11

Information Systems Framework Synthesis on the Base of a Logical Inference

Evgeny Cherkashin, Vyacheslav Paramonov,
Roman Fedorov, Igor Bychkov
Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences, Irkutsk National Research Technical University, Irkutsk Russia

1. Introduction

Any software development life cycle consists of distinct stages and involves various formal and informal models, agents (designers, software developers, users, etc.) and technologies. Various combinations of stages form a number of software development life cycle schemes, such as waterfall and spiral models, V-model, agile and extreme approaches, iterative and incremental development, and various improvement models [1]. All the approaches make use of models of various degrees of abstraction and formalization. We consider a general case of the life cycles as a process of adaptation of new ideas, requirements and specifications, where each adaptation event is a model change. The adaptation implies modification of all the affected models due to the event. Thus, the software development process is represented as propagation of the differences (modifications) within a set of models representing the software under development. The target of the research is to construct an approach to describe the process of the difference propagation as a basis of a corresponding instrumental environment for software development.

Since 2001 OMG exploits Model Driven Architecture (MDA) of software development. MDA [2] is a part of the model considered in the section III. MDA exploits three levels of abstractions to represent software: CIM, PIM and PSM. The Computation Independent Model (CIM) reflects softwareâĂŹs external requirements âĂŞ its interfaces. CIM hides structural elements, and can be used for define specifications and checking requirements.

The software designing technique of MDA is based on multistage transformation of Platform Independent Model (PIM) into a number of Platform Specific Models (PSM). PIM is a model of the software reflecting most of the structural and some semantic aspects of the software, but the model contains no information about implementation of the structures on the target program architecture. UML Class Diagram extended with some tag values and additional stereotypes is an example of PIM [3]. The extension allows one to denote implementation variants and hints for structures. PSM is a model, which canimplemented as source code of the subsystems, e.g.,could be a physical structure of a rational database, which is directly (deductively or by means of code templates) translated into DDL SQL-requests.

The transformation of the PIM into PSMs is carried out under control of a Platform Model (PM) and a transformation scenario. PM contains information and algorithms of PIMâĂŹs

structure analysis and generation of corresponding structures in PSMs. Sometimes PSM is understood as specified variant of PIM. The tag values and stereotypes are used to direct the transformation of a structure into desired frame.
Main advantages of MDA usage in the software development are as follows:

- Design stage independence of the implementation platform; capability to replace the platform without redesigning PIM.
- Formal definition of PM: programmersâĂŹ knowledge is represented as rules and algorithms.
- Raising the automation level of the life cycle: early stage modifications (design stages) are less expensive to implement in PSMs. MDA is a great approach and successfully used in development complex software, but it has significant disadvantage, which we are to overcome:
- Using the MDA in simple projects usually extends time of software construction, although obtained formal PIM and PM models when analyzed could be used in other projects;
- Currently MDA is of little use in already constructed and implemented systems and systems based on stored data manipulation, e.g., existing informational systems, as modification of information data model results in database structure modification like adaptation to new data structures; Modification of PIM and source code is ignored by the procedures of transformations.

At present (2005...??) PMs in most of commercial MDA systems have been implemented on the basis of algorithmic approach. They are not far from CASE systems translating UML diagrams into a source code by various plug-ins. The main idea of MDA is to allow developer to modify PM according his/her preferences and task properties. Our experience shows that usage of present logical languages and PMs based on formalized knowledge [3] allows us to affect the transformation in an efficient way by means of changing a rule set content. Moreover, declarative paradigm of the logical languages naturally forces programmer to create rules processing only small parts of PIM in a multistage fashion with a number of intermediate decisions. This results in dividing the transformation process on stages, where each next stage deals with more concrete structures, which are nearer to PSM. Such approach looks naturally similar to above described application of the theory.
problematic of local firm traditions and open source.

2. MDA today

General classification XMLTRANSFORM, graph conversion (Paramonov disser) SQL-like pattern recodgniniton (kuzmin)
ATL
Stratego

3. Architecture implication

Enterprise information systems (IS) roughly can be divided to three main subsystems: data warehouse, business logics (also referred as business intelligence and sometimes also middleware) and user interface. The design and implementation techniques of the IS are almost always based on this kind division.

