Multilevel modelling of coloured Petri nets

Alejandro Rodríguez[1], Adrian Rutle[1], Francisco Durán[2], Lars Michael Kristensen[1] and Fernando Macías[1]

[1]Western Norway University of Applied Sciences, Bergen
[1]{arte,aru,lmkr,fmac}@hvl.no
[2]Universidad de Málaga, Spain
[2]duran@lcc.uma.es

Abstract. Coloured Petri Nets (CPNs) is a modelling language for distributed systems which has been applied in a multitude of industrial cases. The supporting tool of CPNs is currently lacking important features such as having the possibility of tailoring the tool for specific domains and separation of concerns for facilitating its extensions and adaptation to new domains. In this paper, we present first steps towards using MultEcore as a multilevel modelling technique to tackle some of the CPN Tools main challenges by modelling CPNs and the CPN Tools using domain-specific multilevel modelling hierarchies. In particular, we take advantage of these hierarchies to first allow the definition of domain-specific CPNs languages and second to establish a clear separation between the CPN behaviour and the data types declarations.

1 Introduction

Model-driven software engineering (MDSE) tackles the increasing complexity of software by utilizing abstractions and modelling techniques. Traditional modelling approaches such as the Eclipse Modelling Framework (EMF) [20], the Unified Modelling Language [3] and MetaCase [2], limit the number of abstraction levels. In some cases this limitation leads to an increased model complexity and convolution [11]. The main reason for this is that real-life domains do not necessarily or naturally fit into a limited number of abstraction levels. Indeed, for a huge number of cases, 2-level techniques are not enough to intuitively specify certain domains [4]. Moreover, the construction of Domain-Specific Modelling Languages (DSMLs) gets likewise affected by this limitation in abstraction levels [22]. As a response to these challenges, Multilevel Modelling (MLM) has been proved a successful approach in many situations [11], such as software architecture or enterprise/process modelling domains.

One of the industrial domains in which MDSE has successfully been applied is distributed systems. Coloured Petri Nets (CPNs) [7] is a framework in this domain which facilitates, among others, specification of communication protocols [6], data networks [5] and distributed algorithms [18]. However, the main tool for CPNs, CPN Tools, is not designed to be easily extended with domain-specific features, although DSMLs have proved to be one of the most important tools of MDSE [16]. Moreover, CPN Tools [1] lacks support for basic modelling concepts

such as modularisation and separation of data type declaration from behaviour definition, thus hampering its extensibility and adaptability to new domains. To tackle these challenges, we propose using an MLM approach to provide an infrastructure for CPN Tools using MultEcore, which is designed to combine the best of both the mature ecosystem of EMF and the multilevel modelling approach [13, 14]. Specifically, MultEcore facilitates the definition of CPN-based metamodels—and hence DSMLs—through the support of an unlimited number of abstraction levels. Moreover, it addresses the challenges introduced in the CPNs case through separating CPN behaviour from data types declarations using the so-called supplementary hierarchies. In fact, MultEcore also allows the definition, assignment and reuse of multiple data type hierarchies. The separation of these concerns provides an improved reusability for the construction of domain-specific CPNs.

In Sect. 2 we introduce CPNs main concepts and describe the aspects which motivate this work. Section 3 presents our approach and details our contribution based on multilevel modelling techniques. In Sect. 4 we discuss related work. Finally in Sect. 5 we summarize conclusions and outline directions for future work.

2 Motivation

First, we briefly introduce CPNs and recapitulate its main concepts. CPNs belong to the class of high-level Petri nets which combine classical Petri nets (PNs) [17] with a programming language [21]. The use of a programming language provides the primitives for the definition of data types, for describing data manipulation and for creating compact and parameterisable models. A CPN model describes the conditions (*places*) of the system and the events (*transitions*) that can cause the system to change its state. *Arcs* can connect places to transitions or vice versa, but it is illegal to connect places with places or transitions with transitions. Figure 1 shows a small example of how a simple CPN model looks like with the concrete syntax we have defined. Circles like Clients and Messages represent places, while rectangles like SendRequest and ProcessRe-

