Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 3 (2011) 1597–1601

Procedia Computer Science

www.elsevier.com/locate/procedia

WCIT

CEM: Class executing modelling

Jaroslav Zacek[a*], Frantisek Hunka[a]

[a]*University of Ostrava, Dvorakova 7, 701 03 Ostrava 1, Czech Republic*

Abstract

This paper analysis approaches and possibilities of executive model aimed to MDA approach. The second part of the article proposes guideline to create executive model and describes basic interactions to object oriented approach. Annotations have been used for executive model object extension. Reflection concept has been used for model execution. Proposed model supports new type of extended object with enhanced metadata model as well as regular objects with no additional metadata description. According to use object with no additional description the model supports third-part components and supports reusability. The model will be applied to LFLC package developed by Institute for Research and Applications of Fuzzy Modeling, University of Ostrava.

Selection and/or peer-review under responsibility of the Guest Editor.

Keywords: MDA; metamodelling; annotations; reflection; executable model; design patterns; dynamic calling;

1. Introduction

In present days the accent is on speed and automatization of model transformation to concrete programming language besides model expressivity and domain usability. In addition elevation of abstraction should be applied to make modelling easy and simple. Main advantage of this approach is during initial analysis of application building or when user needs to automatize some processing. During key requirement identification the higher abstraction level is needed. Reducing model abstraction concretizes this initial design with transformations. Transformations are ending on source code level and model is become platform dependent. But in any time user can elevate model of abstraction to higher abstraction level and edit model on higher level. All these tasks can be done using automatized tools and changes are applied on lower source code level. This approach is very useful in agile programming methodologies and enables very fast model changes. One option is to divide models to different levels of abstraction and make a transformation between them. Model transformation process is described in [1] specifications and is known as a Model-driven architecture (MDA).

* Corresponding author. Tel.: +420-777-339-948.
E-mail address: jaroslav.zacek@gmail.com.

1877-0509
doi:10.1016/j.procs.2011.01.055

2. Problem formulation

Basic formulation of executive modelling has been described in introduction. As a context of problem we consider MDA architecture on Fig. 1. A bottom layer contains data and is an instance of M1 layer, which creates a model. There is no execution on M0 layer because M0 contains data with no context and therefore insignificant. Interaction between data is realized on M1 layer, where the classes and their relationships are described. These relationships realize method calling. Fast relationship changing is suitable for modelling. By the thought of changing relationship means change any method calling in any object in the model. Ideally user is able to change relationships and inner class attributes during simulation.

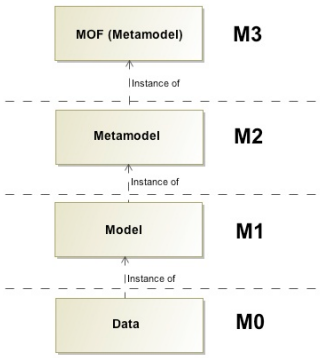

Fig. 1. MDA architecture

This execution approach is usually realized on M1 level, which is closed to platform independent model. User can examine classes and their attribute state, make a direct relationship to another class, watch the simulation progress and ideally read class values in the real time.

2.1. MDA Value proposition

Programming language is an instrument to executive model expressed in UML. This fact was considered as a disadvantage model transformation because by this transformation model becomes platform dependent on operating system or programming language syntax. Programming language lifetime is limited and when new programming language becomes in use the old source code is became useless. Presently using platform independent programming languages minimizes the risk of boundedness source code to the specific platform. Using Java technology in these days minimizes boundedness risk. Company's processes are changing and PIM must respond to these changes. MDA advantage is to preserve high-level views to solve problems - PIM.

2.2. MDA Execution

In original MDA architecture design there was no execution at all. Modelling starts at higher layer and by concretizing model and decomposing (model transformation) new code is generating. Generated code contains class skeleton. Function interactions between classes are represented by UML relationships only and class itself carries no executive information, instantiation approach or input and output methods. Main disadvantage of this approach is that model cannot use components developed before and model cannot be executed and debugged. PSM to PIM transformation can be made from class diagram (low model view), but this transformation is difficult, cannot be done automatically and for right model identification archetypes patterns [1] must be used. To make MDA architecture running automatized an Executable UML extension must be applied.

