Bridging Technical Spaces With A Metamodel Refinement Approach. A BPEL To PN Case Study

Athanasios Staikopoulos[1]

School of Computer Science
University of Birmingham
Birmingham, UK

Behzad Bordbar[2]

School of Computer Science
University of Birmingham
Birmingham, 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 of Business Process Execution Language (BPEL) models to Petri nets.

Key words: Metamodel Refinement, Bridging Technical Spaces.

1 Introduction

Technical Spaces (TS) [11], [14] and Domains [7] deal with the working context where systems and applications are specified and developed from certain perspectives. To benefit from positive aspects of different Technical Spaces

[1] Email: `A.Staikopoulos@cs.bham.ac.uk`
[2] Email: `B.Bordbar@cs.bham.ac.uk`

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 while assisting 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 relating 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: Technical 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. On the other hand, domains [7] are characterised by specific application aspects and not by given programming language constructs. In some respect, Technical and Domain contexts are comparable within metamodelling, where the first focuses on capturing 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 variants is the Model Driven Architecture (MDA) [9], [10] promoted by the Object Management Group (OMG) [20]. 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] is a specification allowing the creation of complicated processes by creating and wiring together different service activities. For example, processes can perform Web service invocations, manipulate data and in general coordinate their activities towards common objectives. Consequently, the specification 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 formulated in various ways similar to workflow specifications [21], forming patterns such as those for simulating loops and parallel execution. In our case study, the BPEL specification determines the technical space and domain of interest where our process models will be based upon.

Petri nets: A Petri net (PN) [15] 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", as "PTArc", or from a "Transition" to a "Place" as "TPArc". Graphically, "Places" are illustrated as circles, "Transitions" as bars and "Arcs" as directed arrows. "Tokens" are represented by black dots in "Places" to simulate the current dynamics (states) of the system. PNs have received considerable attention as proper mathematical tools [22] to formally describe and study information processing systems that are characterised as concurrent, distributed, parallel or non-deterministic. In our case study, PNs specify the semantic domain of business processes (BPEL) used for analysis and validation purposes.

3 Mapping and Bridging Technical Spaces

Technical Spaces and Domains specify contexts that can be represented by equivalent metamodels. In order to re-use their contexts, core characteristics and specialised tools, it makes sense to shift from one Space or Domain to another (please refer to Fig. 1). In this respect, Kurtev et. al. [14] promote 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 exploiting the full potential of each technology.

Furthermore, the Model Driven Development and Engineering can automate the generation of the corresponding target Space or Domain instances [2], [3], [4], by providing a supporting infrastructure and utilising appropriate tools and meta-modelling techniques. Transformations execute the mapping rules established upon well-defined metamodel elements. As a result, they provide bridges enabling the inter-operability of different technologies by formalising the inter-relationships and transformation rules of their Technical or Domain metamodels.

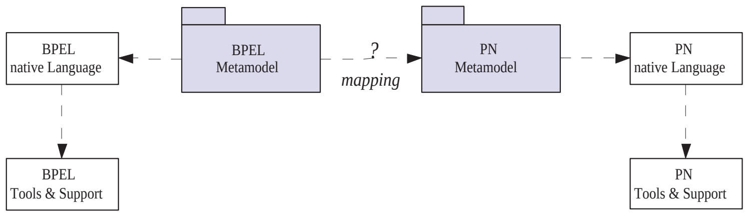
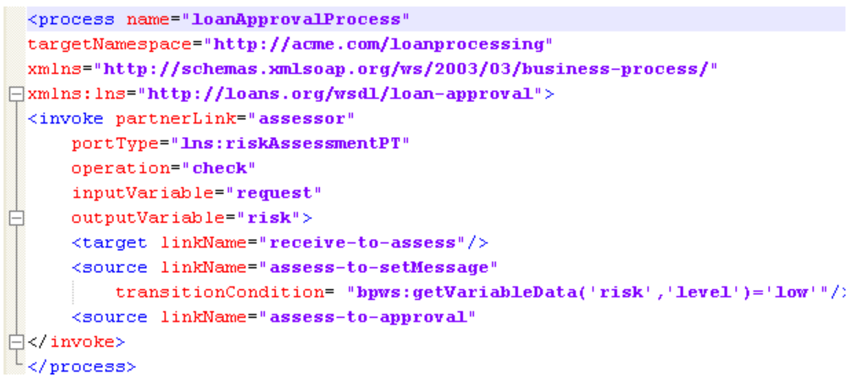


