BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

DYNAMIC SOFTWARE ARCHITECTURES FOR DISTRIBUTED EMBEDDED CONTROL SYSTEMS

DYNAMICKY REKONFIGUROVATELNÉ SOFTWAROVÉ ARCHITEKTURY PRO DISTRIBUOVANÉ ŘÍDÍCÍ SYSTÉMY

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR **Ing. TOMÁŠ RICHTA**
AUTOR PRÁCE

SUPERVISOR **Doc. Ing. VLADIMÍR JANOUŠEK, Ph.D.**
ŠKOLITEL

BRNO 2020

Abstract

This thesis deals with dynamic reconfigurability of distributed control systems. Due to the characteristics of these systems, the Petri nets formalism is used to define their functionality. These are transformed into a interpretable form and then executed by specialized software installed on each system node. Thanks to the properties of used formalism, it is possible to replace the individual parts of the system with new variants. Similarly, it is possible to generate formal specifications for the system's parts from more abstract workflow models and descriptions in the form of domain specific languages.

Abstrakt

Tato práce se zabývá dynamickou rekonfigurovatelností distribuovaných řídících systémů. Vzhledem k charakteristice těchto systémů je pro definici jejich běhu použit formalismus Petriho sítí. Tyto jsou transformovány do proveditelné podoby a následně pak interpretovány specializovaným software nainstalovaným na jednotlivých uzlech systému. Díky vlastnostem použitého formalismu je možné jednotlivé části systému nahrazovat novými variantami. Stejně tak je možné generovat formální specifikace dílčích částí systému z abstraktnějších workflow modelů a popisů ve formě doménově specifických jazyků.

Keywords

Software architectures, distributed systems, control systems, dynamic reconfigurability, formal specifications, model-driven development, model continuity, model execution, model transformation, model migration.

Klíčová slova

softwarové architektury, distribuované systémy, řídící systémy, dynamická rekonfigurovatelnost, formální specifikace, modelem řízený vývoj, kontinuita modelů, vykonávání modelů, transformace modelů, migrace modelů.

Reference

RICHTA, Tomáš. *Dynamic Software Architectures for Distributed Embedded Control Systems*. Brno, 2020. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Doc. Ing. Vladimír Janoušek, Ph.D.

Dynamic Software Architectures for Distributed Embedded Control Systems

Declaration

I declare that I have prepared this dissertation thesis independently, under the supervision of Doc. Ing. Vladimíra Janouška, PhD. Radek Kočí, Karel Richta, Fernando Macías, and Adrian Rutle also provided me with further information. I listed all of the literary sources and publications that I have used.

.....
Tomáš Richta
31.8.2020

Acknowledgements

First of all I would like to thank to all the people who helped me while working on this thesis. This work has also been partially supported by the IT4IXS - IT4Innovations Excellence in Science project (LQ1602). This work has also been partially supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), by BUT FIT grant FIT-11-1, and by the Ministry of Education, Youth and Sports under the contract MSM 0021630528 and partially also by the Norwegian Funds under the academic staff mobility programme (NF-CZ07-INP-5-337-2016)

Contents

Chapter 1

Introduction

With increasing number of interconnected embedded devices, sometimes called the Internet of Things (IoT) or according to a higher level of granularity Distributed Embedded Control Systems (DECS), a demand for software architectures reflecting a heterogeneous characteristics of used devices and environments that dynamically changes according to user's requirements, become more and more important research priority in recent years. This work is going to summarize the approaches to solve the problem of software development, deployment and updating in such a heterogeneous environment, as well as to bring the original solution to this area.

Embedded control systems are important border technology between the physical and information world. The control process itself is described as a control loop that consists of reading data from sensors, updating the decision function, and triggering a number of actuators installed within the physical environment controlled by the system. Most of the control systems are constructed using a set of programmable logic controllers with appropriate software installation. The main purpose of this work is to describe the software part of this construction process with the focus on dynamic reconfigurability of the resulting system using executable models and model continuity approach introducing the formal aspects of software construction into the embedded devices area.

Basic principles of system reconfigurability in this work were adopted form the Reference Petri Nets (RPN) formalism and framework called Renew, where parts of the system specification migrate in the form of tokens. RPNs is a specific type of Coloured Petri Nets (CPN) based on nets-within-nets formalism, where tokens realizing a marking within one network represent other RPN network with arbitrarily deep nesting of nets [39] [41].

This idea makes it possible to construct a system specification from smaller pieces of computation, similarly as it is possible within Hierarchical Petri Nets (HPN) but with dynamic way of nesting and migrating of nets within each other [36]. This is sometimes called code migration and it is used in this work for the distribution of pieces of computation within the system [26], [52], [7]. The problem of code migration would be discussed more in Related Work section.

To be able to change the target system dynamically, according to all changes within its formal specification, the specification itself is not used here for code generation and its further compilation, but rather for its interpretation by the specific target platform forming the heart of the idea prototype implementation. While we deal with embedded devices i.e. with devices with limited resources, the implementation is based on minimalistic interpretable form of the description representing migrating parts of original formal system specification.

As there are plenty of reasons to make it possible to reduce the complexity of the definition of any system, we decided to leverage Workflow system specification approaches to make it possible to define the system in more abstract way. Workflow specifications are then translated into the target system interpretable specifications.

To enable users of the system with the possibility to define its structure and functionality, as well as its changes, we also developed a Domain Specific Language (DSL) that is used as another abstract view of the system specification [72], [20], [85].

The specification defined by the DSL is also translated into the set of RPNs which are first of all used for the system simulation in the Renew simulator workbench, after that it is intended to be used for the translation into the interpretable form, which we call Petri Nets Byte-code (PNBC). PNBC is then distributed among target system nodes according to the system infrastructure specification that is also available in the form of RPNs model.

The PNBC is directly interpreted by a specific virtual machine called Petri Nets Virtual Machine (PNVM) that is responsible for maintaining and running all the pieces of computation deployed within each node. PNBC and PNVM together with the I/O interfaces of the node form so called Petri Nets Operating System (PNOS). All the communication among PNOS nodes is performed by sending simple textual messages via serial lines, or Message Queuing (MQ) tooled distribution bus.

In next section the state of the art of development software for IoT and DECS will be discussed.

1.1 State of the Art

A control system implementation could be divided into the hardware and software part. The hardware part starts with selection of the proper set of modules and its installation within the physical environment, including the sensors and actuators attachment. When there are multiple controllers, the hardware part must also take into account the communication problem. The software part follows with the programming, compilation, linking and installation of each control unit with appropriate part of application or software that controls the hardware.

The system reconfigurability in general is necessary for the ability of the system to adapt itself to changes in environment and also to enable the system maintainer with the possibility to change the system behavior without the necessity of its complete destruction and reconstruction. The main goal of this thesis is to describe the software part of the process, that respects the focus on formal specifications and dynamic reconfigurability.

Because of the strong demand on proper coverage of the system complexity at the beginning of the construction process, there is a need for suitable description tools that preserve the user requirements semantics. During the system lifetime there is also strong demand on its dynamic reconfiguration according to any new requirements and also according to the changes within the physical environment. The dynamic system specification change and following reconfiguration requirements are not easy to satisfy.

1.2 Thesis Motivation

According to described situation, the main motivation of the thesis had arisen to be a well defined way of distributed control system specification and implementation, using formal methods, model continuity, as well as executable models paradigms. The formal

specification of the system gives the model a possibility of formal analysis and thus reduce the errors at the earliest phase of the system construction possible. One of the main goals targeted by the thesis is also to allow the system reconfiguration within its run-time. The solution should also make it possible to the end-user to change the system without deep familiarity with sophisticated information technologies using some intuitive modeling tools. The solution should be also robust enough and easily scalable to more application scenarios within different levels of granularity of constructed systems.

1.3 Thesis Goals

The main goals of this thesis were defined as follows.

- Develop formally defined executable model for running the system specification - reflecting the distributed, concurrent and synchronized features of the system, and be able to run on devices with very limited resources.
- Use component-based architecture and enable for the execution of each system component independently as well as for the possibility of modifying components within the system run-time.
- Define the system construction process taking into account the possibility to involve domain experts to understand its specification and therefore directly participate on the construction process.
- Construct the system using actor properties of every part of the system functionality and with the possibility of its migration across the running model.

This thesis emerged generally as an report from ongoing research and experiments within the area of dynamically reconfigurable distributed control systems of the author. All the described approaches and methods had undergo a certain level of improvements and changes during the thesis collection lifetime. As the evolution itself plays the role in forming the thesis ideas, these changes and improvements are commented and described within the text.

1.4 Used Methods

This work was based on the following procedure.

