A Domain Specific Language for Project Execution Models

Eugen Wachtel, Marco Kuhrmann, Georg Kalus

Technische Universität München, Institut für Informatik – I4
Boltzmannstr. 3, 85748 Garching, Germany
{wachtel|kuhrmann|kalus}@in.tum.de

Abstract: The modeling of strategies for deriving valid project plans is a core task in development process's design. Strategies are used for planning and for the validation of concrete project plan-instances. Especially if using delivery-oriented planning, project managers need to know if they have all required deliverables available or if a scheduled project satisfies the process's requirements. In this paper, we present a domain specific language (DSL) to model and validate a project plan as well as its execution. The DSL will support process engineers during the process design in process improvement or introduction projects. To give a relevant example, we introduce a Microsoft DSL-based language for the V-Modell XT 1.3 describing its project execution strategies (PES) and show a prototype for their design and analysis using the specified DSL.

1 Introduction

Software-intensive systems tend to grow in size and complexity. Along with this development, the size and complexity of development projects increases, too. Software is no longer developed by a single person and also not in a rather ad-hoc fashion. As distribution of projects becomes relevant, most current software development projects need careful planning in order to be successful [BE+06, Sta06]. Capability and maturity certification schemes such as CMMI [CK+06] reflect this necessity and require a project plan for a certification. There are numerous techniques to establish a project plan such as the *precedence diagram method* and similar methods such as *critical path analysis* and *Gantt charts*. Such techniques are described e.g. in [PMI04] and in [Bur03]. There are also various tools that implement these techniques – the most prominent example perhaps being Microsoft Project [CJ07]. These methods and tools support the construction of *valid* project plans in the sense that for example a developer's workload does not exceed his capacities and that there are no mutually exclusive dependencies between work items that would prevent the project plan from ever being executable.

What these project planning methods and tools do not say is *what* actually has to be planned for. The framework for what has to be done is usually laid by the project's development process. Not only does a development process state what has to be done –

depending on where it lays on the scale between agile and heavy weight methodologies – it sometimes also defines in what order this should happen.

1.1 Agile vs. structured planning methods

To anticipate uncertainty inherent in software development projects, agile methods such as [Coc01] and [Sch04] tend to not state much more than to keep planning cycles short and to arrive at certain predefined goals after every cycle (time-boxing, incremental and iterative development etc.). As such, the connection between the project plan and the development process is vague at most.

Heavier and more formally defined development processes such as the V-Modell XT [VMXT] and the Rational Unified Process [Kru00] (and other SPEM-based process models [SPEM]) do state a lot more explicitly what has to be done in a project – and in what order. Here, the project plan can be *derived* from the development process model. Here is, at least in theory, a very clear connection between the elements of the process model and the items in the project plan.

In the V-Modell XT, the connection is established by what is called a *project execution strategy*. A project execution strategy (PES) can be understood as a template for project plans. It defines the sequence in which decision gates have to be passed. As the V-Modell XT also defines connections between *work products* and so-called *decision gates*, project execution strategies and dependent process model elements clearly define which work product has to be finished at what stage in the project. A meta-model defines the strategies, which is highly modular and hierarchical, supporting a flexible configuration. As a downside, this makes the modeling of project execution strategies rather complicated for the process engineer. Complexity is aggravated by the fact that a PES may contain variations, which can be customized for a specific project and thus are uncertain at design time. Therefore, the development of the PES can be fault-prone.

The V-Modell XT comes with a number of predefined strategies. These may not always be suitable to a particular project's needs. The development of a custom PES so far requires detailed knowledge of the V-Modell XT's meta-model [TK09].

The tools coming along with the V-Modell XT so far do not well support the creation of a custom PES. Process engineers have no graphical design mode and need to realize a PES using the native XML structure tree. They are not assisted to decide whether to reuse an existing PES-element or to realize a new one. If assembling a new PES, there is no support to check whether the realization is correct – meaning: Are all model-elements correctly connected? Is a newly designed PES-element consistent related to other elements, which it is connected to? Are all cardinalities (for parallel execution) correctly set? Is the design of a PES-module cycle-free? These and some other problems may occur during the realization – but the V-Modell XT Editor provides no feedback to the process engineer. His current tools to check whether the model and its realization are correct are the export – and especially the integrated drawing module – and the planning module included in the Project Assistant. The first denies the export if there is an error in

the realization. The second one generates invalid schedules if there is an error in the design. Those tools are purely *reactive* and *analytic*. There is no constructive support, yet.

The most recent *Project Assistant*, a tool accompanying the V-Modell XT, contains a graphical editor allowing the creation of project plans that adhere to a given project execution strategy. This is a huge leap forward compared to older versions of that tool which only had very rudimentary support for project planning. However, the template to adhere to – the project execution strategy – has to be defined *beforehand*. It also has to be pointed out that the construction of a project plan with this tool is a one-way operation. Once a project plan is exported, there is no way of checking it's conformity with the PES that it was based on.

