Managing Complexity with Heterogeneous Modeling

Peter Forbrig

University of Rostock
Albert-Einstein-Str. 22
18051 Rostock, Germany
Tel.: +49 381 498 7620

peter.forbrig@uni-rostock.de

Abstract. Complex systems can only be created based on models. A general specification language that is able to specify all aspects of a complex system in a good readable form does not exist. Heterogeneous Modeling with domain-specific languages seems to be a way to tackle complexity.
The paper discusses some aspects of heterogeneous models and provides some examples demonstrating the combination of models based on Xtext and the Eclipse Modeling Framework (EMF).

Keywords: Heterogeneous Models; Domain-Specific Languages; Composition of Languages, Business Process Modeling.

1 Introduction

Complex systems cannot be created without carefully planning. Modelling is an important aspect for this purpose. However, one single modelling approach is often not able to cover the complexity of a system. Even for one aspect of a system, several models with different level of granularity are needed. In case of structural and dynamical aspects, specific modelling mechanisms are necessary to specify each of them. The same becomes obvious for cyber-physical systems. Physics models have to be used in combination with models of the digital world. Challenges center on different specifications of human systems, human-computer interaction, social systems, biological systems, physical systems, etc.

Domain-specific specifications have to be proven useful. Their combination is still a challenge. Domain-specific languages (DSLs) that are specified by grammars might help. Currently, Xtext 16 that is based on EMF (Eclipse Modeling Framework) provides a nice environment for specifying DSLs. The Xtext environment allows the generation of syntax-driven textual editors. As a side effect, a Meta model is generated. Combining grammars of different languages results in a common language and a common Meta model of those languages.

Textual editors seem to be useful in addition to visual editors. Handling of the editor is sometimes easier because of the familiarity with such tools. In 13 an overview of textual modeling languages for UML is provided.

Before examples of heterogeneous textual modeling languages are provided, related work will be discussed. The paper ends with a summary and an outlook.

2 Existing Perspectives of Heterogeneity of Models

Computer scientists have to cope with heterogeneous models. This starts with the different programming languages and ends with different paradigms for specifications.

Lee 11 begins the abstract of his paper with the sentences: “Complex systems demand diversity in the modeling mechanisms. One way to deal with a diversity of requirements is to create flexible modeling frameworks that can be adapted to cover the field of interest.” In other words, complexity has to be tackled with heterogeneous models and modelling languages.

Lee motivates his work with some philosophical remarks. He quotes Kant: “the variety of beings should not rashly be diminished.” (entium varietates non temere esse minuendas) 15 and Einstein: “everything should be made as simple as possible, but not simpler” 13. Later Lee argues with UML in such a way that structural and behavioral specifications have to be combined to specify a complex system. However, for both aspects different paradigms like hierarchical structures and automata have to be used.

Look et al. 12 discuss a concept of integrating six different model languages to model robotics applications. As modelling languages, they use a component & connector architecture description language, automata, I/O tables, class diagrams, OCL, and a Java DSL Their framework Monti-ArcAutomaton supports language aggregation, language embedding and language inheritance. Language aggregation enables establishing a knowledge relationship between the models in terms of references. The models have their own structure. Embedding allows the usage of different languages within the same model. Inheritance utilizes to refining or extending an existing language. All three approaches are discussed based on the consequences for abstract syntax trees.

Lee summarized his paper in the following way: “This paper reviews actor-oriented modeling of complex systems, arguing that it provides a disciplined approach to heterogeneity. The central notion in hierarchical model decomposition is that of a domain, which implements a particular model of computation”. We follow this idea for modeling business processes.

3 Modelling Business Processes with Heterogeneous Models

3.1 Modeling examples

Besides technical innovations, the support of human activities is very often the goal of software development. Business processes ranging from coarse-grained to fine-grained specifications have to be worked with. To demonstrate the usage of heterogeneous domain-specific examples, we will use very much simplified examples.

We will start with two task models for a customer and a salesman. Both are specified as trees with three levels. The first level starts with the root followed by an iteration on the next level. The iterative task is split into two subtasks that have to be executed one after the other.

It is assumed that a customer will ask for information and will get the list of available products. After selecting one product, the customer will get the corresponding price. **Fig. 1** provides the corresponding task models in the notation of the language DSL CoTaL6.

```
role Customer {
    root buy = negotiate{*};
            task negotiate =
                    ask_for_information >>
                    select_product
}
```

```
role Salesman {
    root sell = provide_information{*};
        task provide_information =
                provide_list_of_products >>
                provide_price;
}
```

Fig. 1. Behavioral models for customer and salesman

The behavior of both rules is specified locally. A customer first asks for information and later (temporal operator enabling - >>) selects a product. This can be done iteratively (temporal operator iteration - *). A salesman provides a list of products and later a price for a specific product. The cooperation of both roles is specified by a different model. It is called team model in the context of CoTaL1. The following figure provides a corresponding example.

```
team coop {
    root trade = communicate{*};
       task communicate =
              Customer.ask_for_information >>
              Salesman.provide_list_of_products >>
              exchange_info_about_product;
          task exchange_info_about_product =
                Customer.select_product >>
                Salesman.provide_price;
}
```

Fig. 2. Cooperation models for Customer and Salesman.

