical representation of domain processes

Represent an analytical description of domain processes, using behavioral software models, namely collaboration diagram. Appointment of a collaboration diagram is the next "Collaboration and sequence diagrams are used to capture dynamic interactions between objects and system. A collaboration diagram consists of objects and associations that describe how the objects communicate. An interaction occurs when two or more objects are used together to accomplish one complete task [Gupta, 2012].

Introduce the following notation.

A set of collaboration diagram objects is denoted as *Object*.

A set of collaboration diagram messages is denoted as *Messages*.

Consider a process of some domain entity creation.

Data streams that are occurring before some domain entity creation are denoted as input data streams.

A subset of the set *Object* that contains objects that are used in input data streams is denoted as $Object^{entity}$. $Messages^{entity}$ - is a subset of the set *Messages* that are used in input streams of some domain entity creation. This entity should be matched with some object in collaboration diagram, namely having the same name.

Subsets $Object^{entity}$ and $Messages^{entity}$ contain only those objects and messages that are directly interconnected with considering collaboration diagram object (or domain entity). This interconnection is represented by arrows in graphical notation of collaboration diagram.

Consider any collaboration diagram element $object \in Object$. Describe a process of creating domain entity as a subset of Cartesian product of the following sets: $Object^{entity}$ and $Messages^{entity}$.

$$\left\{ \begin{array}{l} Entity \subseteq Object^{entity} \times Messages^{entity} \\ Object^{entity} = \{object_0, \dots, object_n\} \\ Messages^{entity} = \{message_0, \dots, message_m\} \\ Object^{entity} \times Messages^{entity} = \langle object_0, message_0 \rangle, \langle object_1, message_0 \rangle \dots \langle object_n, message_m \rangle \end{array} \right. \quad (1)$$


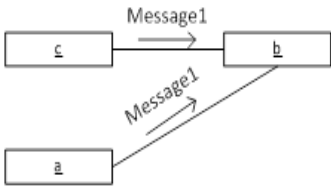
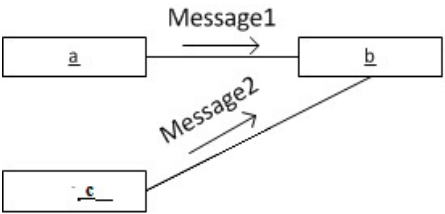
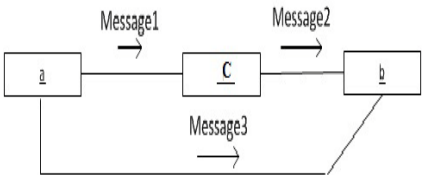
Expression (1) shows that for creation of some domain entity objects and messages can interoperate in any order. It allows usage of alternative data streams for domain entities creation.

Templates for input data streams analysis

Proving the condition of reliability collaboration diagram interconnections shows actual domain data streams. Collaboration diagram analysis shows, that graphical notation proposes several types of input data streams. We

analyze them in order to represent templates for transformation of collaboration diagram fragments into their analytical representation and class diagrams fragments.

Table 1. Templates for input data streams analysis

Graphical representation of input streams	Interconnections between objects
<p style="text-align: center;">1</p>  <p>Figure 1. The first type of input data stream</p>	<p style="text-align: center;">2</p> <p>Template one (Figure 1) shows the first type of input data stream. Analytical description of this template is denoted as</p> $a \rightarrow (Message1) \rightarrow b \quad (2)$ <p>This template shows that in order to create object B it is necessary to use an object A or any of its properties. This template corresponds to composition relation between objects A and B.</p>
 <p>Figure 2. The second type of input data stream</p>	<p>Template two (Figure 2) shows the second type of input data stream. Analytical description of this template is denoted as</p> $a, c \rightarrow (Message1) \rightarrow b \quad (3)$ <p>This template shows that in order to create object B it is necessary to use objects A and C or any of their properties. This template corresponds to composition relations between pairs of objects A, B and B, C respectively.</p> <p><i>The note If the number of input data streams will be more than two an input data stream will be considered as template two.</i></p>
 <p>Figure 3.1. The third type of input data stream</p>	<p>Template three (Figure 3.1) shows the third type of input data stream. Analytical description of this template is denoted as</p> $a, c \rightarrow (Message1, Message2) \rightarrow b \quad (4)$ <p>Template three shows that there are alternative input streams for creation of object B. This template corresponds to aggregation relations between pairs of objects A, B and B, C respectively. <i>The note If the number of input objects data streams are more than two an input data stream will be considered as template three.</i></p>
 <p>Figure 3.2. The fourth type of input data stream</p>	<p>Template four (Figure 3.2) shows the fourth type of input data stream. Analytical description of this template is denoted as</p> $a, c \rightarrow (Message2, Message3) \rightarrow b \quad (5)$ <p>Template four also shows that there are alternative input streams for creation of object B. As analytical description of template four matches with analytical description of template three we consider that this template also corresponds to aggregation relations between pairs of objects A, B and B, C respectively.</p>

Prove that streams of the first and second types correspond to composition relations.