Fig. 1. Mapping BPEL and PN Technical Spaces and representations

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 consistent 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.

In reality, the BPEL language [5] specifies business process models via XML Schemas and technologies as depicted in Fig. 2. Similarly, PNs [15]



```
<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. A sample BPEL scenario

specify processing systems in graphical notations in terms of places, transitions and tokens as shown in Fig. 3.

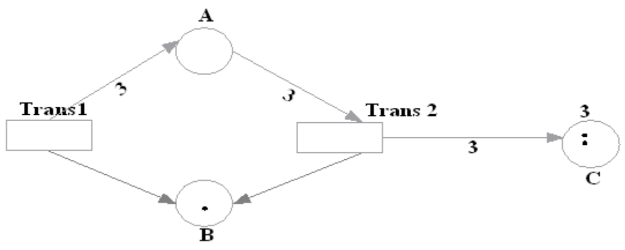


Fig. 3. A sample PN model

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 is conducive to our objective to map them together upon common means, tools and techniques. In order to do so, we assume that metamodelling techniques are sufficiently capable to specify the languages precisely in terms of meta-classes and meta-relationships. For BPEL and PNs we do not encounter any significant problems [18], [19]; following Fig. 4 and Fig. 5 depict their equivalent metamodel representations.

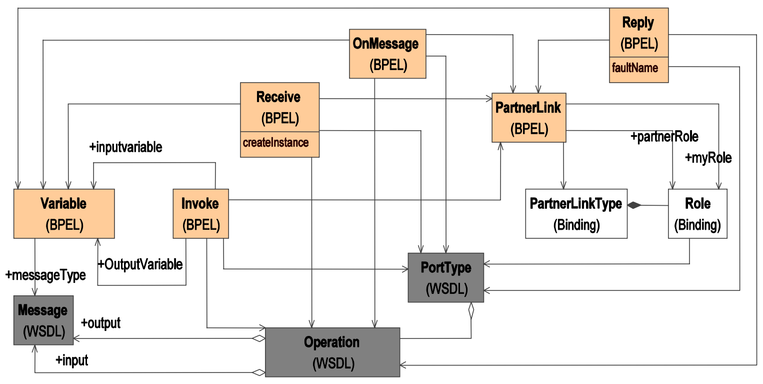


Fig. 4. A metamodel segment of BPEL

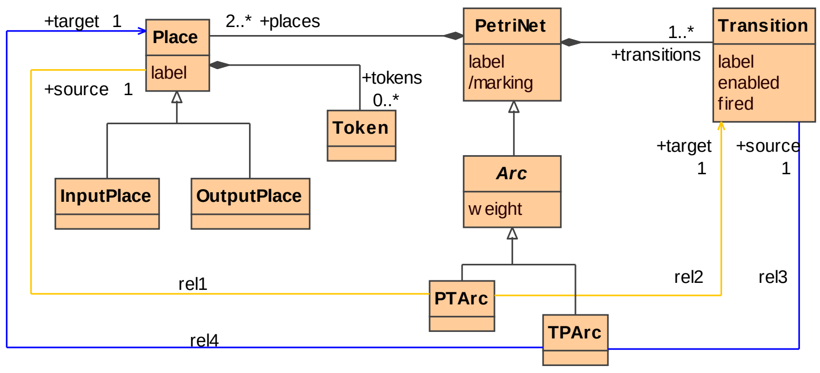


Fig. 5. A metamodel for simple Petri nets

Even if Fig. 4 depicts just a segment of the BPEL metamodel, it is rather clear that the metamodel of BPEL is more sophisticated than the metamodel of PN, including complicated, high-level constructs such as “Invoke”, “Receive” and “PartnerLink”. As a result, there is a visible gap between the two Technical Spaces that needs to be filled.

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 and matching their corresponding conceptual meta-elements is not an easy task. For example, the BPEL metamodel consists of numerous elements such as “Invoke” and intricate structures such as “Sequence” and “While”. As a result, the BPEL metamodel is regarded as much more complicated when compared to a PN metamodel, which is defined upon just few model elements such as “Place” and “Transition”. For that reason, there is a need to systematically analyse the languages and their constituent elements into categories, depending on their characteristics, as well as develop methods that would gradually build and map their corresponding metamodel definitions.

Mapping Technical Spaces: In general, languages and consequently metamodels specify and distinguish their elements into two broad categories: a) the *Primary Elements* that are regarded as atomic, elementary units, representing fundamental concepts of a domain and cannot be analysed further, for example, an *Action* in UML and a *Place* in a PN language and b) the *Composite Elements* that are regarded as more complicated entities, where their definitions are based upon other primary elements and compositions. Usually, they represent more complicated and abstract concepts of their domain, for example, a composite structure in UML or a complete Petri net in a PN language or metamodel. The mapping process is initiated by distinguishing and mapping the corresponding primary concepts and elements at source and target languages. Then, the process continues gradually by allocating the more complicated and compound concepts and characteristics till a complete language mapping is achieved.

