A Metamodel Refinement Approach for Bridging Technical Spaces, a Case Study

A. Staikopoulos and B. Bordbar

School of Computer Science, University of Birmingham, Birmingham, B15 2TT, UK
{A.Staikopoulos, B.Bordbar}@cs.bham.ac.uk

Abstract. To benefit from positive aspects of an existing diverse set of Technical Spaces, it is important to develop methods of automated transformation of models between such domains. Sometimes it is possible to describe Technical Spaces via metamodels. In such cases, the Model Driven Engineering and Architecture pose as a natural candidate for dealing with such transformations between Technical Spaces. This paper deals with the case where the metamodel of the source Technical Space is more complex than the metamodel of the destination. Thus, the gap between the two Technical Spaces is highly non-trivial. The method presented in this paper is based on successive metamodel refinements to bridge this gap. Finally, the method is applied to the transformation from Business Process Execution Language to Petri nets.

1 Introduction

Technical Spaces (TS) [11], [14] and Domains [7] consider the working context where systems and applications are specified and developed, from certain perspectives. To benefit from positive aspects of different Technical Spaces and Domains, applications belonging to one context may need to be transferred to alternative contexts, while using their specified tools and technology. For example, models of Business Process Execution Language (BPEL) [5], as an XML Technical Space, can be translated to Petri nets [15] to allow verification and analysis of the system [6], [18]. Such translation facilitates the cooperation of alternative technologies and techniques rather than their competition while supporting the best possibilities of each domain [14]. To do so, we need to define mappings across the spaces and eliminate their conceptual gaps between the two domains.

The Model Driven Development or Engineering [1], [11], [17] can play an important role, as it provides an approach and a technical framework for establishing bridges between two Technical Spaces, by providing domain integration and interoperability via Metamodel mechanisms [14], [17]. However, if there is a large conceptual gap between the two Technical Spaces, defining a suitable MDA transformation [9] is a highly non-trivial task. This paper deals with the scenario in which the metamodel of the Technical Space of the source is richer than the metamodel of the Technical Space of the destination. The method presented is called *One Step Refinement* of the destination metamodel and is based on the destination enrichment to bridge the gap between the two Technical Spaces. We shall demonstrate our approach by mapping a number of BPEL constructs to Petri nets.

The paper is organised as follows: Section 2 covers the basic concepts involved in the paper, giving a number of definitions and preliminary information. Section 3 compares the Spaces of Business Processes and Petri nets and discusses issues relative to their mapping. Section 4 describes the proposed approach for bridging Technical Spaces and Domains. Section 5 presents how the method is applied with a number of examples. Section 6 provides various discussion points. Finally, Section 7 presents the conclusions drawn during the authors' experimentations and summarises the basic characteristics of the approach adopted.

2 Preliminaries

In this section we shall present a brief overview of the concepts used in the paper:

Technical Spaces and Domains: *T*echnical *Spaces* [11], [14] represent specific working contexts with specific implementation technologies, tools and approaches, where applications are specified, instantiated and utilised from various tools and engines. *Domains* [7] on the other hand represent contexts via specific application aspects and not by given programming language concepts. In some respect, TSs and Domains are comparable when creating contexts as metamodels, where the first is focusing on the technological and implementation issues, while the other on conceptual application representations.

Metamodels and Model Driven Approaches: *Metamodels* are models that formally describe the syntax and semantics of a given context, by modelling languages such as UML [16]. A model has "an instance of" relationship with its metamodel. Metamodelling [1], [13] is an essential foundation for model driven development and architecture. The *Model Driven Development* (MDD) [1], [17] is a model-centric software engineering approach, focusing on the design models instead of the code. One of its most prominent variant is the *Model Driven Architecture* (MDA) [9], [10] by OMG. The MDA specifies a technical framework of standards for designing systems via models. It promotes the creation of highly abstract models that are developed independently of implementation details, which repeatedly and automatically can be transformed by tools to specific implementations and technologies [9], [13]. Similarly to MDD the *Model Driven Engineering* (MDE) [11] is a form of generative engineering building upon the idea of MDA. It supports the integration of Technical Spaces and promotes their synergy in a smooth way by providing bridges [8], [12].

