ЮБИЛЕЙНА НАУЧНА СЕСИЯ – 30 години ФМИ
ПУ "Паисий Хилендарски", Пловдив, 3-4.11.2000

OBJECT MODELLING AND INTELLIGENT RESOLVING: FORMAL MODELS, SOFTWARE ENVIRONMENT AND APPLICATIONS

George Atanassov Totkov, Rossitza Jeliazkova Doneva

The present paper makes an analysis of the conceptual models in various subject domains. As a result it offers a categorial model of a subject domain, which can be used for its formal specification. The corresponding discrete structures model concepts like meta-area, process, language, concept, task, subject domain, etc. The proposed approach allows automatization of the process of designing and building-up of Large integrated development environments for conceptual modelling. The functionality of the environment includes possibilities for building up hierarchies of formal specifications, structures and objects, support systems and methods for automated decision planning in various SD (numerical calculations, solving of problems in the field of the algebra, chemistry, physics, plane geometry, etc.).

AMS Subject Classification: 68N30 Mathematical aspects of software engineering (specification, verification, metrics, requirements, etc.); 68N19 Other programming techniques (object-oriented, sequential, concurrent, automatic, etc.); 68T30 Knowledge representation; 68T20 Problem solving (heuristics, search strategies, etc.)

Introduction. In terms of logic the presented conceptual models [1, 6] of subject domains (SD) can fall into one of the following two types – basic (modelling of internal knowledge about the meta-domain and the computer system itself) and conceptual (external knowledge of a specific SD). According to the proposed model the meta-domain is presented in terms of a many sorted algebraic system (MAS) [7], called **basic MAS.** Every concept (or task) in a SD is specified by subset of a logical language of first order. The SD is modelled by conceptual schemes (hierarchical systems of packages – concepts and tasks, grouped thematically) in object-oriented manner. The concept (so called "abstract class") is modelled as heterogeneous discrete structure with attributes – objects of the classes built up already and with integrity constraints – relations among the attributes in the terms of a MAS, specified before. The elements of the SD – types, classes and tasks (including sorts, functions, relations) are treated as objects within one common hierarchical system. Within this approach each object can be treated as a new class (parameterisation) and it can participate in the building of the hierarchy of inheritance. The developed approach allows designing and building-up of integrated environment for conceptual modelling in various SD. The proposed system can be defined as object-oriented system, in which the inheritance *object_to_object* generalises simultaneously *class_to_class* and *class_to_instance* inheritances. An important consequence is the fact that each class – an successor of a class, already built preserves the categorial type of its predecessor. As a result the built-in typology is invariable and doesn't change during the subsequent descriptions. Most essential for the applications is the fact that the successor classes of the superclasses define new sorts, functions and predicates within the frame of the system. An important feature of the offered models is the extensibility – new classes are specified on the ba-

sis of those that already exist and the latter are modified by addition, removal and change of attributes, constraints or objects in which case the meta-type is preserved.

Formal Models. The conceptual scheme of the SD is built upon basic set of built-in types and classes including sorts, functions, and relations.

The set $T_b = \{CAT, NAME, FEAT, PROC, VAR, CONSTR\}$ of built-in meta-types (base categories) and subcategories determines the beginning of the hierarchy of the modelled types, where CAT means a subset of conceptual categories, $NAME$ – of names; $FEAT$ – of indicators (properties, characteristics and relations), $PROC$ – of the processes, VAR – of variables and $CONSTR$ – of the constructors of types.

The categories CAT are $SBJD$ – subject domain, CM – conceptual model, CSH – conceptual scheme, SSH – conceptual subscheme, $PACK$ – package, CL – abstract class, OBJ – dynamic class (object), $TASK$ – task, $COBJ$ – conceptual object, $CSTR$ – categorial structure, FM – functional model, FCM – functional calculating model.

The category $NAME$ of the names of the elements of the SD has a particular place. For every other category, the a corresponding subcategory of $NAME$ is introduced.

$FEAT$ represents a subset of built-in types of categorial characteristics and properties. $FEAT = \{DATA, FUN, PRED, ROLE\}$, where $DATA$, FUN, $PRED$ and $ROLE$ are the corresponding subcategories for the subjective, functional, predicate and role characteristics. All of the successor of $DATA$ are abstract classes of the modelled SD. An immediate successor of $PRED$ is a type Eq (a relation of equivalence in $DATA$). The type Eq is ultimate predecessor of the hierarchy of the predicates of the equivalence, automatically specified for each representative of $DATA$ subcategories. On its part, FUN gives rise to hierarchy with 12 successors – the functional subtypes $f_1, f_2, \ldots, f_{12}$, everyone of which modelled by an object with specified structure and methods of management of the interface protocol. Each functional category is implemented by an object with characteristics priority, commutativity, associativity, etc. For every concrete SD the basic MAS is expanded if necessary and is denoted with $FEAT_{sd}$.