Consider a stream of the first type. A set from n operations that are executed by object B is denoted as $E^b = \{op_1, op_2, \dots, op_n\}$. Number of operations in the set E^b is denoted as N^b ; number of operations that are depended upon object A or its properties is denoted as $N^b(a)$.

In order to prove composition relations it is necessary to prove that $N^b(a) = N^b$.

Consider that $N^b(a) < N^b$. Then it is necessary to find at least one input interconnection between object B and other collaboration diagram objects. Then an analytical representation of input stream will meet to (4) or (5). But input stream corresponds to the condition (2). That is to create an object B we need to involve an object A or some of its properties. This implies that the object B depends upon the object A or some of its properties and equality is proved.

Proof that input stream of the second type is considering to composition relations between all object pairs of objects A, B and A, C is similar. Data streams are considered pairwise.

Thus, analytical representation of the second type of data streams is denoted as follows:

$$\left\{ \begin{array}{l} Entity = b \ b \subseteq Object^b \times Messages^b \\ Object^b = \{a, c\} \\ Messages^b = \{Message_1\} \\ Object^b \times Messages^b = \langle a, c, Message_1 \rangle \end{array} \right. \quad (6)$$

Prove that streams of the third and fourth types correspond to aggregation relations.

In order to define aggregation relations it is necessary to prove that $N^b(a) < N^b$. It shows, that there are some operations, that are made by object B and these operations do not depend upon object A .

Consider a case with two input streams, then

$$N^b(a) + N^b(c) = N^b \quad (7)$$

As a collaboration diagram meet the following requirements: completeness, information content, accuracy and not contradictory an object B can be created with the help of two operations, namely

$$a \rightarrow (Message_1) \rightarrow b, \text{ or} \quad (8)$$

$$c \rightarrow (Message_2) \rightarrow b \quad (9)$$

Let us assume that the object B was created by means of operation (8). Then not all operations depend upon the object C or its properties, otherwise the equality (7) is not proved.

Thus, analytical representation of the third and the forth types of data streams are denoted as follows:

$$\left\{ \begin{array}{l} Entity = b \ b \subseteq Object^b \times Messages^b \\ Object^b = \{a, c\} \\ Messages^b = \{Message_1, Message_2\} \\ Object^b \times Messages^b = \langle a, Message_1 \rangle \langle c, Message_2 \rangle \end{array} \right. \quad (10)$$

The note in order to denote a message its number on collaboration diagram is used.

Method of domain models designing

1. Obtaining information about domain entities in terms of algebra describing software static models.

Domain entities are described as classes of this algebra.

Define a class C as a subset of Cartesian product of the following sets: properties – A, fields – X and methods -B

$$\begin{cases} C \subseteq A \times X \times B \\ A \times X \times B = \{ \langle \alpha_1, \chi_1, \beta_1 \rangle, \langle \alpha_1, \chi_1, \beta_2 \rangle, \langle \alpha_1, \chi_2, \beta_1 \rangle \dots \langle \alpha_n, \chi_m, \beta_k \rangle \} \end{cases} \quad (11)$$

where n – is a number of class C properties, m – is a number of its fields, k – is a number of methods. Every triple can contain one empty set. All properties and method of a class C with modifier private are denoted as follows $C^{private}$, public - C^{public} and protected - $C^{protected}$ respectively. Class C is denoted as follows:

$$C = C^{public} \cup C^{private} \cup C^{protected} \quad (12)$$

At least one set from the expression (12) can't be empty.

All elements of a set X (fields of class C) are related to $C^{private}$, that is

$$C^{private} = C^{private} \cup X \quad (13)$$

If when the description starts the name of class is known class is denoted as follows $C(name)$ [Chebanyuk, 2013].

A set of problem domain entities is denoted as follows:

$$\Theta = \{C(name)_1, C(name)_{2,\dots}, C(name)_p\}$$

where p – is the number of domain entities.

2. Designing a collaboration diagram showing processes and data streams of domain model
3. Forming a set P from those collaboration diagram objects that match with domain entities names.

$$P = \Theta \cap Object, P \neq \emptyset, Object \neq \emptyset, \Theta \neq \emptyset \quad (14)$$

The power of the set P shows how many domain entities match with names of the collaboration diagram objects.

4. For every $object \in Object$ analytical description of input data streams according to (1) is formed.
5. Clarification of domain entities description

Consider a class C(Name) satisfying the following conditions $C(Name) \in P, C(Name) = Entity$.