1. Analysis of recent and historical approaches to dynamic software modification,
2. narrowing the research focus towards distributed embedded control systems and devices with limited resources,
3. a survey on formalization of the dynamic software updating and compilation of well-structured summary of used methods,
4. designing and implementation of author's prototypical and unique solution of defined problem,
5. identification and definition of different application areas and simulating these according to discussions with experts from selected areas,

6. constructing the experimental installation and preparing several running examples,

7. discuss the solution usage consequences and positive side-effects of defined solution and identification of other possible usages and applications as well as extensions.

1.5 Thesis Structure

The chapter Related Work describes the relevant work of other authors within the fields of dynamic reconfigurability of embedded software as well as the other related areas. The chapter Theoretical Foundations covers the formal apparatus used within the work. The chapter Design Of The Solution describes the characteristics of proposed solution and the details of its construction. The chapter Implementation Details adds some more information about the experimental implementation of the solution. The chapter Applications and Scenarios defines some scenarios of the real-world problems that were experimentally solved using proposed solution. The chapter Experiments and Results shows achieved results from running the experimental system implementations. Finally the chapter Conclusion and Future Work summarizes achieved results and proposes possible future steps within defined research.

Chapter 2

Related Work

The development and deployment of safe and reliable software for embedded control systems remains the actual challenge to the computer scientists. The most important part of the system development process is testing and verification of the system before its final deployment. Also very important remains the possibility of the system to flexible reflect the changes in requirements after the software deployment. For that it is necessary to enable incremental changes to the running system and thus modify its behavior. At the same time we need to maintain the model of the system throughout the whole system development process, to keep the testing and verification possible.

Related work focused on similar problems as this theses could be divided into following areas - embedded and operating systems, software engineering methods applied to the area of embedded systems, Model-Driven Software Engineering (MDSE) methods applied to the area of embedded systems, the usage of higher-level or visual languages for embedded systems specification and implementation, the dynamic reconfigurability within embedded systems, multi-agent approach to the reconfigurable embedded systems development, system partitioning, code generation, and also the reconfigurable hardware.

2.1 Distributed Embedded Control Systems

Distributed embedded control systems (DECSs) consist of a set of nodes that either provide for some functionality to the system or ensures the control over some particular device to which they are connected. The functionality-providing node could for example offer the storage service for the devices without persistent memory, or some more complicated computations (like encrypting/decryption) for the nodes with low computational power. Some of the nodes nodes are attached to the I/O of the device, like sensor, motor, pump, valve, boiler, or switch, providing the signals for the device controlling or reading the data from sensors.

Therefore the overall business logic of the system is spread among the nodes and manifest itself within the controlled environment by achieving the goals of the system, like living comfort for the house inhabitants, energy consumption optimization, or power plant energy production. The functionality itself is defined as a functionality of every node of the system together with the communication among nodes.

There are several communication buses' standards within e.g. home automation industry for the inter-devices communication, e.g. KNX, OPC, BACNet, etc. [31]. These communication buses are suitable to satisfy the reliability and security of the communica-

tion between the nodes themselves. But there is a huge gap regarding any standards for used nodes and control units software equipment. Simple nodes are only reflecting some basic commands received via buses, but there is a lack of computational facilities within most of them. The control unit then remains the only responsible entity within the system.

The complexity of embedded systems has increased in a way, that this area obviously has to undergo similar transformation process as classical software systems passed after the software crisis [16], that lead to an emergence of software engineering disciplines and object-oriented programming languages. Some literature even mentions the complexity as an essential characteristics of modern computing platforms for embedded systems. It is assumed that this complexity being underestimated could lead to the fact that these systems will increasingly become unreliable - with increasing complexity, system reliability and safety becomes a major problem.

The complexity of embedded systems also lead to the component-based system construction, which needs the techniques to integrate components while preserving essential properties of system behaviour [11]. The introduction of appropriate levels of abstraction in modeling and the associated concept formation helped to reduce the emerging complexity by focusing on the relevant properties and omitting irrelevant detail, thus leading to a simpler representation of the evolving artefacts [48].

While earlier embedded systems were usually isolated pieces of software, typical today's embedded system software takes about gigabytes of binary code operating over dozens of devices and these numbers will probably even arise. This type of systems could be found in houses, cars, ships, plants, and many other complex devices. The aim of reducing the complexity of distributed embedded system construction, as well as the necessity to satisfy the predictability, correctness and reliability of such a system caused the focus of the model-driven software construction research towards the embedded systems [91].

2.2 System Dynamic Reconfigurability

Dynamic reconfigurability is necessary for the system ability to adapt itself to changes in environment and also to provide the user with the possibility of changing the system behavior while it is in run time, without the necessity of complete destruction and further reconstruction or even its restart. One of the focuses of this thesis is to describe the software part of construction process and the system maintenance features, that respect a focus on formal specification and dynamic reconfigurability. The main operation principle of resulting system could be described on tasks of system construction - installation, and its reconfiguration. The installation of the system starts with placing proper nodes to the target environment. The physical communication between nodes using different wired or wireless communication technologies should be established. Each node should be installed with proper software, enabling the installation and reconfiguration of the system. The system reconfiguration should be performed on each defined level of the system architecture. All the parts of the system could be changed and then passed to the particular system node to change the behavior of the system.

The usage of formal modeling within the control systems development as well as the dynamic reconfigurability features of such a software is not a new idea. Research activities in this topic are primarily focused on two possible ways - the direct or indirect approach. The direct approach offers specific functions or rules, allowing to modify system structure, whereas the indirect approach introduces mechanisms allowing to describe system reconfiguration. The main difference consists usually on the level of reconfigurability implemented.

Direct methods use formalisms containing intrinsic features allowing to reconfigure the system. Indirect methods use specific kind of frameworks or architectures, that make it possible to change the system structure.

In our field of research the first group consist of formalisms based usually on some kind of Petri Nets. Reconfigurable Petri Nets [33], presented by Guan and Lim, introduced a special place describing the reconfiguration behavior. Net Rewriting System [59] extends the basic model of Petri Nets and offers a mechanism of dynamic changes description. This work has been improved [57] by the possibility to implement net blocks according to their interfaces. Intelligent Token Petri Nets [93] introduces tokens representing jobs. Each job reflects knowledge about the system states and changes, so that the dynamic change could be easily modeled. All the presented formalisms is able to describe the system reconfiguration behavior, nevertheless only some of them define the modularity. Moreover, the study [5] shows, that the level of reconfigurability is dependent on the level of modularity and also that there are modular structures that are not reconfigurable. Another approach introduced by Kahloul et. al. uses classical P/T Nets and specific production rules and graph transformation techniques to modify manufacturing process defined using Petri Nets, i.e. to modify the manufacturing system that controls it [56].

The second group handles reconfiguration using extra mechanisms. Model-based control design method, presented by Ohashi and Shin [68], uses state transition diagrams and general graph representations. Discrete-event controller based on finite automata has been presented by Liu and Darabi [58]. For reconfiguration, this method uses mega-controller, a mechanism, which responses to external events. Real-time reconfigurable supervised control architecture has been presented by Dumitrache [17], allowing to evaluate and improve the control architecture. All the presented methods are based on an external mechanism allowing system reconfiguration. Nevertheless, most of them do not deal with validity and do not present a compact method.

So far, we have investigated formalisms and approaches to the control system development. They have one common property, they are missing complex design and development methods analogous to software engineering concepts. Of course, the methods and tools that are applied in ordinary software systems are not as simply applicable to embedded systems. Nevertheless, we can be inspired with software engineering approaches and adopt them to the embedded control systems [69]. To develop embedded control system, the developer has to consider several areas. We can distinguish five areas [69] as follows—*Hardware*, *Processes* (development processes and techniques), *Platform* (drivers, hardware abstraction, operating systems), *Middleware* (application frameworks, protocols, message passing), and *Application* (user interface, architecture, design patterns, reusing).

Former MDSE approach of embedded systems construction was typically based on metamodeling and model transformations using code generators [91]. These approaches enable the reconfigurability during compile time. But there are also approaches that use higher-level interpreted languages, like SensorScheme. The interpreted characteristics of higher-level languages enables not only for higher abstraction of concepts, but also for platform independence and dynamic features of languages, like dynamic loading and execution of code while the system is in run time. Similar approach we use in this work.

The dynamic reconfigurability of the system could be also provided by the agent and multiagent architecture as a basic system construction framework. But this way the system functionality changes according to the agents characteristics and therefore partially unpredictable. Therefore we focused more on the dynamic change of the system by its user and according to his or her requirements.

The dynamic reconfigurability of system could be also achieved by the proper usage of constants and data persistence means, like sharing the state of the application within the database. Today's software tools already offer Database Management Systems (DBMS) suitable for embedded systems, but dynamically reconfiguring the node behavior would also must take into account storing some form of code within the database to be interpreted later by some virtual machine, which is very similar to our concept. Changing only some coefficients used within the computations is not considered as an dynamic change of the node functionality in our approach.

