ISSN 1392 – 124X INFORMACINĖS TECHNOLOGIJOS IR VALDYMAS, 2004, Nr. 4(33)

COMPOSITION OF TASK-DIALOG MODEL IN THE SPACE OF PROCESSES

Diana Zobakiene, Tadas Zobakas, Saulius Gudas

Kaunas University of Technology, Information Systems Department
Studentu st. 50-309, LT-51368 Kaunas, Lithuania

Abstract. The paper presents the method of composition of task-dialog model used in the design process of user interface. The task-dialog model is based on the process model and composed in the space of processes. The space of processes specifies the knowledge of problem area and is described by hierarchical, non-hierarchical, and process time constructs. This formal structure and the rules of usage define composition process of task-dialog model and ensure fast and smooth user interface development.

1. Indroduction

With a growing interest in integrated modelling of user interface (UI) and information system (IS) there is an increasing need for methods and tools that can control the whole modelling process. The most perspective methods and tools are model-based user interface development environments (MB-UIDEs). These tools provide facilities for user interface designers to describe the UI using a number of abstract models. This means that MB-UIDEs are able to prototype the entire interface of an application by interacting with an appropriate set of tools that manipulate these declarative models [10].

Different notations and tools that are used for the UI design and their underlying software create a problem of the usage of MB-UIDEs. As a consequence of these non-integrated design environments, the same structures of information domain get modelled twice. Moreover, these models may be composed in slightly different ways due to different notations [10].

Unified Modelling Language (UML) [13] is the standard language for object-oriented modelling of software, but the UI, as an important part of most software, should as well be modelled in one or another way. However, UML has neither special diagrams nor methodology for UI modelling. In some UI modelling methods [11] the basic diagrams are UML use cases and/or class, activity diagrams as additional diagrams. Use case diagrams are “an architecture centric processses", while Rational Unified Process (RUP) is considered to be "Use Case Driven Process". Therefore, the user is never considered important in Rational's thinking about design and, in other words, RUP is fundamentally not a user centred design methodology [12].

User interfaces convey the output of applications and the input from application users. For this reason, UIs have to cope with the complexity of both the applications and the users. Therefore, despite its not user-centred design aspects, UML is the best choice in object-oriented modelling of software and UI.

Separate application and UI development processes, different notations used and object-oriented UI project are the main problems that are solved with the new UI modelling method.

The aim of this paper is to present new principles of task-dialog model composition using UML applications model in the space of processes. These principles are part of the new UI modelling method.

This paper has introduction and 5 sections. An overview of UI models and the relationship among them is provided in Section 2. The space of processes is presented and specified by the terms of category theory in Section 3. The composition process of task-dialog model is described in Section 4. A case study of ordering system is presented in Section 5. Conclusions are drawn in Section 6.

2. User Interface Models and Their Interaction

The problem of conciliating application and user interaction complexities is that MB-UIDEs usually have several models describing different aspects of the UI, that models are related to the application models indirectly. UI development methods [9] have diverse models (task, object, user, dialog, etc.), depending on

notation and methodology used. Table 1 presents 4 generalized models that describe various aspects of the user interface [11].

Table 1. The component models of a user interface

Model	Constructors	Function
Application model	Class, attribute, operation, relationship	Describes the properties of the application relevant to the UI.
Task-dialogue model	Task, goal, action, sequencing, task pre-condition, task post-condition	Describes the tasks that users are able to perform using the application, as well as how the tasks are related to each other.
Abstract presentation model	View, abstract interaction object	Provides a conceptual description of the structure and behaviour of the visual parts of the user interface. There the UI is described by the terms of abstract objects.
Concrete presentation model	Window, concrete interaction object, layout	Describes in details the visual parts of the user interfaces. There is explained how the UI is composed in terms of widgets.

Usually user interface modelling starts with the analysis of information specified in the application model. In software modelling process the application model is separated into two models: business model and process model. The business model is concerned with value exchanges among business partners, while the process model focuses on operational and procedural aspects of business communication [3]. The business model is important in composition of user's model, which in some UI modelling methods and tools describes users profiles and roles he is allowed to perform.

In this paper presented UI modelling method the process model is more important than the business model, because UI is supposed to be more related to procedural rather than personal aspects of a problem area. The process model is specified in object oriented UML notation and should have one class diagram and at least one sequence diagram. These diagrams are the main source of information for the TDM development. The TDM is generated from the application model (in our case it's the process model only) as a set of UIs in the space of processes.

UI knowledge base is a meta-model which is based on the formal EMC (Elementary Management Cycle) structure [4]. UI knowledge base structures UI modelling information that is applied to the generated TDM and can produce a list of generation errors. Thus UI developer is forced to return to application model and correct it. The UI life cycle shown in figure 1, is partly iterative (transitions from/to application model & TDM) and partly consecutive (transitions from TDM to abstract presentation model and from abstract to concrete presentation model) [6].

Both abstract and concrete presentation models are supposed to implement MDA (Model-Driven Architecture) principles [8]. These two models are out of scope of this paper. Composition of the TDM from the application model and UI knowledge-based correction of TDM are the key processes for successful GUI development. So this paper is focused on a TDM and application models and EMC-based UI knowledge structuring, as shown in Figure 2.

