ePNK Applications and Annotations

The ePNK is an Eclipse-based framework and platform for developing and integrating Petri net tools and applications. One of its core features is that new types of Petri nets can be realized and plugged into the ePNK without any programming by providing a model of the concepts of the new type, the so-called Petri net type edition (PNTD). Moreover, the ePNK allows developers customizing the graphical appearance of the features of a new Petri net type. The main idea and features of the ePNK have been presented before [1, 2]. One important aspect of the ePNK, however, has not been discussed yet: realizing new applications for the ePNK and, in particular, visualizing the result of an application in the graphical editor of the ePNK by using annotations, and interacting with the end user using these annotations.

In this paper, we give an overview of the concepts of ePNK applications by discussing the implementation of a simulator for YAWL nets [3].

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Software Engineering
Pragmatics annotated coloured petri nets for protocol software generation and verification

Pragmatics Annotated Coloured Petri Nets (PA-CPNs) are a restricted class of Coloured Petri Nets (CPNs) developed to support automated generation of protocol software. The practical application of PA-CPNs and the supporting PetriCode software tool have been discussed and evaluated in earlier papers already. The contribution of this paper is to give a formal definition of PA-CPNs, motivate the definitions, and demonstrate how the structure of PA-CPNs can be exploited for more efficient verification.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Software Engineering, Bergen University College
Authors: Simonsen, K. I. F. (Ekstern), Kristensen, L. M. (Ekstern), Kindler, E. (Intern)
Pages: 1-27
Publication date: 2016

Host publication information
Title of host publication: Transactions on Petri Nets and Other Models of Concurrency XI
Publisher: Springer
Editors: Koutny, M., Desel, J., Kleijn, J.
ISBN (Print): 978-3-662-53400-7
ISBN (Electronic): 978-3-662-53401-4
Series: Lecture Notes in Computer Science
Volume: 9930
ISSN: 0302-9743
Main Research Area: Technical/natural sciences
DOI: 10.1007/978-3-662-53401-4_1
Source: FindIt
Source-ID: 2346506584
Publication: Research - peer-review › Book chapter – Annual report year: 2016

Application and Theory of Petri Nets and Other Models of Concurrency: Special Issue of Selected Papers from Petri Nets 2014

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Software Engineering, Iowa State University
Authors: Ciardo, G. (Ekstern), Kindler, E. (Intern), Penczek, W. (Ekstern)
Number of pages: 2
Pages: V-VI
Publication date: 2015
Conference: 35th International Conference on Application and Theory of Petri Nets and Concurrency, Tunis, Tunisia, 23/06/2014 - 23/06/2014
Main Research Area: Technical/natural sciences

Publication information
Journal: Fundamenta Informaticae
Volume: 140
This book constitutes revised selected papers from the six International Workshops on Behavior Modelling - Foundations and Applications, BM-FA, which took place annually between 2009 and 2014. The 9 papers presented in this volume were carefully reviewed and selected from a total of 58 papers presented at these 6 workshops. The contributions were organized in topical sections named: modelling practices; new ways of behaviour modelling: events in modelling; and new ways of behaviour modelling: protocol modelling.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Software Engineering, University of Osnabrück, Open University of the Netherlands, Metamaxim Ltd.
All electronic communication relies on communication protocols. It is therefore very important that protocols are correct and that protocol implementations are reliable. Coloured Petri Nets (CPNs) have been widely used to model, analyse and verify communication protocols. However, relatively limited work has been done on transforming CPN model to protocol implementations. The goal of the thesis is to be able to automatically generate high-quality implementations of communication protocols based on CPN models.

In this thesis, we develop a methodology for generating implementations of protocols using a sub-class of CPNs, called Pragmatics Annotated CPNs (PACPNs). PA-CPNs give structure to the protocol models and allows the models to be annotated with code generation pragmatics. These pragmatics are used by our code generation approach to identify and execute the appropriate code generation templates. The templates hold the information needed to transform the model to a fully working protocol implementation for a target platform. The code generation approach coupled with PA-CPNs provide a flexible way to perform code generation for communication protocols. The code generation approach has been implemented in a prototype tool called PetriCode.

We defined several criteria for our code generation approach, the approach should be scalable so that it can be used to generate code for industrial sized protocols. The models should be verifiable and it should be possible to perform efficient verification on the models. The approach and the models that are employed for code generation should be platform independent in the sense that it should be possible to generate code for a wide range of platforms based on the same model. The generated code should be integrable meaning that it should be able to use different third party libraries and the code should be easily usable by third party code. Finally, the code should be readable by developers with expertise on the considered platforms.
In this thesis, we show that our code generation approach is able to generate code for a wide range of platforms without altering the PA-CPN model that describe the protocol design. The generated code is also shown to be readable and we demonstrate that a generated implementation can be easily integrated with third party software. We also show that our approach scales to industrial sized protocols by applying our approach to generate code for the WebSocket protocol. The WebSocket protocol creates a message-based two-way channel that can be used by web applications. This allows web applications to communicate with the server much more efficiently than using the traditional request-response pattern for certain application types such as games and rich web applications. Finally, we conclude the evaluation of the criteria of our approach by using the WebSocket PA-CPN model to show that we are able to verify fairly large protocols.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Software Engineering, Bergen University College
Authors: Simonsen, K. I. (Intern), Kristensen, L. M. (Ekstern), Kindler, E. (Intern)
Number of pages: 296
Publication date: 2015

Publication information
Place of publication: Kgs. Lyngby
Publisher: Technical University of Denmark (DTU)
Original language: English

Series: DTU Compute PHD-2014
Number: 345
ISSN: 0909-3192
Main Research Area: Technical/natural sciences
Electronic versions: phd345_Simonsen_KIF.pdf
Publication: Research › Ph.D. thesis – Annual report year: 2015

Pragmatic Annotated Coloured Petri Nets for Protocol Software Generation and Verification
PetriCode is a tool that supports automated generation of protocol software from a restricted class of Coloured Petri Nets (CPNs) called Pragmatic Annotated Coloured Petri Nets (PA-CPNs). Petri-Code and PA-CPNs have been designed with five main requirements in mind, which include the same model being used for verification and code generation. The PetriCode approach has been discussed and evaluated in earlier papers already. In this paper, we give a formal definition of PA-CPNs and demonstrate how the specific structure of PA-CPNs can be exploited for verification purposes.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Software Engineering, Bergen University College
Authors: Fagerland Simonsen, K. I. (Ekstern), Kristensen, L. M. (Ekstern), Kindler, E. (Intern)
Pages: 79-98
Publication date: 2015

Host publication information
Editors: Moldt, D., Rölke, H., Störrle, H.