Refinement Approach: The *refinement* paradigm has become an inevitable step in software development and process, aimed to solve complicated problems. In general, *refinement* refers to the verifiable transformation of a higher level specification (abstract) into a lower level (concrete) specification [23]. Logically the process of *refinement* involves implication, specified as a set of rules. The *stepwise refinement* allows the process to be done in successive stages so that details are added incrementally.

Metamodel Refinement: Our approach applies the *refinement* paradigm at metamodel level, where a more general metamodel specification such as *PN* is refined to a more detailed metamodel specification such as *xPN*. The transformation (implication) is specified by QVT and OCL rules or alternative by graph transformations. The successive refinement of the initial metamodel would create a final metamodel specification, permitting a complete and thorough mapping to a different domain, which in this case is represented by the BPEL metamodel. Therefore, the successive metamodel refinement approach provides a method for bridging the conceptual gap among the Technical Spaces of BPEL and PN that are rather different. More specifically, the method proposed to bridge the discrepant Technical Spaces of BPEL and PN is applied upon the destination PN metamodel, as depicted in Fig. 6 and it is described as follows:

Approach Description: 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). At this example, 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 leads 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.

As a result of the refinement, the technical spaces of M_κ and N are now close enough to be mapped completely, as depicted by ψ_κ. Now, if all step

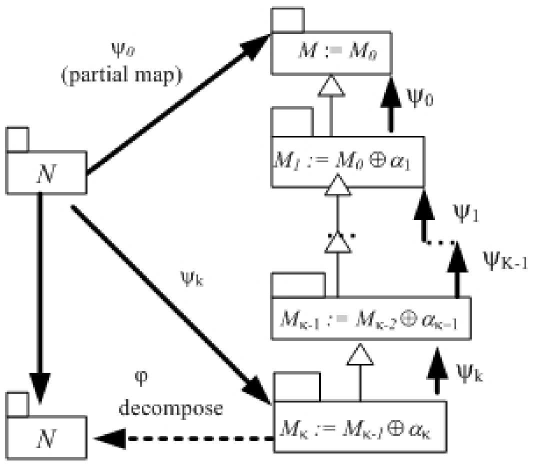


Fig. 6. Refinement of the Destination Metamodel

refinements from M to M_κ are decomposable (see Definition 4.1), then it is possible to define model transformations ψ_ι from M_ι to $M_{\iota-1}$ $(1 \leq \iota \leq \kappa)$ such that the mapping $\varphi = \psi_0 \circ \psi_1 \circ \psi_2 ... \psi_\kappa \circ \psi$ maps N to M.

Definition 4.1 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.

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

The successive refinement of the destination metamodel can be implemented either by profiles (referred as light-weight 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 meta-models respectively. The aim is to establish a bridge between these Technical Spaces by successively refining the destination M (PN) metamodel. To demonstrate the method and raise a number of related issues, the BPEL "Invoke" activity will be mapped to an equivalent Petri net representation.

