Petri Nets-Based Development of Dynamically Reconfigurable Embedded Systems

Tomáš Richta, Vladimír Janoušek, Radek Kočí

Brno University of Technology, Faculty of Information Technology,
IT4Innovations Centre of Excellence
Božetěchova 2, 612 66 Brno, The Czech Republic
{irichta,janousek,koci}@fit.vutbr.cz

Abstract. This paper deals with the embedded systems construction process based on the system specification modeled as a set of Petri nets. Modeling of the system starts with Workflow Petri Nets specification describing the main modules and processes within the system. Workflow model is then transformed to the multilayered Reference Nets structure, that is finally used for the code generation of the target system. The main goal of our approach is to provide for dynamic reconfigurability of the system deployment according to the changes within its specification. Dynamic reconfigurability means the possibility of system changes within its runtime. This is achieved by the decomposition of the whole functionality of the system to small interpretable pieces of computation. This approach also introduces several layers of reconfigurability using different translation rules operating on each layer. The heart of the system lies within the reference nets hosting platform called Petri Nets Operating System (PNOS) that includes the Petri Nets Virtual Machine (PNVM) that performs the very Reference Nets interpretation.

Keywords: workflow modeling, reference nets, embedded systems, model-based software engineering, code generation, model transformation

1 Introduction

Control systems are important border technology lying between the physical and information world. The whole control process is described as a control loop that consists of reading data from sensors and triggers 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 suitable software installation. At a higher level of abstraction, programmable logic controller and its software installation could be seen as an embedded system. In this paper the considered target platform for system installation is set of minimalistic and low energy consumption hardware devices, e.g. Atmel microcontrollers equipped with wireless transmission modules that are often used within Wireless Sensor Networks (WSN).

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 among them. The software part then follows with the programming and installation of each control unit with appropriate application that controls hardware. The main purpose of this paper 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. The reconfigurability 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.

Because there is 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 are also strong demands on its dynamic reconfiguration according to the 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. Within this paper we introduce our solution of the described problem using the Workflow Petri Nets model [1] and MULAN-like multilayered Reference Nets control system structure[5], which is constructed according to the workflow model and then translated into the executable form. The system prototype then runs within the target platform simulator that deduces the requirements for the hardware part of system installation. The main characteristics of the system - its dynamic reconfigurability - is based on the ability of nets to migrate among places as tokens, which was inspired by [5]. The new or modified nets could be sent over another nets to its target place to change the system behavior. Within our solution, the nets are maintained by Petri Nets Operating System (PNOS) and interpreted using the Petri Nets Virtual Machine (PNVM).

Next two sections describe used formalisms and the reconfigurable system architecture. The following section describes the whole system development process using a running example, and the last section contains the evaluation and conclusion.

2 Formalisms and Tools

2.1 Workflow Management

Will van der Aalst defined the way to construct workflow models using Petri Nets[1]. His work is also well formally defined and so the workflow models could be used for the processes verification and validation purposes. The way of modeling the system in this way is also similar to the BPMN workflow models, so it could be easily used by the business process modeling experts. For that reason we decided to use the YAWL notation[2] and Workflow Nets formalism[1] in the beginning of the embedded control system construction process. There are two main concepts from this theory that we use at the moment - basic transition categories (AND-split, AND-join, OR-split and OR-join) and the concept of workflow subprocess.

2.2 Reference Nets

The second part of the system construction then consists of the transformation of Workflow Petri Nets into the multilayered reference Nets model of the system that comply with the nets-within-nets concept defined by Rudiger Valk [3] formalized as Reference Nets[4]. The problem of generating code from formal specification to the running prototype of target system is mainly based on the decomposition of the whole net to a set of subnets, which is also called partitioning problem. For this purpose we use similar concept to the MULAN architecture defined by Cabac et al.[5]. This architecture divides the model into four levels of abstraction - infrastructure, agent platform, agents, and protocols. Our architecture is very similar, we use layers for the infrastructure, platform, main processes and subprocess. Each of those layers is mapped to the target platform and the transformation is used for the code generation. The main goal of the architecture is to enable changes within the system specification during its run-time. This is mainly achieved by the platform, process and subprocess abstraction levels that specify the functionality of the system.