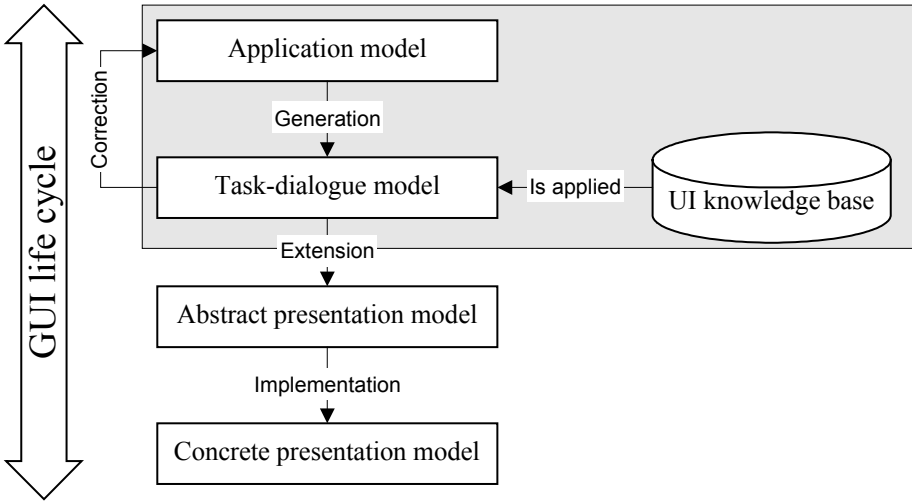


Figure 1. UI model interaction

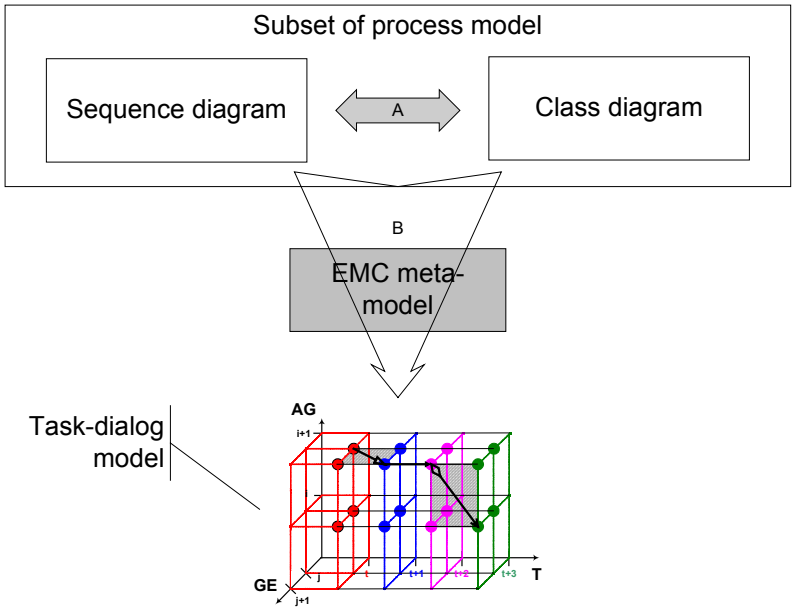


Figure 2. The basic principles of UI modeling

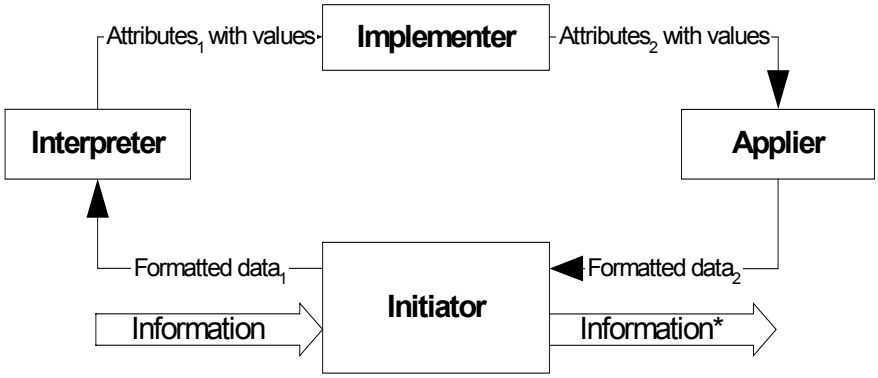


Figure 3. Basic architecture of modified EMC

The composition process of the TDM consists of preparation, generation and check phases. Arrow A in Figure 2 represents the preparation phase, when process model syntactic quality is assured (see Section 4.1 for details). The process model aggregates the basic information of a problem area, which is used in the TDM generation.

In the second phase, the TDM is generated as a set of UIs in the space of processes using the process model (arrow B in Figure 2). The Section 4.2 describes the rules and conditions of transformation process.

The space of processes, its elements (EMCs) and relationships among them are considered a predefined information structure, which is stored in the knowledge base (EMC meta-model).

3. The Space of Processes

The space of processes is a formal structure consisting of elementary management cycles (EMCs) and relationships among them [4,5]. The EMC and the space of processes [4] itself are described in the next sections.

3.1. The Elementary Management Cycle

The main element of process control method is the closed process control loop, which is called elementary management cycle (EMC) [14]. First of all EMC is dedicated to specify a control of material processes in enterprise management [4]. Due to a different purposes of IS development, where representation of information and human-computer interaction processes is more important than material processes, a slightly different terms and conception of EMC was developed [15]. The basic architecture is shown in Figure 3.

Both incoming and outcoming information flows (***Information*** and ***Information**** in Figure 4) are treated as sets of attributes that are directly associated to concrete presentation objects at GUI model level.

The whole process is initiated by incoming information or stimulus (when there's no incoming information), which is put as a request to ***Initiator***. The

Initiator has two functionally different parts. The first one is the GUI, from which Initiator takes the values (user enters or selects these values) and transfers them to Interpreter as a formatted data flow. The second part of Initiator is the Dispatcher, which takes formatted data flow from Applier and transfers it to the next GUI in a sequence.

The goal-driven process of the dataflow restructuring to attributes is called ***Interpreter***. The Interpreter reads the dataflow coming from the Initiator and, according to predefined rules, assigns each of data value to the attribute value of the corresponding class. Interpreter can be specified as a set of formal information interpretation rules.