Series: CEUR Workshop Proceedings
Volume: 1372
ISSN: 1613-0073
Main Research Area: Technical/natural sciences
Electronic versions: paper5.pdf
Links:
http://ceur-ws.org/Vol-1372/
Source: PublicationPreSubmission
Source-ID: 117995794
Publication: Research - peer-review › Article in proceedings – Annual report year: 2015
The Event Coordination Notation: Behaviour Modelling Beyond Mickey Mouse
The Event Coordination Notation (ECNO) allows modelling the desired behaviour of a software system on top of any object-oriented software. Together with existing technologies from Model-based Software Engineering (MBSE) for automatically generating the software for the structural parts, ECNO allows generating fully functional software from a combination of class diagrams and ECNO models. What is more, software generated from ECNO models, integrates with existing software and software generated by other technologies.

ECNO started out from some challenges in behaviour modelling and some requirements on behaviour modelling approaches, which we pointed out in a paper presented at the second BMFA workshop [1]; the integration with pre-existing software was but one of these requirements.

Different ideas and concepts of ECNO have been presented before – mostly with neat and small examples, which exhibit one special aspect of ECNO or another; and it would be fair to call them “Mickey Mouse examples”.

In this paper, we give a concise overview of the motivation, ideas, and concepts of ECNO. More importantly, we discuss a larger system, which was completely generated from the underlying models: a workflow management system. This way, we demonstrate that ECNO can be used for modelling software beyond the typical Mickey Mouse examples. This example demonstrates that the essence of workflow management – including its behaviour – can be captured in ECNO: in a sense, it is a domain model of workflow management, from which a fully functioning workflow engine can be generated.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Software Engineering, Technical University of Denmark
Authors: Jepsen, J. (Ekstern), Kindler, E. (Intern)
Pages: 133-164
Publication date: 2015

Host publication information
Title of host publication: Revised Selected Papers of the International Workshops on Behavior Modeling – Foundations and Applications (BM-FA 2009-2014)
Publisher: Springer
Editors: Roubtsova, E., McNeile, A., Kindler, E., Gerth, C.
ISBN (Print): 978-3-319-21911-0
ISBN (Electronic): 978-3-319-21912-7

Series: Lecture Notes in Computer Science
Volume: 6368
ISSN: 0302-9743
Main Research Area: Technical/natural sciences
Workflow engine, Meta-modelling, Behaviour modelling, Event Coordination, Code generation
DOIs: 10.1007/978-3-319-21912-7_6

Bibliographical note
This book contains revised selected papers of six annual International Workshops on Behaviour Modelling - Foundations and Applications, which were held from 2009 to 2014.
Publication: Research - peer-review; Book chapter – Annual report year: 2015


General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Software Engineering, Iowa State University
Authors: Ciardo, G. (ed.) (Ekstern), Kindler, E. (ed.) (Intern)
Number of pages: 395
Publication date: 2014

Publication information
Publisher: Springer
ISBN (Print): 978-3-319-07733-8
ISBN (Electronic): 978-3-319-07734-5
Original language: English
The purpose of a domain model is to concisely capture the concepts of an application's domain, and their relation among each other. Even though the main purpose of domain models is not on implementing the application, major parts of an application can be generated from the application's domain models fully automatically with today's technologies. The focus of today's code generation technologies, however, is mostly on the structural aspects of the domain; the domain's behaviour is often not modelled at all, or implemented manually based on some informal models, or the behaviour is modelled on a much more technical level.

The Event Coordination Notation (ECNO) allows modelling the behaviour of an application on a high level of abstraction that is closer to the application's domain than to the software realizing it. Still, these models contain all necessary details for actually executing the models and for generating code from them. In order to be able to model the behaviour of a domain, the ECNO makes the events in which the different elements of the domain could engage explicit. The local behaviour of an element defines at which time an element can engage or participate in an event. The global behaviour of the application results from different elements jointly engaging in such events, which is called an interaction. Which events are supposed to be jointly executed and which elements need to join in is defined by so-called coordination diagrams of the ECNO. Together, the models for the local and the global behaviour define the overall behaviour of the domain.

In this technical report, we discuss the main idea and philosophy of ECNO and its notation as well as all the subtle details and concepts - and we motivate the decisions made for its design. Moreover, we discuss the prototypical implementation of ECNO, which consists of a modelling environment based on Eclipse and the Eclipse Modeling Framework (EMF) and an execution engine, which fully supports all the concepts and features of ECNO discussed in this technical report. All the examples are based on EMF, but the ECNO Engine can be used with different other platforms or object-oriented code across different platforms, once some adapters are provided. Though the focus of this technical report is on the general concepts of ECNO, the examples discussed here work for version 0.3.2 of the ECNO Tool and Framework. The ECNO Tool as well as the examples are available from the ECNO Home page: http://www2.imm.dtu.dk/~ekki/projects/ECNO/.

Pragmatics Annotated Coloured Petri Nets for Protocol Software Generation and Verification
This paper presents the formal definition of Pragmatics Annotated Coloured Petri Nets (PA-CPNs). PA-CPNs represent a class of Coloured Petri Nets (CPNs) that are designed to support automated code generation of protocol software. PA-CPNs restrict the structure of CPN models and allow Petri net elements to be annotated with so-called pragmatics, which are exploited for code generation. The approach and tool for generating code is called PetriCode and has been discussed and evaluated in earlier work already. The contribution of this paper is to give a formal definition for PA-CPNs; in addition, we show how the structural restrictions of PA-CPNs can be exploited for making the verification of the modelled protocols more efficient. This is done by automatically deriving progress measures for the sweep-line method, and by introducing so-called service testers, that can be used to control the part of the state space that is to be explored for verification purposes.
A simulator for high-level Petri nets: An ePNK application

The ePNK is a platform for Petri net tools based on the PNML transfer format. One of its important features is its extensibility, which allows developers to plug in new Petri net types and new functions and applications for different kinds of Petri nets. The basic version of the ePNK provides an editor for high-level Petri nets, but no analysis or simulation functionality. In this paper, we present a simulator for high-level Petri nets, which supports most of the built-in operators of ISO/IEC 15909-2. As an additional feature, this simulator allows the simulation of so-called network algorithms. In this paper, we briefly show how to use this simulator from the end user's point of view. Moreover, we discuss some of the concepts underlying this simulator and its implementation.