1.2 Organization of the Paper

In this paper we present a domain specific language (DSL) to model and validate project execution strategies as well as the corresponding project plans. The domain specific language is implemented in a prototypical tool that can visually display project execution strategies and allows the creation of custom PES with a graphical designer. So it targets process engineers that suffer from using the reference editor and demand more support for the design of a PES. Most importantly, DSL and the prototype ensure the validity of the created PES. It also allows the creation of project plans based on a given PES. Furthermore, a project plan that is described in the domain specific language presented here can be checked for consistency against project execution strategies.

The paper is organized as follows: In section 2 we give a short introduction to the domain model by introducing the domain specific language with its two facets to model project execution strategies and project plans. In section 3 we give an overview about how the domain specific language was implemented using the Microsoft DSL-Tools. We then present the prototypical editor for the domain specific language and highlight a couple of noteworthy implementation details. The paper is concluded in section 4 with a discussion of the presented concept and an outlook to future enhancements.

2 Analysis and Design

In this section we describe the domain model structure (sec. 2.1) for defining project execution strategies (sec. 2.2) as well as project plans (sec. 2.4) and examine the extensions or restrictions (sec. 2.3) that are necessary compared to the V-Modell XT to ensure a valid model by disallowing inconsistent strategies.

2.1 The domain model

The domain model structure of our DSL conceptually consists of two models:

1. *StrategyModel*: This describes how project execution strategies are defined and configured. Therefore it itself consists of two important parts, the first introducing the elements needed to create strategies and the second configuring the implemented strategies to only include the required of all possible workflows. The *StrategyModel* is highly based on the V-Modell XT and therefore for the most part is identical to the V-Modell XT meta-model [TK09] structure for project execution strategies and project types.

2. *PlanModel*: The required elements for defining project plans are described in the *PlanModel*.

To summarize our concept, consider the rather simplified presentation of the domain model in Figure 1. There, the interrelationship between the models can be found in a reference from a project plan to a project type variant. We will explain the two models with their corresponding elements in greater detail in the following chapters, starting with the *StrategyModel* and the project execution strategy definition part.

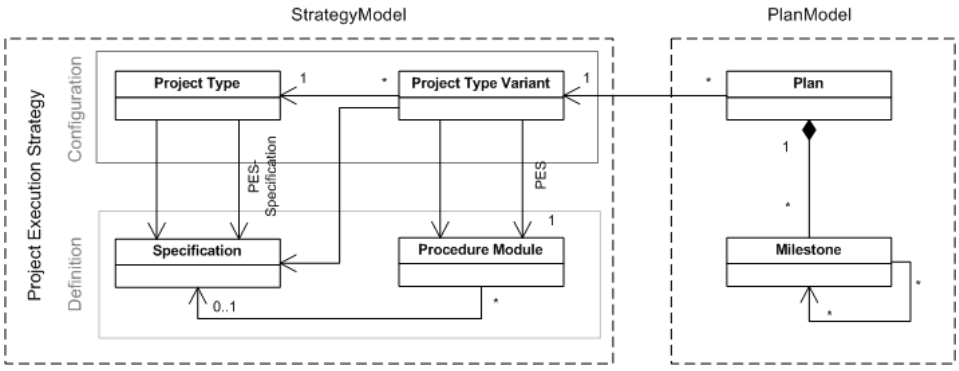

Figure 1 : Domain model structure (simplified)

2.2 Domain Specific Language for Project Execution Strategies

Definition of a PES. The strategy definition part of the *StrategyModel* as shown in Figure 2 in greater detail is based on the meta-model structure [TK09] for project execution strategies [BF09] in the V-Modell XT, which is hierarchical and highly modular. The main element of the model is a *procedure module*, which itself either specifies a *sub-procedure* or a *project execution strategy*. Every procedure module has a reference to one *specification*, which describes the requirements on a procedure module concerning contained decision gates and their sequence. Further, a procedure module consists of a number of elements: It has exactly one startpoint and one endpoint, which are necessary for hierarchical nesting. In that case, the startpoints and endpoints also indicate the entry and exit points. Between the startpoints and endpoints, different elements can be placed: *procedure specification points, procedure decision gates, splits* and *joins*. The first two items represent actual steps in the sequence; the second two are required to model parallelism:

- Every *procedure specification point* has a reference to a specification and therefore integrates further procedure modules. This accounts for a hierarchical ordering, allowing for project specific customizations, which are further specified during the configuration of a PES and thus are unclear at design time of the strategy.
- Every *procedure decision gate* has a reference to a decision gate given by the process model and therefore has to be added to a procedure module to include a decision gate in a project execution strategy. In contrary to a procedure specification point, a procedure decision gate cannot be further refined and therefore describes a concrete step in the process.
- *Splits* are planning elements, which allow creating parallel processes. Every split has exactly one split entrance and one-to-many split exits. To model parallel processes, transitions have to be first introduced between the procedure elements defined above and a split entrance. The split then serves as a starting point for the outgoing parallel processes, which are included as further transitions from split exits to additional procedure elements. To specify the cardinality of the parallelism, split outs allow setting the minimum and maximum number of possible outgoing transitions.
- *Joins* are the opposite of splits and serve the purpose of uniting parallel processes defined by splits. Every join has one-to-many join entrances and exactly one join exit. Same as a split exit, a join entrance specifies the number of minimal and maximum parallel processes it can unite.