$ROLE$ is a special category modelling the so-called roles of the elements in the SD. For the expression of the semantic relations among the elements of the constructional objects and the objects themselves in the set of names $NAME$, the subcategory "names of the roles"is introduced.

The fourth metacategory $PROC$ is presented by two categories – P_{SYS} (for models of various, internal system processes) and P_{METH} (for methods with subcategories PM and GM – private and general methods). The internal processes model separate periods of the life cycle of the representatives of the categories. They are related to the quantitative transformations from one category to another. The separation of the processes has rather a methodical meaning – a number of processes can be included in the MAS signature, and the categories of the corresponding arguments can be added to the set of sorts S and interpreted appropriately. Important places within the hierarchy beginning with P_{SYS} have some "internal" processes, participating in the environment organization and management as printing, converting, etc.

The elements of P_{METH} are used for automatic synthesis of solutions for the specified problem (element of the $TASK$ category). The methods of the type PM are applied to the internal model of the problem (from the FM category). The PM methods are implanted into the in-

ternal model of the problem and in this way the functional-computational net (*FCM* category) is created. The semantics of the nodes is modelled by dynamic recalculation of the list of corresponding local characteristics according to their type. That list is used especially for designing and carrying out of the global strategy with heuristic considerations.

The category $VAR = \{V_x : x \in SUBJ = CAT \cup FEAT_{sd}\}$ is presented by $SUBJ$ – an indexed family of categories.

The elements of *CONSTR* are operations over elements of other categories for constructing new ones. Basic constructors are **set**, **sequence** and **name**. In this way new concepts, tasks, topics and inheritance graph etc. could be specified. The classes and tasks connected in the *a_kind_of* hierarchy are grouped in packages. Packages are organized in hierarchies and build-up the conceptual scheme of the SD.

Let SD be the set of all elements of the modelled SD, and x be its arbitrary element. x could be identify and revealed as individual concept in the subject domain by the roles played by other *SD* elements in the x life cycle and in the processes participated by x. Let $pr(x)$ be the set of all *SD* elements playing any role related to x as a concept. If $pr(x) = \varnothing$, then x will be called an **elementary element**, otherwise – compound element. In the first case x is a representative of a determinate category of T_b.

The set of all elements of SD that have identical roles r in relation to x will be denoted with $pr_r(x)$. As a result, the categorial structure $cstr(x) = \{< r, z >: z = pr_r(x), z \neq \varnothing, r \in N_r\}$ is connected uniquely with every compound object x of SD.

The set of all elements quoted explicitly in the x structure will be denoted with $ref(x)$.

Let $T = D_c(N_{r,}\ T_b)$ be the set of the categorial structures over T_b that are constructed using the set of roles N_r. Then $T_c = D_c(N_{r,}\ T_b) \setminus T_b$ will be the set of compound categorial structures (types).

Definition 1. The ordered pair $x =< n, str >$ will be called **abstract class** (relating to T_b), if $ref(x) \subset T_b$. Where $n \in N_{CL} = NAME$ is **the name of the class** and $str \in D_c(N_{r,}\ T_b)$ is his **categorial structure** (denotations: $n = name(x), str = cstr(x)$).

The set of all compound classes is an interpretation of T_c. With the purpose of convenience that interpretation will be denoted with T_c again. For every abstract class x, the set of all positions in the x structure that are belong to a sort of S will be denoted with $S(x)$. $S(x)$ could be interpreted as particular set of S -sort variables. Consequently the various types of substitution from $S(x)$ into the set of terms ($Term_{sd}$) over $FEAT_{sd}$ could be considered [3].

Definition 2. Let $x =< n, str >$ be an abstract class relating to T_b and h be a substitution from $S(x)$ into $Term_{sd}$. The ordered triple $o =< n_o, x, h >$ will be called a **dynamic class** (**object**) with name $n_o \in NAME$, **substantial part** x and **generic substitution** h. The object o is successor of the class x (relating to h).

Let Obj be an arbitrary set of objects. The following denotations are introduced: $ref(o) = h$, $ref(Obj) = \bigcup_{o \in Obj} ref(o)$, $name(o) = n_o$, $spart(o) = x$, $name(Obj) = \bigcup_{o \in Obj} name(o)$ and for categorial structure: $cstr(o) = cstr(x) \in D_c(N_r, T_b)$.