Next, we provide some basic information regarding the Technical Spaces that will be used in our case study. These are as follows:

Business Process Execution Language: The *Business Process Execution Language for Web Services* (WS-BPEL or BPEL for short) [7] specifies the process instance to be mapped. The BPEL provides an XML notation and semantics for specifying business process behaviour, based on collaborating participants (external Web services). The behaviour is defined upon a set of interconnected hierarchical activities that are formed in various ways similar to workflow patterns, simulating loops and parallel execution for example.

Petri nets: The *Petri nets* (PNs) [15] specify the semantic domain of the process. A PN is a particular kind of directed graph consisting of places (p), transitions (t), which are connected with arcs either from a place to a transition (PTArc) or from a transition to a place (TPArc). Graphically, places are drawn as circles, transitions as bars and arcs as directed arrows. Tokens are represented by black dots placed in places to simulate the dynamics of the system. PNs can also be considered as a mathematical tool to describe and study information processing systems that are characterised as concurrent, distributed, parallel or non-deterministic.

3 Mapping and Bridging Technical Spaces

As TSs and Domains represent metamodelling contexts, it makes sense to shift from one Space or Domain to another in order to use its context, concepts, approach and tools (please refer to Fig. 1). In this respect, Kurtev et. al. [14] forward the idea that there should be more cooperation than competition among alterative technologies/Spaces and similar among domains.

To realise such an idea, we need to establish bridges between the different Technical Spaces, with specified mappings and properties. In that way, Technical Spaces or Domains are no longer isolated islands but cooperating units, allowing original instances to be transferred to alternative representations, with an objective to solve a given problem by using the best possibilities of each technology.

Furthermore, the Model Driven Development and Engineering with transformation approaches and tools can automate the generation of the corresponding target Space instances and domains [2], [3], [4], when they are applied upon well-established points. Thus, the bridges have to be established upon well formalised TS or Domain metamodels capturing precisely the TS and Domain languages, technologies and characteristics.

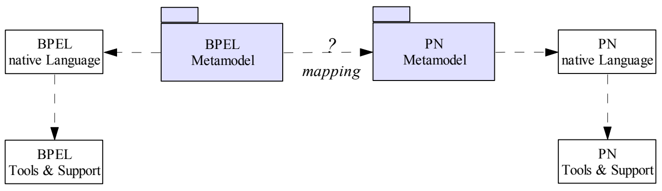
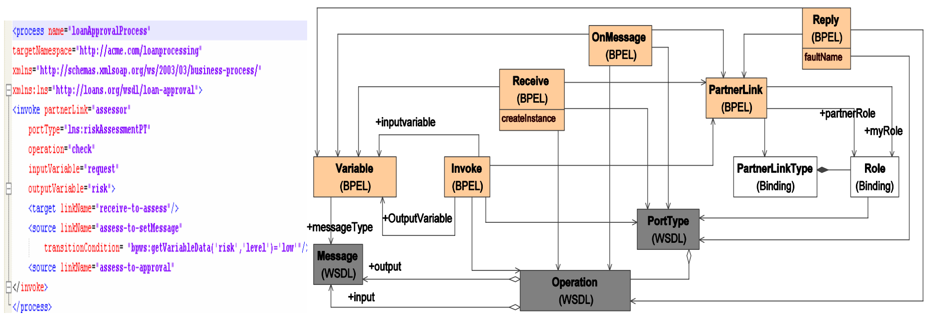


Fig. 1. Mapping BPEL and PN Technical Spaces and representations

Sometimes, however, it is not possible at the first attempt to map directly two metamodels together, as the Technical Spaces or Domains may be rather different and not in accordance with each other. Such problems often occur when one Space may define or possess characteristics that the other one does not define or accounts for. In this paper, such problems are investigated and an approach on bridging the gap among quite different Spaces is proposed.