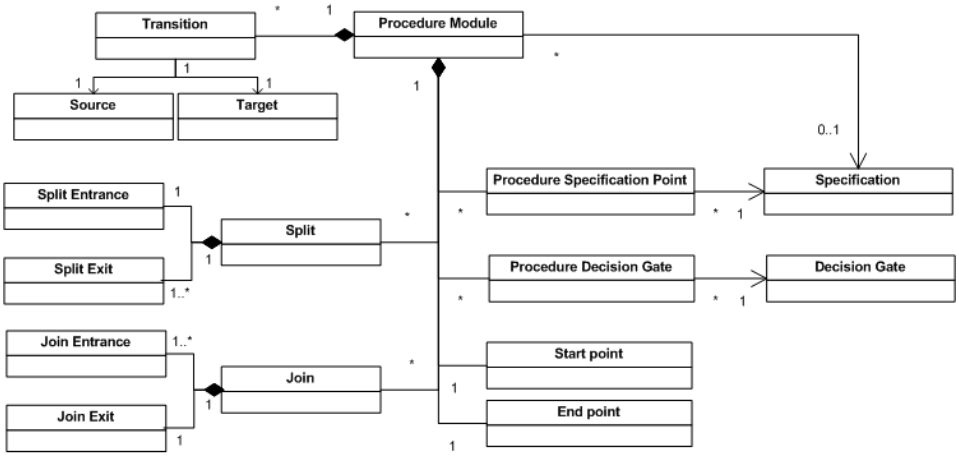

Figure 2: *StrategyModel* (strategy planning part)

The procedure elements defined above are connected through *transitions*, which are directed connections between the elements and allow for a flexible modeling of procedure modules.

Configuration of a PES. So far, we have described how project execution strategies can be defined using the elements introduced above. The so modeled strategies need to be further configured, because they allow workflows, which might not be needed or even explicitly unwanted. Therefore, the exact specifications as well as the allowed procedure modules based on that specification have to be set. To allow that, we introduce the project type definitions part of the *StrategyModel*, which configures the implemented strategies. It consists of two elements: project types and project type variants.

A *project type* defines a class of similar projects by providing guidelines stating with specifications that are mandatory to be applied and therefore have to be included in the strategy. Further, a project type has exactly one reference to a specification thas has to be fulfilled by the final project execution strategy.

A *project type variant* characterizes the project in more detail and therefore determines the concrete procedure modules of the final project execution strategy. Every project type variant references exactly one project type and one distinguished procedure module (as PES) as well as a number of specifications and procedure modules, which additionally have to be included in the project.

In summary, project types and project type variants provide the required configuration of the project execution strategies stating the possible workflows.

2.3 Validation for Project Execution Strategies

We have now introduced the domain model of our DSL, but yet have not discussed how to ensure validity of the model. Thus, we will examine the issues of the V-Modell XT in that area and propose extensions or restrictions to our domain model to guarantee validity.

To create a new project plan according to the V-Modell XT one can either use already implemented strategies during the tailoring or create a new strategy, which has to be consistent to the model. The second option requires a deep understanding of the allowed processes. The creation is neither fully assisted by the *V-Modell XT Editor*, which is the standard tool for process engineers, nor does the structure of the model enforce validity. For instance, it is possible and not explicitly disallowed by the meta-model to add a transaction from one element of a procedure module to an element of a different procedure module. Further it is allowed to create impossible paths like adding a transition from a start point to a join entrance or supplying different cardinalities for parallel processes. These errors are not discovered at design time, making the modeling of project execution strategies difficult and error-prone.

In our domain specific language we aim at validating the PES at *design time* according to the underlying DSL, for identifying inconsistencies and preventing the user from saving faulty data. To allow this, the DSL contains extensions going further than what the meta-model of the V-Modell XT defines. Some extensions were introduced to disallow inconsistencies, such as:

- A transition outgoing from a start point cannot have a join entrance as a target since no parallelism has been modeled yet (so there is none to unite).
- A transition outgoing from a split exit cannot have an end point as a target since a split creates parallel processes and needs a join to unite them.
- Transitions between a join exit and join entrances as well as between split exits and a split entrance of the same join or split element cannot be allowed since this would account for infinite paths without a defined project finish.
- A split exit cannot exist without the corresponding join entrance and vice versa. The cardinalities describing the number of parallel processes between a split exit and a join entrance cannot be specified differently.
- A transition from one element of a procedure module can never go to elements of different procedure modules.
- A procedure specification point is not allowed to have the same specification reference as the procedure module it is included in since that would account for infinite integration.
- Every path through a procedure module has to be possible in the sense that it has to begin at a start point and stop at an end point.