2.3 Systems Modeling

Systems modeling is a discipline covering all the necessary knowledge and practices for creating artificial conceptual models of real-world systems. Typical approach is to divide between modeling the structure of the system and its dynamic behavior. Both together the structure and its dynamic should represent the definition of system functionality. There are usually some modeling means to decompose the functionality as well to partition the system structure into easily manipulable pieces. All the functionality blocks forming some abstract concept are usually mapped to some functional requirement defined by the future user of the system, or customer.

There are plenty of notations, and formalisms for systems modeling, but they differ in the level of exactness and therefore some sort of straightforwardness of inducing the final implementation from the model itself. Among mainly used notations in the industry there is a Unified Modeling Language (UML) and Business Process Model and Notation (BPMN) notations worth of mentioning. UML aims to be strong enough to model the structure of the system, as well as its behavior. The BPMN is more focused on the business level behavior of the system. Both define enough tools for system description making it possible for systems analysts and designers to define and discuss the implementation with computer programmers. On the other hand, such a non-formal notations will always leave some space for uncertainty of the final solution functionality accurateness. Compared to that there are some formal approaches that enable for well-defined system implementation results. Among these, there are e.g. agent based modeling, data modeling and mathematical modeling.

The IEEE recommendation defines the system as its aspects and the environment [32]. It mainly focuses on the common way to talk about system structure and behavior. The most important concept in system modeling is abstraction that enables for the simplification of complicated problems as well for wrapping some unnecessary details into more abstract concepts. Not even the structure of the system has its own hierarchy of abstractions. As well the behavioral complexity of the system with e.g. non-deterministic behavior, and other difficult-to-characterize properties are necessary to cover.

Two key concepts play a role when modeling different levels of abstraction, those are: view and viewpoint and black-box and white-box modeling, which will be described below [1]. Next chapter will briefly describe the component-based software engineering that brings important key concepts into the system decomposition problematic.

2.4 Model-based Systems Engineering and Formal Specifications

Within the Model-Based Engineering (MBE), or Model-Driven Engineering (MDE) and Model-Driven Development (MDD) domains, the emphasis lays in using the visual modeling tools to leverage the benefits of common and easily understood concepts in the same way as in system description during the System Development Life Cycle (SDLC). The important part of the definition of MBE is that The Model-Based Engineering paradigm is model-based to the extent that the visual modeling artifacts that it generates are sufficiently precise and complete that they can serve as a software or systems blueprint for improving SDLC efficiency and productivity. The paradigm is considered to be model-driven to the extent that it at least partially automates (i.e., „drives“) the SDLC via requirements that are precisely and completely specified as part of the system model, and which can be fully traced across the SDLC [62].

2.5 Domain Specific Modeling

When dealing with implementation of complex systems covering some specific area of human activity, it usually come down to the problem domain that is not understandable to everybody, but domain specialists educated and experienced in certain problematic. The system specification could then boil down or just be partially solved by so called Domain-specific modeling (DSM), i.e. programming using Domain-specific languages (DSL), which is a computer language specialized to a particular application domain, in contrast to any general-purpose language (GPL), that could be used for any problem. The important thing here is to divide the problem solution between the software tool that is able to somehow interpret or consume the DSL and behave according to what is defined in its statements, or whatever structures they use. The typical approach is to support higher-level abstractions than typical general modelling languages tobe able to specific the problem domain in terms and costructs it contains [43].

Typically within the domain specific modeling there is a code generation inherently included. It is because the Domain-specific languages (DSLs) are typically not executable nor interpretable itself. While it is quite challenge to interpret the DLS directly, it is much more simple to translate it to some classical language that is compilable or interpretable rather straightforwardly. The main benefit of this approach is the possibility to involve the domain expert into the development process as well as the reducing of possible problems and bugs created by programmer when implementing the specified system description, thus directly improving the quality of the software and code.

Chapter 3

Theoretical Foundations

This chapter collects all the necessary theoretical background for the reader to be able to understand later parts of the thesis. It mainly focuses on Petri Nets as a modeling formalism. Several types of Petri Nets and their features are discussed. First of all the classical basic Petri Nets and their features, together with the high-level Petri Nets specification making it possible to add types to tokens and places. Then a specific type of High-Level Petri Nets called Reference Nets, or Elementary Object System is defined. And also an other type of Petri Nets definition, based on the aggregation of particular places and transitions patterns into a new nets components. These components form e.g. logical operations used within workflow modeling. Therefore are these called Workflow Nets.

3.1 Petri Nets

Petri Nets, called by Carl Adam Petri is specific mathematics language for modeling discrete-event systems, particularly suitable for describing distributed parallel systems. A Petri net could be displayed as directed bipartite graph of two types of nodes - places and transitions. Places are typically displayed as circles and transitions as bars. Places represent distributed state of the model and could carry so called tokens. Transitions represent events within the system and could be invoked arbitrarily when all the preconditions of the transition are satisfied. Preconditions are defined by places connected to the transition by oriented arcs. The precondition is satisfied when there is so called token within the place. The transition invocation produces tokens on all the post-conditions places, connected with the transition by oriented arcs. Every result of the transition invocation forming the distributed state represented by tokens placed on places is called marking. Using the combination of places and transitions it is possible to describe the execution flow of system components or parts. Compared to the other graphical tools for describing the the execution flow like e.g. UML, BPMN, etc., Petri Nets have an exact meaning and mathematically defined execution semantics, also with matured mathematical theory for the execution flow analysis. One of the many specifics of Petri Nets is the inherent nondeterministic transitions execution policy. Any executable transition could be fired on each step.

3.2 Reference Nets

A specific type of High-level Petri Nets are called Reference Nets. In this thesis this type of HLPN is used as a basic formalism for system model specification. Reference Nets allow to construct a system hierarchically, in several levels. Nets can migrate among places in other nets and thus it is possible to dynamically modify functionality of system components, specified by this kind of nets [12]. The formalism is based on nets-within-nets concept introduced by Valk [88]. This concept enable Petri Nets to be nested in other Petri Nets in the form of tokens.

3.3 Workflow Nets

Workflow modeling is very popular for its aim to precisely define the functionality requirements using intuitive and human-readable form, while offering enough precision to be interpretable by machines. For its formal characteristics and large research background we adopted for the purposes of our research Wil van der Aalst's specification for system workflow modelling, so called Extended Workflow Petri Nets [2]. Aalst's work is well-defined and resulting workflow models could be used for the system processes verification and validation purposes [4]. The main advantage of using Workflow Petri Nets is the possibility of system specification and its adaptation by the non-technically educated domain specialists. This approach is very similar to the BPMN workflow models, so it might be easily adopted by business process modeling domain experts. For that reason we decided to use the Aalst's YAWL notation [3] and Workflow Petri Nets formalism [2] in the early beginning of the system construction process. There are two main concepts from this theory that we use at the moment - two basic transition categories - *split* and *join* behavior (AND-split, AND-join, OR-split and OR-join), and the concept of workflow subprocess (sub-task).

Chapter 4

Design of the Solution

This chapter describes the design of the dynamically reconfigurable embedded system construction process. It covers the decomposition of the problem into specific parts and their further transformations and interactions.

4.1 The Development Process

The development process is described in Figure 4.1. It starts with the system specification using Reference Nets framework Renew [51] which is then followed by the transformation of the models into the interpretable form. It is also possible to generate native code, and deploy it directly to the chip, but this approach dramatically reduces the level of reconfigurability. Statistics gathered from simulation experiments can be used for verification purposes and also can support decisions about type of hardware for target system implementation. The hardware components for the target implementation are installed with the specific interpreter implemented to be able to run translated nets. Finally the whole system is installed according to its model by sending appropriate nets definitions and instructions to all subsystems. All the parts of the system could by reinstalled later within the system run time. This is how the deployment and maintenance of the system is achieved.

The multi-layered nature of the system and responsibilities of particular levels are described in Figure 4.3.

4.1.1 Communication Model

The communication within described model is intended to be based on textual messages constructed according to the defined rules and the structure of the system. We call this language **protocol** and it consists of commands and addresses, according to the structure and capabilities of each of involved nodes. Regardless on the way the code is generated all the abstraction levels communicate with each other using described uplinks and downlinks. The communication principles are described in Figure 4.3.

The communication is basically initiated by sub-processes installed within processes. According to the instructions within the protocol, processes send messages to other processes. Processes could also communicate with platform in which they are installed to install other processes, or receive and send data. They could also communicate with PNOS on which the platform is installed to install other platform. The communication between nodes is accomplished by packets that are in upper layers interpreted as mentioned textual messages.