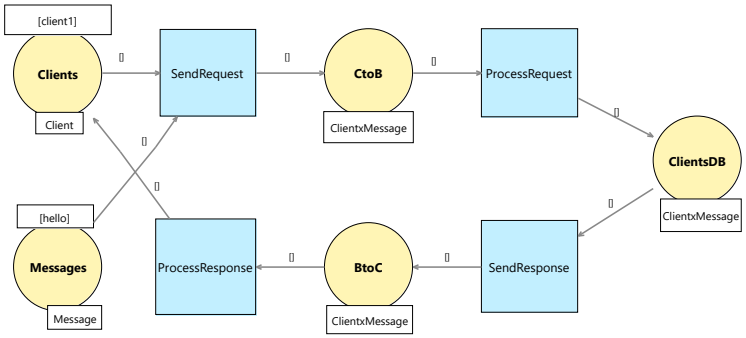
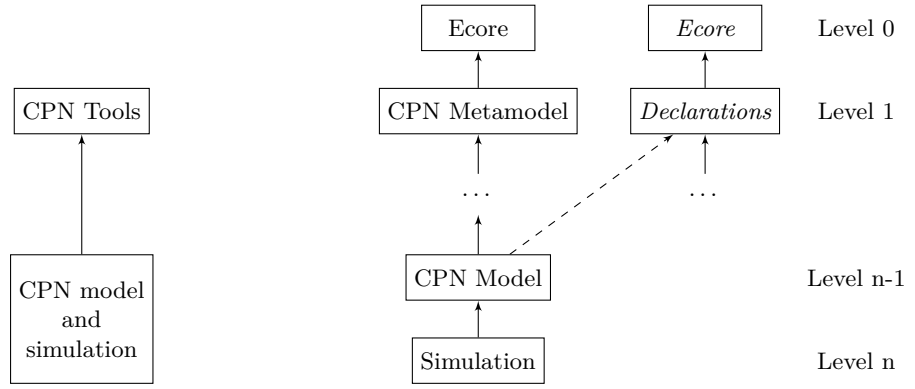


Fig. 1: CPN Model - Concrete syntax view



(a) CPN Tools (b) MultEcore approach

Fig. 2: MLM approach for CPNs

sponse represent transitions; places and transitions are connected by means of arcs.

Each place can hold an arbitrary number of *tokens*, providing the *marking* of the place. The tokens are generated by analyzing the initial marking expressions, which can be either function calls or literal expressions. The set of possible token colours is specified by means of a type (as known from programming languages), and it is called the *colour set* of the place. Client, Message and ClientxMessage are three colour sets which are written in the white rectangles below the places. Message is represented by a content; the Client colour set consists of a name; ClientxMessage is a structure composed by a Message and a Client (see details in Sect. 3.3 where colour sets are defined as classes). In Fig. 1, there are two initial marking expressions [client1] and [hello] that will generate a Client and a Message token, respectively. The number of tokens and their colours on the individual places represent the state of the system (called the marking of the model).

CPN Tools [1, 9] is a tool that supports the construction, simulation (execution), state space analysis, and performance analysis of CPN models. Although CPN Tools has been successfully used for modelling distributed systems, there are several challenges which motivate us to create an MLM infrastructure.

Implicit levels. In the current CPN Tools implementation there are multiple implicit levels of abstraction which are forced into a 2-level modelling technique (see Fig. 2a). From a multilevel modelling perspective, we identify at least four levels as presented in Fig. 2b. The first two levels are intended to represent Ecore—the metamodel of EMF—and the general concepts of CPNs, respectively; it is also possible to introduce more levels between these two in order to specify and reuse general Petri nets-like behaviour [19]. Below these, one can define several levels to specify domain-specific versions of CPNs, which in turn can be used to define CPN-based DSMLs. We estimate that, in general, two or three levels in between will be enough to specify a certain CPN domain (note that

MultEcore does not restrict the number of levels). These DSMLs may declare special kinds of places and transitions with special behaviour and constraints; for example, a domain specific transition that may only allow outgoing arcs. The Level n-1 is a specific CPN model that will be executed at Level n. The lowest level is generated by executing the initial markings which are put on the places in Level n-1; these are expressions that specify the tokens which allow us to execute the model. The hierarchy shown in Fig. 2b can be assigned to the so-called application hierarchy in MultEcore, which specifies the CPN DSML (see Sect. 3.3).

Token instantiation. The concept of token also fits perfectly within the usage of multilevel modelling since it is defined in the Level 1, together with the rest of basic concepts of CPNs. However, its instantiation needs to be done (at least) two levels below. This is because simulating the model requires first the analysis of the initial marking expressions. Therefore, it is only allowed to instantiate tokens in the Level n. This restriction can be established by using the concept of *potency* [15]. In the case of MultEcore, a potency specification includes three values: the first level where one can directly instantiate an element (min), the last level where one can directly instantiate an element (max), and the number of times the element can be indirectly re-instantiated (depth). For example, in Fig. 3 the potency for the class Token will be 2-*-1, since we want to allow only one instantiation (depth = 1) of Token two (min = 2) or more (max = *) levels below—where '*' means unbound [14, 15].