To realise and present the approach the authors experiment on bridging the Technical Spaces of BPEL and PNs with a case study.

The BPEL language [5] (see Fig. 2(a)) specifies business process models via XML Schemas and technologies. Similarly, PNs [15] (see Fig. 3(a)) specifies processing systems in graphical representations. As both Spaces and Domains are originally based upon different languages, we need to represent them in a common formalism, for example a UML metamodel representation [16]. That will assist our objective to map them together upon common means, tools and techniques. In order to do so, we assume that metamodelling techniques are capable enough to specify the languages precisely. For BPEL and PNs we do not have such problems [18], [19] and following Fig. 2 and Fig. 3 depict such different representations among native and metamodel representations.



```
<process name="loanApprovalProcess"
targetNamespace="http://acme.com/loanprocessing"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:lns="http://loans.org/wsdl/loan-approval">
<invoke partnerLink="assessor"
    portType="lns:riskAssessmentPT"
    operation="check"
    inputVariable="request"
    outputVariable="risk">
    <target linkName="receive-to-assess"/>
    <source linkName="assess-to-setMessage"
        transitionCondition= "bpws:getVariableData('risk','level')='low'"/
    <source linkName="assess-to-approval"
</invoke>
</process>
```

Fig. 2. Sample of BPEL (a) and a metamodel segment of BPEL (b)

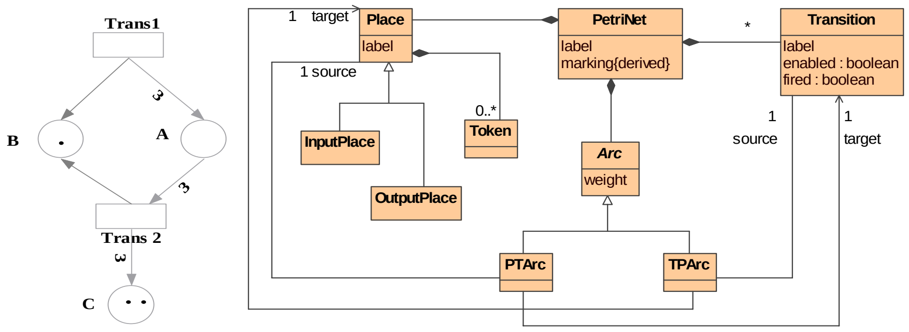


Fig. 3. Sample of PN (a) and a metamodel of simple Petri net (b)

It can clearly be seen that the metamodel of BPEL is more sophisticated than the metamodel of PN, including complex, high-level constructs such as "*Invoke*", "*Flow*" and *"Scope"*. As a result, there is a visible gap between the two Technical Spaces.

4 A Method for Bridging Technical Spaces

In order to map two different Technical Spaces represented by metamodels, one needs to identify and match their corresponding metamodel elements and supported characteristics. However, identifying corresponding meta-elements is not an easy task. In this case the BPEL metamodel is very complex when compared to PNs, comprising complicated structures, whereas the PN metamodel defines just few model elements.

To bridge the discrepant Technical Spaces, we propose a method that is based upon the successive refinement of the destination metamodel. The method is depicted in Fig. 4 and is described as follows:

Refinement of the destination: Assuming that the aim is to define a model transformation from a Technical Space modelled via a metamodel *N* (in this example a BPEL) into a Technical Space modelled via a metamodel *M* (a simple PN). In this paper, the emphasis is placed upon one-way mappings from *N* to *M,* where the metamodel of the source *N* is more expressive or richer than the metamodel *M*, in the sense that there is a considerable number of metamodel elements of *N,* which cannot be directly mapped into metamodel elements of *M.*

In some cases, it might be possible to map *some* of the model elements of *N* into *M* as depicted by the ψ_0 mapping. Now, consider a model element β_1 of *N* that cannot be directly mapped into any model elements of *M*. However, suppose that it is possible to construct β_1 via model elements of $M{:=}M_0$. This lead to one step refinement $M_1 := M_0 \oplus \alpha_1$ such that β_1 can be mapped to α_1. The process can be repeated until an extension M_κ (after κ stages) is created, such that *all* model elements of *N* can be mapped successfully and meaningfully into model elements of M_κ.