General information
State: Published
Organisations: Department of Applied Mathematics and Computer Science, Software Engineering
Authors: Kindler, E. (Intern), Laganeckas, M. (Ekstern)
Pages: 3–19
Publication date: 2013
Main Research Area: Technical/natural sciences

Publication information
Journal: Petri Net Newsletter
Volume: 82
ISSN (Print): 0391-1804
Original language: English

Bibliographical note
Technical contribution.
Source: dtu
Source-ID: u::9990
Publication: Research - peer-review › Journal article – Annual report year: 2013

Code Generation for Protocols from CPN models Annotated with Pragmatics

Model-driven engineering (MDE) provides a foundation for automatically generating software based on models. Models allow software designs to be specified focusing on the problem domain and abstracting from the details of underlying implementation platforms. When applied in the context of formal modelling languages, MDE further has the advantage that models are amenable to model checking which allows key behavioural properties of the software design to be verified. The combination of formally verified models and automated code generation contributes to a high degree of assurance that the resulting software implementation satisfies the properties verified for the model. Coloured Petri Nets (CPNs) have been widely used to model and verify protocol software, but limited work exists on using CPN models of protocol software...
as a basis for automated code generation. In this report, we present an approach for generating protocol software from a restricted class of CPN models. The class of CPN models considered aims at being descriptive in that the models are intended to be helpful in understanding and conveying the operation of the protocol. At the same time, a descriptive model is close to a verifiable version of the same model and sufficiently detailed to serve as a basis for automated code generation when annotated with code generation pragmatics. Pragmatics are syntactical annotations designed to make the CPN models descriptive and to address the problem that models with enough details for generating code from them tend to be verbose and cluttered. Our code generation approach consists of three main steps, starting from a CPN model that the modeller has annotated with a set of pragmatics that make the protocol structure and the control-flow explicit. The first step is to compute for the CPN model, a set of derived pragmatics that identify control-flow structures and operations, e.g., for sending and receiving packets, and for manipulating the state. In the second step, an abstract template tree (ATT) is constructed providing an association between pragmatics and code generation templates. The ATT then directs the code generation in the third step by invoking the code templates associated with each node of the ATT in order to generate code. We illustrate our approach using an example of a unidirectional data framing protocol.
Generating Protocol Software from CPN Models Annotated with Pragmatics

Model-driven software engineering (MDSE) provides a foundation for automatically generating software based on models that focus on the problem domain while abstracting from the details of underlying implementation platforms. Coloured Petri Nets (CPNs) have been widely used to formally model and verify protocol software, but limited work exists on using CPN models of protocols as a basis for automated code generation. The contribution of this paper is a method for generating protocol software from a class of CPN models annotated with code generation pragmatics. Our code generation method consists of three main steps: automatically adding so-called derived pragmatics to the CPN model, computing an abstract template tree, which associates pragmatics with code templates, and applying the templates to generate code which can then be compiled. We illustrate our method using a unidirectional data framing protocol.

An ECNO semantics for Petri nets

The Event Coordination Notation (ECNO) allows modelling the behaviour of software on top of structural software models - and to generate program code from these models fully automatically. ECNO distinguishes between the local behaviour of elements (objects) and the global behaviour, which defines the coordination of the local behaviour of the different elements. The global behaviour is dened by ECNO's coordination diagrams, whereas the local behaviour of the different elements can, for example, be modelled by a simple form of Petri nets, ECNO nets. The ideas of ECNO have already been presented in earlier work. In this paper, we will show that the ECNO, in turn, can be used for modelling the behaviour of Petri nets in a simple and concise way. What is more, we will show that the ECNO semantics of Place/Transition Systems can easily be extended to so-called signal-event nets.
Business Process Management: 10th International Conference, BPM 2012 Proceedings

General information
State: Published
Organisations: Department of Informatics and Mathematical Modeling, Computer Science and Engineering, Software Engineering, Bergen University College
Authors: Simonsen, K. I. (Intern), Kristensen, L. M. (Ekstern), Kindler, E. (Intern)
Number of pages: 3
Publication date: 2012
Main Research Area: Technical/natural sciences

Code Generation for Protocols from CPN models Annotated with Pragmatics

General information
State: Published
Organisations: Department of Informatics and Mathematical Modeling, Computer Science and Engineering, Software Engineering, Bergen University College
Authors: Simonsen, K. I. (Intern), Kristensen, L. M. (Ekstern), Kindler, E. (Intern)
Number of pages: 3
Publication date: 2012
Main Research Area: Technical/natural sciences

Modelling Foundations and Applications: 8th European Conference, ECMFA 2012; Kgs. Lyngby, Denmark, July 2-5, 2012; Proceedings
The 20 revised full foundations track papers and 10 revised full applications track papers presented were carefully reviewed and selected from 81 submissions. Papers on all aspects of MDE were received, including topics such as architectural modelling and product lines, code generation, domain-specific modeling, metamodeling, model analysis and verification, model management, model transformation and simulation. The breadth of topics, as well as the high quality of the results presented in these accepted papers, demonstrate the maturity and vibrancy of the field.
Modelling Local and Global Behaviour: Petri Nets and Event Coordination

Today, it is possible to generate major parts of a software system from models. Most of the generated code, however, concerns the structural parts of the software; the code that concerns the actual functionality or behaviour of the software system is often still programmed manually. In order to address this problem, we developed the notation of coordination diagrams, which allows us to define the global behaviour of a software system on top of existing class diagrams. One of the major objectives of coordination diagrams was to make it easy to integrate them and the code generated from them with existing structural models, with existing code, and with other behavioural models. Basically, coordination diagrams define how the local behaviour of the individual parts of the software is coordinated with each other. The main concepts of coordination diagrams and their semantics are stabilising now: We call it the Event Coordination Notation (ECNO).