With the restrictions listed above it is possible to create a valid PES and check it at design time.

2.4 Domain Specific Language for Project Planning

In order to create a project plan, the PES has to be already implemented and valid. A generated project plan does not necessarily contain any information about the specific decision gates it references in the V-Modell XT. Therefore, a validation of such a plan, considering whether it suits a PES or not, cannot be done easily. In our DSL we aim at validating project execution strategies as well as project planning, which also includes the validation of generated plans against the respective strategies. To achieve this, we introduce the *PlanModel* designed for modeling project plans as shown in Figure 1.

A project plan, from our viewpoint, is given by a number of milestones and the defined transitions between them. A transition in that sense describes the sequential ordering between milestones. Every milestone defines a date and includes a reference to a decision gate in the project execution strategy. Therefore, a project plan can also be seen as a sequence of decision gates. Further, every project plan has a reference to a project type variant connecting a project plan with a configured PES and specifying the possible workflows, which allows to validate the given plan against the appropriate project execution strategy.

3 Implementation

In this section we present a prototype (*PESEditor*) for our domain specific language by introducing the framework used to create it, the Microsoft DSL-Tools (sec. 3.1), and by

further examining interesting and important technical aspects (sec. 3.2), which we observed while implementing the concepts described in section 2.

3.1 Microsoft DSL-Tools

The Microsoft DSL-Toolset [GSC04, CJ+07] provides a framework to create and distribute domain specific languages. This includes a graphical designer to define the domain model and its appropriate graphical visualization. The domain model is the DSL core and therefore implements its concepts. In that sense it consists of elements, which constitute the model as well as rules how those elements may be interconnected through relationships. Furthermore, the domain model includes definitions for the serialization, the toolbox creation and the validation process. To create a domain model, the DSL-Tools provide a domain class object, which can hold a number of properties as needed by the DSL, and three different relationship types, which can be instantiated between the domain classes:

1. *Embedding Relationship*: A domain class serves as a container for another one
2. *Reference Relationship*: A domain class references another domain class
3. *Inheritance*: A domain class is inherited from another domain class

We have now described how the meta-model information can be transformed into the domain model of the DSL-Tools. Further we will take a look at how instantiated information based on the domain model can be presented.

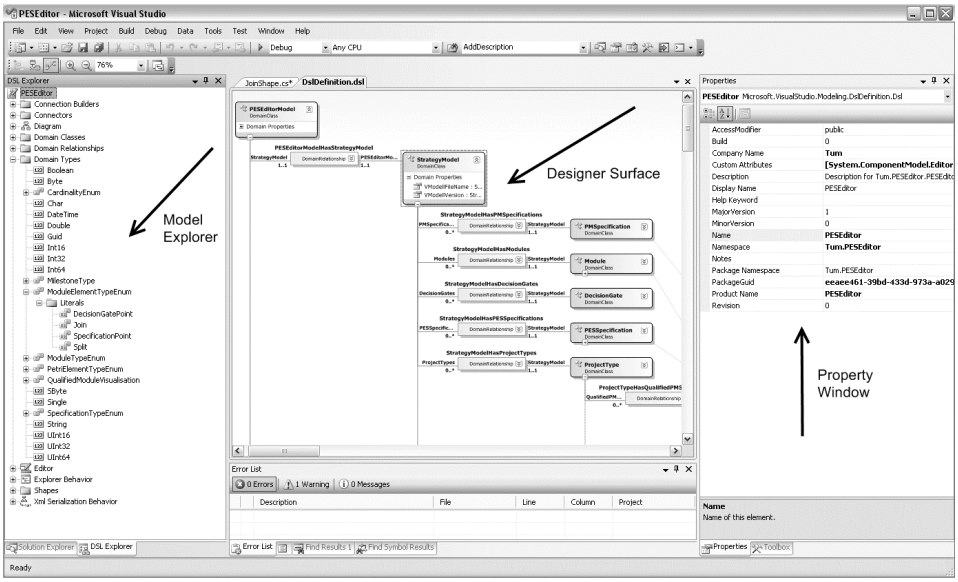

Figure 3. Visual Studio IDE for DSL design

Visualizing a DSL. The user interface of the Microsoft DSL-Tools contains three different windows to visualize and manipulate the model (Figure 3):

1. *Designer Surface*: Each element and relation of the domain model can be mapped to a specific presentation via shapes or connectors. DSL-Tools contain a number of predefined shapes like compartment-, geometry-, image- or port shape as well as connectors to present relations.
2. *Property Window*: Displays the properties of the selected model element.
3. *Model Explorer*: Displays the whole domain model in a tree structure that allows creating, editing or deleting domain classes directly through a context menu and the property window.

