# Model-driven Generation of Graphical Maps for e-Contents

Antonio Natali[1], Enrico Oliva[1] and Cristina Bonanni[2]

[1]ALMA MATER STUDIORUM–Università di Bologna, Cesena, Italy
antonio.natali,enrico.oliva@unibo.it
[2]IBM Italia, Tivoli Software, Roma, Italy
cristina.bonanni@it.ibm.com

**Abstract.** In this paper we present the application of a model driven software development approach (MDSD) to the generation of graphical (interactive) maps. The approach is rooted on the Eclipse Modeling Framework (EMF) and on a textual representation of the metamodel handled by the openArchitectureWare (oAW) suite of plug-ins. The metamodel represents our domain-specific language (DSL), which extends the concepts of e-Content organization defined in the Darwin Information Typing Architecture (DITA). While the usage of a textual representation is chosen to facilitate the task of the e-Content designer, graphical maps are introduced to enhance understanding and to improve the ability to acquire knowledge. The MDSD approach allowed us to hide into code generators all the details required to overcome the abstraction gap between the domain language and the implementation platform.



## 1 Introduction

Model driven software development (MDSD) [1] represents a strategic issue in modern software development for two main reasons; first of all it makes explicit (trough the models) knowledge that usually remains implicit during a conventional process and secondly it promotes automatic generation of (well structured) code to overcome the abstraction gap between the models and the implementation platform.

The Eclipse Model Framework (EMF) [2] supports the development of *platform independent models* by means of Ecore [3] which is the implementation of the Object Management Group's (OMG) Meta Object Facility (MOF) specification [4]. A model expressed in Ecore is usually introduced to define a meta-model, i.e. the abstract syntax of a domain specific language (DSL)

Domain specific languages are at the centre of MDSD in order to capture the key aspects of an application domain and to promote the automatic generation of platform dependent layers.

We have adopted the MDSD approach in the context of a joint project between the University of Bologna and the IBM Rome Tivoli Laboratory. The main aim of this project is to exploit the Darwin Information Typing Architecture (DITA) [5] and Eclipse to build a Manual Management System (MMS) intended as a specialized version of a modern eLearning Management System (LMS). As implementation platform we make reference to the Eclipse UA Infocenter - the Internet-based scenario

of the Eclipse user assistance (UA) - and a conventional eLearning platform like Moodle [6].

Working with a domain specific language poses the problem of the concrete syntax of the language. The editors automatically generated by EMF are tree-based editors; these editors, like those based on the graphical interface provided by Graphical Modeling Framework (GMF) [7] could be quite handy to use for a software developer; however the end user could feel more comfortable working with a textual representation, once it can be automatically associated to a graphical map tailored to the essential concepts in the domain. The graphical map provides both a more understandable vision of the model and a set of general-purpose interactive functionalities as zooming, filtering and detailing-on-demand that can improve understanding.

For this reason we have decided to define a textual editor for a DSL devoted to the specification of the organization and semantic properties of contents represented in machine-readable forms (*e-Contents*) and to exploit MDSD to generate in automatic way one or more graphical maps intended as different views of the user model. To achieve the goal, we are using the Xtext framework of the oAW [8] suite and the Prefuse [9] library as the operational implementation framework for map drawing.

In this paper we discuss the usage of EMF-oAW for the construction of a textual editor with code completion, syntax highlighting, together with the automatic generation of the artifact required for the specific operational platform, with particular reference to the Prefuse visualization engine.

The work is structured as follows: section 2 presents an overview of system functionality from the user point of view; section 3 describes the rules to construct the grammar and the editor; section 4 focuses on the map generation and section 5 draws some conclusion.

## 2 Overview of the system functionalities

The organization and navigation of e-Contents has a dramatic impact on the user's ability to acquire knowledge; in particular, the usage of graphical maps for structuring and organizing information to enhance understanding and learning is an open research issue [12].

In our approach, the conventional navigation loop "select a topic from an index, open the topic, and read the content" is replaced with the concept of semantic, customizable navigation into *reading paths*, either pre-planned by the e-Content designer or even dynamically built by the reader. Thus, our DSL provides the following main concepts (that extend those introduced in DITA [5] ):

- Each content Unit is composed of a set of logical contents called *Items* or *Topics*; each Item is associated to one or more resource File that can be written in any format readable by a conventional browser.
- Each Item can be associated to a (empty) set of *metadata* including binary logical relations with other items.
- The author can define one or more *reading paths*.
- *Graphical maps* are exploited to arrange the items into custom-related highly-readable contexts.