Class data engineering, decision and other processes are the execution process of an EMC, which is called ***Implementer***. The Implementer is treated as a set of operation rules on the class attributes and their values.

The goal-driven process of structuring values of class attributes to dataflow is called ***Applier***. The Applier takes the values of class attributes from Implementer and, according to predefined rules, forms a dataflow. The Applier can be specified as a set of formal rules on the values of the class attributes.

By the terms of category theory [2], ***EMC*** is a category, which has 4 objects, i.e. $EMC(0) = \{INI, INT, IMP, APP\}$, where $INI \neq \varnothing$, $INT \neq \varnothing$, $IMP \neq \varnothing$, $APP \neq \varnothing$. Arrows of category ***EMC*** are one-directional arrows between neighbouring EMC objects, i.e. $EMC(R) = \{INI\_INT, INT\_IMP, IMP\_APP, APP\_INI\}$ [14].

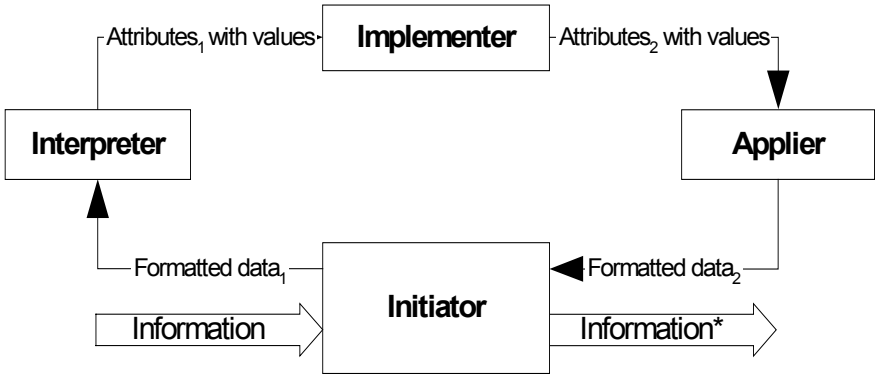


Figure 4. Basic architecture of modified EMC

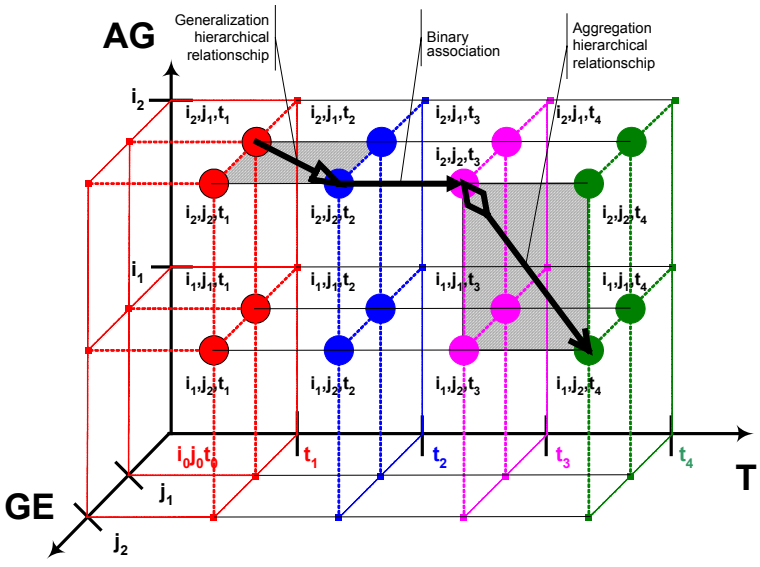


Figure 5. The space of processes with the typical relationships

3.2. Specification of the Space of Processes

This paper presents the user interface modelling method, when UI engineering is based on the EMC and the space of processes. This space consists of three planes built up by three semantically different axes:

- ***Time axis*** (T) defines a sequence of UIs (screen forms) in an execution time. All non-hierarchical relationships (binary association, iteration and multiply executions) are shown on this axis.

Meanwhile, hierarchical relationships between UIs are displayed on the two planes formed between T and AG axes and T and GE axes, respectively.

- ***Generalization axis*** (GE) specifies the correlation between two objects by the terms of generalization action. For instance, when some properties of 2 problem area objects "Person" and "Company" are generalized into another object "Customer", then the corresponding UIs are generalized also.
- Aggregation axis (AG) specifies the correlation between two objects by the terms of aggregation action. For instance, when the problem area object "Order rows" is aggregated into another object "Order", then the corresponding UIs are aggregated also.

An example of the space of processes and typical relationships is depicted in Figure 4.

The lack of some UI development methods is that one class is treated as one UI [1]. This point of view is incorrect in the space of processes, since in this space each *EMC* cycle specifies one class activity or one GUI in the implemented system. So, an abstract *EMC* specifies a dynamic property of an object (one class operation). Usually, one object has more than one dynamic property, which is complex. This results to more than one EMC in order to describe the behaviour of one object.

All UIs related to one problem area are treated as a super-category *TDM* (task dialog model). The sequences of user actions are the objects of the category *TDM*, i.e. $TDM = \{\mathrm{Seq}_1,...,\mathrm{Seq}_{\mathrm{UASN}}\}$. An abstract *TDM* has unlimited number of user action sequences, i.e. $UASN \in \{1,...,\infty\}$ and number *UASN* shows the maximum number of sequences.

An abstract sequence Seq_X, where $X \in \{1, \ldots, UASN\}$, has unlimited number of objects that are *EMCs*. So, the sequence is treated as a set of *EMCs* that specify an order of user actions, i.e. $\mathrm{Seq}_X(0) = \{EMC_1, \ldots, EMC_{EMCN}\}$, where $EMCN \in \{1,...,\infty\}$. The category Seq_X has one attribute:

- ***Name*** – $Seq_X(Name) \neq \varnothing$.

