# Practitioners View on Domain Specific Business Process Modeling

Janis BICEVSKIS, Jana CERINA-BERZINA, Girts KARNITIS, Lelde LACE,
Inga MEDVEDIS and Sergejs NESTEROVS
*University of Latvia*

**Abstract.** Practitioners view on modeling with Domain Specific Languages (DSLs) is presented in this paper. It is shown, that unlike general-purpose modeling languages (UML [1], BPMN [2]), DSLs provide means for concise representation of semantics of the particular business domain, enabling development of consistent and expressive business process models. Resulting models can be used not only as specifications for information systems, but also for generation of implementation artifacts, creating documentation, testing and for other purposes. Thus one of the most principal goals of Model Driven Architecture (MDA [3]) – the development of model-based information system – is achieved.

**Keywords:** Business Process Modeling, BPM, Domain Specific Languages, DSL, Model Driven Architecture, MDA.

## Introduction

A perfect information system, which is consistent with all customer requirements, reliable and flexible, still remains only a dream for IT developers. Main obstacle in the way to this dream is inability of customers, who do not know information technologies in details, to define their requirements clearly and to communicate them to the developers. Traditionally, information systems have been developed in compliance with some standardized documentation, for example, software requirements specifications, but, in practice, requirements, if they are formulated in natural language, tend to be inaccurate and ambiguous. The situation is worsened by often changing customer requirements. All this places software developers in unenviable situation – they must develop software according to inaccurate and changing specifications.

One possible solution to the problem is to use a formal language to define requirements and to make a model for the system. This can eliminate ambiguity of requirements, and can enable direct translation of specification into application. Thus, the problem of changeability of requirements becomes resolvable – changes can be introduced into model, and then application can be automatically generated.

However, despite decades-long efforts, these problems still are not completely solved. Traditional CASE technologies have given only partial results (see, for example, Oracle Designer [4]). Fierce competition in IT market demands information systems of exceptional quality, and support from traditional CASE technologies is not sufficient to provide adequate user interface, high usability, maintainability, performance... Flexibility against changing requirements is still limited. For example, if requirements are changed, and these changes cannot be represented in the formal

specification (as specification language is not sufficient), they must be incorporated directly into the source-code of the system, and, in case of automatically generated source-code serious problems can arise. CASE tools are also quite conservative and slow to catch up the fast development of the programming technologies. As a result, only a fraction of applications can be generated from specifications.

IT experts are still looking for new ideas. One of the recent developments is Model Driven Architecture (MDA [3, 5]). In order to make application development more flexible, this approach splits the process into two steps. First, the platform-independent model (PIM) is made in some general-purpose or domain-specific modeling language, for example, UML. Second, PIM is translated to a platform-specific model PSM, thus obtaining an executable application. This separation allows for more flexible generation of applications and development of user-friendly information systems. MDA approach evolves very fast, but some challenges are also at a glance.

General-purpose modeling languages, including UML, often used to make PIMs, are difficult to grasp for non-IT professionals, the future users of information systems. Even if they read and accept the models, their understanding is not deep enough, and they undervalue consequences of decisions behind these models. Code, generated from the PSMs, still does not produce usable and reliable software. Information systems are not flexible, and it is hard to achieve compliance with the models. If changes are made directly in the generated code, consequent re-generation can void them.

Trying to follow the path of MDA often ends with UML specifications, that is just one of the sources of information for developers of the software. Only in some projects UML models are directly translated into software ([6]).

Facing these problems in the practice again and again, and having advanced tool-building platform ([7]) at hand, we tried to solve them building domain-specific business process modeling tools, according to the following principles:

1. Tools must be comprehendible to non-IT specialists, because modeling will be done mostly by domain professionals.
2. Specific requirements of business domain and of information system development must be taken into account.
3. Modeling of real-life situations in the business domain must be possible, as well as ensuring and showing to users that information systems treat these situations correctly.

We have made a graphical domain-specific language, which, we hope, can easily be understood by non-IT professionals. This language is domain-specific: first, it can be used only for modeling of business processes, and, second, it can be used only in specific organizations. The language, called ProMod, is extended subset of BPMN (Section 1). Language deals mostly with behavioral aspects and do not try to cover entire enterprise architecture as for example ArchiMate ([8]) do. Key feature of ProMod is that semantics of graphical primitives is deeply specific for organizations where it is intended to be used. Business process model is a set of diagrams, interconnected with tree-like structures of enterprise data. Unlike many general-purpose modeling tools ensuring only syntactical correctness of models, ProMod provides tools for checking semantic consistency and completeness with domain-specific rules – this kind of validation is impossible in a general-purpose language.