Definition 3. The set of objects Obj will be called **compatible system of objects** if the following conditions are satisfied:

i) $|Obj| < \infty$ (finiteness);

ii) $o_1, o_2 \in Obj, name(o_1) = name(o_2) \rightarrow o_1 = o_2$ (uniqueness);

iii) $o_1, o_2 \in Obj, name(y_1) = name(y_2), y_i \in ref(o_i), i = 1, 2 \rightarrow y_1 = y_2$ (completeness);

iv) $ref(Obj) \subseteq name(Obj) \cup T_b$ (connectivity);

v) $o \in Obj \rightarrow \{o\} \cap ref(o) = \varnothing$ (be acyclic).

Definition 4. The **task** is an ordered pair $Task =< n, Obj >$, where $n \in NAME$ and Obj is a compatible system of objects.

Denotations: $n = name(Task)$, $Obj = cstr(Task)$.

Definition 5. The **conceptual object** is a task or an abstract class.

Definition 6. The **package** $Pack$ is a set of conceptual objects that satisfies the conditions:

i) $|Pack| < \infty$ (finiteness);

ii) $c_1, c_2 \in Pack, name(c_1) = name(c_2) \rightarrow c_1 = c_2$ (uniqueness).

Definition 7. The sequence of packages $PS = seq(P_1, P_2, \ldots)$ will be called **conceptual subscheme** if the following conditions are satisfied:

i) $|PS| < \infty$ (finiteness);

ii) $c_1, c_2 \in PS, name(c_1) = name(c_2) \rightarrow c_1 = c_2$ (uniqueness);

iii) $c \in P_i, P_i \in PS \rightarrow \{c\} \cap ref(c) = \varnothing$ (be acyclic);

iv) $c \in P_i, P_i \in PS, x \in ref(c) \rightarrow x \in P_j$ for certain $j > i$ (consistency).

Definition 8. Let an partial ordering $\prec$ to be settled in the set of packages P_{csh}. The pair $csh =< P_{csh}, \prec >$ will be called **conceptual scheme** if the following conditions are satisfied:

i) $|P_{csh}| < \infty$ (finiteness);

ii) $c_i \in P_i, P_i \in P_{csh}$ $(i = 1, 2), P_1 \prec P_2, name(c_1) = name(c_2) \rightarrow c_1 = c_2$ (uniqueness);

iii) $c \in P, P \in P_{csh} \rightarrow \{c\} \cap ref(c) = \varnothing$ (be acyclic).

iv) $c \in P_i, P_i \in P_{csh}, x \in ref(c) \rightarrow \exists P_j \in P_{csh}, x \in P_j, P_i \prec P_j$ (consistency of relation $\prec$ with the class inheritance).

Definition 9. Every set of conceptual schemes is a **subject domain**.

On the basis of the above theoretical study of categories and objects in a various SD the corresponding model of large integrated development environment for conceptual modelling of SD [3] is offered.

Software Environment. A proper architecture [3] of the computer environment for designing, constructing and development of intelligent object-oriented systems able to support knowledge-base and automatic synthesis of solutions is proposed as consequence. The model

of the environment consists of 2 parts – **kernel** and **superstructure**. Models of meta-knowledge (knowledge, independent from the applications) such as an abstract MAS, a task-solver interpreting the decision planning system, a processor for supporting conceptual schemes of SD and an adaptive interface has been realized in the **kernel**. The **superstructure** contains models of the meta-domain and of particular methods for problem solving, a conceptual model of the SD etc.

An esential characterristic of the environment is the in-built ability for **spiral iterative development** in the course of the time of every application designed and supported by the means of the environment. The latter in particular means that the model and the architecture of each next application is a development (in time, resources, on another level) of some previous application.

A methodology which is offered for building up of applications not only enables the creation of independent applications (diversified SD) on common base – a specific base MAS but on the vice versa lets the specifications of a SD be used with diversified base MAS. Another important feature is the ability for automatic synthesis of solutions for tasks, specified as classes in the SD. As a result there is an advantageous side effect of the adopted approach – the adaptivity of input-output interface to different SDs.

The architecture of the software environment (Table 1) can conventionally be treated as 3-level entity – **application**, **meta-expert** and **base** level, which on their part are subdivided into 7 sublevels – **users**, **conceptual**, **expert**, **interface**, **logical**, **physical** and **system**.