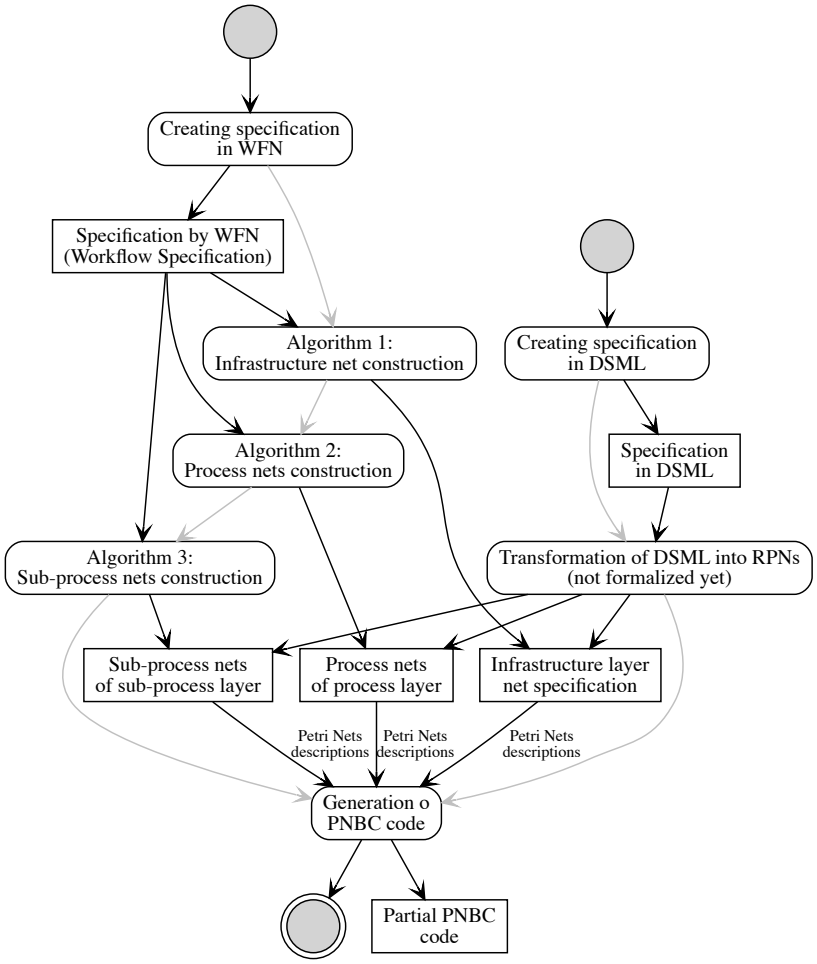

Figure 4.1: System development process

4.2 System Model Definitions

In this section all the necessary extensions to previously defined theoretical foundations are defined. We have here extended communicating workflow net, to be able to communicate among nets. Then there is workflow specification definition added which enables for nesting workflow nets together. And finally there is workflow system specification that makes it possible to combine multiple workflow specifications. Besides the model interpretation problem there is a model construction part, which heavily relies on abstract model transformations. In this work, there are two translation phases. The translation of the Workflow Petri Nets model into the Reference Petri Nets model and translation of the Reference Petri Nets model into its interpretable form. The first transformation phase takes into account

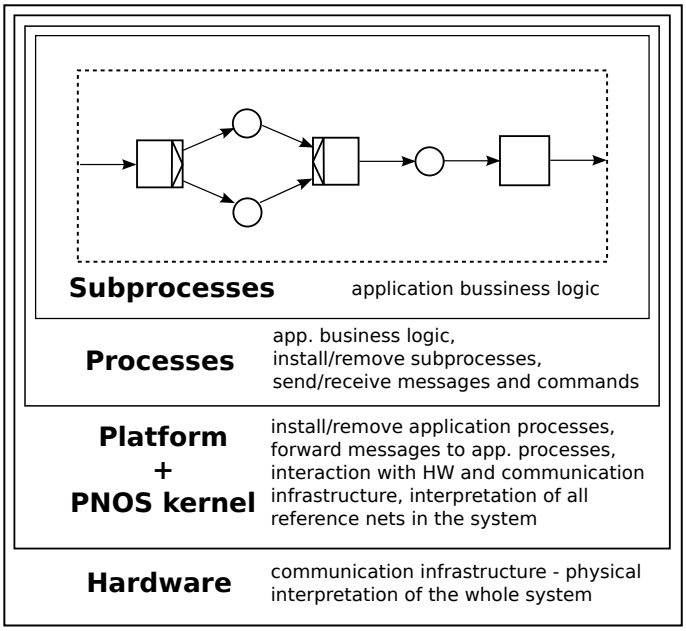

Figure 4.2: System layers and their responsibility

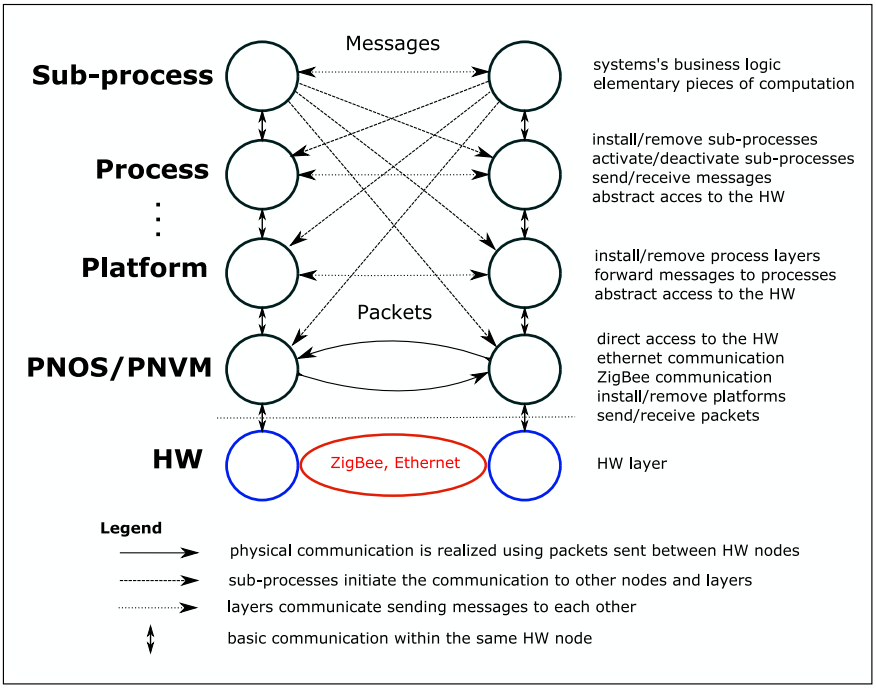

Figure 4.3: Communication schema

the set of workflow specifications described within the workflow model of the system and produces target node representations. Such a representation should contain the basic PNOS I/O functionality, and the platform functionality, which means the ability of receiving nets specifications, nets instantiation, removing nets instances, removing nets specifications, etc.

Using this functionality the node main processes should be installed. It usually consists of the description of sub-processes interactions and ordering. Then the main processes

of each node are installed with translated sub-processes. The communication between resources is represented by transitions, which are not part of any other role and serve as a data transport part of the system. Particular data types should be described in the terms dictionary, that holds all the necessary information needed for nets translation, that is not included within the diagram. Regarding the workflow model, also other specific rules for the communication protocol could be derived. Let us introduce some basic definitions of formalisms used during the system development. Our approach follows the previous definitions and adds some more rules to enable the extended workflow models with communication features to satisfy the developer ability to combine multiple workflow specifications.

4.3 Domain Specific Languages

The increasing complexity of software systems requires emerging methodologies and techniques for software engineering. In previous section the workflow modeling and formally defined approach to the interpretable system description has been shown. This section introduces more practically oriented view of the problem, forming part of this thesis introducing another method of the system specification. This method is also based on model-driven software engineering (MDSE), which tackles software complexity by employing models as first-class entities in all development phases. By raising the abstraction level, many details of the implementation itself could be hidden, which has also the benefit of improving the communication between technical staff and the domain experts. Among advantages of MDSE, there are also improved productivity, re-usability and code quality, separation of concerns as well as easier to react on changes [91].

Obviously, it is hardly possible to expect software engineers to become experts in the domains which they write software for. The same is true for domain experts: they most probably will not understand program code, logic, software modeling or object-orientation. Software projects which do not succeed in involving the domain experts in the production line has a higher probability of failure. One of the methods promoted by MDSE to solve this problem is an introduction of domain specific modeling languages (DSMLs).