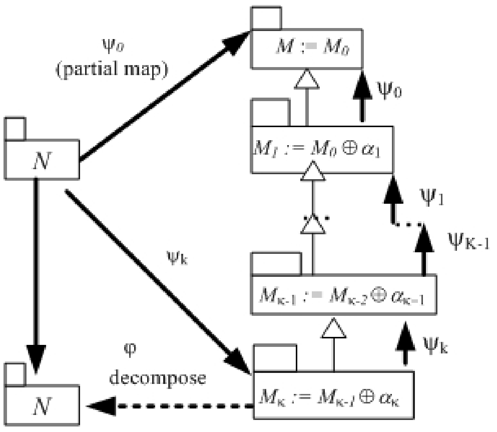


Fig. 4. Refinement of the Destination Metamodel

In this case, the technical space *Mκ* and *N* are near enough to be mapped. Now, if all step refinements from *M* to M_κ are decomposable (see *Definition1)*, then it is possible to define model transformations ψ_1 from M_ι to $M_{\iota-1}$ ($1 \leq \iota \leq \kappa$) such that the mapping $\varphi = \psi_0 \circ \psi_1 \circ \psi_2 \circ \ldots \psi_\kappa \circ \psi$ maps N to M.

Definition1: Assuming that M is a metamodel and $M \oplus \alpha$ is a *One Step Refinement* of M by adding α. Then, we say $M \oplus \alpha$ is ***decomposable***, if we can define a model transformation from $M \oplus \alpha$ to M.

Consequently, a metamodel extension $M \oplus \alpha$ may be decomposable, when the newly introduced concept α can be represented with the original metamodel elements of M, without losing any essential information during the decomposition.

The successive refinement of the destination metamodel can be implemented either by profiles (referred as light-way extensions) or more conservative metamodel extensions (based upon heavy-weight extensions) [13].

5 Applying the Method Adopted – Case Study

Let us consider the Technical Spaces N and M described by the BPEL and PN metamodels respectively. The aim is to establish a bridge among these Spaces by successively refining the destination M (PN) metamodel. To demonstrate the method and raise a number of related issues, the BPEL *"Invoke"* activity to an equivalent Petri net representation will be mapped.

The BPEL *"Invoke"* activity provides a two-way operation (request/reply) between a business process and a Web service participant. Its operational semantics will block the process till the participant replies back with an answer [5]. The BPEL metamodel of Fig. 2(b) describes *"Invoke"* and its associated elements, as input and output variables used by operations provided by participants.

Within the destination metamodel (PN) *"Invoke"* does not have a clear counterpart. The PN metamodel as previously discussed (please refer to preliminary section), consists of just few elements.

Assuming that we want to extend the Technical Space M of a simple PN with the notion or concept α of *"Invoke"* operation. In order to satisfy the previously described characteristics, the extension metamodel $M \oplus \alpha$ will be refined as the one depicted in Fig. 5(a) and its equivalent PN instance representation as the one depicted in Fig. 5(b).

