Towards Transformations from BPMN to Heterogeneous Systems

Tobias Küster and Axel Heßler

DAI-Labor, Technische Universität Berlin
Faculty of Electrical Engineering and Computer Science
{tobias.kuester|axel.hessler}@dai-labor.de

Abstract. By now, the mapping from BPMN to BPEL has been implemented in numerous tools, greatly assisting the business architect in the creation of BPEL processes. However, most of these tools are tailored especially for this transformation, neglecting the original purpose of BPMN: Providing a language independent process model. To address this shortcoming, a pure BPMN editor is needed, being dynamically extensible with several export features and added editing functionality. In this paper, we present a tool that follows this approach, not only providing a compelling transformation to BPEL but at the same time being extensible to other languages.

Key words: BPMN, Process Design Tools, Model Driven Development, Transformations, Multi-agent Systems

1 Introduction

The goal of process modelling, as of Model Driven Engineering in general, is to provide an abstract view on systems, and to design those systems in a language and platform independent way. For that purpose the Business Process Modelling Notation (BPMN) [10] has been standardised by the Object Management Group. It can be understood intuitively by all business partners, even those who have great knowledge in their domain but do not know too much about Service Oriented Architecture (SOA) or programming in general. At the same time, BPMN is formal enough to provide a basis for the later implementation and refinement of the business process. Given a respective mapping, a BPMN diagram can be used for generating readily executable code from it. A brief introduction to BPMN is given for instance in [16].

Today, the Business Process Modelling Notation and the specified mapping to the Business Process Execution Language (BPEL) are supported by a growing number of tools — we will have a closer look on some representatives later in Section 4. However, the problem with the majority of existing tools is that while they do provide the usual transformations from BPMN to BPEL, they are focused only on this one aspect of BPMN. Often the editors and even the underlying metamodels are adapted to BPEL in many ways. While this may be desired in order to provide highest possible usability and to support the user in the creation of executable BPEL code, the consequence is that business process diagrams created with these tools can neither be transformed to other executable languages, nor can the process model be used with other tools that might provide different transformations. Thus, while process modelling and BPMN should be independent of a specific executable language, the *tools* are not.

The solution to this problem is to keep both the underlying BPMN metamodel and the diagram editor free from influences from the BPEL world and to use pure BPMN instead, so that diagrams created with such a tool will be truly independent of any concrete language — apart from what influenced the BPMN specification in the first place. Based on this, several mappings to different target languages can be implemented and integrated into the editor as plugins, which may also contribute to the editor in order to support the business architect with language-specific editing features.

Following this approach, the *Visual Service Design Tool* (VSDT) has been implemented as an Eclipse plugin, inherently providing the necessary modularity, as we will see in Section 2. For the export of BPMN diagrams to executable languages a transformation framework has been designed, which we will describe in more detail in Section 3. The transformation has been subdivided in distinct stages, so that significant parts of it are reusable, e.g. the challenging transformation of the control flow. Thus the actual mapping to a given language can be integrated in a very straight-forward way. While the usual mapping from BPMN to BPEL has been realised as a proof of concept (see Section 3.2), the main intent behind the VSDT is to provide a transformation from business processes to multi-agent systems such as the JIAC language family [14]. The respective mappings are currently under development and will be discussed briefly in Section 3.4. Our ultimate goal is to provide transformations not only in different, but also in heterogeneous systems — just like they are used in the real business world. Future work in order to achieve this goal, as well as a conclusion to this paper, will be discussed in Section 5.

2 The Visual Service Design Tool

The first version of the VSDT has been developed as a diploma thesis [7] in the course of the *Service Centric Home* (SerCHo) project at TU Berlin in early 2007. As the work continued it matured to a feature-rich BPMN editor with an extensible transformation framework and has already been used in a number of service orchestration scenarios in the context of a smart home environment,[1] one of which will be shown later in Section 3.3.