The usage of maps enhances the semantics aspects related to the e-Contents and allows the creation of content views tailored to specific user needs. A conventional index can be considered as one of several, possible pre-planned *reading paths* that can be offered to the reader in order to achieve specific learning goals, such as theoretical background rather than practical skill, etc.

At the centre of our approach there is a formal model of e-Content organization [11], expressed in `MOF` [4]; a subset of the model is shown in the figure hereunder:

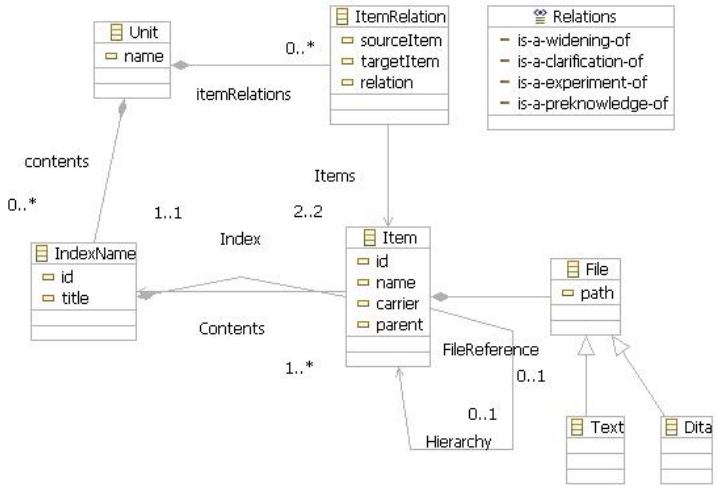


**Fig. 1.** Part of the metamodel expressed as an Ecore diagram.

One of the key-concepts introduced in the formal model is the possibility to express binary logical relation between items. The set of relation types is open; for example, with reference to a relation statement such as S *relType*T possible *relType* values could be:

- *Preknowledge-of*: S assumes that the reader knows what is written in T
- *Clarification-of*: T should make the content of S more clear;
- *Conceptualization-of*: T presents the content of S in more formal way;
- *Widening-of*: T is a study in depth of S;
- *Experiment-of*: T is an experiment related to S;
- *Exercise-of*: T is an exercise related to S;
- *Test-of*: T is an evaluation of the student understanding of S.

Besides reading paths and semantic relations, the model allows users to associate metadata values to the items. Metadata can represent educational data (e.g. `semanticDensity`, `difficulty`, etc), audience-related information (e.g. `beginner`, `expert`), and so on.. A relation can be also associated with one or more conditions involving the metadata; in other words the concept of relation is not absolute, but can depend on (meta)data values.

The `MOF` model is meta-model that provides our `DSL`. Thanks to the concrete textual syntax associated to the language, users can specify their content structure in a textual form like the following:

```
econtent Computer Science
item func title "Function" meta difficulty high audience beginner
item stat title "Statement" meta difficulty low audience beginner
item prog title "Programming" meta difficulty medium audience expert
item val title "Evaluation" meta difficulty medium audience beginner
item adt title "Abstract Data Type" meta difficulty
item java title "Java" meta difficulty high audience beginner
item jadt title "Java ADT" meta difficulty high audience expert
item list title "List" meta difficulty medium audience expert
item arr title "Array" meta difficulty medium audience expert
readingpath labpath { java prog jadt adt list }
readingpath thpath { func stat val adt list arr }
relation func preknowledge-of prog
relation stat preknowledge-of prog
relation stat widening-of val
relation adt preknowledge-of prog
relation list example-of adt
relation arr example-of adt
```

This toy example describes the organization of a computer science course. Each `item` represents a knowledge unit; the items can be arranged in two different *reading paths*, one (`thpath`) intended for theoretical background and the other (`labpath`) for the laboratory. Each item can be associate to one or more metadata introduced by the key word `meta`; in the example, the metadata specify the kind of user (`expert`, `beginner`) for which the item content is written and the difficulty level of an item (`low`, `medium`, `high`).