3 Reconfigurable Architecture

Reference Nets allow to construct a system hierarchically, in several levels. Such an idea is a basis of the MULAN (MultiAgent Nets) architecture developed by Cabac et al.[5]. Thanks to the nature of Reference nets, MULAN allows nets to migrate among places in other nets and thus it is possible to dynamically modify functionality of system components, specified by this kind of nets [5]. We use application-specific main processes and subprocesses, which are hosted on platform that is considered to be a part of the operating system of the node, PNOS (Petri Nets Operating System). The multi-layered nature of the system and responsibilities of particular levels are described in Figure 1.

The main part of the the system is installed over the hardware as a PNOS kernel with platform net, that are both able to host other nets. Each platform then hosts some number of main processes nets that hosts subprocesses. The whole communication is performed by sending messages using serial link. There is also theoretical possibility for subprocesses to contain other subprocesses etc. But presented example does not cover this.

The PNOS contains PNVM (Petri Net Virtual Machine) which interprets Petri Nets that are installed in the system in the form of a bytecode called Petri Nets ByteCode (PNBC). PNOS also provides the installed processes with the access to inputs and outputs of the underlying hardware that are connected

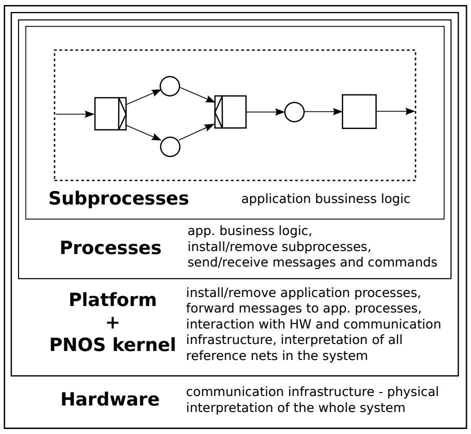
Subprocesses
application bussiness logic
Processes
app. business logic, install/remove subprocesses, send/receive messages and commands
Platform + PNOS kernel
install/remove application processes, forward messages to app. processes, interaction with HW and communication infrastructure, interpretation of all reference nets in the system
Hardware
communication infrastructure - physical interpretation of the whole system

Fig. 1. System layers and their responsibility

to sensors and actuators, and also with the serial communication port that is connected to the wired or wireless communication module (e.g. ZigBee)[8].

The main net (first process) interpreted in PNOS is so called platform net. Platform net is responsible for interpretation of commands which are read from buffered serial line. These commands allow to install, instantiate, and uninstall other Petri nets. The Platform also allows to pass messages to the other layers, which are responsible for application-specific functionality. Since we need reconfigurability in all levels, the installation and uninstallation functionality is implemented in each level.

4 The Development Process

The whole process of system development is described in Figure 2. It starts with the specification of the main system workflow and its subprocesses. Resulting workflow model is then transformed to the layered architecture and might be further debugged using the Renew reference nets tool [6]. After this, the final set of Reference Nets is translated to Petri Nets ByteCode (PNBC) that is then used either for the target prototype simulation using SmallDEVS tool [7] and also to be transfered to the nodes of the system infrastructure. Here it serves as a reconfigurable part of the running system.

More detailed description of the whole PNOS architecture and functionality could be found in [8].

4.1 Running Example

As a running example, we use a subset of a home automation system. The home automation is partly based on the optimization of the energy consumption from multiple sources. There are diverse primary sources of the energy, but within

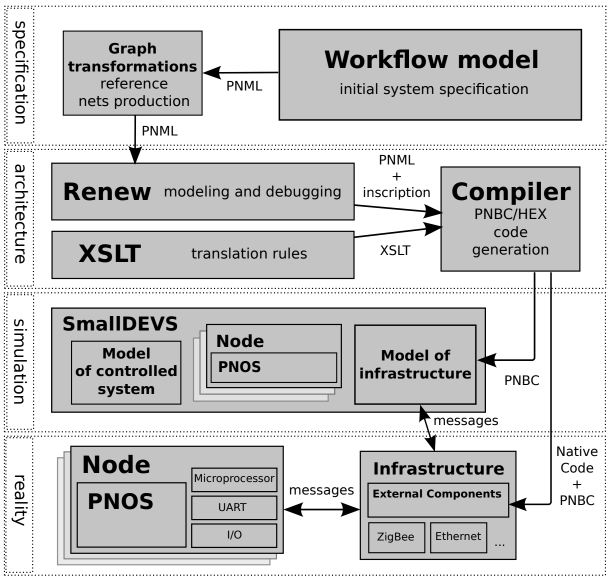

Fig. 2. System construction

our example we concern on the photothermic solar energy panels used for warm water and heating circuits energy supply. The home automation problem used as an example is described in more detail in [9], where also some preliminary ideas about the system design and code interpretation principles are proposed. In this paper we present refined and improved version of the design process and its evaluation.