ECNO’s coordination diagrams define the global behaviour of a system only: they define how the local behaviour is coordinated and jointly executed in so-called interactions. In principle, ECNO is independent from a specific notation for modelling the local behaviour. For our experiments with ECNO, however, we implemented a simple modelling notation for the local behaviour, which is based on Petri nets: ECNO nets. Together, ECNO coordination diagrams and ECNO nets allow us to completely model a software system, and generate executable code for it.

In this paper, we discuss the general idea of ECNO and of ECNO nets. ECNO nets are implemented as a Petri net type for the ePNK tool, together with a code generator that produces code that can be executed by the ECNO execution engine.
On the dimensions of software documents — An idea for framing the software engineering process.

The ePNK: A generic PNML tool Users' and Developers' Guide for Version 1.0.0

The Petri Net Markup Language (PNML) is an XML based interchange format for all kinds of Petri nets, which was published as International Standard ISO/IEC 15909-2 in February 2011. Technically, ISO/IEC 15909-2 is defining an interchange format for three different kinds of high-level Petri nets and a simple version of Place/Transition systems only. But, one of the objectives of PNML was to provide a means for exchanging any kind of Petri net [10, 29, 1]. To this end, the concept of a Petri Net Type Definition (PNTD) was introduced, which is subject of a newly issued standardisation project: ISO/IEC 15909-3.

There are many tools supporting one form of PNML or the other, and, in particular, there is the PNML Framework [7], which helps tool developers to ease the implementation of PNML by providing a framework and an API for loading and saving Petri net documents in PNML. This framework is based on the Eclipse Modeling Framework (EMF) [2] and has its focus on the underlying meta-models of Petri nets. The PNML Framework, however, is not generic in the following sense: Whenever a new Petri net type is created, the code for the complete tool needs to be regenerated. Moreover, the PNML Framework does not come with a graphical editor for Petri nets.

The ePNK overcomes these limitations: It provides an extension point, so that new Petri net types can be plugged in to the existing tool without touching the code of the ePNK. For defining a new Petri net type, the developer, basically, needs to give a class diagram (actually an Ecore diagram) defining the concepts of the new Petri net type, along with a mapping of these concepts to XML syntax. This type can then be plugged into the ePNK, and the graphical editor of the ePNK will be able to edit nets of this new type with all its features. Likewise, the ePNK allows to plug in new functionality for analysis and verification of Petri nets or any other kind of application for Petri nets. Actually, this was the idea when we started the development of the Petri Net Kernel (PNK) about 15 years ago [17, 12, 20]. At that time, however, we had to implement all of the IDE functionality of such a tool ourselves. Today, we can make use of the Eclipse platform [27], which helps us focusing on the Petri net specific parts; we get all the functionality of a nice IDE, basically, for free. This is why we named the tool ePNK: it can be considered to be an Eclipse-based Petri Net Kernel. But, it is just the spirit and their idea that the PNK and the ePNK have in common; technically, there is not a single line of code from the PNK in the ePNK, and the ePNK is not compatible with the PNK.

What is more, we use the nice features of EMF, GMF, and Xtext for developing the ePNK in a model-based way. In this way, the complete development process of the ePNK is a case study in model-based software engineering using EMF and related technologies. This, actually, was the driving force behind this project. The evaluation and the lessons learned during this project, however, will be reported at an other occasion and to a different audience. This manual focuses on how to use the ePNK as an end user, and shows how a developer can use the extension mechanisms of the ePNK for providing new Petri net types along with their XML syntax, and how to add new functionality to the ePNK. A first version of this manual has been published in February 2011 as IMM-Technical Report-2011-03 already, which referred to version 0.9.1 of the ePNK. The current version of this document refers to version 1.0.0 of the ePNK, which was released in October 2012.

General information
State: Published
Organisations: Department of Informatics and Mathematical Modeling, Computer Science and Engineering, Software Engineering
Authors: Kindler, E. (Intern)
The Event Coordination Notation: Execution Engine and Programming Framework

ECNO (Event Coordination Notation) is a notation for modelling the behaviour of a software system on top of some object-oriented data model. ECNO has two main objectives: On the one hand, ECNO should allow modelling the behaviour of a system on the domain level; on the other hand, it should be possible to completely generate code from ECNO and the underlying object-oriented domain models. Today, there are several approaches that would allow to do this. But, most of them would require that the data models and the behaviour models are using the same technology and the code is generated together. By contrast, ECNO can be used for modelling the behaviour on top of any object-oriented model - or even on top of manually written object-oriented code. This way, it is easy to integrate ECNO models with other technologies, to use ECNO on top of code generated by other technologies or with code that was written manually. In this paper, we rephrase the main concepts of ECNO. The focus of this paper, however, is on the architecture of the ECNO execution engine and its programming framework. We will show how this framework allows us to integrate ECNO with object-oriented models, how it works without any explicit control, and how it easily integrates with traditional programming.

General information

State: Published
Organisations: Department of Informatics and Mathematical Modeling, Computer Science and Engineering, Software Engineering
Authors: Kindler, E. (Intern)
Pages: 8
Publication date: 2012

Host publication information

Title of host publication: BM-FA ’12 Proceedings of the Fourth Workshop on Behaviour Modelling - Foundations and Applications
Publisher: Association for Computing Machinery
ISBN (Print): 978-1-4503-1187-8
Main Research Area: Technical/natural sciences
Workshop: 4th Workshop on Behavioural Modelling - Foundations and Application (BM-FA 2012), Kgs. Lyngby, Denmark, 03/07/2012
Computer applications, Models
DOIs: 10.1145/2325276.2325279
Source: dtu
Source-ID: n:oai:DTIC-ART:compendex/370624891::19906
Publication: Research › peer-review › Article in proceedings – Annual report year: 2012

ePNK: A generic PNML tool - Users' and Developers' Guide: version 0.9.1

The Petri Net Markup Language (PNML) is an XML based interchange format for all kinds of Petri nets, which was published as International Standard ISO/IEC 15909-2:2011 in February 2011. Technically, ISO/IEC 15909-2 is dening an interchange format for three different kinds of high-level Petri nets and a simple version of Place/Transition systems only. But, one of the objectives of PNML was to provide a means for exchanging any kind of Petri net [8, 15, 1]. To this end, the concept of a Petri Net Type Denition (PNTD) was introduced, which is subject of a newly issued standardisation project
ISO/IEC 15909-3. There are many tools supporting one form of PNML or the other, and, in particular, there is the PNML Framework [6], which helps tool developers to ease the implementation of PNML by providing a framework and an API for loading and saving Petri net documents in PNML. This framework is based on the Eclipse Modeling Framework (EMF) [2] and has the focus on the underlying meta-models of Petri nets. The PNML Framework, however, is not generic in the following sense: Whenever a new Petri net type is created, the code for the complete tool needs to be regenerated.