The model written in textual form is the only artifact that should be produced by the author. The content specification file must be given as input to the generator system shown in figure 2; the generator produces in automatic way both the run time support for in the chosen platform of content delivery and a set of graphical maps

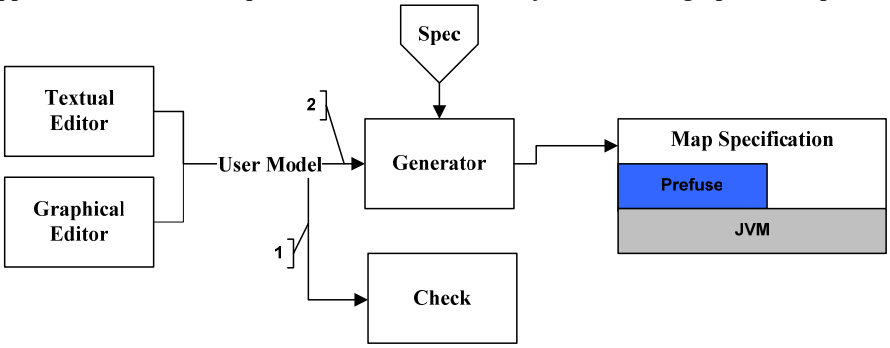


**Fig. 2.** Map generation system.

### 2.1. The role of the maps

Maps are visual methods for understanding and representing knowledge. Visual methods could be used for learning purpose helping human to articulate implicit

knowledge and to formulate new thinking. Typically, most of the visual methods are interactive and provide an overview of the data, including functionalities like zooming, filtering and detailing-on-demand in order to promote cognition; other kinds of maps help users to elaborate concepts, ideas and plans.

There are several *map styles* that can be pragmatically classified as in [12] through two relevant dimensions: complexity of visualization and application area. Each map style can emphasize different aspects of a model by allowing users to perceive concepts in different ways; different styles can be used in combined, complementary way to achieve better results. For example:

- tree view : is a map style organized like classical hierarchical index where the contents are open/closed dynamically according to users requests. This style can be used to show reading paths;
- *radial graph* : is a map style that highlights hierarchies or semantic networks of information centered on a specific item. The selection of an item produces a re-configuration of the map, by focusing on the selected item. This kind of map promote a qualitative elaboration of the information and is well suited, together with filtering/selection features, to represent item relations;
- *tree-based map* : is a space-filling approach to display data as set of nested rectangles. This map style is useful to organize and represent complex structure of information with several connections, categories and users. Typically it is possible to display in a unique map a multiplicity of items so to recognize patterns or content characteristics in few seconds. . This kind of map can be used to organize the views by keeping into account metadata values.

With reference to the toy example, the left part of Figure 3 shows a classical tree view in which the image changes when the user navigates into the structure; the right part shows a radial map including an overview-part of the e-Content and a detailed part centered on `Programming` topic.

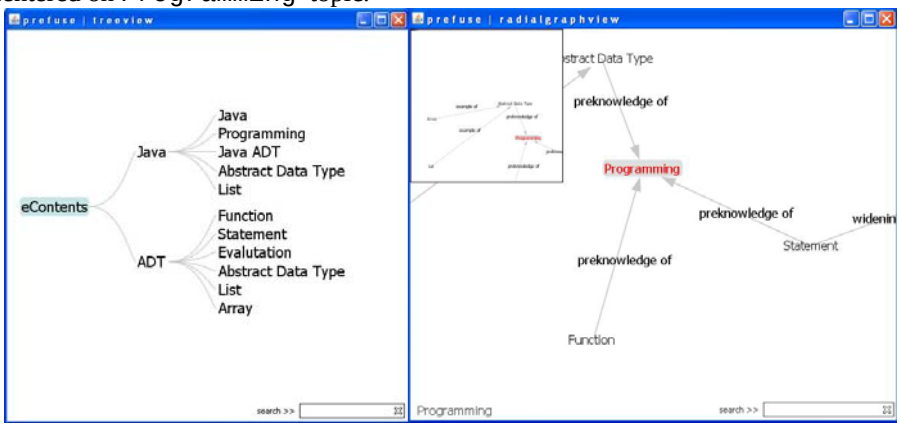


**Fig. 3.** Generated Map: Direct Radial Map with Overview (left); Tree View (right).

The left part of Figure 4 shows the tree-based map related to a reading path; the selected item is represented by the area on the left-up corner, while the dimension and color of rectangles are related to `difficulty` metadata of the corresponding item; the right part of figure 4 shows a radial map centered on the `Programming` item that highlights (by using a filter) the items written with reference to expert people.