Data type declaration. A key difference of CPNs (with respect to traditional PNs) is that one can define data types (colours) for the models. One of the issues of CPN Tools comes from the fact that it mixes the data types together with the behaviour of the system which is represented by the CPN model. Keeping the data types and the behaviour separated provides a great potential for flexibility and reusability. Recall that the CPN model will belong to an application hierarchy, and now the data types that the models in the hierarchy might use can be defined in so-called supplementary hierarchies, making them independent from the CPN models (hierarchy with italics font in Fig. 2b). Using MultEcore, model designers can work with different multilevel hierarchies and fuse them with each other. This allows the concepts to have at least one type from the levels above in the application hierarchy and potentially one other type per incorporated supplementary hierarchy [12–14].

Table 1 summarizes the advantages of using MLM and MultEcore as presented in this paper. The elements appearing in the two first columns come from the analysis of CPNs and the CPN Tools and their challenges. The last two columns show the solutions we provide for each of the identified issues both referring to the technique from the theoretical background and to the concrete mechanism developed in MultEcore.

3 Multilevel infrastructure for CPNs

One advantage of the CPNs language is that it contains few but powerful modelling constructs, hence, the modeller has few constructs that need to master in

Table 1: Main ideas and contributions

<table>
<tr><th colspan="2">CPN Tools</th><th colspan="2">Infrastructure approach</th></tr>
<tr><th>Ideas</th><th>Challenges</th><th>MLM technique</th><th>MultEcore solution</th></tr>
<tr><td>Domain-specific CPN</td><td rowspan="2">2- levels oriented</td><td rowspan="2">Unlimited number of levels</td><td rowspan="2">Decoupling of levels from 0 (Ecore) to n (Simulation)</td></tr>
<tr><td>Behaviour is Multilevel</td></tr>
<tr><td>Separate initial marking and tokens</td><td rowspan="2">Implicit levels</td><td rowspan="2">Deep instantiation</td><td rowspan="2">Restrictions using 3-valued potency</td></tr>
<tr><td>Tokens belong to simulation level</td></tr>
<tr><td>Data types independent of behaviour</td><td rowspan="2">System behaviour and data types blended</td><td rowspan="2">Double typing</td><td rowspan="2">Suplementary hierarchies</td></tr>
<tr><td>Reusability of data types</td></tr>
<tr><td>DSMLs have commonalities</td><td rowspan="2">No mechanism for reusability and domain-specific constraints</td><td rowspan="2">Interlevel constraints</td><td>Interlevel constraints</td></tr>
<tr><td>Reuse of model transformations</td><td>MCMTs</td></tr>
</table>

order to use the language. Despite this strength, creating tools with concepts and rules tailored for specific domains would increase the potential application of CPNs. However, CPN Tools currently lacks this kind of functionality, which we propose to solve by introducing a multilevel infrastructure in MultEcore. In the following, we will explain this infrastructure and argue for several advantages which we gain by employing a multilevel modelling tool. Namely, explicit levels of abstraction paving the way for CPN-based DSMLs, clearer specification of behaviour by restricting instantiations of the *Token* concept and reusability of data types declarations by specifying them in separated MLM hierarchies.

3.1 Metamodel

Figure 3 shows the CPN metamodel (defined with MultEcore) which corresponds to the Level 1 (see Fig. 2b). In the metamodel we can see a root node representing a Net. A net contains Nodes (which can be either a Place or a Transition), Arcs and Expressions. Expressions are used to represent CPN inscriptions (expressions) which in the current CPN Tools are defined using the programming language Standard ML (SML). Also, a Place might hold Tokens.

Since the top level in MultEcore is Ecore, the nodes and arrows in this model are typed by the Ecore types EClass and EReference, respectively. These types are declared in blue ellipses (for nodes) and in labels in italics (for arrows). Arrows can support containment, with the same semantics of containment for EReference [20]; e.g. instances of Place must always be contained in instances of Net. We can also have inheritance arrows with the usual visualisation and semantics. The source class will have the target one as super type (declared in italics). Node is an abstract class (so it cannot be instantiated) from which Place