The models of the **application level** reflect the outlooks of the final user of a SD, the **meta-expert models** specify the knowledge-base for the SD and the meta-domain considered, and the **base models** realize logical and physical structures providing effective management of the knowledge-base.

The design, construction and development of each level or sublevel is carried out according to the object-oriented principle.

The **base model** is the base of effective construction and management of the knowledge in the SD. It consists of 3 sub-levels – **system** (closely related to the computer environment), **phisical** (supporting internal presentation of the used data structures) and **logical** (realizing the offered formal models).

LEVELS		**SUB-LEVELS**	**OBJECT-ORIENTED ENVIRONMENTS**				**CLASSES**	**SUP-PORTED**
s u p e r .	APPLI-CATION	USER	CLASSES TASKS	CLASSES TASKS	…	…	^Class ^Problem	Classes and tasks specifi-cations of SD
		CONCEP CEP-TUAL	SUBJECT DOMAIN	SUBJECT DOMAIN	…	…	^Class ^Problem ^Package ^Schema	Conceptual schemes and automated problems solving in SD

<table>
<tr><td rowspan="2">. . . struktur e</td><td rowspan="2">META-EXPERT</td><td>EXPERT</td><td>MSS</td><td colspan="4">SOLVER</td><td rowspan="2">X_Kind
X_Oper
X_ProcNode
X_ProcNet
X_CompModel</td><td rowspan="2">Meta-environment
X Solver</td></tr>
<tr><td>INTER FACE</td><td>LANGUAGE</td><td colspan="2">METHODOLOGY
DIALOG
EDITOR</td><td>HELP</td><td>METHODS
…</td></tr>
<tr><td rowspan="3">KERNEL</td><td rowspan="3">BASE</td><td>LOGI-CAL</td><td colspan="5">LOGICAL MODELS</td><td>Kind Oper SKnot SKnotDesSort SNet ProcNode ProcNet CompModel Schema Editor Dialog</td><td>Formal model Structures Abstract MSS Common problem solver Universal interface</td></tr>
<tr><td>PHYSI-CAL</td><td colspan="5">PHYSICAL MODELS</td><td>PList NetNode Net</td><td>Internal repre-sentation of data structures</td></tr>
<tr><td>SYSTEM</td><td colspan="5">Borland Delphi 5.0 WINDOWS’98</td><td>Class</td><td>Object-oriented software envi-ronment</td></tr>
</table>

Table 1. Architecture of Software Environment for Conceptual Modelling

The choice of a **system sublevel** defines to a great extent the characteristics of the environment. A highly developed and effective object-oriented environment for programming and an appropriate operation system have been chosen in this case.

The **physical sublevel** supports the physical models of the abstract data structures. Due to the high degree of abstraction of the models building up the next (logical) level, classical dynamic structures are preferred here – structures as **PList**, **NetNode**, **Net**, etc.

On the **logical sublevel** the experts and the users ideas are integrated into one schema which is independent from the specific SD, the knowledge-base management system and the selected internal presentation. An abstract MAS (**Kind**, **Oper**), a task solver (**CompModel)** able to support an internal functional model (**SNet**, **SKnot**, **SKnotDesSort**) and methodics for problem solving (**ProcNode**, **ProcNet**) of tasks, universal interface (**Editor**, **Dialog**) and a processor for supporting conceptual schemes of SD (**Schema**) are modelled here, based on object-oriented principle (by introduction of appropriate classes). Important elements of this level are the linguistic component and the global strategy (as a functional characteristic of **CompModel**) for solving problems in the SD. The programming tools built up on the base level make up the kernel of the system (**Shell**).

Within the **meta-expert level** unambiguously and uncontradictorily are integrated the views and the ideas of different users and experts about the meta-domain considered (a set of SDs, admitting formalising on the base of one MAS a common methodics for task-solving and unified interface).

The **expert sublevel** models the meta-knowledge (common concepts, hierarchy and classifications), disregarding the particular concepts, methods and facts of a specific SD. An object-oriented environment, supporting knowledge of the chosen base MAS (**X_Kind**, **X_Oper**) is build on this level by specification of logical models. It also supports methodics and methods about automatic synthesis of task-solutions in the SD (**X_ProcNode**, **X_ProcNet**, **X_CompModel**).

On the **interface sublevel** the input/output system of the meta-environment is specified. It is build up as a typical system application. The adapted approach allows quick adaptation of the environment to various applications. The designed and realized interface is used by all SDs on the next application level. The specified classes here (**X_Editor**, **X_Dialog**, **X_Help**) add to the object-oriented environment for conceptual specifications of the expert level with additional abilities for dialog and communication with the final user.