Basically, DSMLs are modelling languages which define the structure, semantics and constraints of models related to a particular application domain [25]. DSMLs facilitate domain experts with means to model their own systems by offering capacity for high-level abstraction, user friendliness and tailoring to the problem space. Hence as opposed to general purpose modeling languages, in DSMLs the concepts and language constructs come from the particular domain to which the system is dedicated. An important factor for the success of DSMLs is the existence of good language workbenches such as MetaEdit [43], MultEcore [60], DPF Workbench [54], USE [30], etc.

A natural application of MDSE and DSML is the specification of home automation configurations since these systems usually consist of various embedded devices with different manufacturer models, which makes their communication, configuration, reconfiguration, etc., a challenging task. Moreover, in most cases the domain experts in home automation are home owners from whom we should not be expecting technical expertise. A flexible yet extensible and user-friendly DSML in this regard would be huge gain for both installation engineers and home owners. One of our contributions in this paper is such a DSML which, through abstraction, will enable users to configure their broad range of devices without being bothered with the technicality [79].

Chapter 5

Implementation Details

This chapter describes implementation details of presented design of the solution. At first it introduces all the hardware constraints that have been defined within thesis goals in more detail. Then it goes more deeply into the code generation and interpretation of generated target system implementation.

5.1 Hardware Infrastructure

In this section we are going to briefly describe the hardware constraints defined as main focus considerations for the thesis itself. From the point of view of distributed embedded control systems, there are plenty of aspects that should be taken into account. First of all of those is the latency, i.e. the time it takes to the system reaction to some impulse. Second one is the rate of data processing, e.g. whether it is necessary to process some data each hour or there is a demand to do that at 4000 Hz frequency.

Particularly when dealing with home automation problems, it is worth of spending some more time while having more flexibility regarding the dynamic reconfigurability compared to more rigid, but fully reliable systems as car or boat driving software. For example for gathering the high-frequency data the typical HW equipment on an Anchor Handling Tug Supply vessel (AHTS) with Dynamic Positioning (DP) features is an industrial PC with Intel Atom N270 fanless configurable controller with 2 PCI slots and 2GB memory from ADLINK Technology.

On the other hand, for the humidity, temperature, gases and other environmental variables on very low frequency measurement, let's say 10 Hz the ATMega chip typically installed on Arduino or Libelium devices will sufficiently do. The difference here is quite huge and it is quite easy to imagine any device half way through this spectrum. In our work, we would like to cover all those device with the same approach. To be able do this we needed to cover the most weak devices at the beginning. Next section will describe particular examples of devices in more detail.

5.2 Petri Nets Operating System (PNOS)

During the development of the solution the concept of Operation System-Like environment for the Reference Petri Nets interpretation and manipulation emerged. The basic principle was coined as a Petri Nets Operating System (PNOS) which means, that this part of the system should represent the basic embedded operating system principles - provide means for

input and output of I/O data, communication tools, multiprocessing support and memory management. In following sections this concept will be described in more detail. The aspects of the PNOS functionality, including PNVM and PNBC, will be demonstrated by example.

5.3 Petri Nets Byte Code (PNBC)

Part of the system specification is described as PNBC (Petri Nets Byte Code). PNBC was developed to solve the key problem of the thesis, which is interpretation of the RPN models within devices with limited resources.

The language itself uses a set of special characters to express the beginning and end of some structure. Following text will explain the language elements as well as its grammar. The interpretation details follows.

5.4 Petri Nets Virtual Machine (PNVM)

This part of the work describes the BNBC interpreter called PNVM that is part of the PNOS and is responsible for running the RPN nets on each node of the system. Because the memory management is critical when writing the software for embedded systems, while interpreting the PNBC on devices with very limited resources, it was necessary to keep the memory management under control to the maximum level possible. Therefore the dynamic memory allocation must have been avoided and specific memory management targeted directly to the Petri Nets management was developed. This part of the work is partially based based on work of one of our students, who translated original PNVM previously implemented in Smalltalk into the C code [64]. The translation was performed using generated Smalltalk Slang sources, that are equivalent to the C language semantics, so they could be easily transferred into C program. On the other hand, this way of constructing the virtual machine was abandoned, because the post-processing of generated code appeared as non-trivial. Also the memory management on devices with limited resources is much different from the Smalltalk approach. So only the initial implementation of the PNVM was held using this transformation, but further development is conducted now directly in C language. Following section describes the virtual machine functionality in more detail.

Chapter 6

Application Scenarios

In this chapter we are going to describe all the discovered and tested application scenarios that form the example domains of projects possibly implemented based on this thesis results. At first there is a main motivational home automation scenario which solves the problem of changing weather as well as user's demands by introducing easily manageable solution. Second part describes the scenario, where there was an idea to broaden the scope of the proposed system construction mechanism, using to bigger scenarios and different application areas - particularly the maritime logistics in Norway.

6.1 Control Systems for Home Automation

In the area of home automation, there are two main approaches in the house control mechanism construction - centralized and decentralized one. Centralized approach is based on one central control unit, that is connected with all the devices and sensors in the house. Such a control unit gathers all the data from sensors and process it according to some predefined set of rules. Based on the results, the control unit produces commands for target action devices. Decentralized approach is based on some sort of bus that connects many devices behaving independently according to the data and commands sent over the bus. Both approaches employ microcontrollers as a means of computation units to control devices. These microcontrollers are usually programmed using languages as Assembly language or C. Sometimes there exists a visual tool for controller programming, that produces the compiled binary to be uploaded to the chip.

This approach causes the typical maintenance roundtrip to be divided into the planning and programming phase, and then to the installation and run time phase. If there are some bugs encountered within run time, it is necessary to change the program, recompile binaries and install it to the devices. That forms two main disadvantages in usage of such a control system: lack of autonomous and dynamical reconfigurability in changed conditions, and lack of means for system formal and simulation based validation and verification before it is finally deployed. This leads to the considerable extent of discomfort for the owner and maintainer of the control system. Our work aims to overcome these disadvantages by introducing the control system, that regardless whether centralized or decentralized is constructed based on the formally specified and verified design and is able to adapt itself to changing conditions without the necessity of maintainer direct intervention. The maintainers role is to correct the system remotely e. g. from his place of work.

As already been discussed, the idea behind our solution is to construct model of described control system as a well defined and sound formal specification and then run this model with high degree of flexibility in reconfiguring it in run time. In our approach the target system is divided into the set of specific abstraction levels and each abstraction level is then mapped to the target platform. The transformation is then defined, which can be used for code generation from particular abstraction level network. Enabling changes within the model during the system run time is achieved by the concept of a multi-level abstraction, where the functionality of the system is determined by currently present agents and protocols, that they interpret. Each modification to the system is performed in following steps: 1) modeling or remodeling of the particular artifact (net), 2) code generation, and 3) forwarding the net to the system.

6.2 Data-Driven Maritime Processes Management

In this part of the work a decision support system for maritime traffic and operations, based on formal models and driven by data from the environment will be briefly described and used as an example. To handle the complexity of such a system description, we work with a decomposition of the system to set of abstraction levels. At each level, there are specific tools for system functionality specification, respecting particular domain point of view. From the business level point of view, the system consists of processes and vehicles and facilities over those the processes are performed. From the engineering point of view, each process consists of a set of devices, that should be controlled and maintained.

Software engineering point of view operates on reading and converting bytes of data, storing them into variables, arrays, collections, databases, etc. For complex trading processes management purposes we need to cover all levels of abstraction by specific description, suitable to model and automate the operations on each particular level. As a case study we use salmon farming in Norway. The system implementation is based on *Reference Petri Nets* and interpreted by the *Petri Nets Operating System* (PNOS) engine. This approach brings formal foundations to the system definition as well as dynamic reconfigurability to its run time and operation. This example emerged as a result of authors internship on NTNU: Norwegian University of Science and Technology.

Chapter 7

Conclusion and Future Work

This work aimed to and introduced the basics of the methodology for automated conversion of formal system specifications to the executable implementation that preserves the dynamic reconfigurability of the original model, i.e. changes within its run time. Present implementation uses the Raspberry Pi and Arduino platforms as hardware platforms for target system deployment. The architecture enables to run and simulate the control system specification as a model. The same model in the form of running implementation works on top of the network of Arduino and Raspberry Pi boards. All the changes to the running system are preformed by the model modification. The modification could be done directly, or mediated by introduced higher-level abstractions - Workflow Nets, or DSMLs. In this section the main achieved results are concluded.