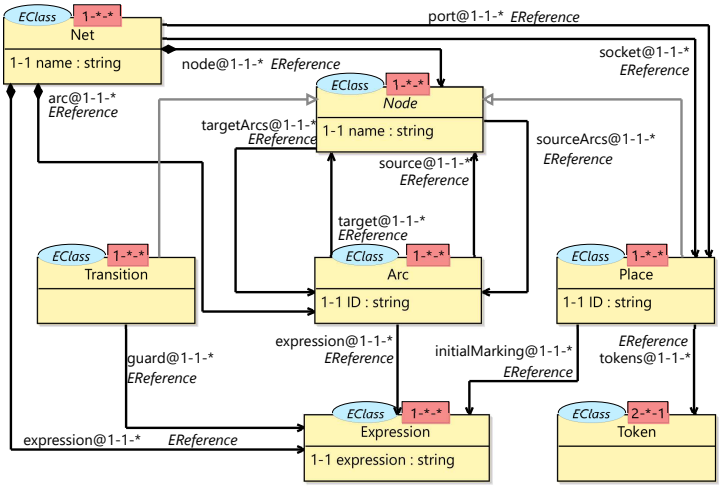


Fig. 3: CPN Metamodel

and Transition inherit all the attributes and methods. The 3-valued potencies (min, max, depth) of the nodes are defined in red rectangles on top of the nodes. Potency on arrows are defined after their names, separated by '@'. In the case of attributes, the value of depth is always 1 since we do not instantiate an instance (e.g., "Bob") of a data type (e.g., string). Hence, only start and end potencies are depicted in front of the name for attributes, with default value of 1-1.

3.2 CPN model

From the metamodel in Fig. 3, or a specialised version of it, one can create, for example, an editor which can be used to specify CPN models. These models (corresponding to Level n-1 in Fig. 2b) represent the systems which we model, be it a protocol or an industrial control system. The model in Fig. 4 depicts such a model which conforms to the metamodel in Fig. 3 and describes the abstract syntax of the model represented in Fig. 1. We have highlighted the names of the places and transitions that are explicitly shown in Fig. 1. This abstract syntax is the one that MultEcore provides out-of-the-box. However, it is straightforward to define a concrete, user-friendly syntax as the one presented in Fig. 1. The model specifies a scenario where an arbitrary number of clients can send messages to a broker, which acknowledges back with the corresponding messages to the clients.

3.3 Data type hierarchy

Recall that a place in a CPN model (an instance of the node Place in the metamodel) must be specified with a data type which restricts the type of tokens that the place accepts. In our proposal, behaviour description is separated from data types declarations; the former is specified in an application hierarchy consisting of CPN metamodel(s) and models, the latter is specified in its own hierarchy,

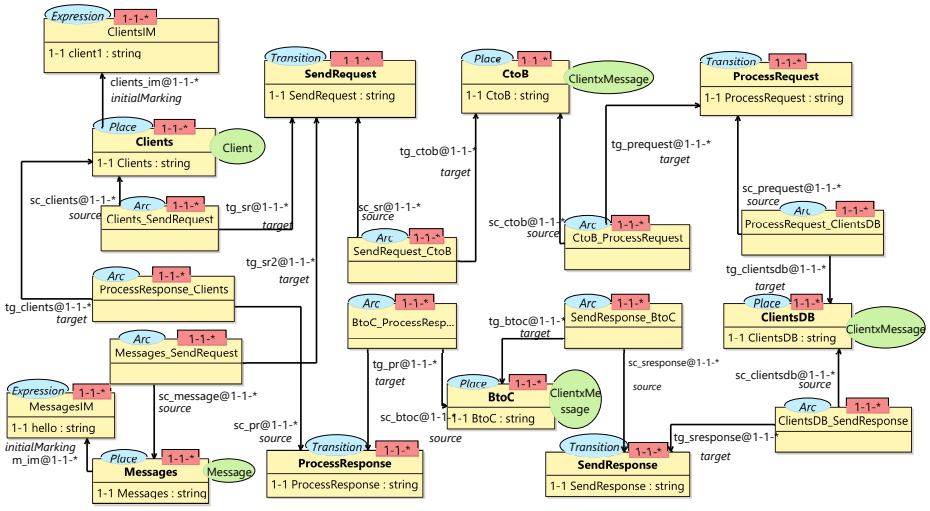


Fig. 4: CPN Model - MultEcore view

which we call supplementary hierarchy. As we see in Fig. 4, some model elements have two types: for a node, the blue ellipse specifies its type in the application hierarchy, while the green ellipse specifies the type in a supplementary hierarchy. One can see that Clients, Messages and ClientsDB places hold Client, Message and ClientxMessage types, respectively. Figure 5 shows these three data types declared as classes in an Ecore model. The Message class has an attribute content of type String. Places of type Message will carry tokens of such type, so they will have the shape of a typical string. The Client class has declared one attribute, name, which will represent the name of the client. ClientxMessage describes pairs of Client and Message elements. The same treatment of type compatibility applies to arc expressions. There must always be a correspondence between the supplementary types of arc expressions and places. For example, if the supplementary type of a certain place is String, the arc expression must evaluate also to String.