Home automation process could be described as an workflow model using the Workflow Petri Nets described previously. Next section shows the workflow model and its description.

4.2 House Workflow Model

Within this section, the workflow model of the part of house automation system - the photothermic solar panel and hot water storage tank - is described, using the Workflow Petri Nets defined by Van der Aalst[1]. The Figure 3 describes two swimlines that represent two modules - solar panel and water tank. Each swimline consists of the main process of the module, that is constructed using a set of subprocesses. Within the solar panel module, there is a task of sending data and measure temperature subprocess. In the water tank module, there is a task of receiving the data and two subprocesses - measure temperature and adapt settings. Measure data subprocess and the receive task are connected with the adapt setting subprocess using the OR transition. Particular subprocesses descriptions are shown in next figures.

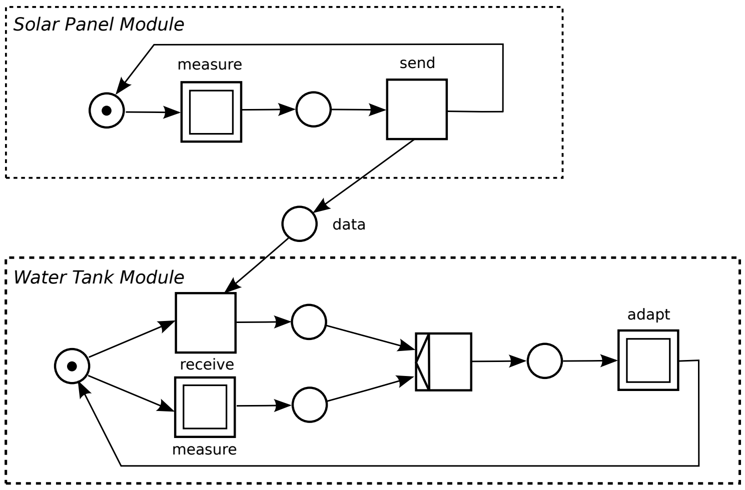

Fig. 3. House workflow example

In Figure 4 the measure subprocess was modeled also using the Workflow Petri Nets. It consists of two tasks - reading the data and converting it to the temperature value. Reading the data means getting the voltage from the input and the conversion means the necessary calculations to produce the human readable results.

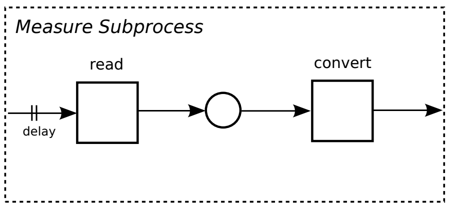

Fig. 4. Measure subprocess net

The other subprocess shown on Figure 5 consists of the task of temperatures reading and comparing them to use the result for the adequate reaction of the automation system. If there is higher temperature on the solar panel than within the water tank, corresponding circular pump is started to move the hot water form panel to the tank.

In this way the system specification is basically defined. But there are some other prerequisites, e.g. we need to know about the technical aspects of reading and writing the input/output data. This information should be obtained from the customer and must be included as a part of the PNOS system. At this moment, these rules are stored in a proprietary format alongside the nets specifications,

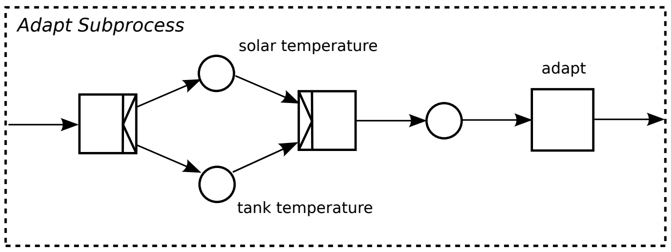

Fig. 5. Adapt subprocess net

but in future we plan to add them as a next layer of the system called drivers. The following section describes the derived four level reference nets architecture, which is produced from described workflow model. The process of conversion of workflow model into the multilayered Reference Nets system is done using some coarsely defined rules, but in future it should be based on formally defined translation rules.