Data warehouse is realized as Ãă rational database with a query language, Đţ.g., SQL. Business logics is responsible to support object-relational layer and define object interaction at runtime. User interface is used to modify data in the data warehouse, control the process of the IS functioning and initiate processes of solution of specific tasks such as data-analysis procedures and report generation. The division is principally compatible with standard MVC-scheme (Model-View-Controller). Data warehouse is Ãă Model. User interface is Ãă View. Business logics is Ãă Controller.

The design approaches to realize an the IS can be divided to two categories. The first category is a construction of the IS on the base of some universal library of modules. This approach embodies famous enterprise IS development platform SAP R/3 [1]. The second category is Ãă construction of the IS from the ground using Ãă CASE-system, Đţ.g., Rational Rose. The RAD-systems (Rapid Application Development) supposed to belong to the first category.

First approach can be seen as Ãă process of experience accumulation in various IS manufacturing and consequent implementation of library modules with Ãă reasonable level of abstraction. Each module implements Ãă class of operations or business processes, and can be adjusted to the peculiarities of the business processes of Ãă concrete enterprise. In this case the IS is Ãă composition of previously specified modules implementing Ãă system of business processes Đ¿ĐźĐżĐţ enterprise.

Second approach is to use some abstract modeling language to express Ãă design of the system on Ãă humanunderstandable level, and generate the source code of the IS according to the design. The UML language **?** is widely accepted standard of visual model formal representation.

Most of present CASE tools contain the subsystems of code generation. The UML language is Ãă general design description language and has no special features oriented on enterprise IS description, Đţ.g., some many"=to-one associations are relations of an instance attribute to its list of values from Ãă reference book. In this case Ãă special choosing widget can be used in Ãă user interface to fill in the attribute with value from the list. Code generators incorporated in the CASE"=tools will generate source code of SQL database definition script and Java classes. The database and Java classes will not interact without modules implementing business logics, coupling the database and classes' instances. User interface usually generated as Ãă result of Ãă database definition script analysis.

Another fundamental problem is the database structure modification in IS administration, which results in necessity to find and modify source code of business logics and user interface. It seems, that the problem cannot be solved with present CASE"=tools.

In late 2001 OMG (Object Management Group) proposed MDA (Model Driven Architecture) **?**. Software development in the MDA starts with Ãă construction of Platform"=Independent Model (PIM) of an application's business functionality and behavior, typically built in the 11ĐIJ1 language. This model remains stable as technology evolves. MDA development tools, available now from many vendors, convert the PIM first to Ãă Platform"=Specific Model (PSM) and then to Ãă working implementation on virtually any middleware platform: Web Services, XML/SOAP, EJB, ĐąâĎŰ/.Net, etc. Portability and interoperability are built into the architecture. OMG's industry"=standard modeling specifications support the MDA: UML; the MetaObject Facility (MOF); the Common Warehouse Metamodel (CWM); and ĐěĐIJĐň Metadata Interchange (XMI). OMG Task Forces organized around industries including Finance, Manufacturing, Biotechnology, Space technology, and others use the MDA to standardize facilities in their domains.

The ann of our research is to construct Ãă generator software, which can generate Ãă kernel of the IS on the base of an MDA. The generation is carried out by means sequential specification of the PIM, resulting PSM and, then, the source code generation. To approach the aim an enrichment of UML is considered. The enrichment does not require to change the UML standards. MDA"=model of the IS of consists of UML Class diagrams enriched by stereotypes and tag values, PIM"=PSM conversion knowledge base formalized as Prolog language rules, and constraints on attribute values and associations represented with the OCL"=language. Other kinds of diagrams are not used on the present stage of this research.
Hereafter we will use the following point of view to the structure of the IS. The IS is constructed of the following program aspects: data warehousing, business"=logics and user interface. The aim is to generate all the program aspects from one PIM model simultaneously so that there were no functional contradictions among their source codes.