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. 4 describes "Invoke" and its associated elements, as input and output variables used by operations provided by participants. The BPEL "Invoke" activity is considered as a primary BPEL element, specifying a particular type of behaviour interaction, which may be reused by other composite and more complicated elements defining groups and coordinating business activities such as "Flow" and "Sequence".

Metamodel Refinement - A Synthesis Perspecive: Within the destination metamodel (PN) the “Invoke” concept 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, the “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. 7 and its equivalent PN instance representation, qualified with an according PN representation as the one depicted in Fig. 8.

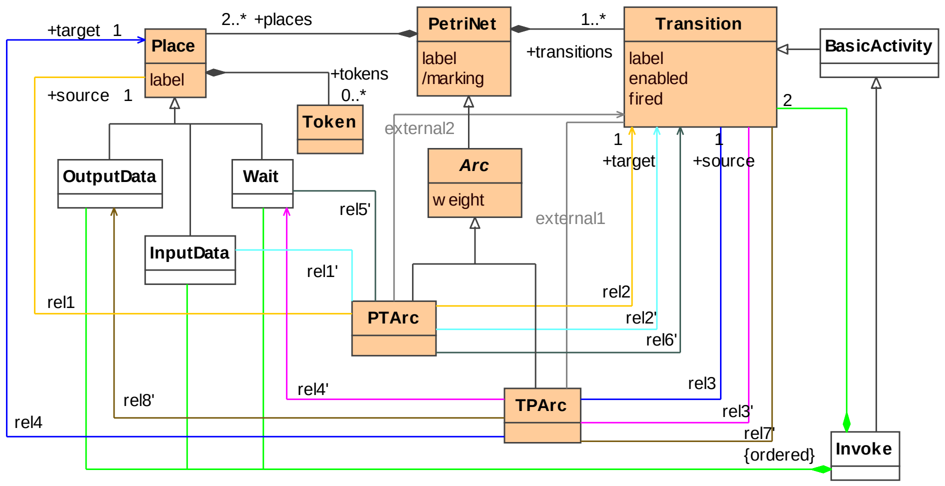


Fig. 7. A PN metamodel extension for “Invoke”

The introduced PN “Invoke” element represents a new primary concept, created as a self-contained element, allowing to be reused by more complicated 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 the process with its external participant.

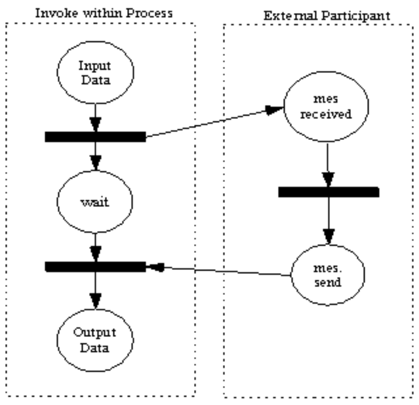


Fig. 8. A PN instance (extension) representation for “Invoke”

The PN model of Fig. 8 depicts an instantiation of its metamodel for performing the “Invoke” operation as previously described, according a PN notation.

Metamodel Composition - The Synthesis Rules: The following Table 1 highlights in detail which elements of M are refined to elements of the $M \oplus \alpha$ increment, representing the “Invoke” concept when applying the *one step refinement* approach.

Refined Elements (M)	Refining Elements ($M \oplus \alpha$)
Transition	Transition, Invoke
Place	Place, Wait, InputData, OutputData
rel1	rel1, rel1’, rel5’
rel2	rel2, rel2’, rel6’, external2
rel3	rel3, rel3’, rel7’, external1
rel4	rel4, rel4’, rel8’

Table 1
Refined Elements for “Invoke” concept

At $M \oplus \alpha$ increment, new associations such as *rel1’* and types such as “Invoke” are introduced. The *rel1* association is now refined to a *rel1* (following the previously defined rules), *rel1’* and *rel5’* (where new rules are introduced). In particular, the *rel1’* redefines the *rel1* (refer to Fig. 5) applied among ordinary “Places” and “Transitions”. The *rel1’* is now applied among “Places” that are of “InputData” type and “Transitions” that are contained by other “Transitions” having an “Invoke” type. Similar redefinition rules are applied to the rest associations, however, there are associations of particular interest involving "Places" and "Transitions" which belong to different PN models, thus they are regarded and denoted as "external" associations, such as *external1*.

The refinement rules can be formalised in a QVT/OCL language and executed by a transformation tool such as ATLAS [24]. For example the refinement rules for "Place" can be described as follows:

Example 5.1 Pseudo-code