Narrow usage domain of ProMod raises a question of amortization of efforts, i.e. whether the benefits gained are worth the time and money spent developing the language. Using transformation driven architecture to build DSL tools can solve this problem. We used metamodel-based graphical tool-building platform GrTP and it

enabled us to create both graphical editor and consistency checker in a reasonably short time. This paper deals mostly with ProMod DSL – for detailed discussion of tool-building platform GrTP see [7], [9] and [10].

The paper is organized as follows. Section 1 contains description of our business domain and main features of ProMod DSL. Section 2 is a brief introduction to tool-building platform GrTP. Section 3 discusses current usage and ideas about future development of ProMod DSL.

## 1. Features of the Domain-specific Modeling Language

General-purpose modeling languages offer a fixed set of primitives: objects, attributes, connectors etc., with predefined semantics, that cannot be extended, or can be extended in some limited predefined manner (as, for example, stereotypes and profiles in UML [11], [12] or styling of graphics and option to adding attributes in BPMN [2], [13]). Such notation is suitable for modeling in general, still much of the domain-specific information remains outside the models. In the domain-specific languages we are looking for ways to extend the set of modeling primitives with ones, specific to the particular business domain ([14]).

### *1.1. Modeling Domain: State Social Insurance Agency*

The domain, where we are looking for specific concepts, repetitive patterns and clichés of business organization to enrich the modeling language, is Latvian State Social Insurance Agency (SSIA) – a government institution, providing pensions, benefits, allowances etc. Like in many government institutions, all activities in SSIA are strictly prescribed by legislation and local instructions, but, unlike most government institutions, SSIA is a client-oriented enterprise – its main function is to service clients. Servicing clients in a vast majority of cases means processing documents: a client claims for some social service and provides appropriate documents, these documents go thru a workflow, and in the end approval or rejection letter is sent to the client.

The domain of social security is sensitive sphere in Latvia and it is a target of frequent political decisions and new regulations resulting in frequent changes of information systems.

SSIA has recognized need for the management of business processes. Many of the business processes have already been defined in richly annotated IDEF0 ([15]) diagrams. Currently SSIA is planning to include business process models into the instructions for the stuff and to use them as essential part of the requirement specifications for the information systems. In the same time, as the numbers of diagrams is constantly growing, and the diagrams are independent VISIO files, it becomes harder and harder to keep them consistent.

### *1.2. Overview of Language ProMod*

ProMod is based on a subset of BPMN and keeps its more frequently used graphical symbols: activities, events, sequence and message flows, data objects etc. These symbols sometimes have specific semantics for SSIA. For example, occurrence of an event means, that a person or an organization has brought a package of documents – events are also color-coded to show whether they originate within SSIA or come from

another institution. Message and sequence flows always carry documents and are marked as significant (bold arrows) or insignificant, and so on.

Graphical symbols have rich set of attributes. Activity, for example, in addition to traditional attributes (name, textual description, performer…) has also domain-specific attributes (regulations defining it, document templates involved, customer services fulfilled…) and modeling process attributes (acceptance status, error flags, version…).

A model in ProMod is a set of diagrams representing business processes (Figure 1). There are three types of diagrams (all behavioral according to UML taxonomy):

1. Business process diagrams are used to describe business processes for employees of the organization. They are simple and easy to read.
2. Information system diagrams are also used for process modeling, but are intended to describe the process in a way, suitable for development of information systems. For further differentiation between these two types of diagrams see Section 1.6.
3. Customer service diagrams provide description of the business processes from the viewpoint of services provided to the customers – see Section 1.5.

Besides these diagrams, structured lists are also part of the model, representing:

1. Structure of organization,
2. Regulations and local instructions defining the business processes,
3. Information artifacts – documents and pieces of information,
4. Services provided to customers.

These lists are not only convenient way to enter values of attributes – they by themselves carry essential data, establish connection between various fragments of model and are used for consistency checking and reporting.