Within this work an analysis of recent and historical approaches to dynamic software modification, mainly focused on distributed embedded control systems and devices with limited resources, has been contributed. According to the goals of the thesis a great focus was targeted to the formalization of system specification as well as the dynamic software updating. The thesis introduced original solution to the problem of running specifications on low-level hardware, as well as to the problem of involving domain experts into the development process. The work introduces the Reference Petri Nets based approach, that enables model preservation during the whole system development life-cycle. The solution is based on so called Petri Nets Operating System (PNOS) that consists of basic I/O and communication means and also of the so called Petri Nets Virtual Machine (PNVM), that is able to interpret the original Petri Nets Byte Code (PNBC). PNBC serves as an intermediate language, that could be produced from many sources, as well as interpreted by many interprets. A prototypical solution has been also prepared. Both main targets were also experimentally applied to two domain areas - Home Automation and Maritime Logistics. Domain experts in both areas were directly interviewed and their knowledge was used to test achieved results, during the methodology development. The running example of Home Automation problem was implemented using defined methodology and experimentally runs within the Real World installation. Also some practical results were collected and presented.

Among the main methods the work uses models transformations and target system prototype code generation, model execution, and model continuity. Development process starts with the Workflow Model or Domain Specific Model of the system specification. Workflow model of the system describes the functionality from user's or domain specialist's point of view. Using defined methods, the Workflow Model or Domain Specific Model are further transformed to the multi-layered architecture based set of Reference Petri Nets.

The system is constructed in several layers. Each layer of the system is translated to the specific target representation called PNBC, which is interpreted by the PNVM, that is a part of the PNOS, that is installed on all nodes of the system. Targeted dynamical system reconfigurability is achieved by the possibility of PNBC net templates and instances replacement with their new versions. After the replacement, PNVM interpretation engine starts to perform a new version of partial functionality of the system. That makes the dynamic reconfigurability possible.

The work also describes the process of construction of basic elements of Domain Specific Language (DSL) for domotic systems configuration and reconfiguration called DexML. The idea here was to impart some formally well defined concepts to the informal DSL definition by its translation to formally well established form. This additional goal was achieved only partially, because the transformation is still defined in non-formal way, therefore it is not possible to ensure that the resulting system reflects the source DSL model. On the other hand, our up to date architecture and a set of tools enable the end users of simple IoT systems to define their structure and behavior using readable DSL and then transform it into the runnable target system implementation, leveraging the PNOS architecture defined by our research previously. Because of the possibility to simulate the generated model, or goal is at least partially fulfilled. The other advantage is leveraging the dynamic reconfigurability features of the PNOS, enabling the user with the possibility to change the DSL model and then generate modified set of Petri Nets that could be sent to the target system changing its behavior while it is in run time.

This work defines the basic principles of dynamically reconfigurable distributed embedded control system construction process and mechanisms. Planned future work could be divided into following areas: implementing more PNVM versions using different languages and different platforms, finishing the DSL formalization process and generalize core DSL parts to be applicable to different scenarios, introduce run time verification features to the running specifications leveraging its formal properties, and finally use the formal properties of the system for proving its trustability.

Bibliography

[1] 1522 IEEE Trial-Use Standard for Testability and Diagnosability Characteristics and Metrics. 5 2005. ISBN 0-7381-4492-4. 35 pp.

[2] van der Aalst, W.; van Hee, K.: *Workflow Management: Models, Methods, and Systems*. Cambridge, MA, USA: MIT Press. 2004. ISBN 0262720469.

[3] van der Aalst, W.; ter Hofstede, A.: YAWL: yet another workflow language. *Information Systems*. vol. 30, no. 4. 2005: pp. 245 – 275. ISSN 0306-4379.

[4] van der Aalst, W. M. P.: Verification of workflow nets. In *Application and Theory of Petri Nets 1997*, edited by P. Azéma; G. Balbo. Berlin, Heidelberg: Springer Berlin Heidelberg. 1997. ISBN 978-3-540-69187-7. pp. 407–426.

[5] Almeida, E. E.; Luntz, J. E.; Tilbury, D. M.: Event-Condition-Action Systems for Reconfigurable Logic Control. *IEEE Trans. on Automation Science and Engineering*. vol. 4, no. 2. 2007: pp. 167–181.

[6] Alur, R.; ; Esposito, J.; et al.: Hierarchical modeling and analysis of embedded systems. *Proceedings of the IEEE*. vol. 91, no. 1. Jan 2003: pp. 11–28. ISSN 0018-9219.

[7] Asperti, A.; Busi, N.: Mobile Petri nets. *Mathematical Structures in Computer Science*. vol. 19. 2009: pp. 1265–1278.

[8] Bagnato, A.: *Handbook of Research on Embedded Systems Design*. Advances in Systems Analysis, Software Engineering, and High Performance Computing:. IGI Global. 2014. ISBN 9781466661950.

[9] Bartocci, E.; Falcone, Y. (editors): *Lectures on Runtime Verification - Introductory and Advanced Topics. Lecture Notes in Computer Science*, vol. 10457. Springer. 2018. ISBN 978-3-319-75631-8.

[10] Bierman, G.; Hicks, M.; Sewell, P.; et al.: Formalizing Dynamic Software Updating. In *Proceedings of the Second International Workshop on Unanticipated Software Evolution (Warsaw), in conjunction with ETAPS*. April 2003. 17pp.

[11] Bouyssounouse, B.; Sifakis, J.: *Embedded Systems Design: The ARTIST Roadmap for Research and Development*. Lecture Notes in Computer Science / Programming and Software Engineering. Springer. 2005. ISBN 9783540251071.

[12] Cabac, L.; Duvigneau, M.; Moldt, D.; et al.: Modeling Dynamic Architectures Using Nets-Within-Nets. In *Applications and Theory of Petri Nets 2005*, edited by

G. Ciardo; P. Darondeau. Berlin, Heidelberg: Springer Berlin Heidelberg. 2005. ISBN 978-3-540-31559-9. pp. 148–167.

[13] Capra, L.; Cazzola, W.: A Petri-Net Based Reflective Framework for the Evolution of Dynamic Systems. *Electr. Notes Theor. Comput. Sci.*. vol. 159. 2006: pp. 41–59.

[14] Di Modica, G.; Pantano, F.; Tomarchio, O.: *SNPS: An OSGi-Based Middleware for Wireless Sensor Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg. 2013. ISBN 978-3-642-45364-9. pp. 1–12.

[15] Dias, J. P.; Ferreira, H. S.: State of the Software Development Life-Cycle for the Internet-of-Things. *CoRR*. vol. abs/1811.04159. 2018. `1811.04159`.

[16] Dijkstra, E. W.: The Humble Programmer. *Commun. ACM*. vol. 15, no. 10. October 1972: pp. 859–866. ISSN 0001-0782.

[17] Dumitrache, I.; Caramihai, S.; Stanescu, A.: Intelligent agent-based control systems in manufacturing. In *Proceedings of the 2000 IEEE International Symposium on Intelligent Control. Held jointly with the 8th IEEE Mediterranean Conference on Control and Automation (Cat. No. 00CH37147)*. IEEE. 2000. pp. 369–374.

[18] Emadi, S.; Shams, F.: A Comparison of Petri Net Based Approaches Used for Specifying the Executable Model of Software Architecture. In *Proceedings of the International MultiConference of Engineers and Computer Scientists 2007, IMECS 2007, March 21-23, 2007, Hong Kong, China*, edited by S. I. Ao; O. Castillo; C. Douglas; D. D. Feng; J. Lee. Lecture Notes in Engineering and Computer Science. Newswood Limited. 2007. ISBN 978-988-98671-4-0. pp. 1104–1109.

[19] Emadi, S.; Shams, F.: A new executable model for software architecture based on Petri Net. *Indian Journal of Science and Technology*. vol. 2, no. 9. 2009: pp. 15–25.

[20] Eterovic, T.; Kaljic, E.; Donko, D.; et al.: An Internet of Things visual domain specific modeling language based on UML. In *2015 XXV International Conference on Information, Communication and Automation Technologies (ICAT)*. Oct 2015. pp. 1–5.

[21] Fabry, R. S.: How to Design a System in Which Modules Can Be Changed on the Fly. In *Proceedings of the 2Nd International Conference on Software Engineering*. ICSE '76. Los Alamitos, CA, USA: IEEE Computer Society Press. 1976. pp. 470–476.

[22] Fant, J. S.; Gomaa, H.; Pettit, R. G.: A comparison of executable model based approaches for embedded systems. In *2012 Second International Workshop on Software Engineering for Embedded Systems (SEES)*. June 2012. pp. 16–22.

[23] Flach, P. A.: *Simply Logical Intelligent Reasoning by Example*. New York, NY, USA: John Wiley & Sons, Inc.. 1994. ISBN 0471941530.

[24] Foukalas, F.; Ntarladimas, Y.; Glentis, A.; et al.: Protocol Reconfiguration Using Component-Based Design. In *Distributed Applications and Interoperable Systems*, edited by L. Kutvonen; N. Alonistioti. Berlin, Heidelberg: Springer Berlin Heidelberg. 2005. ISBN 978-3-540-31582-7. pp. 148–156.