```
--Select all place instances. For each of them ...
--if there is not an associated invoke element return Place
--otherwise if the place is the first element of the invoke places
--(returns an ordered set) return InputData
--else if it is the second element return Wait
--otherwise return OutputData
```

OCL Description

```
package xPN
context Place:: getPlaceType():Place
post: let p:Set(Place)= Place.allInstances in
result =
  if p.invoke.isEmpty() then p.oclAsType(Place)
  else
    if p.invoke.places.first()=p then p.oclAsType(InputData)
    else
      if p.invoke.places.at(2)=p then p.oclAsType(Wait)
      else p.oclAsType(OutputData)
      end
    end
  end
endpackage
```

Finally, the set of the refined rules specifies the *one step refinement* of α (representing the Invoke concept within PNs), mapping M to $M \oplus \alpha$.

Within the PN domain, the $M \oplus \alpha$ metamodel extension is now a M refinement defining an "Invoke" element and enabling the resulting PN metamodel to be mapped effectively with its corresponding BPEL "Invoke" elements. For example, Fig. 4 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 characteristics. So, it justifies our initial aim to use it for analysis and verification purposes. For example, the internal states and transitions of "Invoke" (see Fig. 7) 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.

Metamodel Decomposition: 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 (please refer to preliminary section). In that case we have to rewrite the metamodel and assist as much as possible the semantic interpretation of its decomposed elements:

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" and three specialised "Places". The element may also have internal defined OCL statements, which need to be transferred equivalently as well. As "Invoke" is of transition type 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, see Fig. 9. As the "composition relation" accommodates the need to depict the constituent elements of the "Invoke" structure, which actually no longer exists ("Invoke" is decomposed), then there is no reason to be retained.

However, 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 via parsing. Similarly, the decomposed "Places" may be identified if they are appropriately annotated by QVT/OCL rules (provide algorithm) which construct a conceptual context (e.g. "Invoke") as in parsing. In this case the most relevant type or element for "OutputData", "InputData" and "Wait" is that of a "Place". As a "Transition" is allowed to have associations with "Places" via "Arcs", there is no problem to preserve their relationships and model them analogously, however now they belong to the "Place" context and not to "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 of them are of the same type "Place". However, we may address such problems at some extent, if we annotate the association ends with their type names or use the attribute "Label" (see Fig. 5) 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 it. Alternatively, we may mark the modelling element with stereotypes, which effectively is similar to trying to make the metamodels profileable.