A set of methods $B^{C(Name)}$ of this class is supplemented by elements of a set $Messages^{entity}$, namely

$$B^{C(name)^I} = B^{C(name)} \cup Messages^{entity} \quad (15)$$

6. Defining association relations between domain entities.

Consider a pair of objects. A class C(Name) satisfying the following conditions

$$C(Name) \in P, C(Name) \neq Entity, C(Name) \in Object^{entity} \quad (16)$$

and collaboration diagram object named Entity. Association relations are set between all classes, that are satisfied to the condition (16) and this object.

According to algebra, describing software static models association relationships are denoted as follows:

$$F(C_0^I)^{acc} = F(C_0^I) \cup \Omega, \Omega \neq \emptyset \quad (17)$$

where $F(C_0^I)^{acc}$ - is a functionality of class C_0^I when it is interconnected by relationship of association with class C_0^II . Classes C_0^I and C_0^II , that are not interconnected by relationship of inheritance. Define a set of C_0^I

methods in class that are called from class C_0^I as Ω . If classes C_0^I and C_0^{II} are interconnected by relationship of association, then the functionality of C_0^I spreads on methods from the set Ω .

7. Clarifying association types between domain entities

Consider a class $C(name) \in P$. Denote a set of its properties as $A^{C(name)}$. Consider an element $\alpha^{C(name)} \in A$. Denote a collaboration diagram object with the same name as property $\alpha^{C(name)} \in A$ as OB.

Both graphical and analytical representations of object OB are analyzed. Then we compare input stream of object OB with analytical (6), (10) and graphical representation (Figure 1-3) of input streams. If object OB input stream matches to (6) or (10) representation we set proper type of association relation between classes OB and $C(name) \in P$.

This operation is made for every element of the set P.

Analytical description of aggregation and composition relations was also proposed by algebra describing software static models [Chebanyuk, 2013].

Functionality of a class C_0^I when it is interconnected by relationship of aggregation with class C_0^{II} is denoted as follows:

$$F(C_0^I)^{aggr} = F(C_0^I) \cup (F(C_0^{II}) \setminus F(C_0^{II}^{private})) \quad (18)$$

where $F(C_0^I)^{aggr}$ - is a functionality of class C_0^I when it is interconnected by relationship of aggregation with the class C_0^{II} . When object of type C_0^{II} is created in a method of class C_0^I , it is possible to call all methods of this object C_0^{II} , excluding private.

Considering, that aggregation and composition relationships are differ by its content, not by structure when classes C_0^I and C_0^{II} are interconnected by relationship of aggregation, the functionality of class C_0^I is spreads similar to (14) and is denoted as follows:

$$F(C_0^I)^{comp} = F(C_0^I) \cup (F(C_0^{II}) \setminus F(C_0^{II}^{private})) \quad (19)$$

where $F(C_0^I)^{comp}$ - is a functionality of class C_0^I when it is interconnected by relationship of composition with the class C_0^{II} .

The note if between some domain entities association relation was set and after that relation of composition or aggregation was defined more strong relation remains, namely association changes to composition or aggregation. Aggregation can be changed into composition.

Designing domain model for the domain "designing leather goods for cutting schemas"

1. Obtaining information about domain entities in terms of algebra describing software static models.

Consider main domain entities: details, linear effect – L, double grid – W, layout – Li, section – Sec and cutting schema -Sh. Math apparatus for designing of single and double grid was described in paper [Chebanyuk, 2013]. Analytical description of classes $C(Laying)$, $C(Leather_laying)$ and $C(Leather_layng_two)$ also was proposed in this paper.

Consider domain entity – linear effect.

According to [Чупринка, 2011] defining of grid vectors is based on dense combinations of rectangles that have been described around details. When two details are combined together there are six types of linear effects (Figure 4).

The note: if detail is rotated on angle 180 by axis X it's considered as detail of the second type.

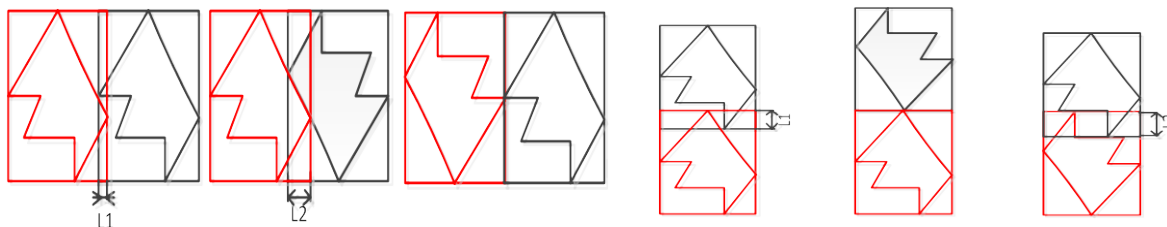


Figure 4. Six types of linear effects when different details are combined together

Class Linear, namely $C(\text{Linear})$, has the next properties – details that are used for defining a linear effect, namely $C(\text{detail1})$, $C(\text{detail2})$; grid for constructing of a laying, namely $C(\text{grid})$, vectors for designing of a grid, namely a_1, a_2 and g [Chebanyuk, 2013] and a percent of material usage, namely P .