Moreover, the PNML Framework does not come with a graphical editor for Petri nets. The ePNK overcomes these limitations: It provides an extension point, so that new Petri net types can be plugged in to the existing tool without touching the code of the ePNK. For defining a new Petri net type, the developer, basically, needs to give a class diagram (actually an.ecore-diagram) dening the concepts of the new Petri net type, along with a mapping of these concepts to XML syntax. This type can then be plugged into the ePNK, and the graphical editor of the ePNK will be able to edit nets of this new type with all its features. Actually, this was the idea when we started the development of the Petri Net Kernel (PNK) about 15 years ago [12, 10, 13]. At that time, however, we had to implement all of the IDE functionality of such a tool ourselves.

Today, we can make use of the eclipse platform [14], which helps us focusing on the Petri net specific parts: we get all the other functionality of a nice IDE, basically, for free. This is why we named the tool ePNK: it can be considered to be an eclipse based Petri Net Kernel. But, it is just the spirit and idea that the PNK and the ePNK have in common; technically, there is not a single line of code from the PNK in the ePNK, and they are not compatible. What is more, we use the nice features of EMF, GMF, and Xtext for developing the ePNK in a model-based way. In this way, the complete development process of the ePNK, is a case study in model-based software engineering using EMF and related technologies. This, actually, was the driving force behind this project. The evaluation and the lessons learned during this project, however, will be reported at another occasion and to a different audience. This manual will focus on how to use the ePNK as an end user, and it will show how a developer can use the extension mechanisms of the ePNK for providing new Petri net types along with their XML syntax, and how to add new functionality to the ePNK.

**General information**

**State:** Published  
**Organisations:** Software Engineering, Department of Informatics and Mathematical Modeling  
**Authors:** Kindler, E. (Intern)  
**Publication date:** 2011

**Publication information**

**Place of publication:** Kgs. Lyngby, Denmark  
**Publisher:** Technical University of Denmark, DTU Informatics, Building 321  
**Original language:** English

**Series:** IMM-Technical Report-2011-03  
**Main Research Area:** Technical/natural sciences

**Electronic versions:**

tr11_03.pdf  
Links:  

**Bibliographical note**

This is a previous version of IMM-Technical Report-2012-14: “The ePNK: A generic PNML tool Users’ and Developers’ Guide for Version 1.0.0”

Source: orbit  
Source-ID: 275136  
Publication: Research › Report – Annual report year: 2011

**Integrating Behaviour in Software Models: An Event Coordination Notation: Concepts and Prototype**

One of the main problems in model-based software engineering is modelling behaviour in such a way that the behaviour models can be easily integrated with each other, with the structural software models and with pre-existing software. In this paper, we propose an event coordination notation (ECNO) that deals with this problem. We present the main concepts and rationales behind this notation and discuss a prototype and run-time environment that executes these models, and provides an API so that other parts of the software can be easily integrated. The core concepts of the ECNO seem to be stabilizing now, and the prototypic implementation of ECNO and its runtime environment show that the concepts of ECNO work. Still, there are some design issues and open questions that we discuss in this paper.

**General information**

**State:** Published  
**Organisations:** Software Engineering, Department of Informatics and Mathematical Modeling  
**Authors:** Kindler, E. (Intern)  
**Pages:** 41-48  
**Publication date:** 2011
Integrating Design Decision Management with Model-based Software Development

Design decisions are continuously made during the development of software systems and are important artifacts for design documentation. Dedicated decision management systems are often used to capture such design knowledge. Most such systems are, however, separated from the design artifacts of the system. In model-based software development, where design models are used to develop a software system, outcomes of many design decisions have big impact on design models. The realization of design decisions is often manual and tedious work on design models. Moreover, keeping design models consistent with all made decisions is a time-consuming manual task that is often performed in peer reviews. In this thesis, a generic technology has been developed for extracting model differences from models and transferring them to other models. These concepts, called model-independent differences, can be used to specify realizations of decisions in design models. This way, recurring realization work of design decisions can be automated. Since the concepts are generic and not bound to design decisions, other recurring work on models can be automated as well, for instance, design patterns and refactorings. With such a technology at hand, design decision realizations can easily be specified and parts of the realization work can be automated. A binding is produced as a by-product that links documented decision outcomes to design model elements which are affected by the respective decisions. With a set of constraints, such a binding can be used to validate the consistency between the design and made design decisions. Whenever the evolving design models become inconsistent with realized decisions, developers are notified about the violations. The violations can be fixed by correcting the design, adjusting the binding, or by ignoring the causes. This substitutes manual reviews to some extent. The concepts, implemented in a tool, have been validated with design patterns, refactorings, and domain level tests that comprise a replay of a real project. This proves the applicability of the solution to realistic examples. The implementation of model-independent differences, called MPatch, is further contributed to the Eclipse open source project.

Modelling Local and Global Behaviour: Petri Nets and Event Coordination

Today, it is possible to generate major parts of a software system from models. Most of the generated code, however, concerns the structural parts of the software; the parts that concern the functionality or behaviour of a system are still programmed manually. In order to overcome this problem, we are developing the concept of coordination diagrams that define the global behaviour on top of structural software models. Basically, these diagrams define how the local behaviour of an element is coordinated with the behaviour of the elements it is connected to. The exact concepts of these coordination diagrams and their notation is still under development, but there exists a first prototype for experimenting and for fine-tuning its features. We call it the Event Coordination Notation (ECNO). For experimenting with the ECNO, we...
implemented also a simple modelling notation for the local behaviour, which is based on Petri nets. In this paper, we briefly discuss the general idea of the ECNO and then present ECNO nets that define the local behaviour of elements. They are implemented as a Petri net type for the ePNK tool, together with a code generator that produces code that can be used in the ECNO framework and runtime environment. This way, all the behaviour of a system can be modelled – and code can be generated that easily integrates with the structural models and existing software.