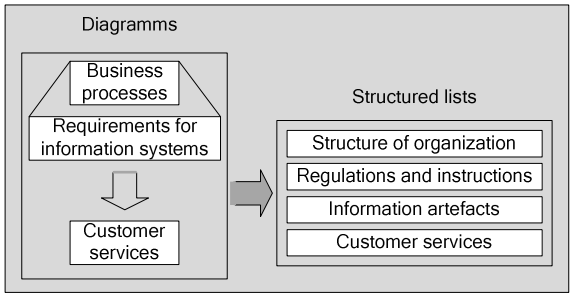


**Figure 1.** Diagrams and structured lists

### *1.3. Refinement of Business Process Models with Enterprise Information*

Information stored in the structured lists is needed not only for modeling of business processes – in fact it is the basic enterprise information. In most organizations this information can be found in some information systems, but unfortunately these systems have not been built with business process modeling in mind. Traditional modeling tools often are incapable to connect to these data bases and obtain the data. Models, therefore, remain isolated from the real world, and if, for example, enterprise information is changed, it is easy to forget to change the models accordingly.

In domain-specific modeling tools it is natural to provide means for data exchange with enterprise information systems. ProMod provides these means, giving so the following advantages:

- Enterprise information can easier be maintained in information systems specially designed for it – there the information is connected to another data, quality can be checked and responsibility for maintenance can be assigned.
- Information is not duplicated, if modeling tools take it from information systems.
- Business models are closely connected to real life of the organization, and the risk for them to become outdated and inadequate is smaller.
- If needed, it is possible to integrate business process models into information systems, to show step-by-step progress of the instance of business process.
- It is possible to analyze business processes in context of enterprise data, showing, for example, which other regulations and which steps of business processes will be affected, if some part of regulation is changed (that, by the way, is especially important in SSIA, because of frequent and voluntary changes in regulations).

In addition it is possible to interconnect different diagrams and graphical symbols and to make new types of diagrams according the domain-specific logic. Customer service diagrams, for example, consists of action symbols, defined in other diagrams, and joined together, because they are needed for particular customer service (as mentioned in Section 1.2, customer services are part of enterprise data).

### *1.4. Domain-specific Consistency Rules for Business Processes*

Important aspect of modeling is consistency of created models. According to the scope consistency can be:

1. In the level of element, for example, whether all mandatory attributes have been entered.
2. In the level of diagram, for example, whether diagram begins with an event.
3. In the level of model, for example, whether all referenced sub-processes are defined somewhere.

Above mentioned are universal consistency rules, but in ProMod, rules, reflecting specificity of SSIA are more essential. As the main goal of business processes is to process customer documents, it must be checked, whether all documents, provided by customer, are used in some step. It must be checked, whether set of documents from event starting the business process match to set of documents used in decomposition of its first step. It must be checked, whether all steps in all business processes are needed for some customer services, and so on.

As enterprise data is linked to the model, it is possible to check, whether performer of the step still exists in SSIA, whether organizational structure of SSIA have not been changed, whether regulations, defining business process, have not been expired… As the possibilities of domain-specific consistency checking seem to be unlimited, ProMod provides means for easily adding new rules.

In ProMod consistency checking is not performed during creation or modification of the diagrams – consistency check is a separate action, and inconsistent models can be stored in the system and kept for a while. This approach gives some benefits:

1. It is possible to start from rough sketches made by insurance professionals of SSIA, work with them and in the end turn them into consistent models.
2. This approach is more suitable for non-IT professionals, because they tend to concentrate on the main ideas and think, that consistency details are boring.

*1.5. Business Processes and Customer Services: Two Views on the Same Model*

At the very first steps of the business process modeling, when trying to identify and name business processes, there are two essentially different options. First, we can look at the organization from management's perspective and classify processes according to the way they are performed. Second, we can take perspective of customers and classify processes according to services or products they are producing.

Management's perspective seems more natural, and has been chosen in SSIA. We believe that it will be the case in most government institutions. If one reads statutes of SSIA, the first higher level business processes are obviously the functions mentioned there: grant social insurance services, provide consultations, register socially insured persons, etc. (see ProMod business process diagram in Figure 2.a). Customer's perspective would lead to business processes related to the customer services: grant retirement pension (including consultations, registration and everything else), grant disability pension or grant childbirth benefit.

If, following the management's perspective, we perform top-down decomposition of high-level abstract business processes; we see that many steps are independent of the customer services they provide. For example, steps "Receive documents", "Register documents" and "Send resolution" (Figure 2.b) are almost the same whether request for retirement pension or disability pension is being processed. Differences in processing various customer services show up only in some steps (Figure 2.c) and mostly in deeper level of decomposition.

Essential drawback in taking management's perspective is that in the resulting models customer services are not clearly represented – they are "dissolved" in detailed lover-level diagrams. Customer, for example, is always interested in one service at a time and business process diagrams are not helpful to him.