[25] Fowler, M.: *Domain-Specific Languages*. The Addison-Wesley signature series. Addison-Wesley. 2011. ISBN 978-0-321-71294-3.

[26] Fuggetta, A.; Picco, G. P.; Vigna, G.: Understanding code mobility. *IEEE Transactions on Software Engineering*. vol. 24, no. 5. May 1998: pp. 342–361. ISSN 0098-5589.

[27] Gaudel, M. .: Formal specification techniques. In *Proceedings of 16th International Conference on Software Engineering*. May 1994. ISSN 0270-5257. pp. 223–227.

[28] Girault, C.; Valk, R.: *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.. 2001. ISBN 3540412174.

[29] Girault, C.; Valk, R.: *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications*. Springer Publishing Company, Incorporated. first edition. 2010. ISBN 3642074472, 9783642074479.

[30] Gogolla, M.; Büttner, F.; Richters, M.: USE: A UML-based specification environment for validating UML and OCL. *Sci. Comput. Program.*. vol. 69, no. 1-3. 2007: pp. 27–34.

[31] Granzer, W.; Kastner, W.: Information modeling in heterogeneous Building Automation Systems. In *2012 9th IEEE International Workshop on Factory Communication Systems*. May 2012. ISSN Pending. pp. 291–300.

[32] Group, I. A. W.: IEEE Std 1471-2000, Recommended practice for architectural description of software-intensive systems. Technical report. IEEE. 2000.

[33] Guan, S. U.; Lim, S.: Modeling adaptable multimedia and self-modifying protocol execution. *Future Generation Comp. Syst.*. vol. 20, no. 1. 2004: pp. 123–143.

[34] Hayden, C. M.; Saur, K.; Hicks, M.; et al.: A study of dynamic software update quiescence for multithreaded programs. In *2012 4th International Workshop on Hot Topics in Software Upgrades (HotSWUp)*. June 2012. ISBN 978-1-4673-1764-1. pp. 6–10.

[35] Hicks, M.: *Dynamic Software Updating*. PhD. Thesis. Department of Computer and Information Science, University of Pennsylvania. August 2001. winner of the 2002 ACM SIGPLAN Doctoral Dissertation award.

[36] Huber, P.; Jensen, K.; Shapiro, R. M.: Hierarchies in coloured Petri nets. In *International Conference on Application and Theory of Petri Nets*. Springer. 1989. pp. 313–341.

[37] ISO Central Secretary: Systems and software engineering - High-level Petri nets - Part 1: Concepts, definitions and graphical notation. Standard ISO/IEC 15909-1:2004. International Organization for Standardization. Geneva, CH. 2004.

[38] Jensen, K.: High-Level Petri Nets. In *Applications and Theory of Petri Nets, Selected Papers from the 3rd European Workshop on Applications and Theory of Petri Nets, Varenna, Italy, September 27-30, 1982*, *Informatik-Fachberichte*, vol. 66, edited by A. Pagnoni; G. Rozenberg. Springer. 1982. ISBN 3-540-12309-1. pp. 166–180.

[39] Jensen, K.: Coloured petri nets. In *Petri nets: central models and their properties*. Springer. 1987. pp. 248–299.

[40] Jensen, K.: *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Volume 1*. Springer Publishing Company, Incorporated. 2010. ISBN 3642082432, 9783642082436.

[41] Jensen, K.: *Coloured Petri nets: basic concepts, analysis methods and practical use*. vol. 1. Springer Science & Business Media. 2013.

[42] Keim, D.; Andrienko, G.; Fekete, J.-D.; et al.: Visual analytics: Definition, Process, and Challenges. In *Information Visualization*, *LNCS*, vol. 4950. Springer. 2008. pp. 154–175.

[43] Kelly, S.; Tolvanen, J.-P.: *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr. 2010. ISBN 978-0-470-03666-2.

[44] Kheldoun, A.; Zhang, J.; Barkaoui, K.; et al.: A High-Level Nets based Approach for Reconfigurations of Distributed Control Systems. In *ADECS 2014, Proceedings of the 1st International Workshop on Petri Nets for Adaptive Discrete-Event Control Systems, co-located with 35th International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2014), Tunis, Tunisia, June 24, 2014.*, *CEUR Workshop Proceedings*, vol. 1161, edited by M. Khalgui; Z. Li. CEUR-WS.org. 2014. pp. 36–51.

[45] Kočí, R.; Janoušek, V.: Simulation Based Design of Control Systems Using DEVS and Petri Nets. In *Computer Aided Systems Theory - EUROCAST 2009*, edited by R. Moreno-Díaz; F. Pichler; A. Quesada-Arencibia. Berlin, Heidelberg: Springer Berlin Heidelberg. 2009. ISBN 978-3-642-04772-5. pp. 849–856.

[46] Köhler, M.; Rölke, H.: Properties of Object Petri Nets. In *Applications and Theory of Petri Nets 2004*, edited by J. Cortadella; W. Reisig. Berlin, Heidelberg: Springer Berlin Heidelberg. 2004. ISBN 978-3-540-27793-4. pp. 278–297.

[47] Köhler-Bußmeier, M.: A Survey of Decidability Results for Elementary Object Systems. *Fundam. Inform.*. vol. 130, no. 1. 2014: pp. 99–123.

[48] Kopetz, H.: The Complexity Challenge in Embedded System Design. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. May 2008. ISSN 1555-0885. pp. 3–12.

[49] Kummer, O.: *Referenznetze*. PhD. Thesis. University of Hamburg, Germany. 2002.

[50] Kummer, O.; Wienberg, F.; Duvigneau, M.; et al.: Renew -User Guide. 2001.

[51] Kummer, O.; Wienberg, F.; Duvigneau, M.; et al.: Renew – The Reference Net Workshop. In *Tool Demonstrations. 24th International Conference on Application and Theory of Petri Nets (ATPN 2003). International Conference on Business Process Management (BPM 2003).*, edited by E. Veerbeek. Department of Technology Management, Technische Universiteit Eindhoven. Beta Research School for Operations Management and Logistics. June 2003. pp. 99–102.

[52] Laid, K.; Allaoua, C.: Coloured Reconfigurable Nets for Code Mobility Modeling. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*. vol. 1, no. 7. 2007: pp. 1939 – 1944. ISSN eISSN:1307-6892.

[53] Lam, W.: *Hardware Design Verification: Simulation and Formal Method-based Approaches*. Prentice Hall Modern Semiconductor Design Series: PH Signal Integrity Library. Prentice Hall Professional Technical Reference. 2005. ISBN 9780131433472.

[54] Lamo, Y.; Wang, X.; Mantz, F.; et al.: *DPF Workbench: A Diagrammatic Multi-Layer Domain Specific (Meta-)Modelling Environment*. Berlin, Heidelberg: Springer Berlin Heidelberg. 2012. ISBN 978-3-642-30454-5. pp. 37–52.

[55] Lamsweerde, A. v.: Formal Specification: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering*. ICSE '00. New York, NY, USA: ACM. 2000. ISBN 1-58113-253-0. pp. 147–159.

[56] Laïd Kahloul; Allaoua Chaoui; Karim Djouani; et al.: Using High Level Nets for the Design of Reconfigurable Manufacturing Systems. In *ADECS @ Petri Nets*. CEUR-WS.org. 2014.

[57] Li, J.; Dai, X.; Meng, Z.: Improved net rewriting system-based approach to model reconfiguration of reconfigurable manufacturing systems. *The International Journal of Advanced Manufacturing Technology*. vol. 37, no. 11. Jul 2008: pp. 1168–1189. ISSN 1433-3015.

[58] Liu, J.; Darabi, H.: Control reconfiguration of discrete event systems controllers with partial observation. *IEEE Trans. Systems, Man, and Cybernetics, Part B*. vol. 34, no. 6. 2004: pp. 2262–2272.

[59] Llorens, M.; Oliver, J.: Structural and Dynamic Changes in Concurrent Systems: Reconfigurable Petri Nets. *IEEE Trans. Computers*. vol. 53, no. 9. 2004: pp. 1147–1158.

[60] Macías, F.; Rutle, A.; Stolz, V.: MultEcore: Combining the Best of Fixed-Level and Multilevel Metamodelling. In *Proceedings of the 3rd International Workshop on Multi-Level Modelling co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2016), Saint-Malo, France, October 4, 2016.*, *CEUR Workshop Proceedings*, vol. 1722, edited by C. Atkinson; G. Grossmann; T. Clark. CEUR-WS.org. 2016. pp. 66–75.

[61] Makris, K.: *Whole-program Dynamic Software Updating*. PhD. Thesis. Arizona State University. Tempe, AZ, USA. 2009.