According to the cooperation model several communications can be performed. A communication is started by asking for information by a customer. A salesman will afterwards provide a list of products. From this list of products, the customer will select one product and the salesman will provide afterwards a price for this product.

The role models and the cooperation model are different languages. However, references from the team model to task of the role models exist. This is possible because of a joined grammar and consequently with a joined Meta model that is automatically generated based on that grammar.

Editors for domain-specific languages that were implemented with Xtext allow the generation of code for other tools. Each specification is stored by the editor as instance of the Meta model, which has much in common with an abstract syntax tree. For the introduced language DSL-CoTaL code generation for CoTaSE 4, HAMSTERS 8 and CTTE 5 were implemented. The animated visualization of the specification in the tool CoTaSE is presented in **Fig. 3**.

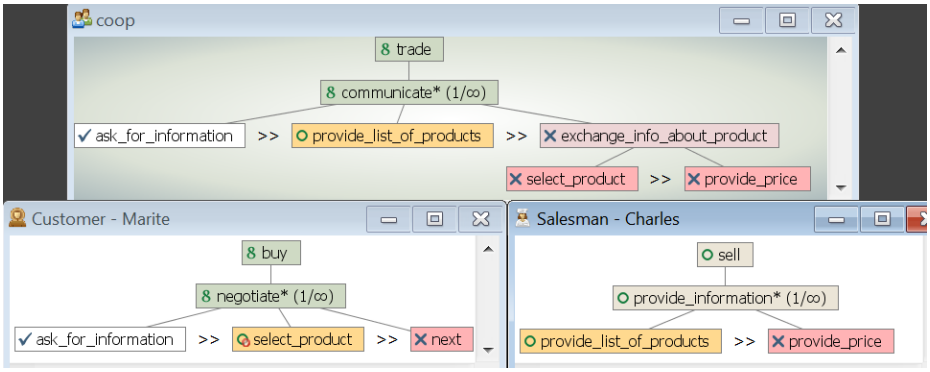

Fig. 3. Visualization of the cooperative model in CTTE.

The team model reflects the state of the cooperation. In the current state Marite asked for information and has to wait until she can select a product. Charles can provide a list of products.

It might be the case that some developers are more familiar with statecharts than with task models. Therefore, it can be useful to provide both views or allow the developer to specify the view

he/she is most familiar with. **Fig. 4** shows the specification of the behavior of a salesman in a DSL for task models and a DSL for statecharts.

```
role Salesman {
    root sell = provide_information{*};
        task provide_information =
                provide_list_of_products >>
                provide_price;
}
```

```
activity_state request_information
  provide_list_of_products => request_price
end
activity_state request_price
  provide_price => request_information
end
```

Fig. 4. Behavioral models for a salesman as task model and statechart model.

Previous examples focused on the task flow only. However, in business processes there are also objects involved. They can be specified in conjunctions with the tasks and used in preconditions or in object flows. Two different languages (object specification and task specification) were embedded in one general language. Relations between objects are omitted because of simplicity.

```
object product {
    attribute: String name, String price
}
role Salesman {
    root sell = provide_information{*};
        task provide_information =
                provide_list_of_products >>
                provide_price;
                pre provide_price -> product.price != ""
}
```

Fig. 5. Precondition for a task specified on the value of attributes of an object.

Specifications of the object domain can also be done on the class level. **Fig. 6** demonstrates on the left hand side the application of the GoF design pattern observer 7.

```
class Product{
    property name
    property price
    operation getPrice
}

class Customer{
    property name
}

subject Product observers {Customer}
```

```
public class Customer implements Observer {
    private Object name;
    private Product product;

    public Customer() {}

    public Object getName() {
        return this.name;
    }
    public void setName(Object name) {
        this.name = name;
    }

    public void update() {}

    public Product getSubject() {
        return product;
    }
    public void setSubject(Subject s) {
        product = (Product) s;
    }
}
```

Fig. 6. Specification of a pattern application (left) and the resulting java code (right).

The classes Product and Customer are specified with their attributes on the left hand side of **Fig. 6**. Additionally, the operation getPrice is specified for the class Product. Finally, the application of the design pattern observer is provided in a different language. The class Products is identified as having the role subject of the pattern. Several observers are possible. However, in the provided example there exists one observer class only. The class Customer acts in this role.

On the right hand side of **Fig. 6** one can see the consequences of the observer pattern application on the Java source-code level. The class Customer got the reference to a product as attribute. Additionally the operation update was generated. Both modifications of the class Customer are related to the pattern specification. The rest is due to general rules for code generation of classes.