4.3 Layered Reference Nets Architecture

The multilayered system architecture derivation starts with the subprocess nets. In Figure 6 there is the measure subprocess reference net derived from the measure subprocess. This net is constructed adding the initial and final uplinks and places. These uplinks serve as a starting and finishing transitions called from the main process of the module. There are also primitive system functions calls, that operate directly with the underlying operating system. Resulting value token is prepared and sent using uplink : *output*(). All the subprocess protocol nets are named using the name place and corresponding uplink.

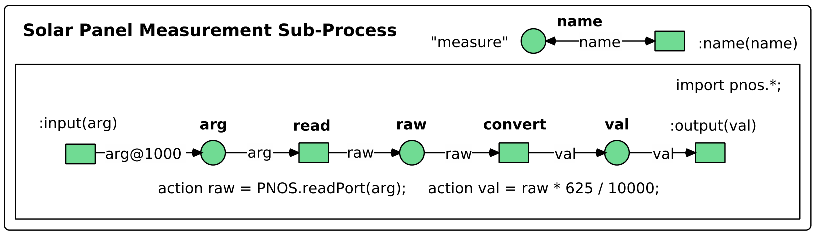

Fig. 6. Measure subprocess net

The solar panel main process described in Figure 7 is derived from the solar panel swimline in the workflow model. It consists of the place, where all the subprocess nets are stored and according to their names are called in particular order. Synchronization place is added between the subprocess protocol nets calls matching the solar panel main process swimline place. The name of the protocol net is derived from the name of the workflow subprocess, and it is not necessary to be human readable.

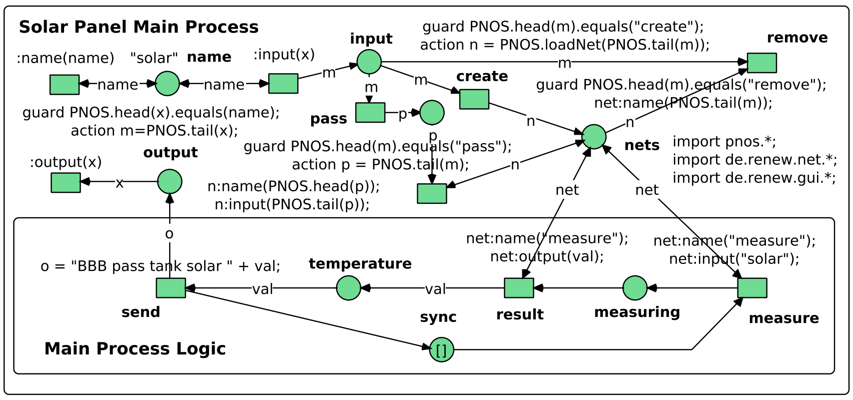

Fig. 7. Solar panel main process

The measurement subprocess protocol net has already been described, so the last net that remains is the settings adaptation subprocess protocol net. It is described in Figure 8 and communicates with the operating system calling the proper signals according to the decisions made in transitions.

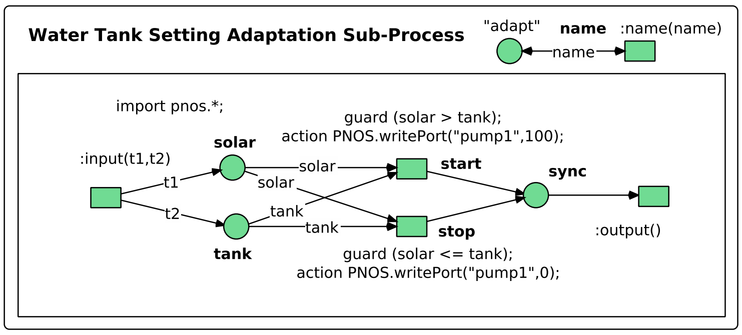

Fig. 8. Adapt subprocess net

The water tank main process reflects the main process in the workflow model. It calls all the subnets and performs the synchronization of subprocesses using two temperature places, that are then synchronized within the adapt subprocess. It is described in Figure 9.