In the space of processes each EMC_Z, where $Z \in \{1, \ldots, EMCN\}$, is specified using 3 position attributes:

- ***T*** dimension (index ***t*** in the Figure 5) specifies EMC_Z position in execution time, i.e. $EMC_Z(T) \in \{1, \ldots, \infty\}$.
- ***AG dimension*** (index ***i*** in the Figure 5) specifies EMC_Z position taking in account the aggregation relationship with former EMC. The position can change as follows $EMC_Z(AG) \in \{-\infty, \ldots, \infty\}$.
- ***GE dimension*** (index ***j*** in the Figure 5) specifies EMC_Z position taking in account the generalization relationship with former *EMC*. The position can change as follows $EMC_Z(GE) \in \{-\infty, \ldots, \infty\}$.

Depending on outcomming information or actions performed by a user, some *EMC*s should be repeated more than one time (e.g. mistake corrections). In this case there exist two levels of abstraction, where one *EMC* (higher abstraction level element) is represented by the sequence of the same *EMC* with different incoming information (lower abstraction level elements). From this follows the conclusion, that the execution time of higher abstraction level *EMC* (t dimension) is a sum of the execution time of each element of lower abstraction level *EMC*s. This two level *EMC* structure should be understood as the terms of dimension and a dimension member in OLAP technology [7].

The set of the lower abstraction level EMCs is represented as an iteration axis in the space of processes. It's treated as extension of time moment or in another words the moment of execution time t_1 is split into a set of execution times, i.e. $t_1 = \{it_1,...,it_n\}$ (see Figure 6).

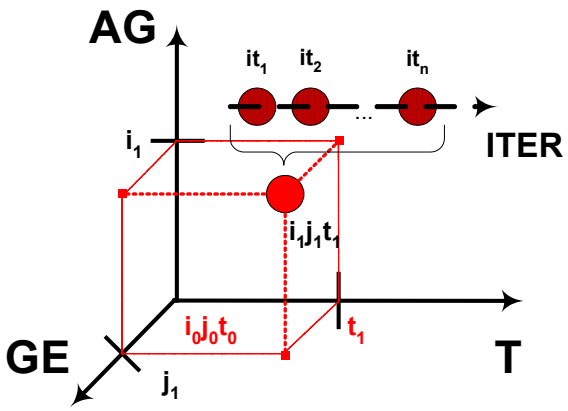


Figure 6. A set of *EMC*s in the iteration axis

So, iterated EMC_Z can consist of a set of the same *EMCs*, i.e. $EMC_Z = \{EMC_Z(IT_1), \ldots, EMC_Z(IT_{ITN})\}$. This means that each EMC_Z has one additional attribute, which specifies its position in the iteration set, in the case it exists, i.e. $EMC_Z(IT) \in \{0, \ldots, \infty\}$.

Both hierarchical and non-hierarchical relationships among the objects of the problem area reflect on user interface scenario. Therefore, they are important part of the space of processes and are treated as arrows between objects of category $\mathrm{Seq_X}$, i.e. $\mathrm{Seq_X}(R) = \{Rel\}$, where $Rel = \{Rel_1, \ldots, Rel_{RN}\}$ and $RE \in \{(EMCN - 1), \ldots, (EMCN * 2 - 1)\}$. Each relationship Rel_Y, where $Y \in \{1, \ldots, RN\}$, has at least 6 attributes:

- ***Type*** describes a nature of relationship in the space of processes. Each relationship can be aggregation, generalization, binary association, iteration or multiple executions, i.e. $Rel_Y(Typ) \in \{AG, GE, Bin, Iter, Multi\}$.

- The ***subject*** of relationship is an *EMC* from a set of *EMCs*, i.e. Rel_Y (*Subject*) ∈ {EMC_1, …, EMC_{EMCN}}.
- ***Subject multiplicity*** describes the existence conditions of subject elements, i.e. Rel_Y (*Subject_multi*) ∈ {0..1, * , 1..*}.
- The ***object*** of relationship is an *EMC* from a set of *EMC*s, i.e. Rel_Y(*Object*) ∈{EMC_1, …, EMC_{EMCN}}.
- ***Object multiplicity*** describes the existence conditions of object elements, i.e. Rel_Y (*Object_multi*) ∈ {0..1, * , 1..*}.
- ***Direction*** shows type of action (forward, return or both). The direction is supposed to have one of 3 values, i.e. Rel_Y (*Direction*) ∈ {–1, 0, 1}. When the transition is from relationship object to the subject, it has the meaning 'forward" and Rel_Y (*Direction*) = 1. In the case the type of action is "return", then Rel_Y(*Direction*) = –1.

3.3. Types of Relationships in the Space of Processes

All types of relationships in the space of processes [15] must fit the following requirement – both subject and object of relationship is an *EMC*, i.e. Rel_Y(*Subject*) = EMC_X and Rel_Y(*Object*) = EMC_Z, where EMC_X, EMC_Z ∈{EMC_1, …, EMC_{EMCN}}.

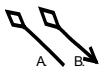
Hierarchical relationship, where Rel_Y(*Typ*) ∈{*AG*, *GE*}, in addition to 6 attributes (see Section 3.2) has 2 specific attributes. These additional attributes specify hierarchical direction of relationship between subject and object of the relationship:

- ***„Parent"*** is the object from higher hierarchical level, which can be subject or object of the relationship, i.e. Rel_Y(*Parent*) ∈ Rel_Y(*Subject*), Rel_Y(*Object*) and Rel_Y(*Parent*) ≠ Rel_Y(*Child*).
- ***„Child"*** is the object from lower hierarchical level, which can also be subject or object of the relationship, i.e. Rel_Y(*Child*) ∈ {Rel_Y(*Subject, *Rel_Y(*Object*)} and Rel_Y(*Child*) ≠ Rel_Y(*Parent*).