[62] Margaria, T.; Steffen, B.: *Leveraging Applications of Formal Methods, Verification and Validation. Modeling: 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings*. Number pt. 1 in Lecture Notes in Computer Science. Springer International Publishing. 2018. ISBN 9783030034184.

[63] Mascardi, V.; Martelli, M.; Sterling, L.: Logic-based Specification Languages for Intelligent Software Agents. *Theory Pract. Log. Program.*. vol. 4, no. 4. July 2004: pp. 429–494. ISSN 1471-0684.

[64] Minář, M.: *Interpret Petriho sítí pro řídicí systémy s procesorem Atmel*. Master's Thesis. Brno University of Technology. Faculty of Information technology. Department of Intelligent Systems. 2013.

[65] Morrisett, G.; Walker, D.; Crary, K.; et al.: From System F to Typed Assembly Language. *ACM Trans. Program. Lang. Syst.*. vol. 21, no. 3. May 1999: pp. 527–568. ISSN 0164-0925.

[66] Necula, G. C.: Proof-carrying Code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '97. New York, NY, USA: ACM. 1997. ISBN 0-89791-853-3. pp. 106–119.

[67] Nikolopoulos, C.: *Expert Systems: Introduction to First and Second Generation and Hybrid Knowledge Based Systems*. New York, NY, USA: Marcel Dekker, Inc.. first edition. 1997. ISBN 0824799275.

[68] Ohashi, K.; Shin, K. G.: Model-based Control for Reconfigurable Manufacturing Systems. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation, ICRA 2001, May 21-26, 2001, Seoul, Korea*. 2001. pp. 553–558.

[69] Oshana, R.; Kraeling, M.: *Software Engineering for Embedded Systems: Methods, Practical Techniques, and Applications*. Newton, MA, USA: Newnes. first edition. 2013. ISBN 0124159176, 9780124159174.

[70] Ostermayer, L.; Seipel, D.: A Prolog Framework for Integrating Business Rules into Java Applications. In *Proceedings of 9th Workshop on Knowledge Engineering and Software Engineering (KESE9) co-located with the 36th German Conference on Artificial Intelligence (KI2013), Koblenz, Germany, September 17, 2013.*. 2013.

[71] Owe, O.; Schneider, G.; Leucker, M.; et al.: The 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS'07) A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*. vol. 78, no. 5. 2009: pp. 293 – 303. ISSN 1567-8326.

[72] Patel, P.; Cassou, D.: Enabling high-level application development for the Internet of Things. *Journal of Systems and Software*. vol. 103. 2015: pp. 62 – 84. ISSN 0164-1212.

[73] Raistrick, C.; Francis, P.; Wright, J.: *Model Driven Architecture with Executable UML(TM)*. New York, NY, USA: Cambridge University Press. 2004. ISBN 0521537711.

[74] Ray, C.; Gallen, R.; Iphar, C.; et al.: DeAIS project: detection of AIS spoofing and resulting risks. In *OCEANS 2015-Genova*. IEEE. 2015. pp. 1–6.

[75] Richta, T.; Janoušek, V.: Operating System for Petri Nets-Specified Reconfigurable Embedded Systems. In *Computer Aided Systems Theory - EUROCAST 2013*. LNCS 8111. Springer Verlag. 2013. ISBN 978-3-642-53855-1. pp. 444–451.

[76] Richta, T.; Janoušek, V.; Kočí, R.: Code Generation For Petri Nets-Specified Reconfigurable Distributed Control Systems. In *Proceedings of 15th International Conference on Mechatronics - Mechatronika 2012*. Faculty of Electrical Engineering, Czech Technical University. 2012. ISBN 978-80-01-04985-3. pp. 263–269.

[77] Richta, T.; Janoušek, V.; Kočí, R.: Petri Nets-Based Development of Dynamically Reconfigurable Embedded Systems. *CEUR Workshop Proceedings*. vol. 2013, no. 989. 2013: pp. 203–217. ISSN 1613-0073.

[78] Richta, T.; Janousek, V.; Kocí, R.: Dynamic Software Architecture for Distributed Embedded Control Systems. In *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'15), including the International Workshop on Petri Nets for Adaptive Discrete Event Control Systems (ADECS 2015) A satellite event of the conferences: 36th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2015 and 15th International Conference on Application of Concurrency to System Design ACSD 2015, Brussels, Belgium, June 22-23, 2015.*, *CEUR Workshop Proceedings*, vol. 1372, edited by D. Moldt; H. Rölke; H. Störrle. CEUR-WS.org. 2015. pp. 133–150.

[79] Richta, T.; Macías, F.; Rutle, A.; et al.: Domain Specific Modelling for Reconfigurable Distributed Embedded Control Systems. In *ACIIDS 2018*. Faculty of Electrical Engineering, Czech Technical University. 2018. ISBN 978-80-214-5543-6. pp. 447–452.

[80] Richta, T.; Wang, H.; Osen, O.; et al.: Data-Driven Maritime Processes Management Using Executable Models. In *Computer Aided Systems Theory – EUROCAST 2017*, edited by R. Moreno-Díaz; F. Pichler; A. Quesada-Arencibia. Cham: Springer International Publishing. 2018. ISBN 978-3-319-74727-9. pp. 134–141.

[81] Rodríguez, A.; Fernández-Medina, E.; Piattini, M.: CIM to PIM Transformation: A Reality. In *Research and Practical Issues of Enterprise Information Systems II*, edited by L. D. Xu; A. M. Tjoa; S. S. Chaudhry. Boston, MA: Springer US. 2008. ISBN 978-0-387-76312-5. pp. 1239–1249.

[82] Rose, K.; Eldridge, S.; Chapin, L.: The internet of things: An overview. *The Internet Society (ISOC)*. 2015: pp. 1–50.

[83] Rozenberg, G.: *Behaviour of elementary net systems*. Berlin, Heidelberg: Springer Berlin Heidelberg. 1987. ISBN 978-3-540-47919-2. pp. 60–94.

[84] Rutle, A.; MacCaull, W.; Wang, H.; et al.: A Metamodelling Approach to Behavioural Modelling. In *Proceedings of the Fourth Workshop on Behaviour Modelling - Foundations and Applications*. BM-FA '12. New York, NY, USA: ACM. 2012. ISBN 978-1-4503-1187-8. pp. 5:1–5:10.

[85] Salihbegovic, A.; Eterovic, T.; Kaljic, E.; et al.: Design of a domain specific language and IDE for Internet of things applications. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. May 2015. pp. 996–1001.

[86] Stoyle, G.: *A Theory of Dynamic Software Updates*. University of Cambridge. 2007.

[87] Thiagarajan, P. S.: Elementary Net Systems. In *Petri Nets: Central Models and Their Properties*, edited by W. Brauer; W. Reisig; G. Rozenberg. Berlin, Heidelberg: Springer Berlin Heidelberg. 1987. ISBN 978-3-540-47919-2. pp. 26–59.

[88] Valk, R.: Petri Nets As Token Objects: An Introduction to Elementary Object Nets. In *Proceedings of the 19th International Conference on Application and Theory of Petri Nets*. ICATPN '98. Berlin, Heidelberg: Springer-Verlag. 1998. ISBN 3-540-64677-9. pp. 1–25.

[89] Valk, R.: Petri Nets As Token Objects: An Introduction to Elementary Object Nets. In *Proceedings of the 19th International Conference on Application and Theory of Petri Nets*. ICATPN '98. London, UK, UK: Springer-Verlag. 1998. ISBN 3-540-64677-9. pp. 1–25.

[90] Vieira, R.; Moreira, A.; Wooldridge, M.; et al.: On the Formal Semantics of Speech-Act Based Communication in an Agent-Oriented Programming Language. *J. Artif. Int. Res.*. vol. 29, no. 1. June 2007: page 221–267. ISSN 1076-9757.

[91] Voelter, M.; Salzmann, C.; Kircher, M.: *Component-Based Software Development for Embedded Systems: An Overview of Current Research Trends*. chapter Model Driven Software Development in the Context of Embedded Component Infrastructures. Berlin, Heidelberg: Springer Berlin Heidelberg. 2005. ISBN 978-3-540-31614-5. pp. 143–163.

[92] Wang, H.; Zhuge, X.; Strazdins, G.; et al.: Data Integration and Visualisation for Demanding Marine Operations. In *Oceans 2016: MTS/IEEE Oceans Conference*. 2016.

[93] Wu, N.; Zhou, M.: Intelligent token Petri nets for modelling and control of reconfigurable automated manufacturing systems with dynamical changes. *Transactions of the Institute of Measurement and Control*. vol. 33, no. 1. 2011: pp. 9–29.