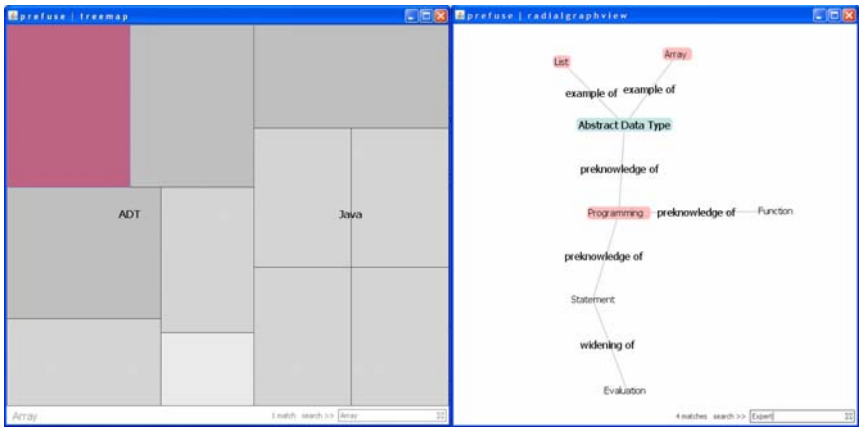


**Fig. 4.** Generated Map: Tree Map (left); Radial Map with meta filter (right).

Once made interactive, the maps could also be used by the user as a new form of input device for his/her specifications.

## 3 Editor and Grammar Construction

The construction of a text editor for a specific grammar is not an easy task. Tools like parser generators (`javaCC`, `ANTLR`) allow the construction of a parser and the data structure used by the parser to build the `AST` (Abstract Syntax Tree) leaving to the user the creation of the whole editor, with syntax highlighting, code completion, model checking, etc.

`oAW` is an Eclipse Project written in Java that supports the parsing of models and a set of tools to check and transform the models as well as to generate code. `XText` [10] is a component of `oAW` that allows the textual definition of `DSLs` using Extended Backus-Naur-Form (`EBNF`) notation. The framework supports the automatic generation of a parser, an `Ecore` metamodel and a specific text editor for Eclipse.

Starting from the Ecore metamodel defined in our previous work [11], we have defined a grammar so that the transformation performed by `oAW` generates the same `Ecore` metamodel. Since the definition of such a grammar cannot be performed by an automatic process, our approach was to introduce a set of guide-rules in order to achieve the goal in the most systematic way as possible.

### 3.1 From the model to a grammar

An `Ecore` metamodel represents the abstract syntax of a (domain specific) language. `XText` allows describing both the abstract (i.e. the metamodel) and the concrete syntax of a `DSL`. A grammar specification consists of a list of production rules written in a `EBNF` notation of form `RuleName:Description`.

The `RuleName` is both the name of the rule and the name of the class (*metatype*) in the metamodel corresponding to this rule (e.g. `Unit`, `Item`, etc. of figure 1). A

`Description` is made up of *tokens* that can be a built-in token `ID` that represents a unique identifier or *KeywordTokens*, *IdentifierTokens*, and *AssignmentTokens* of the form: `Lpart Op Rpart`.

Each assignment token is not only used to create a corresponding assignment action in the parser, but also to compute the properties (represented by the `Lpart`) of the current metatype. Properties can refer to the simple types such as `String`, `Boolean` or `Integer` as well as other complex metatypes (i.e. other rules); what the type actually is depends on the assignment operator `Op` and on the type of the token on the right; with the standard assignment form (`=`) the property type is computed from the token in the `Rpart`; in the `+=` form the type is a `List` of elements whose type depend on the token in the `Rpart.`

In order to re-produce exactly our metamodel [11], the `Description` part of the rule related to a specific metaclass has been defined according the following set of criteria:

1. For each *attribute* define a keyword token (which is exploited by the editor to suggest text completion) through a standard assignment.
2. For each *composition* define a += assignment in which the `Rpart` is the rule associated to the related metaclass.
3. For each *association* (cardinality 1) define a standard assignment that makes reference to the rule associated to the related metaclass.
4. For each *aggregation* define a += assignment in which the `Rpart` is a crossreference to the rule associated to the related metaclass `D` (denoted with `[D]`).
5. For each *generalization* define an abstract rule made of a sequence of alternatives that reflects the type hierarchy; all the properties defined in all subtypes are automatically moved to the common supertype.
6. For each *enumeration* type, define an enumeration rule `Enum` composed of a sequence of choices with all literals as terminal symbols.