The definition of a model presentation has to consider a specific shape mapping and the appearances on the model explorer and on the property window.

Transformation, Validation and Serialization. So far we have described the features and concepts of the graphical editor of the DSL-Tools. After defining a DSL with the editor, the result is transformed into generated source code using the *Text Templating Transformation Toolkit* (short: T4 – an overview can be found here: [Syc07]). The source code can be extended by the language designer, especially because of the "double derived" feature, which allows a programmer to override methods of a domain class or any visualization class to create custom behavior.

When it comes to validation of a DSL, *soft and hard constraints* [CJ+07] have to be considered. Soft constraints are constraints that can be violated at some point where hard constraints are prevented from ever being violated. DSL-Tools provide support for both types of constraints by source code. For soft constraints, new methods have to be added on domain classes, for hard constraints custom code has to be added to the designer.

Serialization in the DSL-Tools uses a *domain specific serializer*, which is automatically generated. It loads and saves the domain model and the diagram presentation information in two different files (structured in a domain-specific XML-Format). For more information on the DSL-Tools refer to [CJ+07].

3.2 PESEditor

To implement the prototype for our DSL, we have to specify a transformation for the *StrategyModel* as well as for the *PlanModel* and introduce the appropriate presentation elements. This can be done using the domain classes we introduced above. Further, we have to define a mapping between the domain classes and their appropriate presentation shapes, as shown in Figure 4.

The procedure module and the project execution strategy shapes are so called *container shapes*. They host elements. The other shapes are *nested child shapes* and cannot be moved out of their container shape (this behavior is not completely supported by the DSL-Tools and needs to be implemented by custom code).

Implementation challenges. During the implementation of the prototype for our DSL we encountered several interesting problems, which are listed below. Some of these and the chosen solutions will be discussed in more detail below.

- Multiple reference relationships vs. source code constraints containment
- Custom serialization of the domain model
- Separation of data and presentation
- Multiple views on one domain model (multiple diagrams)
- Container and nested shapes
- Custom validation
- Drag & Drop from the Visual Studio Toolbox and from the Model Explorer
- Visualization of a virtual connection (there is no real reference relationship instantiated here) between two elements, which contain different reference relationships to the same element
- Automatically generated Layout Manager
- Copy and Paste