The software tools from the base level and the realization of the specific meta-environment make up a **primary application** (marked in Table 1 with **X**), which represents object-oriented environment for conceptual modeling with automatic synthesis of solutions in the meta-domain, modeled here.

Applications. The model of the **application level** is built up on the basis of the concept conceptual scheme (Def. 8), by the tools of the specified environment **X**. The conceptual scheme of a specified SD, together with the specific meta-environment, in which it is realized, is called **secondary application**. The secondary application maintains users' requests for task-solutions (**tasks)** and expert knowledge (**packages**) for grouping concepts (**objects, classes, schemes**). The automatic synthesis of solutions in the specific SD is carried out by the decision planning system in the meta-domain built-up on the previous architecture level.

The application level consists of 2 sublevels – **conceptual** and **user's**. On the **conceptual sublevel** concepts, methods, and conceptual schemes of a specific SD are modelled. The conceptual schemes of SD can be built hierarchically from subareas called packages, containing hierarchy of classes and tasks presenting essential knowledge of concepts and the relations among them. The conceptual scheme and its elements are presented by objects from the corresponding classes, built up on the logical level (**Scheme**, **Package**, **Class**, **Problem**).

On **user's sublevel**, every final user can realize his own model of knowledge on some part of SD adding to it and developing the already existing conceptual scheme in the system or creating new ones.

The architectural layers of the environment allow the development and support of diversified applications. Some of these are typical instrumental computer environments for automatic specifications and synthesis of task-solving in a SD, and others – dialog systems for management of conceptual knowledge-bases of a specific type.

As a result it is possible to build up applications in specifically various SD by an upgrade of separate levels of the system until a primary or secondary applications are obtained. The created primary applications are meta-environments for calculations with real numbers, solving of chemical [5] and plane-geometrical tasks [4]. Specific conceptual schemes for calculation with great precision are realized into them, together with solving of tasks in stereometry, kinematics, chemistry, etc. [2].

Conclusion. The system has the following characteristic features, abilities and peculiarities:

a) automated building up of a base level –a model interpretation of the signature of the base MAS

b) specification of the concepts (classes) and tasks in a selected SD with the help of a special language

c) object-oriented specification of diversified SD on the basic of MAS as hierarchy of packages, classes and tasks

d) in-built context-sensitive help system

e) adaptivity of the I/O interface to diversified SD and meta-domains (according to the basic MAS)

f) automatic translation of specifications of the classes/tasks to internal system presentation

g) automatic synthesis of task-solutions within the modeled SD

h) application for calculation and automatic training, etc.

The characteristics a), d), e) and f) are invariant in terms of diversified superstructure, i. e. they are functions of the base level of architecture; b), c) and g) are functions of a specific meta-expert level, where h) is a properly of the application level which has been built-up.

REFERENCES

1. Brodie M. I., J. Mylopoulos, J.W. Schmidt, Conceptual Modelling, Springer Verlag, N.Y., USA, 1984.
2. Doneva R., Totkov G., OMIR – An Object-Oriented Tool for Conceptual Modelling and Automatic Solving of Secondary School Problems, Pre-Conf. Proc. of the IFIP WG 3.1/3.5 Open Conference "Informatics and Changes in Learning", Gmunden, Austria, Jun 7-11, 1993, p. 3-5.
3. Doneva R., Automated Construction of Intelligent Object-Oriented Environments for Conceptual Modelling, Synopsis of the Doctoral Dissertation, Sofia, 1994.
4. Doneva R., Computer plane-geometry in object environment, I-st National Conference INFORMATICS'94, Sofia, november 8-10, 1994, p. 98-104.
5. Doneva R., Resolving chemical computational problems in object-oriented environment, Mathematics and Education in Mathematics, XXVI Spring Conference of the UBM, 1997, p. 244-249.
6. Tyugu M., Knowledge Based Programming, Addison-Wesley, 1988.
7. Wirsing M., Bergstra J. A. (eds.), Algebraic Methods: Theory, Tools and Applications, 394 LNCS, Springer Verlag, 1990.

George Atanassov Totkov
Plovdiv University, 24 Tzar Assen Str.
BG-4000 Plovdiv, BULGARIA
tel. 268636, e-mail: totkov@pu.acad.bg

Rossitza Jeliazkova Doneva
Plovdiv University, 24 Tzar Assen Str.
BG-4000 Plovdiv, BULGARIA
e-mail: rosi@pu.acad.bg