For example, the rules corresponding to the simplified version of or metamodel depicted in figure 1, can be defined as follows:

```
Unit :
  "econtent" title=STRING
  ("version" version=ID)?
  (contents += Item)*
  (readingpaths += ReadingPath)*
  (relations += ItemRelation)* ;
Item :
  "item" name=ID
  "title" title=STRING
  ("path" files += File)+
  (metadata += Metadata)* ;
Metadata :  "meta"
   ("difficulty" difficulty=DifficulType)?
   ("audience" audience=Audience)? ;
File : STRING(Dita | Html) ;
Dita : ".dita";
Html : ".html";
ReadingPath :
 "readingpath" id=ID title=STRING "{"
 (items += [Item])+ "}";
ItemRelation :
    "relation" firstel=[Item] rel=RelationType secondel=[Item];
Enum RelationType :
    A1="widening-of" | A2="preknowledge-of" | A3="clearification-of"
    | A4="example-of" | A5="parent-of " | A6="composed-by";
Enum DifficulType : dt1="low" | dt2="hight" | dt3="medium";
Enum AudienceType : at1="expert" | at2="beginner" ;
```

- The rule `Unit` defines the whole content with its name, identification and optionally with a version. The declaration based on the += operator corresponds to the containment reference of the same name shown in the model of fig. 1.
- The rule `ReadingPath` corresponds to the concept of reading path composed of a list of item references. It is an example of aggregation.
- The rule `Item` allows the user to add the concrete information about the content as the title, the optional level of difficulty and the reference to one or more files.
- The rules `File` allows to insert the reference to a file in two different formats (as `DITA` or `html`). It is an example of generalization.
- The rule `Itemrelation` allows to specify the semantic relations between two items. It is an example of association with the use of cross reference.

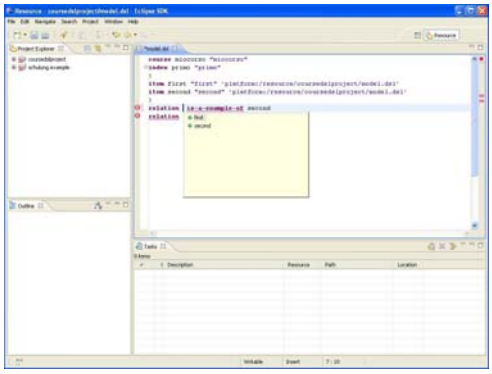

**Fig. 5.** Code completion with use of cross-reference

In figure 5 we report an image of the `GUI` provided by the textual editor automatically built from the `Xtext` specification by `oAW`, together with parser and the `Ecore` metamodel. The figure shows in particular the execution of `ItemRelation` rule where the editor suggests a list of `id` from the inserted `item`.

## 4 Map Generation

Our drawing reference platform is `Prefuse`, a Java open source project that simplifies the creation of custom visualization. `Prefuse` is based on data state model [12] that is a reference model for information visualization maps. It introduces abstractions for filtering data and actions to perform over the contents as layout, assignment color, search and zoom. Programmers can create custom interactive views from the visual data exploiting the highly-configurable rendering engine.

`Prefuse` supports several visualization methods as radial layout, force-direct layout, tree map, tree view and fisheye menu. The first step to generate the map is to load a graph or tree data set from a `XML` file; the second step is to associate to each entity, nodes and edges, a renderer; the third step is to specify a sequence of actions in order to assign visual properties (position, color, size) and filter data; the final step is to create the display by adding interactive functionalities as zoom, focus control and animate transitions.