There exists only one restriction - the subject and object of hierarchical relationship can't be the same *EMC*, i.e. Rel_Y(*Subject*) ∈ Rel_Y(*Object*) or EMC_X ≠ EMC_Z. While keeping in mind that an *EMC* has 4 objects, the hierarchical relationship is detailed as follows: Rel_Y(*Subject*) = EMC_X(*IMP*) and Rel_Y(*Object*) = EMC_Z(*INI*). Table 2 specifies hierarchical relationships in the space of processes.

When the relationship is child-parent relationship (case A in Table 2) Rel_Y(*Subject*) = Rel_Y(*Child*) = EMC_X and $Rel_Y(Object) = Rel_Y(Parent) = EMC_Z$. When the relationship is parent–child relationship (case B in Table 2) Rel_Y(*Subject*) = Rel_Y(*Parent*) = EMC_X and $Rel_Y(Object) = Rel_Y(Child) = EMC_Z$.

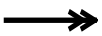
Table 2. Hierarchical relationships

Notation	Specification
	***Generalization (GE)***, when $Rel_Y(Typ) = GE$ is vertically represented bi-directional relationship. For this relationship GE dimension of object EMC is changed, i.e. $EMC_X(GE) \neq EMC_Z(GE)$ and AG dimension is kept the same $EMC_X(AG) = EMC_Z(AG)$. A. The direction of child-parent relationship is according to GE arrow, with positions in the space of processes $EMC_X(GE,T) = (j_1, t_1)$ and $EMC_Z(GE,T) = (j_2, t_2)$. B. The direction of parent–child relationship is opposite to GE arrow, with positions in the space of processes $EMC_X(GE,T) = (j_2, t_1)$ and $EMC_Z(GE,T) = (j_1, t_2)$. Return direction is opposite to type arrow (A) or direction arrow (B).
	***Aggregation (AG)***, when $Rel_Y(Typ) = AG$ is vertically represented bi-directional relationship. For this relationship AG dimension of object EMC is changed $EMC_X(AG) \neq EMC_Z(AG)$ and GE dimension is kept the same $EMC_X(GE) = EMC_Z(GE)$. A. The direction of child-parent relationship is according to AG arrow, with positions in the space of processes $EMC_X(AG,T) = (i_1, t_1)$ and $EMC_Z(AG,T) = (i_2, t_2)$. B. The direction of parent–child relationship is opposite to GE arrow, with positions in the space of processes $EMC_X(AG,T) = (i_2, t_1)$ and $EMC_Z(AG,T) = (i_1, t_2)$. Return direction is opposite to type arrow (A) or direction arrow (B).

Non-hierarchical relationships, when Rel_Y(*Typ*) ∈{*Bin*, *Iter*, *Multi*}, have only 6 attributes: type, subject, subject multiplicity, object, object multiplicity and direction (see Section 3.2). In comparison to hierarchical relationships they can have the same subject and object, i.e. it can be Rel_Y(*Subject*) = Rel_Y(*Object*) or $EMC_X = EMC_Z$.

There exist basic restrictions for all non-hierarchical relationships, i.e. EMC_X(*GE*) = EMC_Z(*GE*), EMC_X(*AG*) = EMC_Z(*AG*). The Table 3 specifies non-hierarchical relationships in the space of processes.

Table 3. Non-hierarchical relationships

Notation	Specification
	***Binary association,*** when $Rel_Y(Typ) = Bin$ and $EMC_X \neq EMC_Z$, is a bi-directional relationship represented in the space of processes horizontally (only T position is changed).
	***Iteration***, when $Rel_Y(Typ) = Iter$, $EMC_X(T) = EMC_Z(T)$ and $EMC_X(IT) \neq EMC_Z(IT)$, is one-directional relationship represented in the space of processes in the additional iteration axis (T, AG and GE positions are not changed). This relationship specifies multiply execution of the same EMC with the different pre-conditions (incoming information).
	***Multiply execution***, when $Rel_Y(Typ) = Multi$ and $EMC_X(T) = EMC_Z(T)$ $EMC_X(IT) = EMC_Z(IT)$, is a one-directional relationship represented in the space of processes in the additional iteration axis (T, AG and GE positions are not changed). This relationship specifies multiply execution of the same EMC with the same pre-conditions (incoming information).

In this section the space of processes, its elements and possible relationships among them are specified by the terms of formal notation – category theory. This formalism is used to describe object-oriented user interface, which will be presented in the next section.

4. Composition of the Task-Dialog Model in the Space of Processes

The task-dialog model is based on the process model, thus syntactic quality of the process model is an important pre-condition in the process of task-dialog model composition.

4.1. Syntactic Quality of Process Model

In this paper presented UI modelling method the process model is treated as a combination of class and sequence diagrams in UML notation. ***Class diagram*** specifies static information of a problem area – data structures (classes and their attributes) and actions with them (class operations). ***Sequence diagram*** specifies an order of system operations, which are directly related to UIs [14].

Syntactic quality of process model means that model is consistent and all its elements are syntactically related to each other. In the quality assurance process the following requirements get proved. It results into the list of missing or redundant parts of process model, which should be necessary in UI development.