Methods of class $C(\text{Linear})$ – an estimation of a material usage percent, namely $\text{estimate}()$; creation of a laying, namely - $\text{create}()$; saving of laying parameters, namely - $\text{save}()$; defining maximum linear effect for one detail combination, namely - $L1_Max()$; defining maximum linear effect for two details combination, namely - $L2_Max()$.

Analytical representation of a class $C(\text{Linear})$:

$$\begin{cases} C(\text{Linear}) = A \times B \\ A = \{\alpha_0, \dots, \alpha_6 \mid \alpha_0 = C(\text{detail1}), \alpha_1 = C(\text{detail2}), \alpha_2 = C(\text{grid}), \alpha_3 = P, \alpha_4 = a_1, \alpha_5 = a_2, \alpha_6 = \\ B = \{\beta_0, \dots, \beta_4 \mid \beta_0 = \text{create}, \beta_1 = \text{estimate}, \beta_2 = \text{save}, \beta_3 = L1_Max(), \beta_4 = L2_Max()\} \end{cases} \quad (20)$$

Using grids layings are built. Domain entity $C(\text{Laying})$ was considered in paper [Chebanyuk, 2013]. Layings are building blocks for constructing layouts.

Consider a domain entity – layout. Denote the height of material as – Height_m , length - Length_m . Layout – it is rectangular area of a material, length Length_L ($0 < \text{Length}_L < \text{Length}_m$) and height - Height_L ($0 < \text{Height}_L < \text{Height}_m$) with details that are systematically placed.

Class Layout, namely $C(\text{Layout})$, has the next properties –length and height of layout, namely Height_L and Length_L ; details that are used in layout, namely $C(\text{detail1})$ and $C(\text{detail2})$; square of this details, namely $Sq1$ and $Sq2$; double grid for designing of a laying , namely $C(W)$, density of layout den_L - and a layout period, namely Period .

Methods of class $C(\text{Layout})$ – an estimation of a material usage percent, namely $\text{estimate}()$; creation of a layout, namely - $\text{create}()$; saving of layout parameters, namely - $\text{save}()$; defining layout period, namely – $\text{Define_Period}()$; defining density of layout – $\text{Define_density}()$.

Analytical representation of a class $C(\text{Layout})$:

$$\begin{cases}
 C(\text{Layout}) = A \times B \\
 A = \{\alpha_0, \dots, \alpha_8 \mid \alpha_0 = \text{Height_L}, \alpha_1 = \text{Length_L}, \alpha_2 = C(\text{detail1}), \alpha_3 = C(\text{detail2}), \alpha_4 = C(W) \\
 \alpha_5 = \text{Period}, \alpha_6 = \text{den_L}, \alpha_7 = \text{Sq}_1, \alpha_8 = \text{Sq}_2\} \\
 B = \{\beta_0, \dots, \beta_4 \mid \beta_0 = \text{create}, \beta_1 = \text{estimate}, \beta_2 = \text{save}, \beta_3 = \text{Define_Period}, \beta_4 = \text{Define_density}\}
 \end{cases} \quad (21)$$

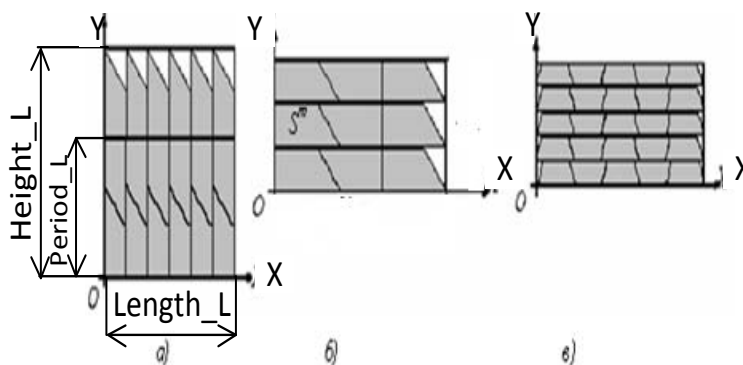


Figure 5. Examples and parameters of layouts

A set of layouts is saved in data structure – List. Operations with the list of layouts are represented in class $C(\text{Lay})$.