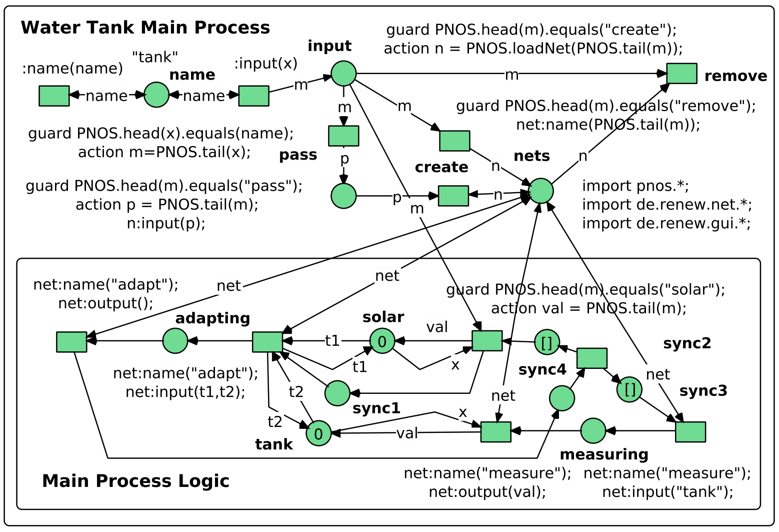

Fig. 9. Water tank main process

Above the last net, called infrastructure, there is a part of the underlying operating system called the platform net that describes the main required functions of the operating system needed by the application processes installed on it. The platform net is shown in Figure 10.

Finally the infrastructure layer, that is derived from the main workflow process description, is shown in Figure 11. In our example, it is very simple. Each swimline represents one place, where the module for hosting the platform, main process and protocols will be placed. The communication between the two subprocesses seated in different swimlines is represented here as an communication transition, that should internally call the final transition of the send task, that means the : *output*() downlink and the initial transition of the receive task, that means the : *input*() downlink. Those transitions are part of the platform layer and are propagated to the subprocesses nets.

4.4 Code Generation

Generally, in our approach, each layer of the system can be compiled to target code independently. There are two possibilities: first - the target code can be the native code of the controller processor, and second - target code is a bytecode that is interpreted by some virtual machine. Regardless on the way the code is generated, all the abstraction levels communicate with each other using uplinks and downlinks. The difference is, that levels deployed as interpreted bytecode are more flexible and dynamically changeable than the compiled ones. It is because such a modification needs a heavy compiler and (possibly) over-the-air programming of the node, that consumes a lot of energy. On the other hand

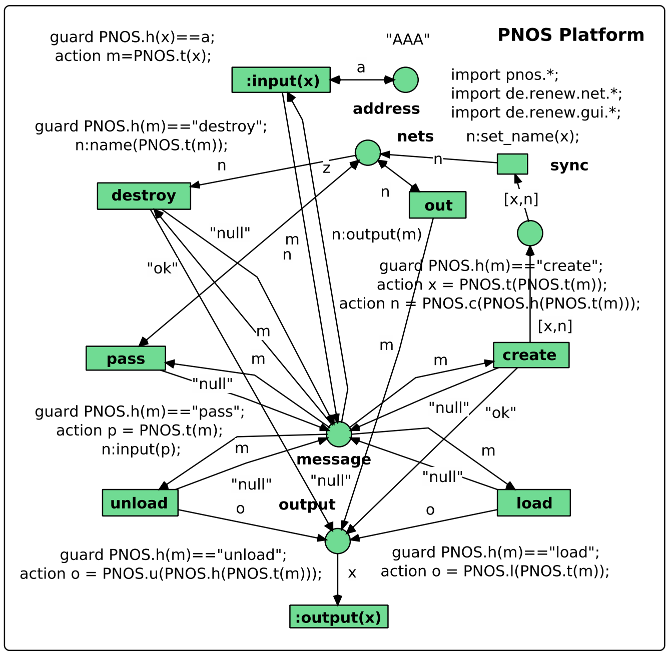

Fig. 10. Platform net

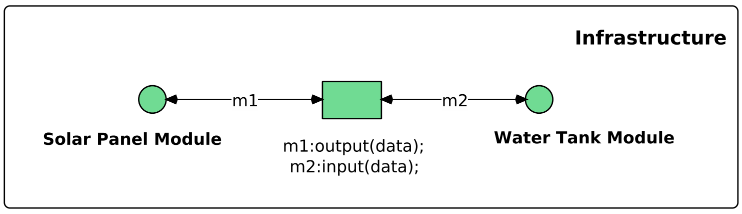

Fig. 11. Infrastructure net

it is possible to send the bytecode to the node as data. It thus allows for very high level of dynamic reconfigurability in the system runtime. E.g. new version of the measure subprocess is produced, then the corresponding Reference Net is derived and proper bytecode is generated. Finally the new version of the measure net is sent to the relevant node, and installed by its platform net.