The introduced *"Invoke"* element represents a new concept, created as a self contained element, allowing to be reused from other language constructs, such as structured activities to make full compact models. The *"Invoke"* element can be described as a specialised type of transition and can be treated as such [6]. It extends *"BasicActivity"* that is the common classifier with other similar activities such as *"Receive"*. More specifically, *"Invoke"* directly defines and contains two simple Transitions, three specialised places identified as *"Wait", "InputData"* and *"OutputData",* as well as two external arcs linking to another PN model, which represents the activity of a participant. In that respect, *"Invoke"* is a composite transition having internal transitions and states, represented by the three place types. The *"Wait"* place represents the situation (state), where the process waits for the participant's answer, by remaining blocked. The *“InputData”* and *“OutputData”* places, together with their tokens, represent respectively the input and output variables used at actual *“in”* and *“out”* parameters of the operation. The operation call is actually represented by the first internal transition and its completion by the second one. The action can be considered as atomic, in the respect that it cannot be externally interfered and represents one unit of action. Finally, the two arcs, one outgoing and one incoming, represent the interaction of our process with its external participant.

The PN model of Fig. 5(b) depicts an instantiation of its metamodel for performing the *“Invoke”* operation as previously described.

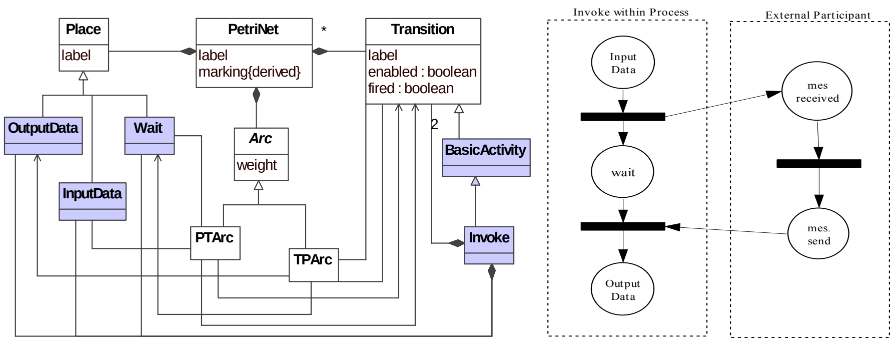


Fig. 5. A PN metamodel extension (a) and instance representation (b) for “invoke”

The $M \oplus \alpha$ metamodel extension is now a M refinement defining an *“Invoke”* element and making the resulting PN metamodel to be mapped effectively with its corresponding BPEL *“Invoke”* elements. For example, Fig. 2 depicts the BPEL relevant metamodel elements. The BPEL metamodel similar to PNs, has an *“Invoke”* element that is associated with *“InputVariable”* and *“OutputVariable”,* a WSDL *“Operation*” provided by a *“PartnerLink”* and representing the actual participant. In that way mappings across the two metamodels can be provided easily, defining the notion of *“Invoke”* and establishing a bridge of collaboration and mapping.

Additionally, the PN metamodel extension, representing the BPEL *“Invoke”* operation, is capable of modelling its dynamic and operational capabilities. So, it justifies our initial aim to use it for analysis and simulation purposes. For example, the internal states and transitions of *“Invoke”* (see Fig. 5) can be mapped to important (run time) aspects, such as reading the operation’s parameter variables and writing the returned results to internal variables. Thus, they allow the simulation and analysis of our BPEL models by specialised tools, within the PN Space.

To decompose the *“Invoke”* notion, we have to represent its context with the original PN metamodel elements such as PN, T, P, TPArc and PTArc (refer to preliminary section). In that case we have to rewrite the metamodel and assist the semantic interpretation of its decomposed elements:

Metamodel Decomposition: If the *“Invoke”* transition is removed, then we have to deal with its internal content, which in this case is defined by two simple transitions, three specialised places and two arcs. An element may also have internal defined OCL statements, which need to be transferred equivalently as well. As *“In-*

voke" is of a type of transition the closer element is a *"Transition"*. However, the notion of *"Transition"* cannot contain *"Places"* or other *"Transitions"* of any kind. Thus, if we preserve such relations we will have a significant violation of its fundamental semantics. As the "composition relation" serves the purpose to depict the constitute elements of an *"Invoke"* structure, which actually no longer exists (*"Invoke" is* decomposed), then there is no reason to be retained.