Consider a domain entity – section. According to definition section (Figure 6) combines details from some layouts [Чупринка, 2011].

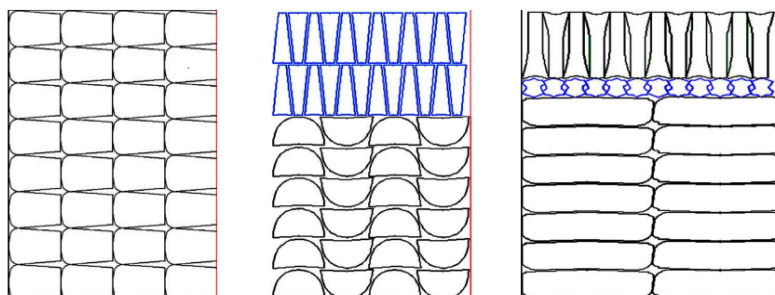


Figure 6. Examples of sections from one, two or three layings

Class Section, namely $C(\text{Section})$, has the next properties – length and height of section, namely Height_S and Length_S; layouts that are used in section, namely list of $C(\text{Lay})$ class; density of section den_S.

Methods of class $C(\text{Section})$ – an estimation of a material usage percent, namely estimate(); creation of a section, namely - create(); saving of section parameters, namely - save(); visualization of section, namely – visualize().

Analytical representation of a class $C(\text{Section})$:

$$\begin{cases}
 C(\text{Section}) = A \times B \\
 A = \{\alpha_0, \dots, \alpha_3 \mid \alpha_0 = \text{Height_S}, \alpha_1 = \text{Length_S}, \alpha_2 = C(\text{Lay}), \alpha_3 = \text{den_S}\} \\
 B = \{\beta_0, \dots, \beta_4 \mid \beta_0 = \text{create}, \beta_1 = \text{estimate}, \beta_2 = \text{save}, \beta_3 = \text{Visualize}, \beta_4 = \text{Swap_Layings}\}
 \end{cases} \quad (22)$$

Cutting schema consist from sections [Чупринка, 2011]. Screenshot with an example of cutting schema is represented on Figure 7.

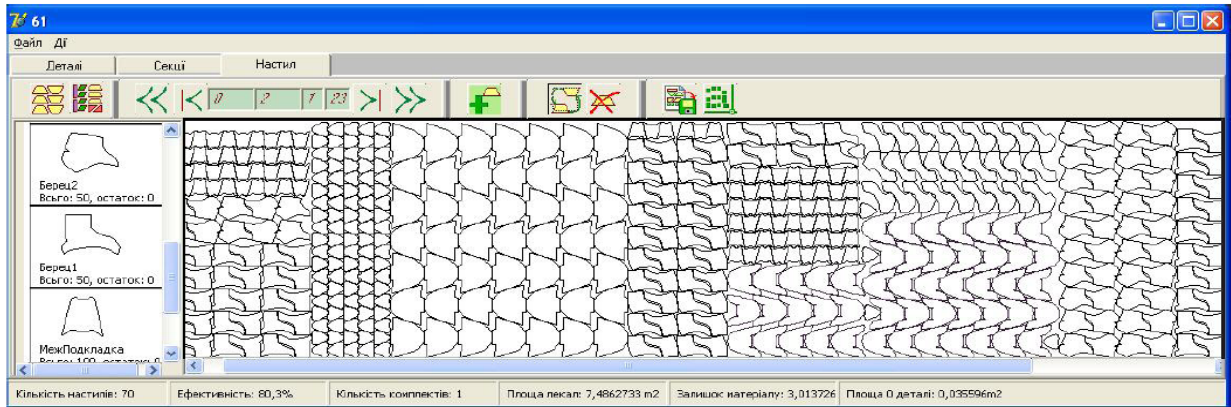


Figure 7. Screenshot with an example of cutting schema

2. Designing a collaboration diagram showing processes and data streams of domain model. Collaboration diagram must meet the following requirements of completeness, information content, accuracy and not contradictory. As collaboration diagram was constructed after detailed analysis of domain entities (20)-(22) it meets all this requirements.