3.4 Behaviour and simulation

The behaviour can be specified with Multilevel Coupled Model Transformations (MCMTs) [15]. MCMTs are based on Graph Transformations – they are Turing complete – so they are powerful enough to represent the CPN semantics.

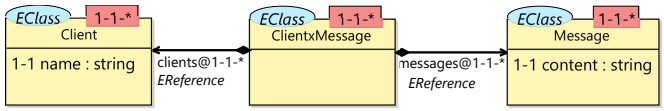


Fig. 5: Data types declarations

MCMTs allow us to specify rules across our multilevel hierarchy and can be applied in the simulation level. In our case the simulation model will be constructed by applying first the initial marking rules and then the behaviour rules. With MCMTs we will be able to introduce new levels without modifying the desired behaviour, and also to override or reuse specified behaviour when defining DSMLs. MCMTs can also be used to specify, check and enforce inter-level constraints. From the visualisation point of view, the simulation can be demonstrated in the instantiation level n-1, but conceptually we have simulation at the lowest level n.

3.5 CPN-based DSMLs and constraints

Using this multilevel infrastructure, one can further specialise the general CPN metamodel in Fig. 3, for example, with special types of places and transitions for a certain domain. These specialisations would be represented in metamodels typed by the general CPN metamodel, and would correspond to DSMLs and editors which can be used to define special CPN models. One could argue that such specializations can be carried on using inheritance in the same metamodel. However, the use of inheritance means that the metamodel becomes more complex, it collects concepts for a certain domain which would not be useful at all for another one or it might create inconsistencies in artifacts created conforming to such metamodel [14].

Recall that a place in a CPN model (an instance of the node Place in the metamodel) must be specified with a data type which restricts the type of tokens that the place accepts. This can be specified using an inter-level constraint which allows us to specify restrictions that need to be evaluated across different levels [14, 15]. This constraint could be expressed using a property/constraint specification language such as the LTL language defined in [14]. However, here we use MCMTs, since the constraint can be expressed by a simple implication. The intuition outlined in Fig. 6 shows that a token (instantiated in the simulation level) can only reside in a place if their data types are identical. The META block displays two levels separated with a red line and specifies that a token cannot be instantiated in the same level as P. In the level above the red line we can see such restriction in the potency on the Token node. In the FROM and TO blocks we can see how the pattern specifies that, for a token z to reside in a place p, both must have the same supplementary type Y. Note that it might be several levels between META and FROM/TO blocks.

CPN-based DSMLs may come with domain specific constraints which reflect the rules of the domain. For example, we could create a metamodel for a DSML in which we specify the place Buffer and the transition SendMessage where only outgoing arcs are allowed from SendMessage to Buffer. Such domain specific restriction can also be specified using an inter-level constraint, where creating an arc from SendMessage to Buffer would be disallowed.

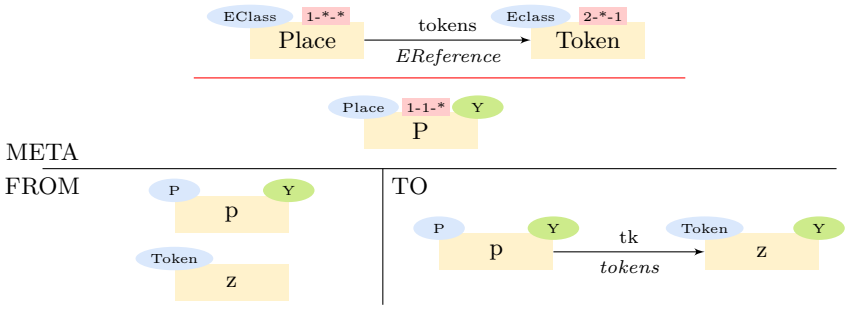


Fig. 6: Inter-level constraint forcing the same data type for places and tokens

4 Related work

We are not aware of other proposals for MLM infrastructures for CPNs. Therefore, in this section we compare our proposal with other MDSE frameworks that facilitate the definition of CPN models.