We currently use the virtual machine and bytecode for all Petri Nets-based code. The only part which is implemented natively, is the PNOS kernel, including PNVM [8]. The example of bytecode follows. It represents simple net that reads data from some sensor and produces relevant output (the net is depicted in Figure 6). In fact, it is a human-readable version of the bytecode. In this representation, numbers are represented as a text and also some spaces and line breaks are added. This means that the contents of the code memory is a bit more condensed. Each byte of the code is either an instruction for PNVM, or data.

```
(Nmeasure
 (measure)
 (arg/raw/val/name)
 (Uoutput(val)()(P3(B1)(V1)))
 (Uinput(arg)()(Y1(B1)(V1)(I1000)))
 (Uname(name)()(P4(B1)(V1)))
 (I(O4(B1)(S1)))
 (Tread(arg/raw)
  (P1(B1)(V1))
  (A(:(V2)(r(V1)))
  (O2(B1)(V2)))
 (Tconvert(raw/val)
  (P2(B1)(V1))
  (A(:(V2)(/(*(V1)(I625))(I10000))))
  (O3(B1)(V2))))
```

The bytecode contains symbols definitions and places definitions, allowed by a code for each uplink (U), initialization (I), and each transition (T). Each transition description consists of preconditions (P), guard (G), action (A) and postconditions (O), in a form of instructions for the PNVM. Each data element is a tuple consisting of a type and a value. Variables are declared as a part of transition code and identified by indexes. When the code of the net template is loaded to code memory of the PNVM, it is indexed in order to allow PNVM to quickly access particular parts of the code, especially places declarations, the uplinks and the transitions code. When the net template is instantiated, a specific part of runtime memory is allocated according to number of places. At the same time, the net transitions are scheduled for execution. Execution of a transition consists of reading its bytecode and attempting to satisfy all preconditions, downlinks and guards using recursive backtracking algorithm.

In guards and actions of transitions it is possible to call primitive operations of the underlying PNOS. Those operations are available in the Reference Nets inscription language as `PNOS.`*operation*, e.g., `PNOS.readPort("solar1")` reads data from virtual port named solar1, `PNOS.writePort("pump1",100)` writes value to the virtual port named pump1, `PNOS.h(m)` gets first space-separated substring from string m, and `PNOS.t(m)` returns the rest of the string m without the first substring.

Those primitive operations are directly mapped to the corresponding bytecode. We use a subset of the Reference Nets inscription language here. It works only on integers and strings as values with corresponding set of basic operations.

The important feature of the system is its reconfigurability. It is based on operations of the operating system that are designated for manipulations with nets (in the form of PNBC) and their instances. Nets could be sent to a node as a part of the command for its installation. The command is executed by Platform net. Using other commands, the platform can instantiate a net, pass a command to it, destroy a net instance and unload a net template - see Figure 10. The PNOS Platform functionality is described in more detail in [8].

4.5 Simulation in SmallDEVS

PNOS-based nodes can be simulated in SmallDEVS environment [7], together with simulation models of sensors and actuators connected to the controlled physical process, as well as with simulation model of communication infrastructure. While Renew is used for application business logic debugging, SmallDEVS is used for realistic simulation of the system with its surroundings. Execution steps delays are incorporated to the simulation model in order to make simulation as realistic as possible. Statistics gathered from simulation experiments can be used for verification purposes and also can support decisions about type of hardware for target system implementation. Hardware-In-the-Loop simulation is also possible.

5 Evaluation

For the testing purposes we use the Arduino and Raspberry Pi hardware platforms with XBee modules for wireless communication. The Arduino is enabled with the ATmega328P chip that introduces some important restrictions to the implementation. The main one is the 2kB SRAM memory that makes extensive use of direct Petri Nets interpretation very difficult. There is a strong limitation for the number of nets and also for the complexity of problems solved. For that purposes we consider now for further testing of the system to use the Raspberry Pi platform, that offer much more memory for the interpretation purposes. The energy consumption of the ARM could be reduced by underclocking, that is part of our future work plans.