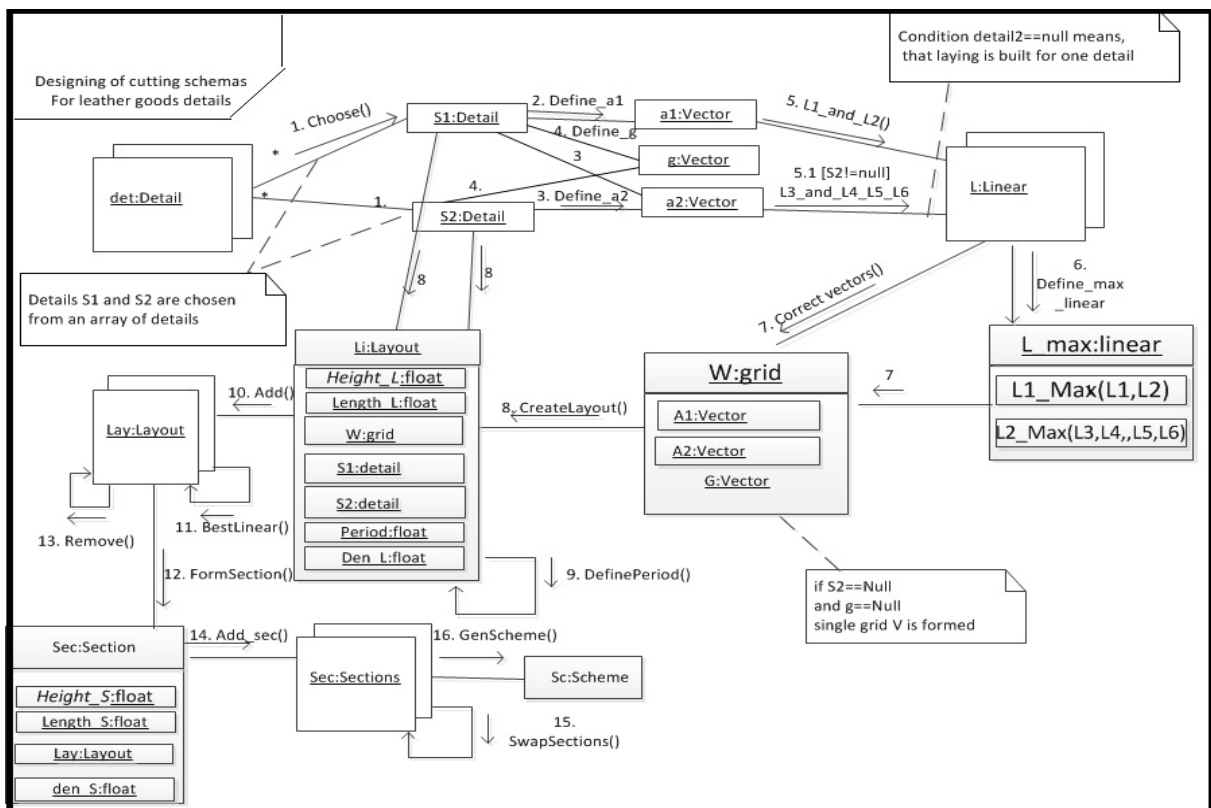


Figure 8. Collaboration diagram, showing streams of domain "designing leather goods for cutting schemas"

3. Forming a set P (14) from those collaboration diagram objects that are matched with domain entities names.

$$P = \{p_0, \dots, p_5 \mid p_0 = \text{det}, p_1 = L, p_2 = W, p_3 = Li, p_4 = \text{Sec}, p_5 = \text{Scheme}\} \quad (23)$$

All collaboration diagram objects match with domain classes.

4. For every $object \in Object$ analytical description of input data streams according to (1) is formed.

Describe a process of domain entities creation (1).

$$\begin{cases} Entity = LL \subseteq Object^L \times Messages^L \\ Object^L = \{a_1, a_2, g\} \\ Messages^L = \{5, 5.1, \dots\} \\ Object^L \times Messages^L = \langle a_1, 5 \rangle, \langle a_1, a_2, g, 5.1 \rangle \end{cases} \quad (24)$$

Consider the expression (24). $Object^L$ - is a set of objects that are used in input data stream when entity L is created, respectively $Messages^L$ is a set of such messages. Elements of the set $Messages^L$ are the numbers of proper messages. Other domain entities are created similarly.

$$\begin{cases} Entity = WW \subseteq Object^W \times Messages^W \\ Object^W = \{a_1, a_2, g, L, L_max\} \\ Messages^W = \{7, 7.1\} \\ Object^W \times Messages^W = \langle a_1, a_2, g, 7 \rangle, \langle L_max, 7.1 \rangle \end{cases} \quad (25)$$

$$\begin{cases} Entity = LiLi \subseteq Object^{Li} \times Messages^{Li} \\ Object^{Li} = \{W, S_1, S_2\} \\ Messages^{Li} = \{8\} \\ Object^{Li} \times Messages^{Li} = \langle W, S_1, S_2, 8 \rangle \end{cases} \quad (26)$$

$$\begin{cases} Entity = SecS \subseteq Object^{Sec} \times Messages^{Sec} \\ Object^{Sec} = \{Lay\} \\ Messages^{Sec} = \{12\} \\ Object^{Sec} \times Messages^{Sec} = \langle Lay, 12 \rangle \end{cases} \quad (27)$$

5. Clarification of domain entities description.

5.1. Consider object L. Type of class L is Linear (Figure 8). According to (15) a set of methods B^{Linear} is complimented by elements and class Linear is complimented by methods:

$$(B^{Linear})^* = B^{Linear} \cup L1_and_L2 \cup L3_and_L4_L5_L6$$

5.2 Consider object Li. Type of class L is Layout (Figure 8). Consider its input message CreateLayout (26). A set of methods B^{Layout} contains a constructor. In this case the set B^{Layout} is not complimented.