Renew [10] is a Java-based high-level Petri net simulator that provides a flexible modelling approach based on reference nets. The tool is not designed to be easily extended to other Petri net formalisms or for defining domain-specific variants for a tailored environment, which is the main focus of our approach.

The ePNK [8] is an Eclipse based platform for developing and integrating Petri nets tools. The tool provides a fixed Petri net metamodel which the modeller needs to extend (using inheritance relations) to provide the new Petri net type. This approach lacks, first, the separation of concepts in different levels of abstractions and, second, facilities for the definition of domain-specific versions since this would imply to modify the metamodel which already mixes two levels of abstraction (the core model and the new defined Petri net type).

5 Conclusions

In this paper we have described how multilevel modelling and the use of MultEcore can improve the CPNs language and associated tools and provide more flexibility, adaptability and reusability for domain-specific variants.

The MultEcore tool allows to work with separate projects—each one representing a hierarchy—and combine them in a flexible way. Our application hierarchy corresponds to the CPN project, that will contain at least the CPN metamodel to define CPN models. The data types to be used in the application hierarchy can be declared in several supplementary hierarchies.

As part of the future work, we plan to create a stronger support for data types declarations. We will incorporate the possibility of using collections (Lists, Maps, etc.) to create complex structures as data types. Moreover, current CPN Tools allows to organize a model into different modules, however they are managed in the same file. We plan to extend our approach to take into account modularity in the models providing an efficient mechanism to structure them.

References

1. CPN tools. http://cpntools.org/.
2. MetaCase. metacase.com/.
3. UML. http://www.uml.org/.
4. C. Atkinson and T. Kühne. Reducing accidental complexity in domain models. *Software & Systems Modeling*, 7(3):345–359, Jul 2008.
5. J. Billington and M. Diaz. *Application of Petri nets to Communication Networks: Advances in Petri nets*, volume 1605. Springer Science & Business Media, 1999.
6. J. Desel, W. Reisig, and G. Rozenberg, editors. *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, volume 3018 of *LNCS*. Springer, 2004.
7. K. Jensen, L. M. Kristensen, and L. Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3):213–254, Jun 2007.
8. E. Kindler. *EPNK: A Generic PNML Tool Users' and Developers' Guide*. DTU Informatics, 2011.
9. L. M. Kristensen and S. Christensen. Implementing coloured petri nets using a functional programming language. *Higher-order and symbolic computation*, 17(3):207–243, 2004.
10. O. Kummer. Renew–the reference net workshop.
11. J. D. Lara, E. Guerra, and J. S. Cuadrado. When and how to use multilevel modelling. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 24(2):12, 2014.
12. F. Macías, A. Rutle, and V. Stolz. Multilevel modelling with multecore a contribution to the multi 2017 challenge.
13. F. Macías, A. Rutle, and V. Stolz. MultEcore: Combining the best of fixed-level and multilevel metamodelling. In *MULTI*, volume 1722 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.
14. F. Macías, A. Rutle, V. Stolz, R. Rodriguez-Echeverria, and U. Wolter. An approach to flexible multilevel modelling. *To appear in EMISAJ, available at http://ict.hvl.no/approach-flexible-multilevel-modelling/*.
15. F. Macías, U. Wolter, A. Rutle, F. Durán, and R. Rodriguez-Echeverria. Multilevel coupled model transformations for precise and reusable definition of model behaviour. *Submitted for publication, online at https://goo.gl/wJ5VQk*, 2017.
16. P. Mohagheghi, W. Gilani, A. Stefanescu, M. A. Fernández, B. Nordmoen, and M. Fritzsche. Where does model-driven engineering help? Experiences from three industrial cases. *Software and System Modeling*, 12(3):619–639, 2013.
17. W. Reisig. *Petri nets: an introduction*, volume 4. Springer Science & Business Media, 2012.
18. W. Reisig. *Elements of distributed algorithms: modeling and analysis with Petri nets*. Springer Science & Business Media, 2013.
19. A. Rutle, F. Macías, F. Durán, R. Rodriguez-Echeverría, and U. Wolter. Describing Behaviour Models through Reusable, Multilevel, Coupled Model Transformations. Proceedings of NWPT 2016, pages 49–51, 2016.
20. D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.
21. J. D. Ullman. *Elements of ML programming*. Prentice-Hall, Inc., 1994.
22. M. Völter, S. Benz, C. Dietrich, B. Engelmann, M. Helander, L. C. L. Kats, E. Visser, and G. Wachsmuth. *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. dslbook.org, 2013.