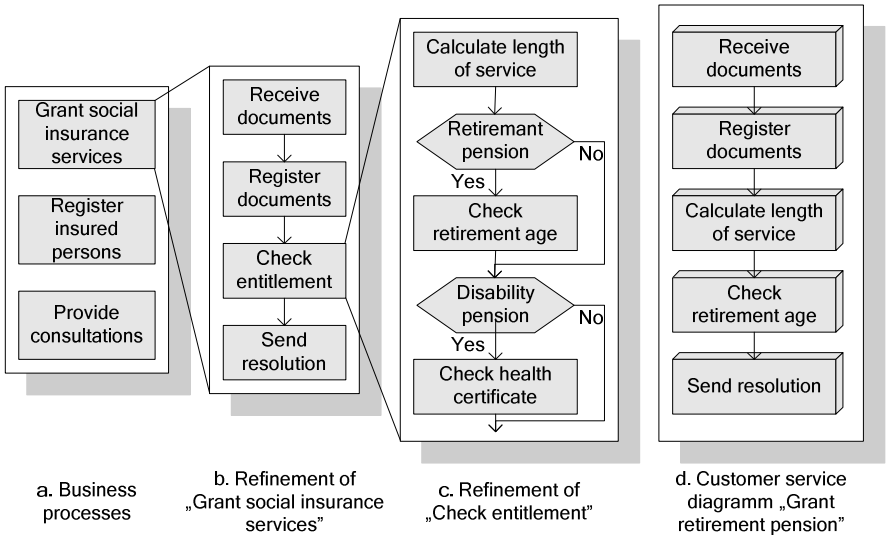


**Figure 2.** Business processes and customer services

To resolve this contradiction between management's and customer's perspectives, in ProMod we have introduced special type of diagrams, called Customer Service

Diagrams. These diagrams are made for every service provided, and contain all steps from various business process diagrams, needed to provide that service (Figure 2.d). This is a distilled value chain for one particular customer service. These diagrams do not contain any new information they are just a different views to business process diagrams, and in our editor they are made semi-automatically. Customer Service Diagrams bear some resemblance to use-cases and communication diagrams in UML.

### *1.6. Modeling for Humans and Modeling for Information Systems*

The MDA approach encourages automatic translation of business process models into implementation artifacts (database objects or executable code). In the situation when both restructuring of the business operations and development of an information system are the goals of the business process modeling, it is tempting to assume, that the same set models can be used for both purposes. This point seems even stronger, because the same modeling language can be used to pursuit both goals. However models, that can be easily read by employees, lack accuracy and detail needed for development of information systems, but models, suitable for development of information systems, are much too detailed and boring, to be read by employees. The difference is not only in the degree of elaboration: style, graphic representation, cultural biases and even human ambitions must be taken into account.

We faced all these challenges in SSIA, where business divisions were responsible for business processes, but development of information systems were split into separate department and partially outsourced. For this reason there are two visually different diagram types in ProMod: one intended for employees, and other – for development of information systems (Figure 3).

Diagrams for employees have limited set of graphical symbols; they are intended to specify sequence of steps, rather than detailed logic; and, in order not to disturb employees of business divisions, they look much like previously used IDEF0 specifications. Level of detail is acceptable, if knowledgeable insurance professionals have no difficulties to follow the business process, using them.

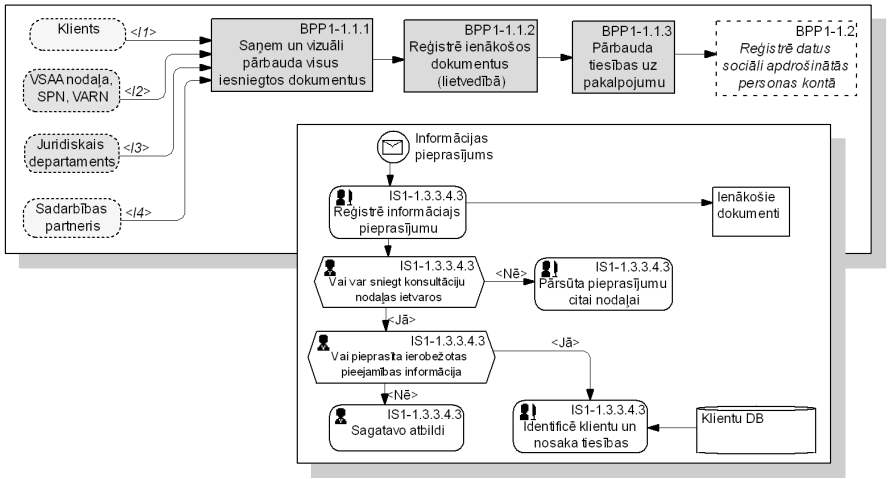


**Figure 3.** Diagrams for employees and for development of information system (real-life example)

Diagrams for development of information systems have richer set of graphical symbols. Level of detail is higher – professional information system designers must have no difficulties to design information system using these diagrams.