2.1 The Metamodel

The BPMN specification [10] describes in detail how the several nodes and connections constituting a BPMN diagram have to look, in which context they may

[1] `http://energy.dai-labor.de`

be used and what attributes they have to provide. However, it does neither give a formal definition of the syntax to be used for the metamodel, nor an interchange format, e.g. using an XML Schema Definition (XSD). Thus the editor's metamodel had to be derived from the informal descriptions in the specification. As it was our main concern to keep as close to the specification as possible, we decided not to reuse the existing Eclipse STP BPMN Editor, which uses a simplified model of BPMN.[2] Instead, almost every attribute and each constraint given in the specification has been incorporated into the metamodel, allowing the creation of any legal business process diagram. Still, some attributes have not been adopted in the metamodel: For instance the possibility to model nested or even crossing Lanes has been dropped, as it turned out that this feature seems to be virtually never used in practical business process design. Further, redundant attributes, such as the Gateway's `defaultGate` attribute, are emulated using getter methods to prevent inconsistency in the diagram model.

Concerning the transformation to BPEL and other executable languages, which in most cases are block-oriented, an extension to the usual BPMN metamodel has been designed, featuring equivalents to the basic block structures, such as sequences, decisions, parallel blocks and loops. These elements are described in a separate metamodel, extending the editor's metamodel. They are used only during the transformation process, especially for the mapping of the structure, as we will see in Section 3.

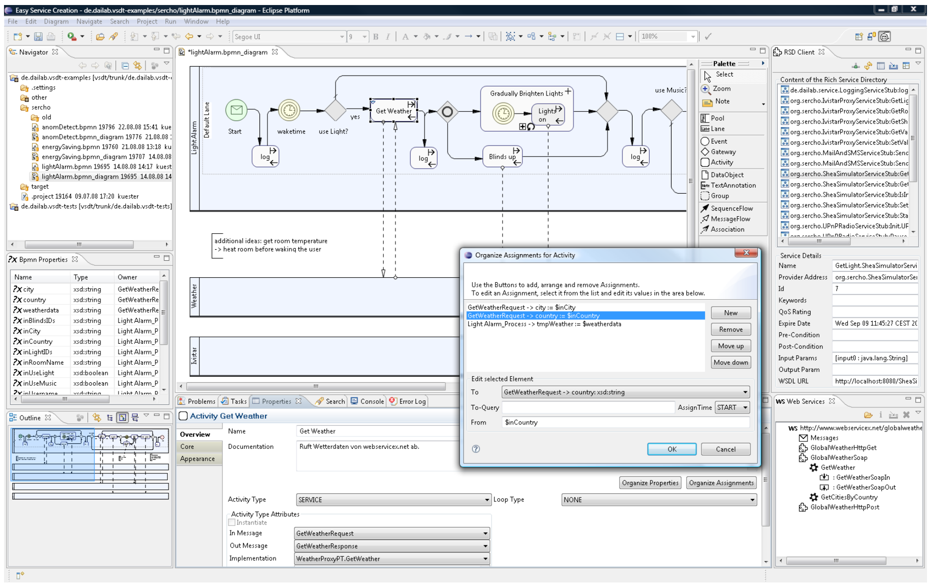
2.2 The BPMN Editor

Like many others, the VSDT editor has been created using the Eclipse Graphical Modelling Framework (GMF), automatically equipping the editor with numerous features, such as support for the Eclipse properties, outline and problem view and unlimited undo and redo, just to name a few. Being embedded in the Eclipse workbench, the editor is easy to use while at the same time providing a powerful tool for professional business architects and service developers.

While GMF provides a solid basis for the editor, several customisations have been made to the code, further improving the editor's overall usability and supporting the creation of new business processes. For example, the generated property tables have been supplemented with custom-made sheets, in which the several attributes are more clearly arranged. For managing the non-visual elements given in the BPMN specification, such as Properties, Messages and Assignments, a number of clear and uniform dialogs has been created. The various constraints given in the specification were translated to several audit constraints used to validate a given business process diagram. A screenshot showing some of the editor's features can be seen in Figure 1.