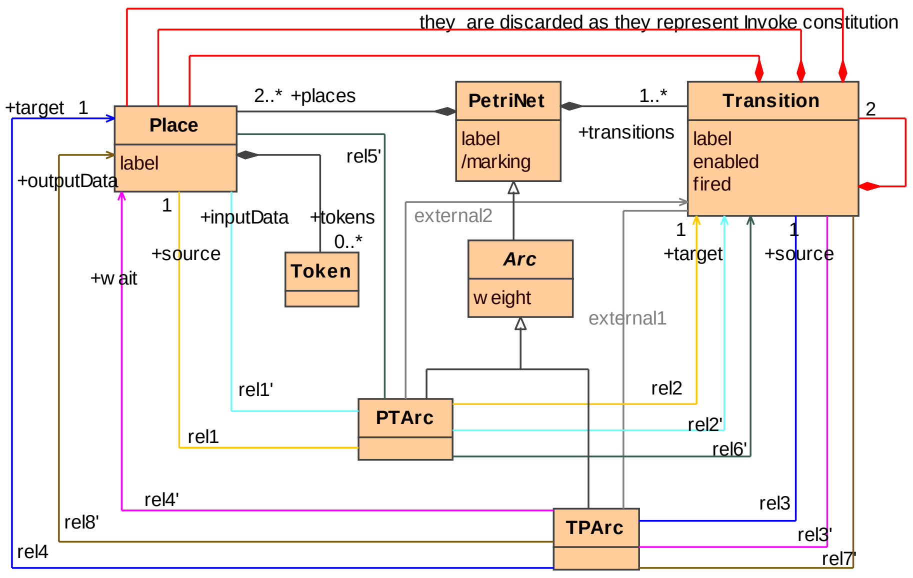


Fig. 9. Decomposing Invoke PN metamodel elements regarding the original

Finally, the mapping of a BPEL "Invoke" element to a decomposed PN extension depends largely on to how one has managed to decompose successively 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 it would be much more difficult to define and more complicated to distinguish than before.

6 Discussion

There are two major prerequisites for the success of our method. The first point is that, as depicted in Fig. 6, 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 source. This is a "bottom up" modelling approach. 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 statements. 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 that 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 4.1, 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 comes to decompose them we may loose semantic information and as a result the model elements may be misinterpreted during instantiation and mapping from tools. For that reason we have to devise ways to incorporate such information to our original metamodels. Various techniques can be applied such as using OCL expressions, labelling association ends and stereotypes. Further research on the matter is required.

7 Summary

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 examples 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] Bezivin, 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/j̃mfavre

[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. Bezivin, 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, *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, van Hee, and Houben, *Modelling and analysing workflow using a Petri-net based approach*, Proc. 2nd Workshop on Computer Supported Co-operative 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)

[20] OMG, *MDA Guide Version 1.0.1*, Object Management Group, Document Number: omg/2003-06-01 June, (2003)

[21] W. M. P. van der Aalst and A. H. M. ter Hofstede, *Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages*, Proc. of the Fourth International Workshop on Practical Use of Coloured Petri Nets and the CPN Tools, Denmark, Aug, p. 1-20, (2002)

[22] Karsten Schmidt and Christian Stahl, *A Petri net semantic for BPEL4WS - validation and application*, Proceedings of the 11th Workshop on Algorithms and Tools for Petri Nets (AWPN'04), Oct, p. 1-6, (2004)

[23] Pons, C., Prez,G., Giandini, R., Kutsche, Ralf-D, *Understanding Refinement and Specialization in the UML*, In. 2nd Int. Workshop on Managing Specialization/Generalization Hierarchies. In IEEE ASE 2003, Canada.

[24] Jean Bezivin, Frederic Jouault, David Touzet, *An introduction to the ATLAS Model Management Architecture*, Lina, Research Report no 05.01, Feb 2005