Both types of diagrams are elaborated by top-down decomposition, and higher level information system diagrams in most cases are subordinated to the lower-level business diagrams.

### *1.7. Using Models to Create Documentation*

Essential part of the software to be created is documentation. In most cases documentation is a set of textual documents or help files. If some model of the system operation is described, then the model itself is shown as a picture inside the document – just a visual extra. If afterwards model of operation is changed, then the textual description is modified and the picture is replaced. Usually models are referred to in many documents: requirements specifications, design description, user guides etc. In practice it often happens, that some changes are reflected in one document, but forgotten in other, yielding to discrepancies between various documents and the system itself.

We propose that models of the system must be used as the primary place to store information about the system, and that the documentation, at least partially, must be generated from the models. This ensures conformance of system, its model and its documentation.

We propose the following scenario for development of the system:

- Client defines his requirements in a form, convenient to him: as textual documents, sketches of models, formats of required reports and so on.
- System analysts create a model of the system containing both formal description (diagrams in the domain-specific language and formalized attributes) and informal description (including requirements given by client: documents, sketches, report formats…). The presence of informal descriptions enables automatic generation of the requirements specification document, containing both formal models and requirements of the client. Such a document is comprehensible to the client, even if pure models are not.
- Due to the rich informal part of requirement specifications, client quickly grasps the essence of the formal models and learns to read them. When client has gained some knowledge about the models, he can even begin to give his requirements in form of models (and in our practice this is often the case). The proportion of formal models grows, but informal descriptions are kept for better understanding.
- In software design phase models are further elaborated and design-specific information is added. The result is complete design model, from which design documentation can be generated (or at least most of it).
- Models of the design phase are used by software developers, ensuring that the same information is used by programmers, designers, system analysts and the client. Models can be used to generate applications, and can be interpreted by application during execution.
- Models contain valuable information for user guides too. If properly extended, they can be used to generate user guides automatically (at least partially), ensuring that user documentation will not be outdated by several software versions.

– If errors in system must be corrected, models are changed accordingly, and documentation is regenerated.
– Change request are supplied by a client as formal models or as informal descriptions and goes thru the above described workflow, and generation of consistent documentation is ensured.

According to this methodology the same set of models is used in all stages of software lifecycle. Of course these are not exactly the same models: initially they contain only information provided by customer, and then they are enriched and elaborated and transformed. The consequent usage of models thru the lifecycle dramatically reduce error rate when compared to use of vaguely connected set of documents for requirement specification, design and user guides. The situation if further improved, if the models are used during execution of applications.

Our approach is model-centric – the model is the central artifact used in all stages of software lifecycle, and in every stage the model is viewed from slightly different viewpoint and enriched with specific information.

## 2. Transformation-Driven Architecture and Tool Building Platform GrTP

Nowadays, when appropriate tool-building platforms are available, it would be unusual to make domain-specific tools from scratch. Therefore, we are using a tool-building platform which is developed in Institute of Mathematics and Computer Science of University of Latvia, and is called GrTP [7, 9, 10]. The universal functionality, which is common in many domain-specific tools, is implemented as part of the platform. Convenient interfaces are provided for adding features needed for specific tools. Using the tool-building platform, the first version of the domain-specific tool can be created in a short time and with reasonable effort. It can then be flexibly adapted to the needs of customers.

The recent version of GrTP is based on principles of the Transformation-Driven Architecture (TDA [7]). In this section, the key principles of the TDA and GrTP as well as their applications in implementation of DSLs are discussed.

### *2.1. Transformation-Driven Architecture*

The Transformation-Driven Architecture is a metamodel-based approach for system (in particular, tool) building, where the system metamodel consists of one or more interface metamodels served by the corresponding engines (called, the interface engines). There is also the Core Metamodel (fixed) with the corresponding Head Engine.

Every engine is responsible for providing some specific functionality to the end-user (for example, a possibility to work with graphical diagrams). Every engine is working only within the boundaries of its corresponding metamodel. That kind of independence allows one to develop and test engines separately thus improving the quality of software. Engines can work together because of their need to support so called “Events – Command” interconnection mechanism, defined by the Head engine. Events provide information about actions of the user, which are stored as instances of the metamodel of the particular engine. Commands provide information about changes in metamodel and requested actions.