As already mentioned, the VSDT was designed to be a pure BPMN editor and independent of BPEL, so the business process diagrams can be transformed to other languages, too, given the respective export plugins. Of course, the downside of this approach is that the editor lacks built-in support for BPEL, e.g. the editor

[2] `http://www.eclipse.org/stp/bpmn`



Organize Assignments for Activity

Use the Buttons to add, arrange and remove Assignments.
To edit an Assignment, select it from the list and edit its values in the area below.

```
GetWeatherRequest -> city := $inCity
GetWeatherRequest -> country := $inCountry
Light Alarm_Process -> tmpWeather := $weatherdata
```

Edit selected Element

To	GetWeatherRequest -> country: xsd:string
To-Query	
AssignTime	START
From	$inCountry

Bpmn Properties

Name	Type	Owner
city	xsd:string	GetWeatherRe
country	xsd:string	GetWeatherRe
weatherdata	xsd:string	GetWeatherRe
inBlindsIDs	xsd:string	Light Alarm_P
inCity	xsd:string	Light Alarm_P
inCountry	xsd:string	Light Alarm_P
inLightIDs	xsd:string	Light Alarm_P
inRoomName	xsd:string	Light Alarm_P
inUseLight	xsd:boolean	Light Alarm_P
inUseMusic	xsd:boolean	Light Alarm_P

Activity Get Weather

Name	Get Weather
Documentation	Ruft Wetterdaten von webservicex.net ab.
Activity Type	SERVICE
Loop Type	NONE
In Message	GetWeatherRequest
Out Message	GetWeatherResponse
Implementation	WeatherProxyPT.GetWeather

Service Details

Name	GetLight.SheaSimulatorServi
Provider Address	org.sercho.SheaSimulatorSer
Id	7
Keywords	
QoS Rating	
Expire Date	Wed Sep 09 11:45:27 CEST 20
Pre-Condition	
Post-Condition	
Input Params	[input0 : java.lang.String]
Output Param	
WSDL URL	http://localhost:8080/SheaSi

```
http://www.webservicex.net/globalweathe
  Messages
  GlobalWeatherHttpGet
  GlobalWeatherSoap
    GetWeather
      : GetWeatherSoapIn
      : GetWeatherSoapOut
    GetCitiesByCountry
  GlobalWeatherHttpPost
```

Fig. 1. The Visual Service Design Tool, with colors and additional markers being enabled. Clockwise: Editor view, RSD client, Web services view, Organize Assignments dialog, customized property sheet, visual outline, properties inspector, navigator.

itself does not validate an expression given in the diagram to conform to the BPEL syntax. However, it is possible to supplement the editor with additional plugins, which can contribute e.g. to the property sheets or provide whole new views with language-specific functionality.

One example of how the VSDT can be extended with features specific to a certain target language — in this case: BPEL — is the RSD View, which can bee seen in Figure 1, too: A client for the *Rich Service Directory*, a special kind of Web service repository. Using the RSD View, existing Web services that have been registered at the RSD server can be inspected and imported into the diagram. In the process, an Implementation object is created for the Web service as well as a set of Message objects, matching the service's input parameters and result. Optionally, also a new Pool will be created for the service, which can be connected to the currently selected Activity via a pair of Message Flows. Further, the Implementation and the Message objects will be associated to the Activity and its type will be set to SERVICE. Thus, the orchestration of existing Web services in a BPEL process can be simplified greatly. Similar features can be created for other target languages, too.

Once the business process diagram is completed it can be validated and exported. As the VSDT is intended to provide export features to arbitrary target languages, and to support the tool smiths in the creation of these features, we have created an elaborate export framework, which we will have a closer look at in the next section.

3 The Transformation Framework

The core of the Visual Service Design Tool clearly is the transformation to executable code. While by now the transformation to BPEL is the only one that can be conveniently used in practice, and thus will serve as an example later in this section, there are currently several other transformations under development.

The transformation framework has been designed from the very beginning to be as *extensible* and *reusable* as possible. For that purpose the process of transformation has been subdivided into several stages, which are sequentially applied to the input model:

1. *Validation*: Validate the input model.
2. *Normalisation*: Prepare the input model for transformation.
3. *Structure Mapping*: Convert the input model to a block-like structure.
4. *Element Mapping*: Perform the actual mapping, create target model.
5. *Clean Up*: Remove redundancies, improve readability, etc.