### 4.1 Code Generation: from Model to Maps

To overcame the gap between the model and the platform we exploit automatic generation of code of oAW, which is based on the declarative language Xpand based on templates (DEFINE blocks) that can be associated to each element of the metamodel. A portion of the templates introduced to generate the XML file for a Radial map is reported hereunder:

```
«IMPORT coursedsl»
«DEFINE main FOR Unit»
«FILE title+".xml"»
  <?xml version="1.0" encoding="UTF-8"?>
  <!-- Radial graph -->
  <graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  <graph edgedefault="directed">
    <!-- data schema -->
    <key id="name" for="node" attr.name="name" attr.type="string"/>
    <key id="type" for="edge" attr.name="type" attr.type="string"/>
    <key id="diff" for="node" attr.name="diff" attr.type="Real"/>
    <key id="audi" for="node" attr.name="audi" attr.type="string"/>
    <!-- nodes -->
    «EXPAND node FOREACH this.contents»
    <!-- edges -->
    «EXPAND edge FOREACH this.relations»
  </graph>
  </graphml>
«ENDFILE»
«ENDDEFINE»
«DEFINE node FOR Item»
  <node id="«this.name»">
    <data key="name">«this.title»</data>
    «EXPAND meta FOREACH this.metadata»
  </node>
«ENDDEFINE» ...
```

The first template (main) is associated to the Unit class (or ule); the EXPAND statement “expands” another DEFINE block (node, edge) in a separate variable context. Since a platform like Prefuse requires different data input format for each map type, different templates are required.
Some customization of maps, such as colors, labels on edges or interactive functionalities are possible only through commands written in the implementation language. To this end Xpand allows to include Java expressions into templates by means of *extensions*.

The generation process, including model checking based on OCL-like expressions is driven by a workflow specification written in a declarative language provided by oAW.

## 5 Conclusions

Thanks to Eclipse EMF and to features like those provided by the plug-in suite of oAW, model driven software development is going to become an effective approach in the field of software construction. In this approach a fundamental role is played by the metamodel that defines the domain-specific language. But the key-point is related to the possibility to exploit tools in order to create in automatic way the code that supports the intended language semantics on different implementation platforms. In this work an user model represents the specification of e-Content organization and

can be written in textual form; the abstraction gap between the language and the platform has been overcome through a set of code generator rules, each related to a specific class of the metamodel.

While a proper design of the metamodel is fundamental to capture the relevant logical aspects of the applications in the reference domain (*business logic*), the specification of code-generation rules can lead to the systematic production of high quality code, by hiding into the generators all the stuff required to make things really working. In order to achieve such a goal, we have designed and developed the generators using agile methods [13] by facing in a systematic way the main "forces" in the domain as suggested by the pattern languages approach [14]. This aspect will be described in a forthcoming paper.

## References


1. Stahl, T., Vötler, M.: Model-Driven Software Development. (2005)
2. Budisnsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.: Eclipse Modeling Framework. (2004)
3. Ecore. http://www.eclipse.org/modeling/emf/?project=emf
4. Mof. http://www.omg.org/technology/documents/formal/mof.htm
5. Harrison, N.: The Darwin Information Typing Architecture (DITA): Applications for globalization. In: Professional Communication Conference IPCC 2005. Volume 10-13., Ieee (2005) 115 – 121
6. Moodle. http://docs.moodle.org/en/Main_Page
7. Eclipse gmf. http://wiki.eclipse.org/GMF_Documentation
8. openArchitectureWare. http://www.openarchitectureware.org/
9. Heer, J., Card, S.K., Landay, J.: Prefuse: a toolkit for interactive information visualization. (2005)
10. Efftinge, S., Völter, M.: oAW xText - A framework for textual DSLs. of Conference Eclipse Summit - Modeling Symposium (2006)
11. Natali, A., Del Cinque, A., Oliva, E.: Using eclipse in building model-driven e-learning supports. In Maresca, P., ed.: Eclipse: a Great Opportunity for Industry and Universities in Italy. Volume 1., Cuzzolin Editore (October 2007) 27–42 1st International Conference on Eclipse Technologies (Eclipse - I 2007), Napoli, Italy.
12. Chi, E.H.: A taxonomy of visualization techniques using the data statereference model. (2000) 69 – 75 15. Cockburn, A.: Agile Software Development. 2 edn. (2007)
13. Alistair Cockburn: Agile Software Development, 2° edition, Addison-Wesley, 2007
14. Franck Bushmann, Kevin Henney, Douglas C. Schmidt: Pattern-Oriented Software Architecture: Volume 5 On Patterns and Pattern Languages, John Wiley & Sons, 2007