5.3. Consider object sec. Type of class L is Section (Figure 8). Consider its input message FormSection (Figure 8). A set of methods $B^{section}$ contains a constructor. In this case the set $B^{section}$ is not complimented.

6. Defining association relations between domain entities.

6.1. Define relations between object L (object of type Linear) and other objects that are elements of the set P. Classes a_1 , a_2 and g are properties of class L (21) and elements of the set $Object^L$. Considering pairs of classes Linear and a_1 , Linear and a_2 , Linear and g . Between classes in this pairs association relations are set. According to representation of association relation (17) we can write:

$$\begin{cases} F(L)^{acc} = F(L) \cup \Omega \\ \Omega = \{\beta_0, \beta_1 \mid \beta_0 = L1_and_L2, \beta_1 = L3_and_L4_L5_L6\} \end{cases} \quad (28)$$

6.2. Define relations between object W and other objects that are elements of the set P.

All properties of class W match with elements of the set $Object^W$ (26). Considering pairs of classes W and a_1 , W and a_2 , W and g , W and L, W and L_max. Between classes in this pairs association relations are set. According to representation of association relation (17) we can write:

$$\begin{cases} F(C(W))^{acc} = F(W) \cup \Omega \\ \Omega = \{\beta_0 = Create_W\} \end{cases} \quad (29)$$

Analysis of other collaboration diagram objects is made similar.

7. Clarifying association types between domain entities

7.1. Consider properties of class Linear that are matched with elements of the set Piths properties are S1 and S2. We analyze analytical and graphical representation of input streams for these objects. Input streams of objects S1 and S2 corresponds to the first data stream (Figure 1). According to this analysis composition relation between classes Detail and Linear is set.

Check this information using knowledge about domain. According to definition of linear effect it is not defined without information about details (Figure 4) this fact proves composition relations between classes W and detail (S1 and S2 are instances of this class).

$$F(C(L))^{comp} = F(C(L)) \cup (F(C(det)) \setminus F(C(det)^{private})) \quad (30)$$

7.2 Consider properties of class W that are matched with elements of the set P. These properties are a_1, a_2, g . We analyze analytical and graphical representation of input streams for these objects. Input streams correspond to the second data stream (Figure 2). According to this analysis composition relation between pairs of classes W and a_1 , W and a_2 , W and g is set. Substitute weak association type to more strong. And composition relations between all these three pairs are remained.

Check this information using knowledge about domain "designing cutting schemas for leather goods details". According to definition of double grid it is consisted from vectors. This fact proves composition relations between following pairs of classes W and a_1 , W and a_2 , W and g .

But it is not necessary to use a procedure of calculating linear effects according to some algorithms of cutting schemas designing. This fact proves association relations between classes W and L.

As a result represent an analytical description of relations is denoted as follows

$$\begin{cases} F(C(W))^{acc} = F(W) \cup \Omega \\ \Omega = \{\beta_0 = Create_W\} \\ F(C(W))^{comp} = C(W) \cup (F(C(a_1)) \setminus F(a_1^{private})) \cup (F(C(a_2)) \setminus F(a_2^{private})) \cup (F(C(g)) \setminus F(g^{private})) \end{cases} \quad (31)$$

Analysis of other collaboration diagram objects is made similar. Represent the analytical description of relations other collaboration object diagrams.

Analytical representation of class Linear relations.

$$\begin{cases} F(C(Li))^{acc} = F(Li) \cup \Omega \\ \Omega = \{\beta_0 = CreateLayout\} \\ F(C(Li))^{comp} = C(W) \cup (F(C(Detail_1)) \setminus F(Detail_1^{private})) \cup (F(C(Detail_2)) \setminus F(Detail_2^{private})) \end{cases} \quad (32)$$

Analytical representation of class Section relations.

$$\begin{cases} F(C(\text{Sec}))^{acc} = F(\text{Sec}) \cup \Omega \\ \Omega = \{\beta_0 = \text{FormSection}\} \\ F(C(\text{Sec}))^{comp} = F(C(\text{Sec})) \cup (F(C(\text{Lay})) \setminus F(\text{Lay}^{private})) \end{cases} \quad (33)$$

Model of domain "designing cutting schemas for leather goods details" designed according to the proposed method is represented on Figure 9.