PNVM/PNOS prototype has been implemented in Smalltalk. The implementation resembles the way how it will be implemented in pure C language in order to make final implementation easy. Up to now, we do not have C implementation ready, because we are still doing minor improvements to the reference Smalltalk version. Nevertheless, the automated generation of C version of PNVM/PNOS is planned for very near future.

With the hardware limitations in mind, we have tested the PNVM/PNOS prototype with a model containing the platform net and other three simple nets (9 transitions in all nets), that are loaded and instantiated successively. The code of nets occupies 718 B, 679 B, 147 B, and 115 B, what is 1659 B of total used memory for code. The simulation generates 4 net instances, containing 14 places. The number of tokens is up to 31, and needs 1547 B of object memory. The history of memory occupation is shown in Figure 12. Peaks in the graph corresponds to receiving a net via serial line and its loading to code memory.

To investigate the time consumption of the simulation, we measured the time needed for each step execution. It comprises evaluation of all transitions in all

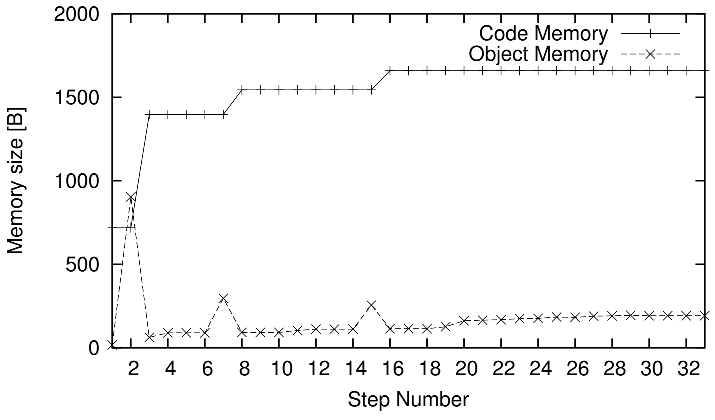

Fig. 12. Memory usage

ten instances. The simulation was executed for 50 times to get average step duration.

The history of simulation steps durations is shown in Figure 13. We can see, that the duration increases depending on number of instances because the number of transitions is increasing. Peeks in the graph correspond with net loading, net instantiation, and uplink execution. These experiments has been done on contemporary desktop computer. On Raspberry Pi the step duration is about 100 times higher, because of slower CPU and slower access to the memory. Nevertheless, we suppose that C version of the PNVM/PNOS for Raspberry Pi will run reasonably faster which will make Arduino and Raspberry Pi platforms well usable for Petri nets-specified control systems.

6 Related Work

The use of high-level languages, especially Petri Nets, allows to build and maintain control systems in a quite fast and intuitive way. There are many approaches to relate high-level languages with embedded devices or microcontrollers. One kind of that approaches is applicable in systems with not very limited resources. For example, Java can be used as a high-level language and works on architecture which can be successfully used in embedded systems [10]. To control robot application, hierarchical Petri Nets are used for middleware implementation in a RoboGraph framework [11]. Another approaches are focused to the devices with limited resources. They obviously use high-level languages or models for system design and the implementation is generated, usually into the C code. An examle is a usage of Timed Petri Nets for the synthesis of control software for embedded systems by generating C-code [12] or Sequential Function Charts [13]. All these approaches allow to design systems using high-level languages or models, but they either do not preserve models in the system implementation,

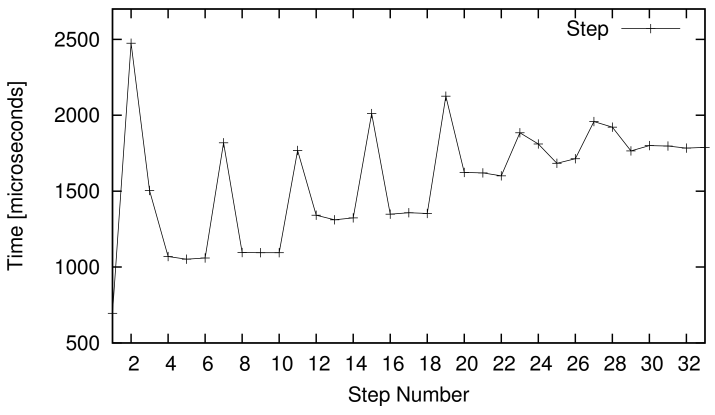

Fig. 13. Time overhead of simulation steps

or are not applicable for systems with limited resources. The approach presented in this paper allows not only for design of systems with limted resources, but also for systems implementation using a high-level language, particularly the Nets-within-Nets formalism, allowing for the dynamic reconfigurability.