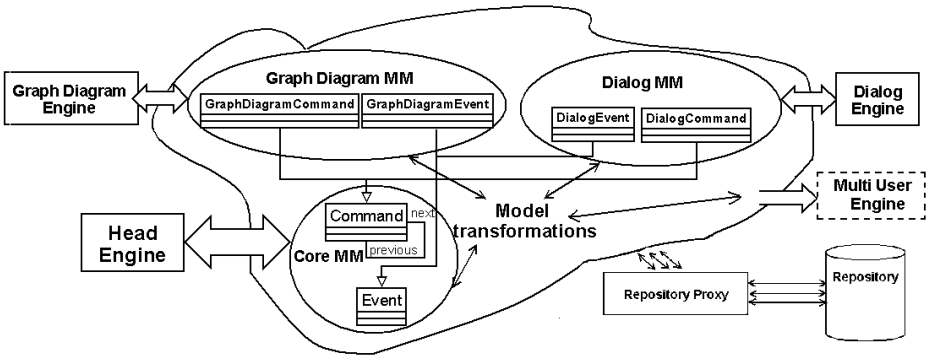


**Figure 4.** Transformation-driven architecture framework filled with some interfaces

Since every engine works only within the boundaries of its metamodel, some tools for connecting these metamodels are needed. Model transformations are used for this purpose. Moreover, using of metamodel transformations is the second basic principle of TDA. Transformations are processing events created by engines. They accomplish changes in metamodel and data and create commands that will subsequently be executed by engines. For the execution of transformations to be successful, it is often needed to add new classes and associations to the metamodel.

It is also possible to connect external engines, which either do not have interface metamodel, or stores only partial data in their metamodel. These engines are called by model transformations using one-way calls.

### *2.2. TDA-based Tool Building Platform GrTP*

Using the TDA approach, we have developed a concrete tool building platform called the GrTP by taking the TDA framework and filling it with several interfaces. Besides the core interface, two basic interfaces have been developed and plugged into the platform in the case of GrTP:

- The graph diagram interface is perhaps the main interface from the end user's point of view. It allows user to view models visually in a form of graph diagrams. The graph diagram engine [7] embodies advanced graph drawing and layouting algorithms [16] as well as effective internal diagram representation structures allowing one to handle the visualization tasks efficiently even for large diagrams.
- The property dialog interface allows user to communicate with the repository using visual dialog windows.

When building domain-specific modeling language for SSIA, it was necessary to create two external engines, which do not use events and commands, but are called by means of model transformations:

- Multi-user engine is based on information of project and graph diagrams and ensures that many users can work together with the same model (one common model on the server and separate local models for different users). Only one user is allowed to edit a diagram at any given moment. Multi-user engine uses its metamodel to save information about correspondence of server and local models.

– Microsoft Word engine is completely external module. It generates Word documents according to data of the metamodel by calling model transformations.

The final step is to develop a specific tool within the GrTP. This is being done by providing model transformations responding to user-created events. In order to reduce the work of writing transformations needed for some concrete tool, we introduce a tool definition metamodel (TDMM) with a corresponding extension mechanism. We use a universal transformation to interpret the TDMM and its extension thus obtaining concrete tools working in such an interpreting mode.

### *2.3. Tool Definition Metamodel*

First of all, let us explain the way of coding models in domain specific languages. The main idea is depicted in Figure 5. The containment hierarchy *Tool* → *GraphDiagramType* → *ElementType* → *CompartmentType* (via base link) forms the backbone of TDMM. Every tool can serve several graph diagram types. Every graph diagram type contains several element types (instances of *ElementType*), each of them being either a box type (e.g., an *Action* in the activity diagram), or line type (e.g., a *Flow*). Every element type has an ordered collection of *CompartmentType* instances attached via its base link. These instances form the list of types of compartments of the diagram elements of this type. At runtime, each visual element (diagrams, nodes, edges, compartments) is attached to exactly one type instance.

The extension mechanism is a set of precisely defined extension points through which one can specify transformations to be called in various cases. One example of a possible extension could be an "elementCreated" extension providing the transformation to be called when some new element has been created in a graph diagram. Tools are being represented by instances of the TDMM by interpreting them at runtime.

Therefore, to build a concrete tool actually means to generate the appropriate instance of the TDMM and to write model transformations for extension points. In such a way, the standard part of any tool is included in the tool definition metamodel meaning that no transformation needs to be written for that part.