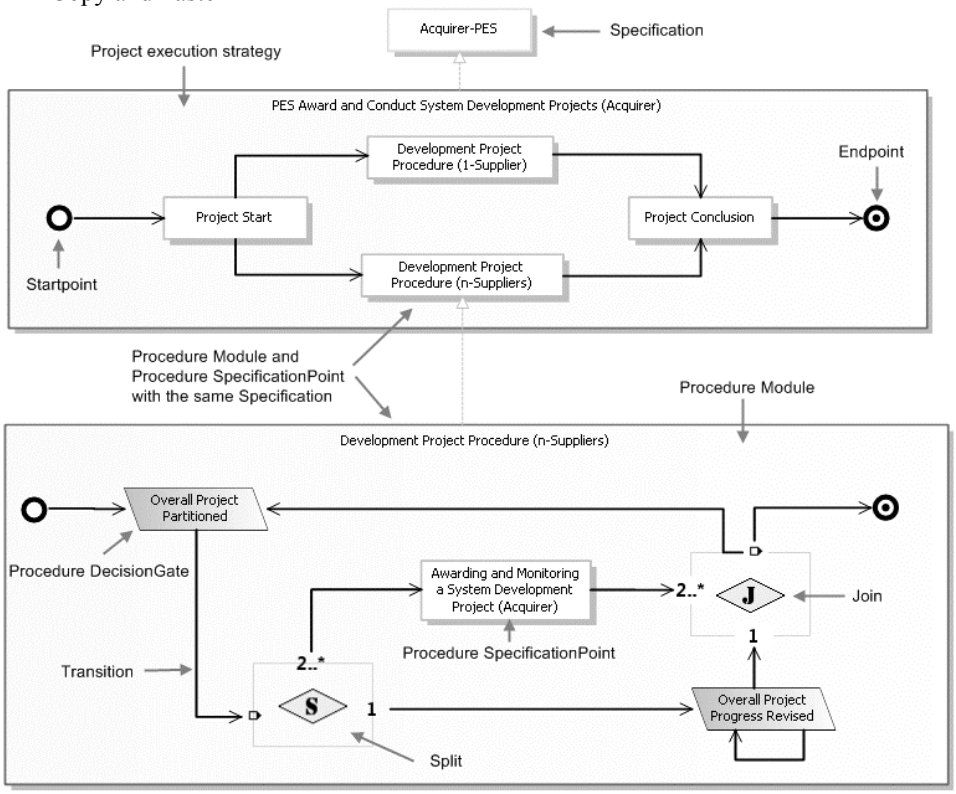

Figure 4: PESEditor presentation elements

Relationships. During the transformation of the domain meta-model into the DSL-Tools domain model according to the defined modifications in section 2, we encountered that to model the described extensions/restrictions we had to add 13 new reference relationships. This obviously accounts for extensive overhead in the modification, e.g. needed while serializing the model. The solution was to create a new domain class `BaseLinkElement` and to inherit all domain classes that could be referenced from that class. Further, a connection between `BaseLinkElements` was introduced and mapped to a

connector for presentation. To achieve the constraints, the method `CanCreateConnection` had to be extended to decline impossible and incorrect references:

```
public override bool CanCreateConnection(ShapeElement sourceShapeElement,
        ShapeElement targetShapeElement, ref string connectionWarning) {
    if (sourceShapeElement is StartpointShape ||
        targetShapeElement is PortShapeJoinIn)
            return false;
            …
    return base.CanCreateConnection(sourceShapeElement,
                                    targetShapeElement, ref connectionWarning);
}
```

Serialization. Thus our DSL is designed to create, edit and display project execution strategies, we had to change the serialization to load and save the domain model data from and to the V-Modell XT xml-definition/model file. In order to do that, we had to write a customized serializer for the domain model. The serializer uses the DSL-file to save properties as well as data, which couldn't be saved in the V-Modell XT file, but were essential for the DSL to function (e.g.: the path to the VMX xml-file). To save and load the diagram data however, we still could use the generated serializer. The main problem we encountered was the conversion of the V-Modell XT *id* attribute (first 13 to 15 byte of a `Guid`) to the DSL-Tools *id* (32 byte `Guid`) and vice versa. This is not an issue while serializing the domain model itself, as those *id*s can be easily converted there. If adding new elements to the domain model, the *id* of the new element is generated automatically by the DSL-Tools and thus is read-only and a 32 byte `Guid`. The solution to this problem was to add a new domain class `BaseElement` and to inherit every other domain class, which needed to be saved later, from `BaseElement`. So whenever a domain class is added to the domain model, its *id* is created as a V-Modell XT id and converted to a `Guid` automatically:

```
public BaseElement(Partition partition, PropertyAssignment[]
        propertyAssignments) : base(partition,
        !IsPropertyAssignmentsArrayEmpty(propertyAssignments) ?
        propertyAssignments : CreateKeyAsPropertyAssignment())
{ … }
```

`IsPropertyAssignmentsArrayEmpty` checks whether the `propertyAssignments` array is empty or not, which is the case when new elements are added to the model. `CreateKeyAsPropertyAssignment` creates a new V-Modell XT *id*, converts it to a 32 byte `Guid` and returns it as a `propertyAssignment`.

Separation of data and presentation. During serialization all available data from the V-Modell XT file are loaded. But there is no need to display all the defined procedure modules, decision gates or specifications on the diagram user interface. However, the DSL-Tools automatically adds shapes once a domain class is instantiated and has a mapping to a specific shape.

To solve this problem, we had to overwrite the diagrams `ShouldAddShapeForElement` and `CreateChildShape` methods and disable the automatic creation of shapes for domain

classes only. Further, to still be able to add elements from the toolbox or from the Model Explorer onto the diagram, the methods `OnDragOver` and `OnDragDrop` had to be overwritten. The interesting part here is that every drag & drop operation from the Model Explorer required to only create the corresponding shape whereas the drag & drop from the toolbox added new domain elements and also used different shapes for the same domain class (procedure module displayed either as a project execution strategy or a sub-procedure module). Therefore, a possibility to recognize the kind and distinguish the source of the drag & drop operation was needed. This was done using the `ModelingToolboxItem`, which represents a toolbox item in the Visual Studio default toolbox. It has the property `Prototype.UserData` that we needed to fill with the toolbox item name to be able to recognize it. After having done that, we utilize the solution in the `OnDragDrop` method:

```
public override void OnDragDrop(DiagramDragEventArgs e)
{
        if (e.Prototype) != null)
          if (e.Prototype.ProtoElements[0].DomainClassId == DomainClassId &&
              e.Prototype.UserData.ToString() == "ToolboxItemName") {… }
}
```

Multiple views. On a domain model multiple diagrams can be used for visualization. In order to display the project execution strategies and the plan modeling in our prototype, we needed to extend the user interface to allow us to create multiple diagrams. Thus, we can use one diagram to create and edit project execution strategies and another one to define project plans, which allows for a very flexible and user-friendly way of processing. The ideas and source code behind the implementation of multiple views on one domain model can be found in [RG09] and seen in display in Figure 5.