General information
State: Published
Organisations: Software Engineering, Department of Informatics and Mathematical Modeling
Authors: Kindler, E. (Intern)
Pages: 42-56
Publication date: 2011

Host publication information
Title of host publication: Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE’11)

Series: CEUR Workshop Proceedings
Number: 723
ISSN: 1613-0073
Main Research Area: Technical/natural sciences
Event coordination, Local and global behaviour modelling, ECNO nets, Code generation, Model-based Software Engineering
Links:
http://conferences.ncl.ac.uk/pn-acsd-11/
Source: orbit
Source-ID: 313749
Publication: Research - peer-review › Article in proceedings – Annual report year: 2011

The ePNK: An Extensible Petri Net Tool for PNML
The Petri Net Markup Language (PNML) is an XML-based interchange format for all kinds of Petri nets, which is an ISO/IEC International Standard 15909-2 now. The focus of this standard is on PNML as an interchange format for high-level Petri nets. PNML, however, is more general and allows exchanging all kinds of Petri nets. To this end, PNML introduced the concept of Petri Net Type Definitions. There are many tools supporting one form of PNML or another. In particular, there is the PNML Framework, which helps tool developers implementing an interface to PNML by providing a framework and an API for loading and saving Petri net documents in PNML. This framework is based on the Eclipse Modeling Framework and has the focus on the underlying meta-models of Petri nets. The PNML Framework, however, is not generic in the following sense: Whenever a new Petri net type is created, the code for the complete tool needs to be regenerated. Moreover, the PNML Framework does not come with a graphical editor. The ePNK overcomes these limitations: It provides an extension-point so that new Petri net types can be plugged into the ePNK without touching the code of the ePNK. For defining a new Petri net type, the developer, basically, needs to give a class diagram defining the concepts of the new Petri net type, along with a mapping of these concepts to XML syntax. This type can then be plugged into the ePNK, and its graphical editor will be able to edit nets of this new type with all its features. This paper present the main idea of the ePNK, and how to use and extend it.

General information
State: Published
Organisations: Software Engineering, Department of Informatics and Mathematical Modeling
Authors: Kindler, E. (Intern)
Pages: 318-327
Publication date: 2011

Host publication information
Publisher: Springer
ISBN (Print): 978-3-642-21833-0
ISBN (Electronic): 978-3-642-21834-7
Series: Lecture Notes in Computer Science
Number: 6709
ISSN: 0302-9743
Main Research Area: Technical/natural sciences
Comparing relational model transformation technologies: implementing Query/View/Transformation with Triple Graph Grammars

The Model Driven Architecture (MDA) is an approach to develop software based on different models. There are separate models for the business logic and for platform specific details. Moreover, code can be generated automatically from these models. This makes transformations a core technology for MDA and for model-based software engineering approaches in general. QVT (Query/View/Transformation) is the transformation technology recently proposed for this purpose by the OMG. TGGs (Triple Graph Grammars) are another transformation technology proposed in the mid-nineties, used for example in the FUJABA CASE tool. In contrast to many other transformation technologies, both QVT and TGGs declaratively define the relation between two models. With this definition, a transformation engine can execute a transformation in either direction and, based on the same definition, can also propagate changes from one model to the other. In this paper, we compare the concepts of the declarative languages of QVT and TGGs. It turns out that TGGs and declarative QVT have many concepts in common. In fact, QVT-Core can be mapped to TGG-rules. We show that QVT-Core can be implemented by transforming QVT-Core mappings to TGG-rules, which can then be executed by a TGG transformation engine that performs the actual QVT-transformation. Furthermore, we discuss an approach for mapping QVT-Relations to TGGs. Based on the semantics of TGGs, we clarify semantic gaps that we identified in the declarative languages of QVT and, furthermore, we show how TGGs can benefit from the concepts of QVT.

General information

State: Published
Organisations: Software Engineering, Department of Informatics and Mathematical Modeling, University of Paderborn
Authors: Greenyer, J. (Ekstern), Kindler, E. (Intern)
Pages: 21-46
Model-based Software Engineering: The challenges of modelling behaviour

The vision of model-based software engineering is to make models the main focus of software development and to automatically generate software from these models. Part of that idea works already today. But, there are still difficulties when it comes to behaviour. Actually, there is no lack in models for behaviour, but a lack of concepts for integrating them with the other models and with existing code. In this paper, we discuss some of the main challenges in behaviour modelling and integration and present some ideas on how to overcome them -- and, still, many open issues.
Process mining: A two-step approach to balance between underfitting and overfitting

Process mining includes the automated discovery of processes from event logs. Based on observed events (e.g., activities being executed or messages being exchanged) a process model is constructed. One of the essential problems in process mining is that one cannot assume to have seen all possible behavior. At best, one has seen a representative subset. Therefore, classical synthesis techniques are not suitable as they aim at finding a model that is able to exactly reproduce the log. Existing process mining techniques try to avoid such “overfitting” by generalizing the model to allow for more behavior. This generalization is often driven by the representation language and very crude assumptions about completeness. As a result, parts of the model are “overfitting” (allow only for what has actually been observed) while other parts may be “underfitting” (allow for much more behavior without strong support for it). None of the existing techniques enables the user to control the balance between “overfitting” and “underfitting”. To address this, we propose a two-step approach. First, using a configurable approach, a transition system is constructed. Then, using the “theory of regions”, the model is synthesized. The approach has been implemented in the context of ProM and overcomes many of the limitations of traditional approaches.
A framework for the definition of variants of high-level Petri nets

General information
State: Published
Organisations: Software Engineering, Department of Informatics and Mathematical Modeling
Authors: Kindler, E. (Intern), Petrucci, L. (Ekstern)
Pages: 121-137
Publication date: 2009
A primer on the Petri Net Markup Language and ISO/IEC 15909-2
ISO/IEC 15909 is an International Standard that is concerned with the high-level Petri nets. Part 1 defines the concepts, the mathematics, and the graphical notation -- and some variants of high-level nets. Part 2 of ISO/IEC 15909, which is currently under the last ballot to be an International Standard, defines a transfer format for high-level nets. The transfer format defined in Part 2 of ISO/IEC 15909 is (or is based on) the lempf(Petri Net Markup Language) (PNML), which was originally introduced as an interchange format for different kinds of Petri nets. In ISO/IEC 15909-2, however, it is used in a setting restricted to high-level nets and a simple version of Petri nets called Place/Transition-Systems. Future parts, of PNML will use the generality of PNML and also standardise some of its other concepts. For example, it is planned that Part 3 will define a module concepts known from modular PNML and will make the concept for defining new Petri net types explicit. In this paper, discuss PNML, its relation to ISO/IEC 15909 and the main ideas for the future extensions of PNML and its standardisation in Part 3 of ISO/IEC 15909-2.