The category theory is used to specify these requirements:

1. All sequence diagram objects (*SO*) are classes (*CO*) from the class diagram, i.e. $SO = \{SO_1, \ldots, SO_{SON}\}$, $SON \in \{1,...,CON\}$ ir $SO \subset CO$.
2. All sequence diagram messages (M) are calls of appropriate class operations (O), i.e. $\forall M_Z = CO_C(O_Y)$, where $Z = \{1,...,MN\}$, $C = \{1, \ldots, CON\}$, $Y = \{1, \ldots, ON\}$ and $CO_C = SO_N = M_Z(Sec\_object)$, where $N = \{1,...,SON\}$.
3. The subject $M_Z(Seq\_subject)$ and the object $M_Z(Seq\_object)$ of each sequence diagram message M_Z ($Z \in \{1,...,MN\}$) must have direct relationship in the class diagram, i.e. $\exists R_X$, when $X \in \{1,...,CRN\}$, then $\{R_X(Rel\_subject), \{R_X(Rel\_object) = \{M_Z(Seq\_subject), M_Z(Sec\_object)$.

The process of syntactic control is performed for the each sequence diagram S_Y, where $Y \in \{1,...,SN\}$. Then each message M_Z from the sequence diagram S_Y, where $Z \in \{1,...,MN\}$, is verified against the 3rd rule above – the relationship between appropriate classes must exist, i.e. $\{R_X(Rel\_subject), \{R_X(Rel\_object) = \{M_Z(Seq\_subject), M_Z(Sec\_object)$, where $X \in \{1,...,CRN\}$.

In the case the relationship is not found in the class diagram, UI developer will be informed on this syntactic error and forced to correct it (return arrow to application model in Figure 1).

The all three rules of process model syntactic quality find out missing or redundant relationships not only in both types of diagrams, but also between the diagrams. However, is the relationship missing in one diagram, or redundant in another – UI developer must decide himself.

4.3. Composition of Task-Dialog Model

When the task-dialog model composition pre-conditions are satisfied (syntactic quality of process model is sufficient) it is possible to proceed to the TDM generation phase. The TDM is generated from process model and placed into the space of processes. By the terms of the category theory the transformation rules are called the functor $F_{TDM}: PM \rightarrow TDM$ (arrows B and C in Figure 1).

The functor F_{TDM} is a set of rules, which are applied to the each sequence diagram S_X, where $X \in SN\}$.

- A separate sequence of user actions is composed from the each sequence diagram in the same space of processes, meanwhile the sequence diagram name is assigned to user actions sequence name, i.e. $S_X(Name) = Seq_X(Name)$.

- Transformation to the TDM starts from the first message in the sequence diagram, which has no previous message, i.e. $M_Z(Seq\_prev) = \varnothing$. Thanks to the attributes (Sequence_prev ir Sequence_next) of the each message it's possible to define the previous and the next messages. The subject and object of each message are transformed to abstract *EMC*s in the space of processes. Each message of the sequence diagram $S_X(M_Y)$, where $Y \in \{1, \ldots, MN\}$ and $X \in \{1, \ldots, SN\}$ is transformed to the message Rel_Z ($Z \in \{1, \ldots, RN\}$) in the space of processes according the following rules:
- Appropriate relationship between classes in a class diagram should be found, i.e. $\{M_Z(Seq\_subject), M_Z(Seq\_object)\} = \{R_X(Rel\_subject), R_X(Rel\_object)\}$. If the type of relationship R_X is hierarchical (AG or GE), then "hierarchical direction" must be stored as the attributes of the relationship between EMCs. There exist additional relationship attributes, i.e. "parent" ($\mathrm{EMC_M}(Parent)$) and "child" ($EMC_M(Child)$). Depending on the type of hierarchical relationship (AG or GE) appropriate dimension (i or j) of the object of relationship is corrected.
- The attributes of the TDM objects (e.g. EMC_M) and relationships (e.g. Rel_M) are assigned values from class and sequence diagram elements: $EMC_M = M_Z$, $EMC_M(AG) = I$, $EMC_M(GE) = J$, $EMC_M(T) = M$, $Rel_M(Object) = EMC_M$, $Rel_M(Subject) = M_Z(Seq\_prev)$, $Rel_M(Object\_multi) = R_X(Object\_multi)$, $Rel_M(Subject\_multi) = R_X(Subject\_multi)$, $Rel_M(Type) = Rel_X(Type)$.

In this section the conditions of syntactic quality of process model and the composition process of the task-dialog model were specified. The process model should be consistent and complete against the rules that specify sufficient conditions of the process model transformation to the task-dialog model. So, the task-dialog model is composed in the space of processes from the syntactic correct process model according the predefined transformation rules.

The way of applying these transformation rules is presented in the next section as a case study of an ordering system.

5. Case Study: Ordering System

The case study of the ordering system is used to show the composition process of task-dialog model using process model.

Suppose one UI function of ordering system (new order from customer) should be modelled. The static structure (class diagram) consists of 5 classes. General information from class *Person* (it describes the information of all private customers) and class *Company* (it describes the information of all business customers) are generalised in a class *Customer*. This class is associated to class *Order* (it describes order information) through binary relationship, when each order must have a customer, who made it, but not each customer must have order made. Each order must have at least one order row (class *OrderRow*). An example of order system class diagram is depicted in Figure 6.

The new order-processing scenario starts from *Order.NewOrder* function call. This function can't be completed without the customer's data, so function *Customer.NewCustomer* is called consequently. From the class diagram (see Figure 7) follows that class *Customer* is a parent class and the class instance Customer must have either *Person* or *Company* instance. That's why either *Person.NewPerson* or *Company.NewCompany* is called to collect customers' information. The instance *Order* also is not complete without order rows (it follows from the class diagram), so the function *OrderRow.NewRow* is called for all new order rows. An example of order system sequence diagram is depicted in Figure 7.