Prototype example. To show an example Figure 5 illustrates a graphical representation of a project execution strategy and a simple project plan as modeled using the actual prototype. The PES displayed here is the "*Introduction and Maintenance of an Organization-specific Process Model*" [VMXT] (upper part). Looking closely at the strategy definition, a missing endpoint can be observed. This is of course detected during the validation process, which results an error list that is shown in the bottom part of the figure. Furthermore a representation of a simple plan is given in the middle part of the figure, which is obviously consistent to the defined PES. The prototype can be used to find inconsistencies and give feedback to the user as displayed in the error list.

3.3 Microsoft DSL-Tools – Lessons learned

Now we want to discuss the advantages and the disadvantages of the DSL-Tools framework as we have encountered during the implementation of our prototype.

The DSL-Tools' main advantages can be found in the abstraction level and in the flexibility and consistency of the framework. The creation of DSLs with the DSL-Tools is less error-prone, as the designer itself provides validation while building the model. Furthermore, DSL-Tools are well-integrated into Visual Studio allowing for the creation of inte-

grated, domain-based editors. Such editors can be used inside the Visual Studio environment to increase productivity.

The disadvantages can be found on the one hand in the needed learning curve and on the other hand in the missing features, which need to be implemented through source code (e.g.: copy & paste; multiple diagrams on one domain model). Many of those features are being worked on and will be implemented in the upcoming version of the DSL-Tools (Visual Studio 2010 SDK [Pri09]).

Despite the listed disadvantages, the DSL-Tools provide a useful framework for creating domain specific languages, which is completely suitable for a great number of models. From our experience, the use of the DSL-Tools in the domain specific development environment can be strongly recommended.

4 Conclusion and Future Work

In this paper we presented a DSL to design and validate project execution strategies and plans. The validation of PES is done at design time. This prevents the process engineer from creating invalid strategies, disallowing the definition of incorrect transitions and providing a validation process to find impossible paths as well as other inconsistencies. The typical use case is located in the process definition phase. A process engineer can validate the strategies under design immediately and in-place and need not to try an export of the process documentation to detect errors. On top of that, not only errors are found and reported as: “PES is invalid” (as done by the reference tools) but the exact cause and origin of the error is provided, too. This real-time feedback, as well as the DSL itself, allows a process engineer to create a new PES by either adjusting or reusing existing ones, or to create completely new ones.

Furthermore, if a project plan is missing, one can be derived using this tool, too. The creation of project plans with the current prototype allows the validation concerning the order of specified decision gates against the corresponding PES. Whenever a plan is not derived according to a strategy, feedback can be given to the user. The feedback can provide information about the last decision gates that could be found in the strategy in the same order as in the plan. This allows the user to restrict his search for missing decision gates based on the feedback. In future work, methods and algorithms can be introduced to propose a solution, suggesting meaningful changes to the plan so that it becomes valid concerning its strategy. The definition of a project plan with the prototype expects a reference to one decision gate per milestone. As all validations for project plans are based on that information, some “real world” plans are currently excluded; hence those are missing references to the corresponding decision gates. Two developments could be argued here to improve the situation:

- First, an extension could be introduced that allows the assignment of missing references for an imported plan.
- Second, methods and algorithms could be developed to actually find and allocate the missing references automatically based on e.g. neighbor analysis.

The validation algorithms for project plans against strategies also open up a new perception on the process itself. Currently, we first implemented a concept and used it to create a plan. The opposite direction, namely the creation of strategies from a given project plan is also an interesting direction in the future. This way, successful plans could be used to create project execution strategies automatically (e.g. via interpolation). Given required algorithms and methods, the tool could be extended to support that kind of processing.