4. Model of IS

The proposed PIM of the IS has two parts: model MS of component structures and the associations and model ĐIJĐą of constraints. The PIM"=PSM conversion is defined by model MI of semantic interpretation of the PIM components and associations in PSM.

4.1 Model of component structures and associations

Model MS is Ãă structure aspect of the IS. We suppose that IS is implemented in object"=oriented programming languages or, at least, in languages supporting some kind of objects (abstract data types) in sense of data encapsulation, Đţ.g., structure programming. MS defines the attribute structure and operations on the objects. Objects have associations with other objects, and the association also defined in MS. The Class diagram of the UML"=language is Ãă way for MS formalization. According to UML specification the objects are represented by Class classifier, which, in general, define Đ£ĐřÑĆĐţ of the object, list of its attributes, and operations on the object. UMLassociations define the relation between objects.
Class diagram can be used to define object structure for almost any kind of software, but in our case the common techniques of the IS design as an enterprise IS should be accounted, Đţ.g., attribute data are usually stored in a relational database. Last means that some additional information associated with the attribute, like attribute Đ£ĐřÑĆĐţ of Ãă table, where the attribute will be stored. The UML"=language supports Ãă possibility to associate the additional infoimation with any element of Class diagram as well as to define new kinds of classifiers. In the first case tag values are used, Đţ.g., tag value display name can associate string "First name" with attribute first name of class Person. In the second case ĐłÑČ means of stereotypes one can define some peculiar properties on Classes, Đţ.g., it can be pointed out that the attributes of Person instances are stored in PERSON table of Ãă rational database. Thus, tag values and stereotypes are used to extend Class diagram with specific information about the IS as Ãă special case of programming system.

4.2 Semantic interpretation of components and associations

Generated source code of program aspects is understood as the semantic interpretation. The semantic interpretation is implemented by means of knowledge base and source code templates. Knowledge base contains formalized qualitative characteristics of relations between structural compositions of Class diagram elements, as well as associations to source code templates in the context of an environment defined with tag values and stereotypes and target

implementation platform. In the process of the compositions recognition some intermediate conclusions are done, specifying the PSM.
Let us consider an example. For Ãă Class marked as «abstract» stereotype usually it is not necessary to generate SQL statement of table definition, as for abstract class has no instances. The SQL's PSM will not contain this class. All the attributes of the class are inherited by descendants, i.e., moved to corresponding descendant table definitions. ĐŘ SQL view is generated to gather all the descendant class instances for Ãă possible query. Thus, according to Class diagram the generating system creates an abstract class for business"=logics program aspect (business logics PSM has an reflection of the class), but creates no corresponding ĐŚĐŕĐň4ĐřĐńĐţ definition. Thus, when Ãă composition of elements has been recognized one of two actions can happen: Ãă fact derivation, which is stored for further usage; source template filling in derived term values and resulting source code generation.

4.3 OCL constraints

Model of constraints is an important part of the IS's PIM model. The following program components are examples of constraint implementations: atomic data types, value ranges, input patterns, database triggers, CONSTRAIN definitions in nowadays SQL. Some elements of Class diagrams can have soundness check constraints, which should be checked by database engine in the case of table modification. UML specification contains the ĐđĐą1 language (Object Constraint Language) specification. The OCL language is useful to define constrains formally. Defined constrains are translated to their source code implementations in program aspects.
Consider the following example. The text "Registration time of Ãă person is earlier than departure time" could be translated to ĐđĐąĐň in some Person object context as follows:
context Person inv: self.reg time < self.dep time The ĐđĐąĐň-expression is an invariant, i.Đţ., holds always during life time of the object after construction. The expression is translated to the following source code:
- for data warehousing program aspect (relational database):
ALTER TABLE PERSON ADD CONSTRAINT REG TIME < DEP TIME
- for user interface program aspect (Đř JavaScript validator within Ãă HTML"=form):
if (!this->reg time<this->dep time) alert(...some warning text...) ;
The warning text is Ãă text associated with the constraint.
ĐđĐą1 supports language structures which can be translated to Ãă complex SQL"=queries. Triggers are generated by means of analyzing combinations of pos t:- and pre: -conditions and pre9"=prefixes. The ĐđĐąĐňexpression are translated in context of MI, also specifying PSM.