The transformation is operating on a copy of the model to be transformed, which can be modified in the course of the transformation without affecting the original diagram. The several stages are realised either as a set of graph transformation rules, a top-down pass through the input model, or a combination of both. For the graph transformation rules the *Tiger EMF Model Transformation* Framework [1] (EMT) has been employed, providing a fast pattern matching and backtracking algorithm for EMF models. In EMT, rules can be specified using a convenient graphical editor. For the VSDT, however, the EMT has been modified so that instead of a Left Hand Side (LHS) with Negative Application Conditions (NACs) and a Right Hand Side (RHS), the rules feature an LHS, NACs and an `execute` method, which may contain arbitrary Java code. Given the several cases to consider in BPMN this has proven more feasible.

3.1 Stages of the Transformation

The Validation, Normalisation and Structure Mapping are to a great part independent of a specific target language, and in most cases the standard implementations provided with the transformation framework can be used. However, it can be advantageous to extend them with additional checks and rules.

For instance, in the *Validation* stage, all identifiers are validated to contain only characters that are legal with respect to the given target language, which can be achieved by extending the standard implementation and using a respective regular expression for the validation of names. Further, the validation includes a pass through the model, checking if each element needed is in place, thus reducing the number of checks necessary in the actual transformation, and providing clearer error messages to the user in case something is missing.

The intent of the *Normalisation* stage is to put the process diagram in a uniform form, and to transform it to what in the following will be referred to as the BPD's *normal form*, a semantically equivalent representation of the diagram following more strict structural constraints than those given in the BPMN specification. The transformation rules that are used in this stage are rather simple. For instance, one rule will check if there are any Activities with multiple incoming or outgoing Sequence Flows attached to it, in which case a Gateway of type XOR or AND will be inserted in between, depending on whether the Sequence Flows have any conditions. Another rule will insert a "no-op" Activity in between any two Gateways that are directly connected by a Sequence Flow. The advantage is that after the application of the normalisation stage there will be much fewer cases to consider in the structure mapping, which will be described in the next paragraph. A simple example of the consecutive execution of normalisation and structure mapping can be seen in Figure 2.

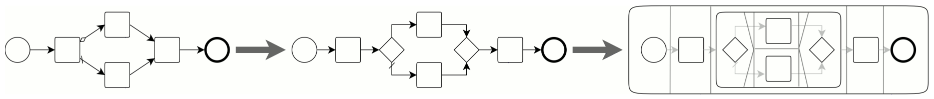


Fig. 2. Simple example of normalisation and structure mapping.

One of the challenges in transforming BPMN to executable languages is the mapping of the process model's graph-oriented structure to a more rigid block-oriented structure. For that reason it is of great benefit making this part, the *Structure Mapping*, independent of the actual target language, so it can be reused in mappings to other block-oriented languages. We decided to follow a *Structure Identification* strategy [9], being independent of BPEL's Link element. As mentioned in Section 2.1, the transformation is using an extension of the BPMN metamodel used in the editor, allowing the introduction of additional elements representing sequences, blocks for parallel and alternative routing, loops, and event handler blocks, i.e. the basic building blocks of block-oriented languages. Now, in the structure mapping stage, the model is searched for graph patterns which are semantically equivalent to these blocks, e.g. two Flow Object nodes connected with a Sequence Flow, or two Gateways connected by a number of branches of Flow Objects. When such a pattern is found, it is replaced with the respective structured element, removing the involved Sequence Flow edges in the course, which are then no longer needed (their conditions, if any, are preserved in the newly created structured elements). With these elements themselves being Flow Objects again, the rules of the structure mapping are applied until the entire process within each Pool has been reduced to a single complex element, e.g. a sequence, or until it can not be reduced any further due to structural flaws. Some examples of BPMN graphs that can successfully be mapped to equivalent block structures and further to executable BPEL code can be seen in Figure 3. Of course, this stage can be adapted or entirely omitted, too, if the target language is structured differently.