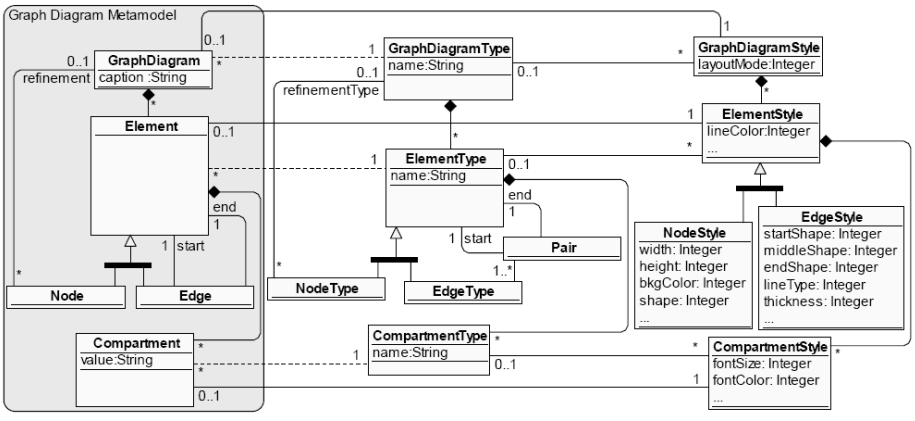


**Figure 5.** The way of coding models

## 3. Some Applications of Domain-specific models

Business process modeling is not an end in itself – models are built to make high-quality and convenient information systems. Sensitivity of social security and frequent changes in regulations (see Section 1.1) require high reliability, flexibility and maintainability of software. Traditional method of software developments have been used for years and have not yielded desired results. Using specifications in natural language, it was impossible to achieve needed accuracy and unambiguity. We have proposed modeling with domain-specific graphical language ProMod as a solution. Applications of the modeling that are the most urgent in SSIA are described below.

**Availability of models to the wide spectrum of users.** Concise description of business processes in graphical diagrams can be used as instructions for employees providing customer services and as information for clients, showing what will be done in SSIA, in order to serve their requests. The best way to spread the models is to make them available on the internet.

ProMod can export diagrams, corresponding information from structured lists and descriptive documents to Web pages. Thought not every diagram is suitable for every reader, and models with varying level of details and models from different perspectives must be built – for example in-depth description for employees of SSIA and simplified version for customers.

**Job descriptions for SSIA employees.** Job responsibilities for many SSIA employees in fact are defined by activities in the business process models – in ProMod Customer Service Diagrams are especially designed to show this. Business process diagrams must be used in job descriptions to make them more concise and easy to read compared to textual instructions. We have conducted survey, which shows ([6]), that 90% of employees in government institutions prefer graphical descriptions to textual. We believe that job descriptions for most of the SSIA employees will be covered by Customer Service Diagrams.

**Software requirement specifications.** Contradiction between inaccurate and changeable requirement specifications, defined in natural language, and need for high-quality information systems is well-known and has already been discussed in this paper. We believe that domain specific business modeling (especially, using ProMod Information System Diagrams) will largely improve situation in SSIA.

**Conversion from models to applications.** We believe that approach: “Less technical programming, more concise specifications”, and development of information systems without technical programming can become possible in the nearest future. Modeling is the first and mandatory stage in this process. In order to use information from business models in applications (no matter, whether they are generated from models or coded manually), it is necessary to transfer this information automatically or manually from repository of the modeling tool into application database. Manual transferring involves re-entering information about objects and their connections, and linking this information to the application data. This is a monotone and quite error-prone job. Transferring information with automated tools would be more efficient. This kind of transformation can be implemented with minor resources, if the modeling tool provides application programming interface to access its repository. So application can work according to models created in graphical language, but its quality (usability, reliability, security, performance etc.) remains independent of capacity of some hypothetic generator to generate a high-quality applications.

This approach has been tested in a number of medium-size projects [6], where information systems are less complex than in SSIA. In SSIA this is a next step to be taken. This approach has proved noticeable viability, and attitude of users towards the graphical models as requirement specifications and as core of user guides was surprisingly positive. Users considered graphical diagrams as highly comprehensible and soon gave up reading thick and boring manuals. Admittedly this approach asks for further development, but we see this as a realistic way to develop user-friendly, flexible and reliable information systems.

**Formal model as testing model.** In software testing the model is often emphasized, according to which testing of the system begins already in requirements specification phase by accumulating test cases for proving compliance of software and specifications. If the formal MDA model can be built by system analysts from use cases, the use cases must contain information needed to test, whether the resulting system operates correctly. Every use case describes sequence of action during some operation, and naturally these actions are used for testing of software, developed from these models. This approach is popular in practice.