Domain specifications by classes and transformations by GoF patterns are provided within the same modelling language. The two different grammars are combined for this purpose and deliver a joined Meta model.

3.2 Discussion and Outlook

Heterogeneous modeling has been used in software development for several decades. However, it seems to become more attractive with the new tools for language engineering. The workshop at EICS 2018 with the title "Workshop on Heterogeneous Models and Modeling Approaches for Engineering of Interactive Systems"[1] supports this impression. Heterogeneous modeling allows the separation of concerns and in this way the management of complexity.

Domain-specific languages allow the embedding of different languages by combining their grammars. This was possible to demonstrate with the small provided examples. Different modeling languages were used for:

- Specifying the communication between different models (team model and role models)
- Specifying alternative specifications for the same purpose (tasks and states)
- Specifying different models with references to each other (task and states)
- Specifying different models with references ant the intention of transformation of an existing specification (GoF patterns).

The overview of textual modeling languages for UML provided by Seifermann and Groenda 13 shows the acceptance of textual model specification. It is not the intention to replace visual representations. Both approaches, textual and graphical, should cross-pollinate each other. However, the textual specification and its representation as instance of a Meta model should be the basis for all implementations.

We already made experiments with modeling languages for user interfaces of questionnaires. Different languages were specified for tasks and domain, abstract user interface, concrete user interface and final user interface 10 using the CAMELEON reference framework 3.

A similar approach might be feasible for business processes and activity specification. One could imagine a specification of scenarios, use cases, and task models with different modeling languages that are embedded in one language and allowing references to each other. Additionally, transformations like for GoF patterns could be specified.

It might also be worth to specify business processes on different level of abstraction. Starting from a simple block sequence. Refinement could reach via several levels the details of BPMN. It might also be worth to specify sublanguages of BPMN for certain specific domains.

4 Summary

The idea of heterogeneous modelling was discussed in the context of textual domain-specific languages. It was demonstrated how the cooperation of different task models can be specified by a team model. Additionally, alternative representations of the same model can be supported by different concepts (task mode versus statechart model). Finally, transformation of models can be

[1] https://sites.google.com/site/wshybridmodels/

specified by models. These approaches are not new. However, they can be combined by Meta models and the corresponding toll support within EMF.

5 References

1. Buchholz, G., and Forbrig, P.: Extended Features of Task Models for Specifying Cooperative Activities. PACMHCI 1(EICS): 7:1-7:21 (2017)
2. Buck, J.T., Ha, S., Lee, E.A., Messerschmitt, D.G.: Ptolemy: A framework for simulating and prototyping heterogeneous systems. Int. Journal of Computer Simulation, special issue on "Simulation Software Development" 4, 155–182, 1994.
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interacting with Computers 15, 3 (2003), 289--308.
4. CoTaSE: https://www.cotase.de/, last visited August 10, 2018.
5. CTTE: http://giove.isti.cnr.it/lab/research/CTTE/home, last visited August 10, 2018.
6. Forbrig, P., Dittmar, A., Kühn, M.: A Textual Domain Specific Language for Task Models: Generating Code for CoTaL, CTTE, and HAMSTERS. EICS 2018: 5:1-5:6
7. Gamma, E., Helm, R., Johnson, R., and Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, 1994.
8. HAMSTERS: https://www.irit.fr/recherches/ICS/softwares/hamsters/contact.html, last visited August 10, 2018.
9. Kramer, M. E., Burger, E., and Langhammer, M.: View-centric engineering with synchronized heterogeneous models. In Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling (VAO '13). ACM, New York, NY, USA, Article 5, 6 pages. DOI: http://dx.doi.org/10.1145/2489861.2489864, 2013.
10. Kühn, M. and Forbrig, P.: Mobile Data Collection Forms Based on DSLs with Different Levels of Abstraction, Proceedings of MIDI 2014, Warsaw, Poland, June 24-25, pp. 1--8, 2014.
11. Lee E.A. (2010) Disciplined Heterogeneous Modeling. In: Petriu D.C., Rouquette N., Haugen Ø. (eds) Model Driven Engineering Languages and Systems. MODELS 2010. Lecture Notes in Computer Science, vol 6395. Springer, Berlin, Heidelberg, pp. 273–287, 2010.
12. Look, M., Navarro Perez, A.,Ringert, J.O., Rumpe, B., and Wortmann, A.: Black-box Integration of Heterogeneous Modeling Languages for Cyber-Physical Systems, in Proceedings of the 1st Workshop on the Globalization of Modeling Languages (GEMOC), 2013.
13. Seifermann, S., and Groenda, H.: Survey on textual notations for the Unified Modeling Language. In 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD), 2016.
14. Shapiro, F.R.: The Yale Book of Quotations. Yale University Press, New Haven, 2006.
15. Smith, N.K.: Immanuel Kant's Critique of Pure Reason. Macmillan and Co., Basingstoke, 1929.
16. Xtext: https://www.eclipse.org/Xtext/, *last visited August 11th 2018.*