After the rule-based structure mapping, in the *Element Mapping* stage, the several BPMN elements can be mapped in a relatively simple top-down pass through the model. We decided to use a top-down pass instead of rules in this

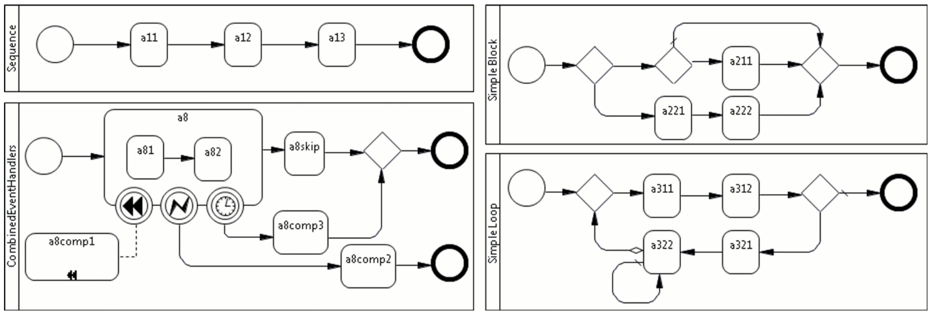


Fig. 3. Some examples of transformable BPMN graphs.

stage, as it is faster and easier to maintain, but the framework does allow for other implementations as well. As the element mapping is very dependent on the actual target language we will go further into detail later, regarding the transformation to BPEL.

Finally, in the *Clean Up* stage, a set of rules is applied on the newly created target model. While this stage is optional, it can be of great use for improving the readability of the generated code while at the same time keeping the required logic out of the earlier stages. For instance, nested sequences will be flattened, or sequences that hold a single element are replaced by that element itself. Further, elements that resulted from no-op Activities inserted in the normalisation stage should be removed again in this stage. As this stage operates on the target model, is has to be implemented anew for each target language.

For implementing a specific transformation, all that has to be done is to specify the element mapping, which can be done in any desired fashion by extending a special abstract class (see Figure 4). In case the target language uses different block concepts, the structure mapping has to be adapted, too, but should still be reusable to some point. In the majority of cases, implementing the other stages is optional.

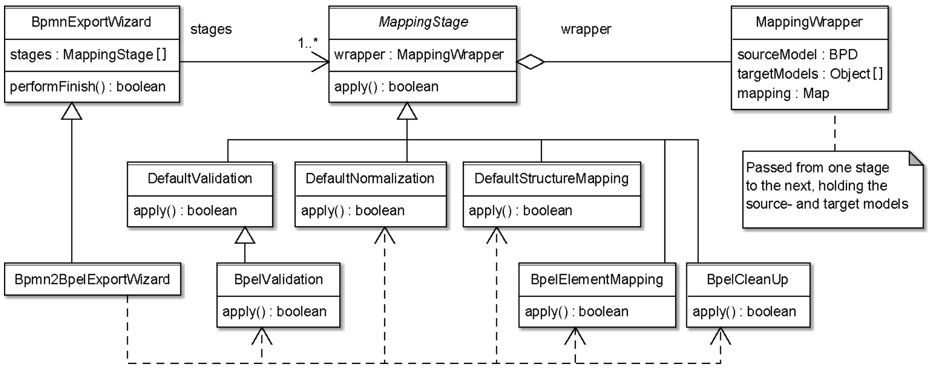


Fig. 4. Essential classes of the transformation framework, including the BPEL case.

3.2 Transformation to BPEL

The transformation to BPEL presented in this work covers nearly the entire mapping as given in the BPMN specification [10, Chapter 11], including event handlers, inclusive OR and event-based XOR Gateways, just to name a few. Still there are some elements for which the mapping is not given very clearly, such as TIMER Start Events, independent Sub Processes or multi-instance parallel loops. While these elements will be transformed as described in the specification, the resulting BPEL processes will require some amount of manual refinement. Besides the BPEL process files a WSDL definitions file is created, holding the message types derived from the process properties and the input and output messages and interfaces (port types) for the several Web services being orchestrated by the process. Still, the WSDL's binding and service blocks and necessary schema types, if any, can not be generated automatically yet, due to insufficient information in the source model. We are currently investigating ways of extending the BPMN metamodel in order to include more information in the model and at the same time making it more independent of the BPEL language.