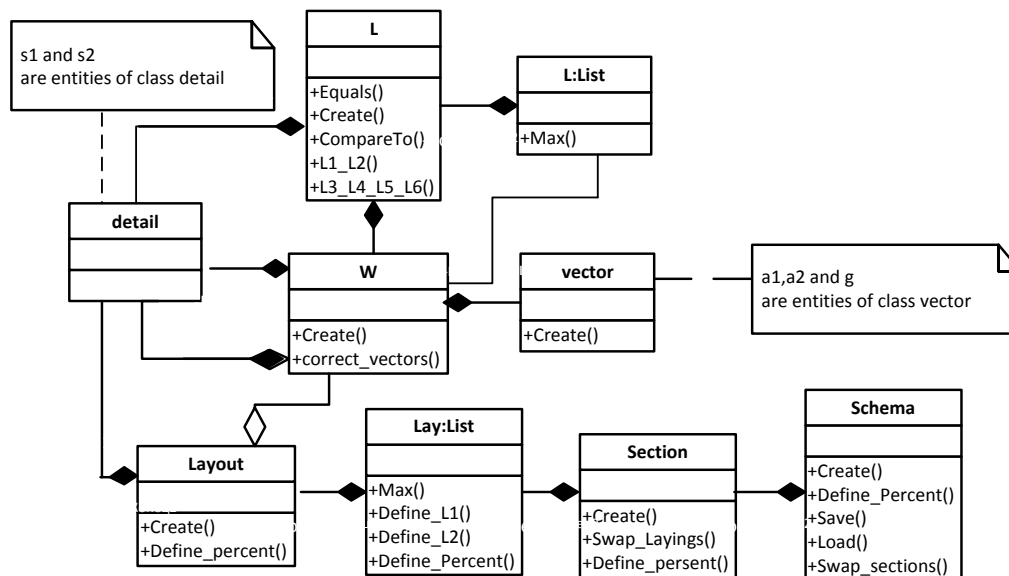


Figure 9. Model of domain "designing cutting schemas for leather goods details"

Conclusion

Review of papers that are devoted to model reuse shows that researchers mostly pay attention to design transformation rules in order to convert software models of one type (for example static model of one type converts to static model of another type).

The peculiarity of the proposed method of domain models designing is that it is necessary to use information both from static and dynamic models. It allows obtaining more initial data that can be considered while domain model designing. And also it influences to the quality of result.

Format of saving information about domain models is compatible with languages for ontology description (for example RDF or OWL). It simplifies the procedure of model refactoring, describing new ontologies, splitting and dividing models and other.

Rules of defining relations between entities that are based on domain streams analysis can be basic for designing transformation technics when other dynamic models are transformed into static. Using of this technics allow to help solving many actual task in MDA area.

Further exploration

Using the method of designing domain models and algebra describing software static models allows to represent frameworks analytically by means of grouping class diagrams constituents according to design patterns and

propose a method of matching problem domain processes to constituents of a class diagram while designing frameworks.

For this it is necessary to propose a concept of mapping processes characteristics and design patterns. This concept also allows defining necessary components from a framework can be reused while analyzing applications functional requirements.

Bibliography

- [Acretoaie, 2013] V. Acretoaie. Delivering the Next Generation of Model Transformation Languages and Tools, European conference of object oriented programming, pp. 2-10, 2013.
- [Chebanyuk, 2013] E. Chebanyuk Algebra describing software static models, INFOS 2013 - "Intelligent Information and Engineering Systems", in press, 2013.
- [Gupta, 2012] S. Gupta, J. Singla. A component-based approach for test case generation, International Journal of Information Technology 5/2, pp. 239-243, 2012
- [Kappel, 2012] Kappel, G., Langer, P., Retschitzegger, W., Schwinger, W., Wimmer, M., Model transformation by-example: a survey of the _rst wave, in: Conceptual Modeling and Its Theoretical Foundations, pp. 197-215, 2012.
- [Whittle, 2009] Whittle, J., Jayaraman, P., Elkhodary, A., Moreira, A., Ara_ujo, J., MATA: A United Approach for Composing UML Aspect Models Based on Graph Transformation, In: Transactions on Aspect-Oriented Software Development VI - Special Issue on Aspects and Model-Driven Engineering, Springer, 2009, pp. 191-237.
- [Чупринка, 2011] В.І. Чупринка, О.В. Чебанюк, Автоматизоване проектування раціональних схем розкрою рулонних матеріалів на деталі виробів шкіргалантереї, Вісник Східноукраїнського національного університету ім; Даля №7(2), 2011,с. 46-50.

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: Domain analysis, Domain engineering, Code reuse.