Difference-based Model Synchronization in an Industrial MDD Process
Models play a central role in model-driven software engineering. There are different kinds of models during the development process, which are related to each other and change over time. Therefore, it is difficult to keep the different models consistent with each other. Consistency of different models is maintained manually in many cases today. This paper presents an approach for automated model differencing, so that the differences between two model versions can be extracted and stored. It can then be re-used independently of the models it was created from to interactively merge different model versions, and for synchronizing other types of models. The main concern is to apply our concepts to an industrial process, in particular keeping usability and performance in mind.

This article was also published at the Tenth Workshop and Tutorial on Practical Use of Coloured Petri Nets and CPN Tools (CPN '09), Århus, Denmark, Oct. 2009, pp. 101-120

Difference-based Model Synchronization in an Industrial MDD Process

Models play a central role in model-driven software engineering. There are different kinds of models during the development process, which are related to each other and change over time. Therefore, it is difficult to keep the different models consistent with each other. Consistency of different models is maintained manually in many cases today. This paper presents an approach for automated model differencing, so that the differences between two model versions can be extracted and stored. It can then be re-used independently of the models it was created from to interactively merge different model versions, and for synchronizing other types of models. The main concern is to apply our concepts to an industrial process, in particular keeping usability and performance in mind.

Keyword: Model Differencing, Model Merging, Model Synchronization
Modelling Constructs

There are many different notations and formalisms for modelling business processes and workflows. These notations and formalisms have been introduced with different purposes and objectives. Later, influenced by other notations, comparisons with other tools or by standardization efforts, these notations have been extended in order to increase expressiveness and to be more competitive. This resulted in an increasing number of notations and formalisms for modelling business processes and in an increase of the different modelling constructs provided by modelling notations, which makes it difficult to compare modelling notations and to make transformations between them. One of the reasons is that, in each notation, the new concepts are introduced in a different way by extending the already existing constructs. In this chapter, we go the opposite direction: We show that it is possible to add most of the typical extensions on top of any existing notation or formalism—without changing the formalism itself. Basically, we introduce blocks with some additional attributes defining their initiation and termination behaviour. This serves two purposes: First, it gives a clearer understanding of the basic constructs and how they can be combined with more advanced constructs. Second, it will help combining different modelling notations with each other. Note that, though we introduce a notation for blocks in this chapter, we are not so much interested in promoting this notation here. The notation should just prove that it is possible to separate different issues of a modelling notation, and this way making its concepts clearer and the interchange of models easier. A fully-fledged block notation with a clear and simple interface to existing formalisms is yet to be developed.

General information

State: Published
Organisations: Software Engineering, Department of Informatics and Mathematical Modeling
Authors: Kindler, E. (Intern)
Number of pages: 634
Pages: 122-141
Publication date: 2009
Towards a Standard for Modular Petri Nets: A Formalisation

General information
State: Published
Organisations: Software Engineering, Department of Informatics and Mathematical Modeling, Université Paris XIII
Authors: Kindler, E. (Intern), Petrucci, L. (Ekstern)
Pages: 43-62
Publication date: 2009

Host publication information
Title of host publication: Proceedings of the Application and Theory of Petri Nets : 30th International Conference
Publisher: Springer
Editors: Franceschinis, G., Wolf, K.
ISBN (Print): 3-642-02423-8
Series: Lecture Notes in Computer Science
Number: 5606
ISSN: 0302-9743
Main Research Area: Technical/natural sciences
Modular Petri Nets, Standardisation, High-Level Nets
DOIs: 10.1007/978-3-642-00899-3_2
Links: http://www.springerlink.com/content/e75372m005642813/
Source: orbit
Source-ID: 253951
Publication: Research - peer-review › Article in proceedings – Annual report year: 2009

Aspect-Oriented Modelling from a Different Angle: Modelling Domains with Aspects
In this paper, we report on a new approach of aspect-oriented modelling, which is particularly suited for domains with naturally born aspects as part of that domain: MoDowA for Modelling Domains with Aspects. Though these models are on a very high level of abstraction and could be made early in the development process, these models are fully operational in that they can be executed by an interpreter. This way, we shed a light on Aspect-oriented Modelling from a new, different angle.

General information
State: Published
Organisations: Software Engineering, Department of Informatics and Mathematical Modeling
Authors: Kindler, E. (Intern), Schmelter, D. (Ekstern)
Pages: 7-12
Publication date: 2008

Host publication information
Title of host publication: Aspect-Oriented Modeling : 12th International Workshop on
Publisher: ACM
Main Research Area: Technical/natural sciences
Conference: Aspect-Oriented Modeling, 01/01/2008
aspect-oriented modelling, domain modelling
Source: orbit
Source-ID: 220248
Diff-based model synchronization in an industrial MDD process

Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting

Requirements and Evaluation of tool papers for PETRI NETS
TGGs for Transforming UML to CSP: Contribution to the AGTIVE 2007 Graph Transformation Tools Contest

Along with the AGTIVE 2007 conference, a Graph Transformation Tools Contest invited tool implementers to present their solutions in order to compare the principles and particular strengths and weaknesses of today's graph transformation tools. This paper documents our contribution to the Tools Contest. The second transformation problem, a transformation from UML activity diagrams to CSP processes, i.e. a transformation between two models, is a typical application for Triple Graph Grammars (TGGs). We present our contributed solution, presenting the TGG rules and the implementation of our TGG interpreter. Moreover, we point out the advantages of our solution as well as some restrictions of the current implementation. This paper will only briefly state the transformation problem and focus on our TGG approach and the discussion of the rules.