In the validation used for the transformation to BPEL, all identifiers are tested to contain only characters that are legal with respect to BPEL. Additionally all expressions used e.g. in Assignments and loop conditions are scanned for occurrences of Property identifiers. So if a Process `Proc` has a Property `foo` and there is an Assignment with an expression like `"foo+1"`, the expression will be changed to `"bpws:getVariableData('Proc_ProcessData','foo')+1"`. Thus the user does not have to care about the way Properties are represented with messages in BPEL but can use a Property's plain name in expressions.

3.3 Example

The following example will show one of the scenarios being used in a smart home environment in the SerCHo project. The resulting BPEL processes were validated and tested with the *ActiveBPEL* designer and process engine.[3]

The BPMN diagram in Figure 5 is showing a "Light Alarm" process, that is used to open the blinds in the user's room to wake her up in a more pleasant way than the usual alarm clocks do. For that purpose, firstly information on the current weather is retrieved using an external Web service. Thereafter, based on the weather data, either the sunblinds are opened, or the ceiling light is turned on, or both. In case the user does not get up, which is checked using an RFID based localisation solution, the stereo is turned on, playing her favourite song or alternatively an unpleasant alarm sound.

For each of the above devices — blinds, lights, localisation, and stereo — Web service interfaces were written, so they can be integrated in a BPEL process. While the WSDL file that is used by the process, holding the definitions for the various orchestrated services, has to be extended with the service bindings, the generated BPEL code resulting from this example is readily executable.

[3] `http://www.activebpel.org`

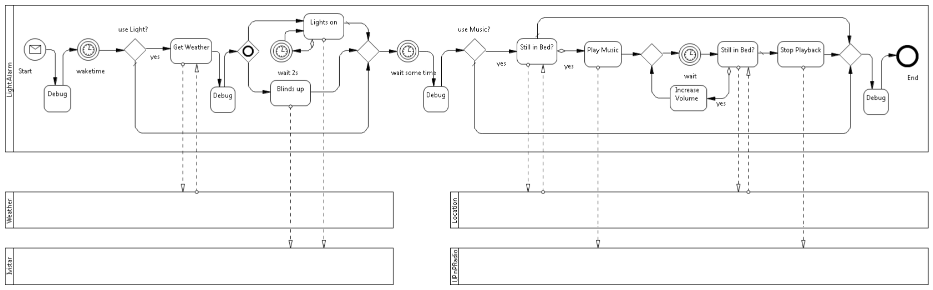


Fig. 5. "Light Alarm" Example Process

3.4 Transformation to JIAC

Concerning our goal of transforming BPMN diagrams to multi-agent systems (MAS) the work is still at an early stage. First, a *normal form* for BPMN diagrams has been investigated, to facilitate the mapping [2]. Later, the first steps of the actual mapping have been developed, basically mapping Pools to agents, Processes and Flow Objects to the agents' plans and the control flow, and Message Flow to the exchange of messages between the agents [3].

A first prototype targeting the agent framework JIAC IV [14] has already been implemented. As the theoretical part of the mapping is not yet fully matured, there is still some work to do. However, with the given transformation framework every addition to the mapping can quickly be adopted.

4 Related Work

The Business Process Modelling Notation has been adopted in a large number of tools. Although many of these are merely diagram drawing tools and do not support the transformation to BPEL, let alone other languages, there are some powerful tools as well. In the following we will introduce some of these. A more extensive list can be found at `http://www.bpmn.org/BPMN_Supporters.htm`.

With the free *eBPMN*, Soyatec provides a very nice BPMN editor, but it does not implement the mapping to BPEL.[4] The same applies to the free community edition of eClarus' *Business Process Modeller*, while the commercial *SOA-Architect* version provides a transformation to BPEL, although it seems to have some limitations.[5] A very mature BPM product can be found in the *Intalio BPMS*.[6] BPEL code is generated on-the-fly and can be deployed to the Intalio process engine. However, the mapping of workflow structures is limited, e.g. we found it impossible to merge a branch originating from an event handler back into the normal flow. Another limitation arises from the tight coupling to the inhouse BPEL engine, which is using some proprietary extensions. Further, Intalio

[4] `http://www.soyatec.com/ebpmn`

[5] `http://www.eclarus.com`

[6] `http://www.intalio.com`

has donated parts of the code to the Eclipse SOA Tools Project (STP). While the *STP BPMN Editor* itself does not provide a transformation to BPEL, Giner et al. were able to combine it with the *BABEL Bpmn2Bpel* tool [4], yet both the editor and the transformation tool are using very simple metamodels.