Though admittedly the testing based on single use cases can be insufficient for high software quality. Essentially higher quality of testing can be achieved using model-driven testing approach. In theory of testing the control flow graph is well-known: actions are vertices, but transitions are edges. One usage of the system is one path thru the graph. Testing of the program is said to be complete, if all transitions (edges) are traversed during some test case (criterion C1).

This approach is suitable also for business processes and other behavioral models. The system can be considered as sufficiently tested, if all possible sequences of actions (according to the model of the system) are tested. As testing according to C1 can be too laborious, testing sometimes is restricted to use cases accumulated during requirement specification phase, but this is clearly much weaker approach.

## 4. Conclusions

The following conclusions can be made from our experience with creating domain specific language ProMod and with business process modeling in SSIA:

- Business process modeling with domain-specific language is preferable, compared to modeling with general-purpose language.
- Domain specific models have wide application: they can be used as core of job descriptions and requirement specification, as source of information for automatic generation of applications, as testing model for model driven testing.
- With tool building platform GrTP domain-specific languages and supporting tools: graphical editor, consistency checker and model-to-application information transfer utility, can be created in short time and with modest resources.
- Business process modeling with consistency checks in a very short timeframe allows to identify contradictions and bottlenecks of processes descriptions
- Move to model-driven architecture profoundly changes information system development technology. If an information system has been developed with traditional methods, serious modifications and enormous resources can be needed.

Practical experience in business processes modeling in large and complex government institution SSIA, confirms feasibility and advantages of model-driven development of information systems.

## Acknowledgments

This research is partly supported by European Social Fund.

## References

1. UML, http://www.uml.org.
2. BPMN, http://www.bpmn.org.
3. MDA Guide Version 1.0.1. OMG, http://www.omg.org/docs/omg/03-06-01.pdf.
4. Oracle Designer, http://www.oracle.com/technology/products/designer/documentation.html
5. Flore, F.: MDA: The Proof is in Automating Transformations between Models. OptimalJ White Paper. http://www.dsic.upv.es/~einsfran/mda/modeltransformations.pdf
6. Cerina-Berzina, J., Bicevskis, J., Karnitis, G.: Information systems development based on visual Domain Specific Language BiLingva. In: Preprint of the Proceedings of the 4th IFIP TC 2 Central and East Europe Conference on Software Engineering Techniques, CEE-SET 2009, Krakow, pp. 128-137. (2009)
7. Barzdins, J., Cerans, K., Kozlovics, S., Rencis, E., Zarins, A.: A Graph Diagram Engine for the Transformation-Driven Architecture. In: Proceedings of the IUI'09 Workshop on Model Driven Development of Advanced User Interfaces, pp. 29-32, Sanibel Island, USA (2009)
8. Lankhorst, M., et al.: Enterprise Architecture at Work: Modelling, Communication and Analysis. Springer (2009)
9. Barzdins, J., Zarins, A., Cerans, K., Grasmanis, M., Kalnins, A., Rencis, E., Lace, L., Liepins, R., Sprogis, A., Zarins, A.: Domain Specific languages for Business Process Management: a Case Study. In: Proceedings of DSM'09 Workshop of OOPSLA 2009, Orlando, USA (2009)
10. Barzdins, J., Kozlovics, S., Rencis, E.: The Transformation-Driven Architecture. In: Proceedings of DSM'08 Workshop of OOPSLA 2008, pp. 60—63, Nashville, USA (2008)
11. Larman, C.: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Prentice Hall (2004)
12. Weiklens, T.: Systems Engineering with SysML. Morgan-Kaufman OMG Press (2007)
13. White, S.A., Miers, D., Fischer, L.: BPMN Modeling and Reference Guide. Future Strategies Inc. (2008)
14. Lan Cao, Balasubramaniam Ramesh, Matti Rossi: Are Domain-Specific Models Easier to Maintain Than UML Models? In: IEEE Software, vol. 26, no. 4, pp. 19--21 (2009)
15. ICAM Architecture Part II - Volume IV - Function Modeling Manual (IDEF0). http://handle.dtic.mil/100.2/ADB0624
16. Freivalds, K., Kikusts, P.: Optimum Layout Adjustment Supporting Ordering Constraints in Graph-Like Diagram Drawing. In: Proceedings of The Latvian Academy of Sciences, Section B, vol. 55, No. 1, pp. 43–51, Riga (2001)