2.3. M3 Action - Model Execution Framework

M3 action, mostly known as a MXF, is a project focused to executive modelling on a higher level of abstraction (M3). A new language has been defined to describe interactions between elements [5]. Language is based on UML Actions/Activities. From executive point of view, a more abstraction view is available compare to Executable UML. MXF and Executable UML cannot change the level of abstraction and models executable model on a single layer. Metaobject instantiation is performed in M3 abstraction level, therefore tool cannot identify design pattern of implementation. Compare to UML MXF supports aspect-oriented programming due to M3 abstraction level.

3. Problem solution

MDA, Executive UML and MXF doesn't include these requirements to executive model:

- Create model form reusable components
- Concerning design patterns
- Flexible change when component is replaced
- Function and debugging with no code compilation
- Change level of abstraction

These requirements can be realized with minimal generality reduction by object metamodel extension and reflection tool application.

3.1. Reflection

Reflection as a term in information science means ability to read and change program structure and behavior during main program running. Considering to object-oriented programming approach, reflection means ability to read and change object attributes, read and execute object method, passing calling results and instantiate new objects. Generally reflection is able to read object metamodel during program running without changing any object attributes. Reflection is widely used with Smalltalk programming language and scripting languages. Reflection can be used as a universal tool to make object persistent [4] or to generate project documentation.

Reflection enables to create new object instance entered by name during running program. Following source codes are in Java programming language, but same function can be done with .NET platform and languages defined under Common Language Specification. Generally there are two requirements to programming languages:

- Ability to read object metadata and work with them as a metamodel (object self-identification)
- Some tool to enable object metamodel extension

Before the instantiation a source of metadata model - metamodel must be discovered.

3.2. Class metamodel

According to [7] a metamodel is a domain-specific language oriented towards the representation of software development methodologies and endeavours. After adjusting to class diagram metamodel we can say that metamodelling is an ability to express interactions between classes from metamodel - inner object state. Metamodelling is the act and science of engineering metamodels. Basic metamodel contains information necessary to class representation in concrete programming language.

Two approaches can be use to get metamodel. Model can be obtained from descriptors made before which are tight with created class. This form of implementation is very simple, however descriptor maintenance becomes difficult. When descriptors are defined in high amount maintenance becomes confusing. If the class doesn't contain descriptors, it cannot be used for metamodel purpose. This type of approach is applied in object-relation mapping called Hibernate.

Second option is use a reflection and read entire object metamodel. These information are obtained during program running and therefore enables dynamic 3rd part library linking with no additional library changes. When class name is provided reflection interface can read all class attributes, methods, return values and modifiers and pass these values to process on a higher level, typically GUI. In some cases detail information must be known to use class metamodelling. Basic metamodel is not sufficient therefore a new tool for user metamodel extension needs to

be found. Reflection must be able to use these extensions during object instantiation and modelling. Annotations are a quite suitable for user metamodel extension. Annotations are special type of syntactic metadata, which can be add to class source code and extend metamodel expressivity.

3.3. Entity view

Graphics representation of basic model scheme suggests Fig. 2. Final list of atomic classes are available. This list represents single classes but relationships are simplified from methods to object links. In simplified model an antecedent has only one consequent and antecedent pass result process to consequent. Reflection provides result passes, instantiation in right way with interaction to design patterns ensures extended metamodel. Model input and output is defined. Every element in model has only one input and one output.