Concerning the transformation from graph-oriented to block-oriented process models, as in the BPMN to BPEL case, Mendling et. al. have evaluated several transformation strategies [9], ranging from a straight-forward mapping of BPMN Sequence Flows to BPEL Links, similar to the one in [17], to a more sophisticated *Structure Identification* strategy, like the one applied in this work, or *Structure Maximisation*, as followed by Aalst and Lassen [15]. In their theoretically well-founded, pattern-based transformation from Petri nets to BPEL, they focus on the readability of the resulting code. However, they do not regard how highly *unstructured* workflows can be transformed to structured ones. As pointed out in [12], there is a "mismatch" between BPMN and BPEL, both on the domain representation and the control flow level, that is not easily to overcome. Many authors have investigated whether different graph patterns can be transformed to an equivalent structured form [5, 8, 13], and came to the conclusion that even slight unstructuredness can require the introduction of additional variables or the duplication of parts of the workflow, even though the workflow models used in these works are much simpler than BPMN. For structuring such workflows, Koehler et al. present a rule-based transformation based on continuation semantics [6]. Another approach is followed by Ouyang et al., using BPEL event handlers as a form of `goto` command [11]. Their examples show how complicated a simple workflow can become when being structured.

Thus, as workflow design will be facilitated greatly if the user is not restricted to the use of block-oriented processes, a transformation of unstructured workflows to readily executable code will be highly desirable, so that such processes can be created by means of Model Driven Engineering.

5 Conclusion

In this paper the *Visual Service Design Tool* (VSDT) has been introduced: a BPMN editor featuring a state of the art transformation to BPEL, while at the same time being easily extensible with export functionality targeting other languages. The editor itself has been designed to be language independent, so it can be used for generating code for any language, given that a respective mapping from BPMN to that language exists. Transformations implementing these mappings can be plugged in to the VSDT together with additional editing features helping the user in the creation of diagrams to be exported to that language. For supporting the developer of these plugins, the VSDT comes with a transformation framework, based on the EMT graph transformation tool. Being subdivided into several stages, large parts of it can be reused throughout different mappings, such as the refactoring of the process graph to block-oriented structures.

With respect to its BPMN editing functionality and the transformation to BPEL, the VSDT does not have to hide behind its commercial competitors.

Implementing the mapping to BPEL as given in the BPMN specification, the tool can be used to generate readily executable code. Still it is recommended to validate the results with a native BPEL editor: While the creation of processes will be easier and faster using the VSDT, its desired independence of a specific language prohibits some BPEL specific features, such as editing assistance for assignment expressions. However, due to the plugin architecture provided by the Eclipse platform, such functionality can be added together with the actual transformation features.

As the key feature of the VSDT is the extensibility with additional export features, further transformations from BPMN to executable languages are currently under development. One of the main goals of our research in this field is a mapping from BPMN to multi-agent systems, combining the intuitive design of business processes with the flexibility of software agent.

5.1 Future Work

Some work still can be done in the field of transformation of unstructured processes. Currently the tool can handle slightly unstructured workflows, such as one Gateway being used for merging multiple decision blocks, but will fail when faced with e.g. overlapping blocks or multiple exits from a loop. Here, further evaluations of the different possibilities to handle such workflows and ways of adapting them to the more complex BPMN diagrams will be necessary.

Concerning the transformation to BPEL, the support for complex data types will need further refinement. Here, the *Rich Service Directory* introduced earlier will be of great use, providing the necessary information about the involved Web services. Finally, the mapping to multi-agent systems has to be completed, and mappings to further languages will be evaluated.