There are cases, where the appearance of *"Invoke"* can be identified as a pattern of loosely combined elements similar to grammars, which identify the tokens of a language. Similarly, the specialised *"Places"* should also be decomposed. In this case the most relevant type or element is that of a *"Place"*. As a *"Transition*" is allowed to have associations with *"Places"* via *"Arcs",* there is no problem for these relationships to be preserved and modelled analogously, however this time they belong to the PN context and not to the *"Invoke"*.

Semantic Interpretation: As the specialised elements such as *"Wait"* do not exist any longer, identifying and distinguishing them may not be an easy task, as now all are of the same type *"Place"*. However, we may address at some extent such problems, if we annotate the association ends with their type names or use the attribute *"Label"* (see Fig. 3) to mark their types. In this case, *"Labels"* are used as "classifiers" to reveal the intention of being of a specific type. Analogously, OCL statements can be used to distinguish them, if the Space or Domain allows them. Alternatively, we may mark the modelling element with stereotypes, which effectively is similar to trying to make the metamodels profile-able.

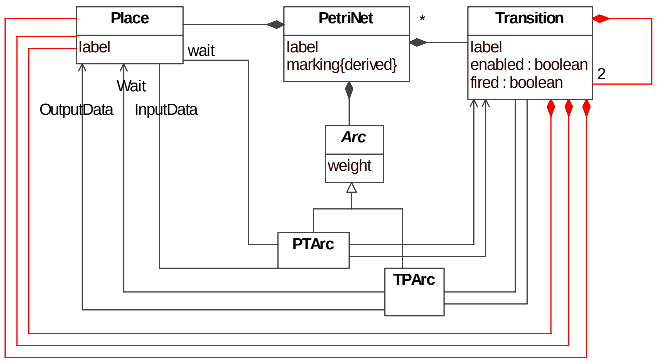


Fig. 6. Decomposing Invoke PN metamodel elements regarding the original

Finally, the mapping of a BPEL "*Invoke"* element to a decomposed PN extension depends largely on to how successively one has managed to decompose the context of the notion introduced and how easily this can be identified. If the previous requirements are met, then it is possible to establish a meaningful mapping. However, this time would be much more difficult to define and more complicated to distinguish than before.

6 Discussion

There are two major prerequisites to the success of our method. The first point is that, as depicted in Fig. 4, we must identify and map the primary and fundamental elements of our Technical Space *N* and then gradually build the more complicated elements on top of the already refined metamodels at the source. This is a *"bottom up"* modelling practice. In particular, there are different ways of refining a source metamodel. For example the authors can model *invoke* in three different ways utilising suitable OCL statement. It seems possible to come up with scenarios that a model element of the source "*can not"* be added to destination to create further refinement, which can be interpreted as, two Technical Spaces are too far from each other. Further research and case studies on the matter are required.

The second major prerequisite is the ability to decompose, see *Definition 1*, a refined metamodel. Currently, it is not very clear to authors under what circumstances it is possible to decompose successively a metamodel. However, it seems crucial that the refinement does not change the semantics of the metamodel, i.e. the semantics of *M*, as a part of $M \oplus \alpha$ must remain unchanged. For example, in PNs, constructs such as *"Place"* may have different semantic interpretations regarding the context which is used. For example, within an *"Invoke"* activity *"Place"* represent the *"InputData"* and *"OutputData"* for an operation, where in *"Switch"* activity they can represent the *"PreCondition"* and *"PostCondition"*. Therefore, when it is to decompose them we may loose semantic information, so the model elements may be misinterpreted during instantiation and mapping from tools. For that reason we have to device ways in order to incorporate such information to our original metamodels. This can take various forms from using OCL statements into labelling association ends or stereotyping. Further research on the matter is required.