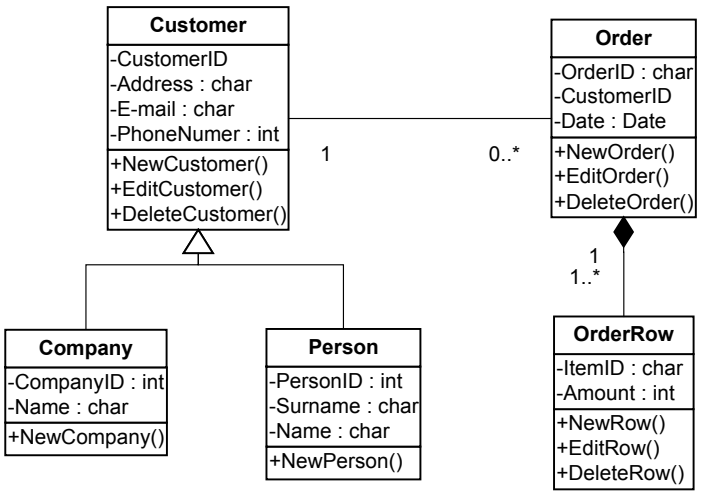


Figure 7. Class diagram of ordering system

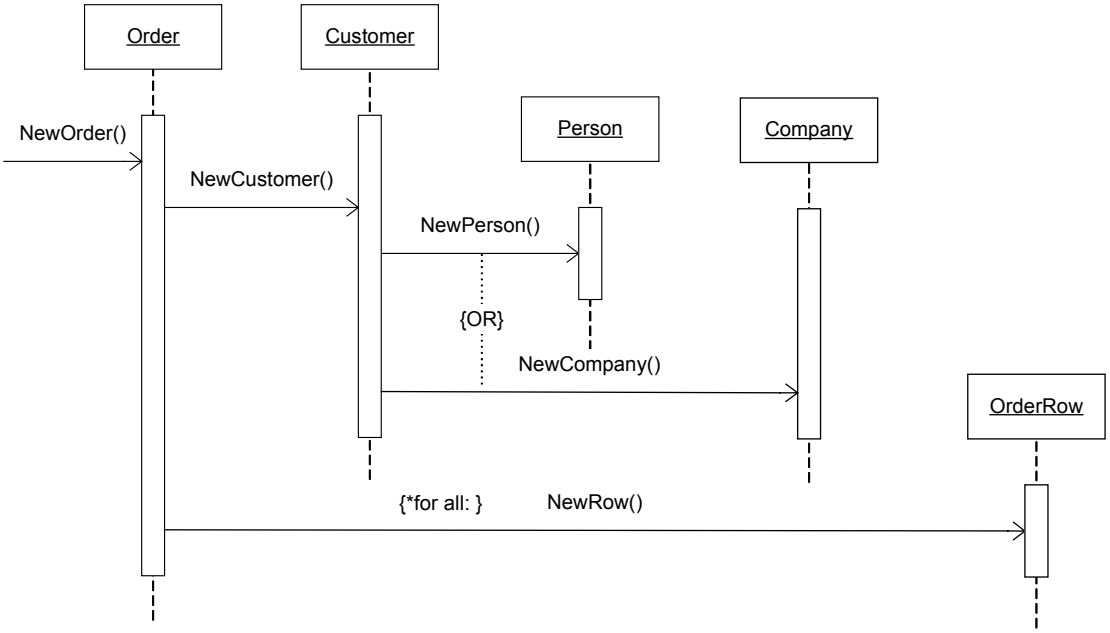


Figure 8. Sequence diagram "new order and new customer"

The task-dialog model composition starts from the first message in the sequence diagram, i.e *Order.NewOrder*. This function becomes the first EMC in the space of processes with the following attributes: $EMC_1 = Order.NewOrder()$, $EMC_1(AG) = 1$, $EMC_1(GE) = 1$ and $EMC_1(T) = 1$.

Function *Customer.NewCustomer* is transformed to the second *EMC* in the space of processes with the following attributes: EMC_2 = *Customer.NewCustomer*(), $EMC_2(AG) = 1$, $EMC_2(GE) = 1$ and $EMC_2(T) = 2$. Also the first relationship between EMC_1 and EMC_2 is created with the following attributes: $Rel_1(Object) = EMC_2$, $Rel_1(Subject) = EMC_1$, $Rel_1(Object\_multi) = \{0..*\}$, $Rel_1(Subject\_multi) = \{1\}$ and $Rel_1(Type) = \{Bin\}$.

The functions *Person.NewPerson* and *Company.NewCompany* form a sequence of *EMC*s $EMC_3 \in \{Person.NewPerson(), Company.NewCompany()\}$, $EMC_3(AG) = 1$, $EMC_3(GE) = 2$ and $EMC_3(T) = 3$. Appropriate relationship is created also: $Rel_2(Object) = EMC_3$, $Rel_2(Subject) = EMC_2$, $Rel_2(Object\_multi) = \{1\}$, $Rel_2(Subject\_multi) = \{1\}$ and $Rel_2(Type) = \{GE\}$.

The last function in the sequence diagram is *OrderRow.NewRow*, which can be repeated more then one time, depending on the number of order rows in the order. This function becomes the last EMC in the space of processes with the following attributes: $EMC_3 = OrderRow.NewRow()$, $EMC_4(AG) = 2$, $EMC_4(GE) = 2$ and $EMC_4(T) = 3$. Appropriate relationship is created: $Rel_3(Object) = EMC_4$, $Rel_3(Subject) = EMC_3$, $Rel_3(Object\_multi) = \{1..*\}$, $Rel_3(Subject\_multi) = \{1\}$ and $Rel_3(Type) = \{AG\}$.

The task-dialog model in the space of processes is depicted in Figure 8.