References

1. Enrico Biermann, Karsten Ehrig, Christian Köhler, Günter Kuhns, Gabriele Taentzer, and Eduard Weiss. Graphical definition of in-place transformations in the eclipse modeling framework. In *Proc. 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS'06)*, Genova, Italy, October 2006.
2. Holger Endert, Benjamin Hirsch, Tobias Küster, and Sahin Albayrak. Towards a mapping from BPMN to agents. In Jingshan Huang, Ryszard Kowalczyk, Zakaria Maamar, David Martin, Ingo Müller, Suzette Stoutenburg, and Katia P. Sycara, editors, *Service-Oriented Computing: Agents, Semantics, and Engineering*, volume 4505 of *LNCS*, pages 92–106. Springer Berlin / Heidelberg, 2007.
3. Holger Endert, Tobias Küster, Benjamin Hirsch, and Sahin Albayrak. Mapping BPMN to agents: An analysis. In Matteo Baldoni, Cristina Baroglio, and Viviana Mascardi, editors, *Agents, Web-Services, and Ontologies Integrated Methodologies*, pages 43–58, 2007.
4. Pau Giner, Victoria Torres, and Vicente Pelechano. Bridging the gap between BPMN and WS-BPEL. M2M transformations in practice. In *Proc. of the 3rd*

International Workshop on Model-Driven Web Engineering (MDWE 2007), Como, Italy, July 2007.

5. Bartek Kiepuszewski, Arthur H. M. ter Hofstede, and Christoph Bussler. On structured workflow modelling. In *CAiSE '00: Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, pages 431–445, London, UK, 2000. Springer-Verlag.
6. Jana Koehler and Rainer Hauser. Untangling unstructured cyclic flows - a solution based on continuations. In R. Meesman and Z. Tari, editors, *CooplIS/DOA/ODBASE 2004*, volume 3290 of *LNCS*, pages 121–138. IBM Zurich Research Laboratory, Springer-Verlag, 2004.
7. Tobias Küster. Development of a visual service design tool providing a mapping from BPMN to JIAC. Diploma thesis, Technische Universität Berlin, April 2007.
8. Rong Liu and Akhil Kumar. An analysis and taxonomy of unstructured workflows. In Wil M. P. van der Aalst, Boualem Benatallah, Fabio Casati, and Francisco Curbera, editors, *Business Process Management*, volume 3649, pages 268–284, 2005.
9. Jan Mendling, Kristian Bisgaard Lassen, and Uwe Zdun. Transformation strategies between blockoriented and graph-oriented process modelling languages, 2005.
10. Object Management Group. Business Process Modeling Notation (BPMN) Specification. Final Adopted Specification dtc/06-02-01, OMG, 2006. `http://www.bpmn.org/Documents/OMGFinalAdoptedBPMN1-0Spec06-02-01.pdf`.
11. Chun Ouyang, Marlon Dumas, Stephan Breutel, and Arthur H. M. ter Hofstede. Translating standard process models to BPEL. In Eric Dubois and Klaus Pohl, editors, *CAiSE*, volume 4001 of *Lecture Notes in Computer Science*, pages 417–432. Springer, 2006.
12. Jan Recker and Jan Mendling. On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages. In T. Latour and M. Petit, editors, *In: T. Latour, M. Petit, eds.: CAiSE 2006 Workshop Proceedings - Eleventh International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD 2006), June 5 - 6, 2006, Luxembourg, pages 521-532.*, pages 521–532, 2006.
13. Wasim Sadiq and Maria E. Orlowska. Analyzing process models using graph reduction techniques. *Inf. Syst.*, 25(2):117–134, 2000.
14. Ralf Sesseler. *Eine modulare Architektur für dienstbasierte Interaktionen zwischen Agenten*. PhD thesis, Technische Universität Berlin, 2002.
15. Wil M. P. van der Aalst and Kristian Bisgaard Lassen. Translating unstructured workflow processes to readable BPEL: Theory and implementation. *Inf. Softw. Technol.*, 50(3):131–159, 2008.
16. Stephen A. White. Introduction to BPMN. Technical report, IBM Corporation, May 2004. `http://bpmn.org/Documents/Introduction%20to%20BPMN.pdf`.
17. Stephen A. White. Using BPMN to model a BPEL process. Technical report, IBM Corporation, March 2005. `http://www.bpmn.org/Documents/Mapping%20BPMN%20to%20BPEL%20Example.pdf`.