AMFIBIA: A Meta-Model for the Integration of Business Process Modelling Aspects

AMFIBIA is a meta-model that formalises the essential aspects and concepts of business processes. Though AMFIBIA is not the first approach to formalising the aspects and concepts of business processes, it is more ambitious in the following respects: Firstly, it is independent from particular modelling formalisms of business processes and it is designed in such a way that any formalism for modelling some aspect of a business process can be plugged into AMFIBIA. Therefore, AMFIBIA is formalism-independent. Secondly, it is not biased toward any aspect of business processes; the different aspects can be considered and modelled independently of each other. Moreover, AMFIBIA is not restricted to a fixed set of aspects; new aspects of business processes can be easily integrated. Thirdly, AMFIBIA does not only name and relate the concepts of business process modelling, as it is typically done in ontologies or architectures for business process modelling. Rather, AMFIBIA also captures the interaction among the different aspects and concepts and therefore fully defines the dynamic behaviour of a business process model, with its different aspects modelled in different notations. To prove this claim, we implemented a prototype of a formalism-independent workflow engine based on AMFIBIA. This workflow engine, also called AMFIBIA, is open for new aspects of business processes and new formalisms can be easily integrated. In this paper, we present the concepts of AMFIBIA and discuss the principles and concepts of its design.
Model checking: Eine Methode zur Verbesserung der Softwarequalität

Model checking is a technique for automatically checking the correctness of systems. This report gives an overview on the state-of-the art of model checking, and the pros and cons of the different approaches for avoiding or reducing the so-called state explosion problem.
Modular PNML revisited: Some ideas for strict typing

The Petri Net Markup Language (PNML) is currently standardised by ISO/IEC JTC1/SC7 WG 19 as Part 2 of ISO/IEC 15909. But, there is not yet a mechanism for structuring large Petri nets and for constructing Petri nets from modules. To this end, modular PNML has been proposed some time ago. But, modular PNML has some problems. These problems along with ideas for their solution will be discussed in this paper. As a first step toward standardising a module concept for PNML in Part 3 of ISO/IEC 15909, this paper proposes a refined concept of modular PNML, which is independent of a particular kind of Petri net, but still has a strict type system. This paper focuses on the ideas and concepts; the technical details still need to be worked out. To this end, this paper also raises some issues and questions that need to be discussed before standardising modular PNML.

Reconciling TGGs with QVT

The Model Driven Architecture (MDA) is an approach to develop software based on different models. There are separate models for the business logic and for platform specific details. Moreover, code can be generated automatically from these models. This makes transformations a core technology for MDA. QVT (Query/View/Transformation) is the transformation technology recently proposed for this purpose by the OMG. TGGs (Triple Graph Grammars) are another transformation technology proposed in the mid-nineties, used for example in the FUJABA CASE tool. In contrast to many other transformation technologies, both QVT and TGGs declaratively define the relation between two models. With this relation definition, a transformation engine can execute a transformation in both directions and, based on the same definition, can also propagate changes from one model to the other. In this paper, we compare the concepts of QVT and TGGs. It turns out that TGGs and QVT have many concepts in common. In fact, fundamental parts of QVT-Core can be implemented by a TGG transformation engine. Moreover, we discuss how both technologies could profit from each other.
Projects:

Delivering the Next Generation of Model Transformation Languages and Tools

Department of Applied Mathematics and Computer Science
Period: 15/01/2013 → 22/06/2016
Number of participants: 5
Phd Student:
Acretoaie, Vlad (Intern)
Main Supervisor:
Störrle, Harald (Intern)
Examiner:
Kindler, Ekkart (Intern)
Chaudron, Michel R. V. (Ekstern)
Taentzer, Gabriele (Ekstern)

Financing sources
Source: Internal funding (public)
Name of research programme: Institut stipendie (DTU)

Relations
Publications:
Model Manipulation for End-User Modelers
Project: PhD

Domain Specific Languages for Waste Management Modelling

Department of Applied Mathematics and Computer Science
Period: 15/12/2012 → 12/12/2016
Number of participants: 5
Phd Student:
Zarrin, Bahram (Intern)
Main Supervisor:
Baumeister, Hubert (Intern)
Examiner:
Kindler, Ekkart (Intern)
Mosses, Peter David (Ekstern)
Wirsing, Martin (Ekstern)

**Financing sources**
Source: Internal funding (public)
Name of research programme: Institut stipendie (DTU) Samf.
Project: PhD

**Behavioural Modelling and Model-based Development of Protocol Software**
Department of Applied Mathematics and Computer Science
Period: 01/03/2011 → 21/11/2014
Number of participants: 6
Phd Student:
Simonsen, Kent Inge (Intern)
Supervisor:
Kristensen, Lars Michael (Ekstern)
Main Supervisor:
Kindler, Ekkart (Intern)
Examiner:
Baumeister, Hubert (Intern)
Daniel, Moldt (Ekstern)
Meling, Hein (Ekstern)

**Financing sources**
Source: Internal funding (public)
Name of research programme: Stipendie fra udlandet
Project: PhD

**Specifying and verifying medical robotics software to ensure harmless operation**
Department of Informatics and Mathematical Modeling
Period: 01/03/2010 → 24/06/2014
Number of participants: 6
Phd Student:
Herbert, Luke Thomas (Intern)
Supervisor:
Hansen, Michael Reichhardt (Intern)
Main Supervisor:
Sharp, Robin (Intern)
Examiner:
Kindler, Ekkart (Intern)
Fränzle, Martin (Intern)
Weske, Mathias (Ekstern)

**Financing sources**
Source: Internal funding (public)
Name of research programme: Institut stipendie (DTU) Samf.
Project: PhD

**Integrating Design Decision Management with Model-based Software Development**
Department of Informatics and Mathematical Modeling
Period: 01/02/2008 → 01/06/2011
Number of participants: 6
Phd Student:
Könemann, Patrick (Intern)
Supervisor:
Baumeister, Hubert (Intern)
Main Supervisor:
Kindler, Ekkart (Intern)
Examiner:
Störrle, Harald (Intern)
Babar, Muhammad Ali (Ekstern)
Paige, Richard F. (Ekstern)

Financing sources
Source: Internal funding (public)
Name of research programme: Institut stipendie (DTU)
Project: PhD