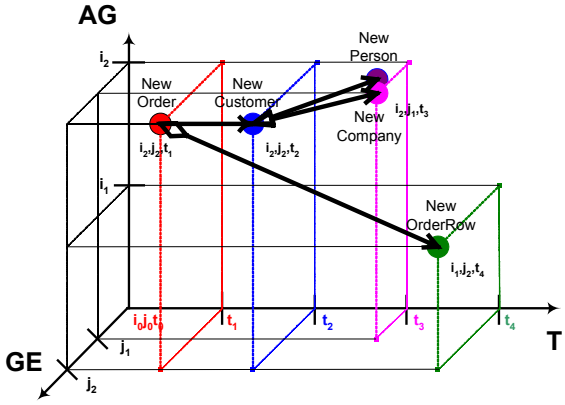


Figure 9. Task-dialog model of "new" order" function

Since the *OrderRow.NewRow* function can be repeated more then one time with the same incoming data, there exist one more specific relationship – multiply execution. It follows from the relationship multiplicity between classes *Order* and *OrderRow* in the class diagram (see Figure 7) and the type of *OrderRow.NewRow* function call "for all" in the sequence diagram (see Figure 8). The subject and object of this relationship is the same EMC_4:

$Rel_4(Object) = EMC_4$, $Rel_4(Subject) = EMC_4$, $Rel_4(Object\_multi) = \{1\}$, $Rel_4(Subject\_multi) = \{1\}$ and $Rel_4(Type) = \{Multi\}$.

The further step is the TDM verification against the rules that specify GUI specific information, e.g. backward navigation in GUI hierarchy or error correction logics. Then the TDM is transformed to the abstract and then the concrete presentation models and transferred to the software code generation.

6. Conclusions

This paper presented the composition process of the task-dialog model. The task-dialog model is composed from the process model while both get developed in object-oriented notations. The process model is specified in UML notation and task-dialog model in the space of processes. The space of process is designed to reflect hierarchical and non-hierarchical relationships between the objects of problem area, in our case, between user interface elements.

In comparison to other UI modelling methods, in this paper presented UI modelling method provides the rules of directly task-dialog model composition from process model, while both models are object-oriented. Object-oriented task-dialog model provides the following benefits and solves the problems that were defined in Section 1:

- The task-dialog model is directly associated to the process model through its generation rules, therefore the same problem area information is not modelled twice, but is reused.
- The task-dialog model is object-oriented, as it exists in the object-oriented space of processes, which represents not only hierarchical, but also non-hierarchical relations between problem area objects. Hierarchical information is considered to affect the sequence of UIs that is why it should be modelled.

The work in this paper focuses on task-dialog model composition process and prerequisites to this process. Future work includes specification of backward navigation in GUI hierarchy and error correction logics, which should enhance the quality of the task-dialog model.

References

[1] **H. Balzert.** From OOA to GUIs: The JANUS System. *Journal of Object-Oriented Programming*. ISSN 0896-8438, *SIGS Publications*, *Vol.*8(1), 1995, 43-47.

[2] **M. Barr, Ch. Wells.** Category Theory for Computing Science. *ASIN* 0133238091, *Prentice Hall*, 1990, 1-84.

[3] **M. Bergholtz, P. Jayaweera, P. Johannesson, P. Wohed.** Process Models and Business Models – a Unified Framework. *Advanced Conceptual Modeling Techniques.* ISBN: 3-540-20255-2. *Heidelberg: Springer-Verlag*, 2002, 364-377.

[4] **S. Gudas.** The Space of Processes as a Framework for Knowledge-Based Requirements Engineering. *Information Technology and control*. ISSN 1392-124X. *Kaunas: Technologija*, *No.*1(7), 1998 ,49-62.

[5] **S. Gudas.** A framework for research of information processing hierarchy in enterprise. *Mathematics and Computers in Simulation*. ISSN 03784754. *Elsevier*, *No.*33, 1991, 281-285.

[6] **S. Gudas, T. Zobakas.** The models of lifecycles for modeling activity process in organisations (in lithuanian). *Information sciences*. ISSN 1392-0561. *Vilnius University Publishing House*, *Vol.*10, 1999, 73-82.

[7] **C.A. Hurtado, A.O. Mendelzon, A.A. Vaisman.** Updating OLAP dimensions. *Data Warehousing and OLAP*. ISBN:1-58113-220-4. *New York: ACM Press*, 1999, 60-66.

[8] **A. Kleppe, J. Warmer, W. Bast.** MDA explained. The model driven architecture: practice and promise. ISBN 0-321-19442-X. *Addison-Wesley,* 2003, 1-54.

[9] **E. Schlungbaum, T. Elwert.** Modellierung von graphischen Benutzungsoberflächen im Rahmen des TADEUS-Ansatzes. *Berichte des German Chapter of the ACM*. ISBN 3-519-02686-4. *Teubner*, 1995, 331-348.

[10] **P.P. Silva.** UMLi: Integrating User Interface and Application Design. *IEEE Software*. ISSN:0740-7459. *Los Alamitos: IEEE Computer Society Press*, *Vol.*20 (4), 2003, 62-69.

[11] **P.P. Silva.** User Interface Declarative Models and Development Environments: A Survey. *Proceedings of DSV-IS* 2000. ISBN. 3-540-41663-3. *Telos: Springer Verlag*, 2001, 204-226.

[12] UI RUPture, Editorial Comment. *http:// www. uidesign.net/2000/opinion/UIRupture.html, 2001.*

[13] Unified modeling language (UML). Version 1.5. *http://www.omg.org/technology/documents/formal/uml.htm, 2003*.

[14] **T. Zobakas, S. Gudas.** The approach for object-oriented model verification (in Lithuanian). *Information sciences*. ISSN 1392-0561. *Vilnius University Publishing House*, *Vol.*23, 2002, 103-112.

[15] **D. Zobakiene.** Object-oriented process modelling using a process meta-model (in Lithuanian). *Information sciences*. ISSN 1392-0561. *Vilnius University Publishing House*, *Vol.*26, 2003, 212-217.

DOI: 10.5755/j01.itc.33.4.11958