5. Logical MDA implementation

Figure 1 illustrates system architecture and interaction of modules of the program generation software.

Fig. 1. System architecture and interaction of modules of the program generation software.

5.1 Architecture of generation software and its basic principles of functioning

The UML Class diagram is defined by visual UML editor, Đţ.g., PoseidonCE **?**. XMI"=translator is Ãă standard DOM2"=compliant translator, we use Ãă Python DOM2 parsing library (ÑĂÑČĐěĐIJĐň"=0.8
The IS PIM Model as object net is conducted by XMIParser.py module opensource library

ArchGenXML **?**, during the conversion OCL"=expressions are extracted and parsed into Ãă syntax parsing trees. The obtained object net is converted to Ãă set of atom predicates of the Prolog language. Thus, the Class diagram during code generation is represented by means of three data formats.
The diagram is sent to the top of Ãă generation module net. Each module recognizes some aspect of IS's design and makes deductions as PSM specification or generates a piece of source code as the constructed PSM implementation. The topmost module make general decisions like whether to consider an attribute as possessed by an object, converts object structure in general. The obtained decisions transferred to other generator modules.
The resulting source code is generated by the lowest modules. Each generator module uses its set of rules, which are stored in the module, so the knowledge base is distributed among the modules.

5.2 Generation module implementation

Each module consists Đ¿Đź ŇĞĐ¿ parts: recognition template and action. Recognition template is Ãă Prolog rule or set of Prolog rules. If Ãă query of the rule is successful for a number of objects, the action parametrized by the objects is initiated. Action can add Ãă fact to the general fact list or generate Ãă piece of source code.
Generation modules are realized as Python language **?** instance methods of Ãă class. The Prolog rules are written in so called docstrings of the methods. The docstrings are defined by special Python syntax constructions and accessible as doc -attribute of methods at runtime. Query to Ãă generator is done by an wide"=class API"=method query, which is Ãă generator. The set (tuple) of objects, on which the query is successful, can be also caught by a for statement. So, for the successful query not only module action is initiated, but some action defined by the f or"=statement.
Consider an example of module construction for generation Ãă SQL"=table for Ãă nonabstract class.
class Ru1esMixing: âĎŰ ruleset for SQL table generation def rule generating class(se1f, c1s): âĎŰ the class attributes are stored in Ãă table, âĎŰ if the class is not abstract, Âź marked by ÂńOODBÂż stereotype element (C1s, ' Class '), 3+stereotype(C1s, 'abstract') ŇŔtereotype(Cls 'OQDB'), 3+internal only(Cls). def rule c1ass arent(se1f, c1s, parent) Âź Parent is one of the nonabstract class arent(C1s, Gparent):Ãćâ Ćňâ Ăİ âĎŰ this two modules have no acti lass SQLTranslator(Translator, Ru1esMixing) âĎŰ SQL-script generation module set
def genClass(self, cls) answer. append(attribs) âĎŰ SQL-script generator for class cls answer = (l âĎŰ the source code is Ãă list of lines if cls in self. generated: âĎŰ if the class is already generated return answer âĎŰ do nothing for , parent in self. query('c1Ãăss ŇĂĐřĐşĐţĐ£ ' (cls, 'âĎŰ')): âĎŰ generate all nonabstract parents name=self.getName(cls) âĎŰ class id at.tribs = self.genSchema(cls) âĎŰ generate Ãăttributes of class answer. append("CREATE TABLE name) answer. append(")self.getTableType(cls)) self.addFact("oodb table('(cls getId(), name)) âĎŰ adding the fact of reflection of âĎŰ the attribute to âĎŰ Ãă corresponding SQL table field self. generated. append(cls) âĎŰ mark the class as already generated return answer âĎŰ return the source code