7 Conclusion

In this paper, we described the process of system construction based on Petri Nets-based models transformations and target prototype code generation. This process starts with the workflow model defined according to the Van der Aalst's WF-Nets that describe the functionality of the system from the customers point of view. This model is then transformed to the multilayered architecture based on Reference Nets formalism. Each layer of the architecture is then translated to the specific target representation. The main part of the system is translated to the Petri Nets ByteCode (PNBC), that is interpreted by the Petri Nets Virtual Machine (PNVM) that is part of the Petri Nets Operating System (PNOS) which forms the remaining part of the system.

The whole system reconfigurability is based on the possibility of PNBC net replacement with the new version where the interpretation after reinstalling starts to perform the new version of the process. In the current version, the reconfigurability is considered to work on the granularity of processes and subprocesses. In further research, we plan to focus on more fine grained reconfiguration, including the platform primitive operations, and also on the processes migration.

Acknowledgment

This work has been 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.

References

1. Van der Aalst, W.M.P., Van Hee, K.M. 2002. Workflow Management: Models, Methods, and Systems. IT press, Cambridge, MA.
2. Van der Aalst, W.M.P., Ter Hofstede, A.H.M. 2005. YAWL: yet another workflow language. Inf. Syst. 30, 4 (June 2005), 245-275.
3. Valk, R. 1998. 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), Jrgen Desel and Manuel Silva (Eds.). Springer-Verlag, London, UK, 1-25.
4. Kummer, O. 2001. Introduction to petri nets and reference nets. SozionikAktuell 1:2001 / Rolf von Lüde, Daniel Moldt, Rüdiger Valk (Hrsg.).
5. Cabac, L., Duvigneau, M., Moldt, D., Rölke, H. 2005. Modeling dynamic architectures using nets-within-nets. In Proceedings of the 26th international conference on Applications and Theory of Petri Nets (ICATPN'05), Gianfranco Ciardo and Philippe Darondeau (Eds.). Springer-Verlag, Berlin, Heidelberg, 148-167.
6. Kummer, O., Wienberg, F., Duvigneau, M., Köhler, M., Moldt, D., and Rölke, H. 2003. Renew - the Reference Net Workshop. Tool Demonstrations. Eric Veerbeek (ed.). 24th International Conference on Application and Theory of Petri Nets (ATPN 2003).
7. Češka M., Janoušek V., Vojnar T. PNtalk - A Computerized Tool for Object Oriented Petri Nets Modelling. Lecture Notes in Computer Science, vol. 1333, DE, p. 591-610, ISBN 3-540-63811-3, ISSN 0302-9743, 1997.
8. Richta, T., Janoušek, V. 2013. Operating System for Petri Nets-Specified Reconfigurable Embedded Systems, To appear in: Proceedings of the 14th Computer Aided Systems Theory, Las Palmas de Grand Canaria, LNCS, Springer Verlag.
9. Richta, T., Janoušek, V., Kočí, R. 2012. Code Generation For Petri Nets-Specified Reconfigurable Distributed Control Systems, In: Proceedings of 15th International Conference on Mechatronics - Mechatronika 2012, Praha, CZ, FEL Č, 2012, s. 263-269, ISBN 978-80-01-04985-3.
10. Pizlo, F., Ziarek, L., et al. 2010. High-level programming of embedded hard real-time devices. In Proceedings of the 5th European conference on Computer systems, ACM New York, 69-82.
11. Fernandez, J.L., Sanz, R., Paz, E., Alonso, C. 2008. Using hierarchical binary Petri nets to build robust mobile robot applications: RoboGraph. In IEEE International Conference on Robotics and Automation. 1372-1377.
12. Rust, C., Stappert, F., Kunnemeyer, R. From Timed Petri Nets to Interrupt-Driven Embedded Control Software. In International Conference on Computer, Communication and Control Technologies (CCCT 2003), Orlando, Florida, USA. p. 6.
13. Bayo-Puxan, O., Rafecas-Sabate, J., Gomis-Bellmunt, O., Bergas-Jane, J. 2008. A GRAFCET-compiler methodology for C-programmed microcontrollers, In Assembly Automation, Vol. 28 Iss: 1, Emerald Group Publishing. 55-60.