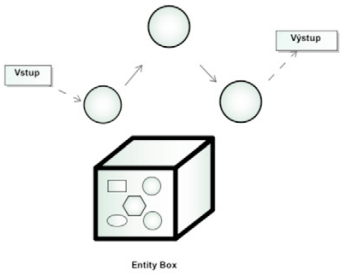

Fig. 2. Basic model

3.4. Class view

For an executive model representation based on reflection and annotation is more useful to create a class view. Every class contains internal and external methods. Internal methods are marked with private modifier, external with public modifier. Same approach is applied to attributes. Modelling starts when user enters initial values and the smallest step in simulation is one executed method. An internal state of object is changed during method execution or return value is generated. Returned value is passed to next class. User can observe every object attribute and read return value after every executed step. This feature enables reflection. User can also change interactions between objects during program running. User is able to use internal methods by changing modifiers. Internal state of object can be edited as well. These features give user ability to create executive model with no source code writing. This can be advantageous when result cannot be predicted but result might influent consequent components - chaining calculation. Nowadays many examples can be found. User gets possibility to create more complex structures and debug these structures after every step with no compiling. Model allows plugging new classes during simulation. Metamodel, read by reflection, allows creating graphical object representation in a model. Final relationships between classes can be saved by structured XML document. XML assigns unique identifiers to classes, defines inputs and outputs and mutual return value passing.

3.5. Elevate level of abstraction

Very important model feature is ability to elevate model of abstraction. In strict metamodelling framework an instance-of operator is allowed only within layers in a same linguistic level. However if we consider ontological level we can use instance-of operator on any layer. By linking on different layers new entities arises. These entities describe [7], namely Clabject (class-object) and Powertypes. Model created by user consists from several classes and interactions between them. Classes are part of entity box. This executive model is transformed to single entity after debugging and testing and carries significance description and defines input and output point. Entity becomes a

part of entity box as a single atomic element and is available to future modelling of executive models. User can edit created entity and modify internal relationships or whole classes.

4. Conclusion

Currently a practice model verification of executive model is in progress on LFLC (Linguistic Fuzzy Logic Controller) package. LFLC package is specialized class package, which is based on fuzzy set theory and fuzzy logic to enable to deduce conclusions on the basis of imprecise description of the given situation using the linguistically formulated fuzzy IF-THEN rules. Computation consists from several following classes. Change computation result means change inferential mechanism mostly. This change is realized by changing one class, or specific method. Rest of the model stays with no change including class and relationships. Currently only programmer can realize this change and there is a compilation process after every performed change. Executive model in this case allows performing a change to user with no programming experience and debug model in a real time with no compiling. User must be familiar only with concrete domain in this case IF-THEN fuzzy rule. LFLC classes are written in C++ programming language therefore an automated tool has been developed to make the process fully automatized. Target platform for executive model is a Java platform therefore tool uses JNI for access C++ classes.

The article is aimed to analyze current approaches to creation of executive model. A weak spots have been identified and new approach of executive model has been designed. This approach enables to create model from 3rd part component as well as from new classes. Model respect design pattern, allows to observing internal object state of all entities in a model in every step and allows changing relationships during model execution. A new interface to metamodel extension has been created to identify design patterns within a model. Interface is realized by annotations therefore available to reflection mechanism during model execution. In section 3.5 is suggested mechanism to elevate level of abstraction of executive model. In the future some visual environment to express executive model must be found. Visual environment should be expressive enough to formulate relationships between classes and simple for a nonprogrammer user. Advanced relationships between classes could be described by Petri's net formalism, where the future research will continue.

Acknowledgements

The paper is supported by IGA PřF University of Ostrava.

References

1. Arlow Jim, Enterprise Patterns and MDA, Addison-Wesley, (2006).
2. Object Management Group, MDA Guide Version 1.0.1., available on-line at www.omg.org/mda, Jan 2010.
3. ISO 10746-2, International Organization of Standardization, available on-line at http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html, Feb 2010.
4. Forman R. Ira, Java Reflection in action, Manning, 2005.
5. Soden Michael , Operational Semantics for MOF Metamodels, available on-line at http://www.metamodels.de/docs.html, Mar 2010.
6. Pecinovský Rudolf, Návrhové vzory, Computer press, 2007.
7. Gonzalez-Perez Cesar, Metamodelling for Software Engineering, Wiley, 2008.