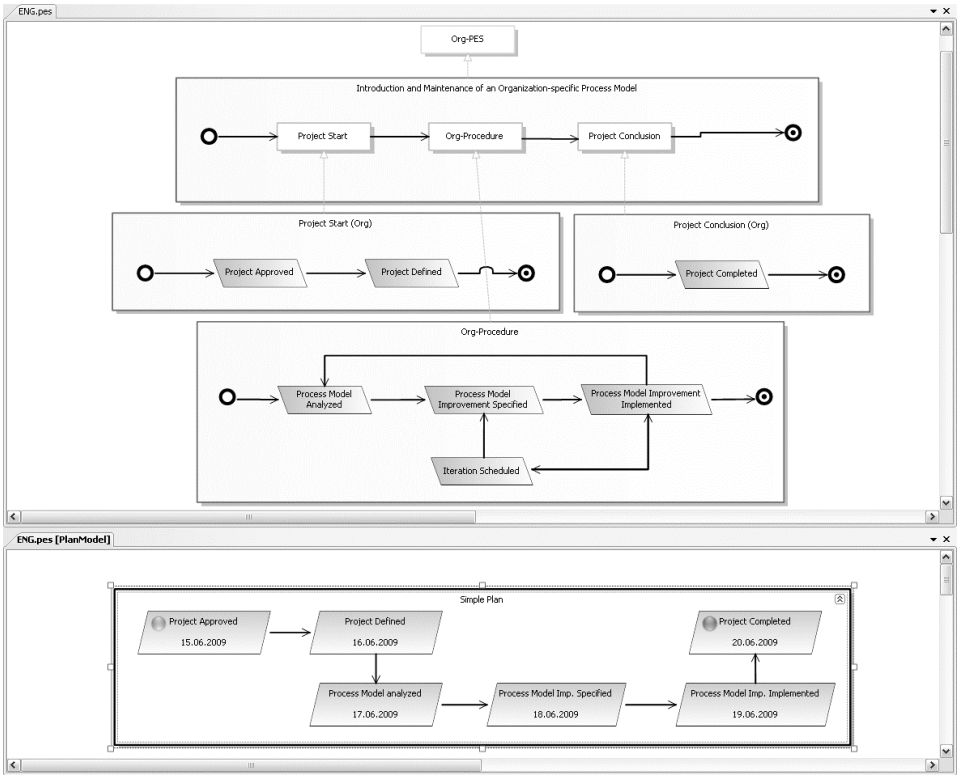

Figure 5: A Project Execution Strategy and a simple plan

The implemented prototype serves as a technology study. It consumes a lot of effort rendering the extension of that kind of technique to the V-Modell XT as a whole too costly. Therefore, a new domain specific language could be introduced to design domain specific languages for the V-Modell XT, allowing higher levels of abstraction and reducing the overhead while implementing different sub-languages. This could also include providing a fully implemented DSL-based editor for the whole process.

References

[BE+06] Buschermöhle, R., Eekhoff, H., Josko, B.: *SUCCESS - Erfolgs- und Misserfolgsfaktoren bei der Durchführung von Hard- und Software-Entwicklungsprojekten in Deutschland*, Offis e.V., Ed. Oldenburg, Deutschland: BIS-Verlag, 2006

[Sta06] The Standish Group, *CHAOS Report*, USA, 2006

[CK+06] Chrissis, M. B., Konrad, M., Shrum, S.: *CMMI: Guidelines for Process Integration and Product Improvement*, ISBN-13: 978-0321279675, Addison-Wesley, Amsterdam, 2006

[CJ07] Chatfield, C., Johnson, T.: *Microsoft® Office Project 2007 Step by Step*, Microsoft Press, 2007

[Coc01] Cockburn, A.: *Agile Software Development.: Software Through People*. Amsterdam, NL: Addison-Wesley Longman, 2001

[Sch04] Schwaber, K.: *Agile Project Management with Scrum*. Redmond, WA, USA: Microsoft Press, 2004

[VMXT] Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung*, V-Modell XT Online Portal*, http://www.v-modell-xt.de/

[Kru00] Kruchten, P., *The Rational Unified Process: An Introduction (2nd edition)*, ISBN-13: 978-0201707106, Addison-Wesley Professional, USA, 2000

[SPEM] Object Management Group, *Software Process Engineering Metamodel*, http://www.omg.org/technology/ documents/formal/spem.htm

[PMI04] Project Management Institute: *A Guide to the Project Management Body of Knowledge*, ISBN-13: 978-1930699458, Project Management Institute, 2004

[Bur03] Burke, R.: *Project Management Planning and Control Techniques*, ISBN-13: 978-0958239158, Burke Publishing, 2003

[CJ+07] Cook, S., Jones, G., Kent, S., Wills, A. C.: *Domain-Specific Development with Visual Studio DSL Tools*, 2007

[RG09] Recchia, P., Guerot, A.: *Multiply Dsl points of view*, http://www.netfxfactory.org/blogs/papers/archive/2009/01/13/multiply-dsl-points-of-view.aspx, 2009

[TK09] Ternité, T., Kuhrmann, M.: *Das V-Modell XT 1.3 Metamodell*, Technical Report, Technische Universität München, TUM-I0905, 2009

[Syc07] Sych, O.: *T4: Text Template Transformation Toolkit*, http://www.olegsych.com/2007/12/text-template-transformation-toolkit/, 2007

[BF09] Bergner, K., Friedrich, J.: *Modulare Spezifikation von Projektabläufen*, Technical Report, Technische Universität München, (to appear) 2009

[GSC04] Greenfield, J., Short, K., Crupi, J.: *Software Factories*, Wiley & Sons, 2004

[Pri09] Jean-Marc Prieur - Announcing the Visual Studio 2010 DSL SDK Beta 1 http://blogs.msdn.com/jmprieur/archive/2009/05/22/announcing-the-visual-studio-2010-dsl-sdk-beta-1.aspx