6. Logical inference

Fig. 2. Test figure

Table 1. Test Table

7. Theoretical considerations

MDA as an intelligent configurator for generative programming
Component Architectures

8. Alternative approaches

Generic programming (Julia, D, Rust)

9. Conclusion

An approach to implement code generators within OMG's MDA architecture is considered for Ãă special class of enterprise information systems. ĐŘ technique and software for generation of sound source code of main subsystems from one formal MDA model is developed. Under the soundness we consider the appropriate reflection of an element of UML Class diagram (Đř part of the model) in the generated subsystems.
The software is based on Ãă system of artiflicial intelligence, namely Ãă knowledge based system. The code generation result from an inference on initial set of facts, representing as Ãă specially enriched UML Class diagram. The generator system supports various levels of abstraction of the information system representation. The levels are defined by set of code generators. This allows one to transit from one implementation technology to another by means of realizing new generator subnet. On each level an intermediate (in sense of abstraction) variant of a Platform Specific Models (PSM) generated, which is specified on the lower levels. Thanks to the feature one can construct Ãă net hierarchy of PSM models, allowing one to generalize design knowledge among Ãă set of common implementation technologies.

9.1 Further pushdown

One of further directions of the project development is to involve in the modeling process other classes of UML diagrams and other OMG's standards, ĐšĐÿÑĄĐň as the MetaObject Facility (MOF), and the Common Warehouse Metamodel (CWM). The another direction is to construct PIM model version difference analysis system. On the base of the analysis, Đţ.g., Ãă SQL script can be generated, which implements Ãă transition of old database structure to new one.
Aggregation and reports

10. Future Works

Software development life cycle has been considered as subject of the theory of complex systems and complexes [4] implying that the software development is a natural process. The life cycle is represented as system of models and morphisms between them. Analysis of the theoryâĂŹs properties realization in the model shown, that the present instrumental software productivity could be extended by means of developing techniques for analysis of the passed life cycle stages, analysis and difference propagation of the models.
To support the propagation of the differences existing examples of the present source code development technologies should be adapted. Namely, known patchfile format is considered and some its modification is suggested to account properties of generated source code. An

approach to realization of the difference propagation is also briefly considered. It is shown, that the transformation technique and file formats are closed with respect to original formats of the software models representation.
One of the aims of the research is to constructsoftware development tools based on analogy. For example, having stores in a revision control systems all the states, models and stages of MDA software development as the differences, it probably be possibleconstruct new sequence of differences for new original model.
To gain some new experience about the topic a pilot project started to automate document preparation innotarial office, where users have a dominant roledeveloping the software function set.

11. Acknowledgment

The research is partially supported by the Council for grants of the President of Russian Federation, state support of the leading scientific schools, project No NSh-8081.2016.9. The presented software is developed using the Shared Computing Center of Integrated Information and Computing Network of Irkutsk Research and Educational Complex http://net.icc.ru.

12. REFS

[1] T. Curran, G. Keller, ĐŘ. 1.Đř44, SAP R/3 Business Blueprint: Understanding the Business Process Reference Model, Prentice Hall PTR, USA, 1997.
[2] G. Booch, J. Rumbaugh, 1. Jacobson, The Unified Modeling Language User Guide, 1st edition, Addison-Wesley Pub ĐąĐ¿., Boston, 1998.
[3] OMG Model Driven Architecture. http: //www.omg.org/mda/
[4] Poseidon for UML - by Gentleware, just model ĐšĐĆâĂİ ĐİĐ¿ÑĆĐţ. http: //gentleware.corn/
[5] ArchGenXML Manual - generating Archetypes using UML. http: //p lone. org/documentation/archetyp manual/
[6] Python Programming Language. http://www.python.org/
MDA2003
kuzmin
italian disser
find something new
logtalk http://logtalk.org/documentation.html A Portable and Efficient Implementation of Coinductive Logic Programming. Paulo Moura. Proceedings of the Fifteenth International Symposium on Practical Aspects of Declarative Languages (PADL 2013), 2013. Springer LNCS 7752. http://dx.doi.org/10.1007/978-3-642-45284-0_6