7 Conclusion

This paper presents a method of bridging two Technical Spaces, which can be expressed via metamodels. The presented method is based on the MDA and aims to address the scenarios that the metamodel of the destination Technical Space has less complex structure than the metamodel of the source Technical Space. To bridge the conceptual gap, the metamodel of the destination is refined by adding model elements corresponding to model elements of the source Technical Space. We have presented samples of application of the approach to bridge the gap between the Technical Space of BPEL and PN.

References

[1] Atkinson, C. Kuhne, T. Model-driven development: a metamodeling foundation, University of Mannheim; Software, IEEE, Publication Date: Sept.-Oct. 2003, Volume: 20, Issue: 5 p. 36- 41, ISSN: 0740-7459, (2003)

[2] Bézivin, J., Hammoudi, S., Lopes, D., Jouault, F.: An Experiment in Mapping Web Services to Implementation Platforms. Technical report: 04.01, LINA, University of Nantes, Nantes, France (2004).

[3] Bordbad, B., Staikopoulos, A.: Modeling and Transforming the Behavioural Aspects of Web Services. In: Proc. 3rd Workshop in Software Model Engineering - WiSME2004, UML (2004)

[4] Bordbar, B., Staikopoulos, A.: On Behavioural Model Transformation in Web Services. In: Proc. Conceptual Modelling for Advanced Application Domain (eCOMO), Shanghai, China, p. 667-678, (2004)

[5] BPEL: BEA, Microsoft, IBM, SAP, Siebel, Business Process Execution Language for Web Services, Version 1.1. (2003)

[6] Chun Ouyang, van der Aalst, Stephen Breutel, Marlon Dumas, Arthur ter Hofstede, Eric Verbeek, Formal Semantics and Analysis of Control Flow in WS-BPEL, (2005)

[7] Greenfield, J, Keith Short: Software Factories, Wiley, ISBN: 0471202843, (2004)

[8] Ivan Kurtev, Adaptability of Model Transformations, PhD Thesis, University of Twente, ISBN 90-365-2184-X, (2005)

[9] J. Miller and J. Mukerji: "MDA Guide Version 1.0.1, OMG Document Number: omg/2003-06-01", OMG, 12.6.2003, http://www.omg.org/cgi-bin/doc?omg/2003-06-01

[10] J. Siegel, Developing in OMG's Model Driven Architecture, Object Management Group, November (2002)

[11] J.M. Favre, "Towards a Basic Theory to Model Model Driven Engineering", 3rd Workshop in Software Model Engineering, WiSME 2004, http://www-adele.imag.fr/~jmfavre

[12] Juliane Dehnert, van der Aalst, Bridging the Gap Between Business Models and Workflow Specifications, International Journal of Cooperative Systems, (2004)

[13] Kleppe, A., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture-Practice and Promise, (2003)

[14] Kurtev, J. Bézivin, and M. Aksit. Technological spaces: An initial appraisal. In Int. Federated Conf. (DOA,ODBASE, CoopIS), Industrial track, Los Angeles, (2002)

[15] Murata, Tadao, Petri Nets: Properties, Analysis and Applications. In: Proceedings of the IEEE, Vol. 77, No. 4, p 541-580, April (1989)

[16] OMG: UML 2.0 Superstructure Specification. Document id: ptc/03-08-02 (2003)

[17] Gitzel, R. and Korthaus, A. (2004): The Role of Metamodeling in Model-Driven Development, In: *Proceedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI2004)*, Orlando, USA, July, (2004)

[18] van der Aalst, W M P, van Hee, K M and Houben, G J, Modelling and analysing workflow using a Petri-net based approach. Proc. 2nd Workshop on Computer- Supported Cooperative Work, Petri nets and related formalisms, p. 31-50, (1994)

[19] Y. Yuhong, A. Bejan, Modelling Workflow within Distributed Systems, 6th International CSCW in Design, Canada, (2001)