Dynamic Software Architecture for Distributed Embedded Control 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 focuses on the field of dynamically reconfigurable distributed embedded control systems construction process and presents a substantial part of the methodology aimed at this application area which is based on formal models, namely some variants of Petri Nets. Initial system specification is represented by a set of Workflow Petri Nets transformed into decomposed multi-layered Reference Petri Nets model, that is used during the generation of interpretable target system components representation. The main objective of presented approach is the introduction of dynamic reconfigurability features into the target system implementation reflecting changes in system specification during its run-time. Reconfigurability is achieved by the system decomposition into smaller interpretable pieces of computation that are installed on and performed by the underlying infrastructure. Introduced approach brings several layers of reconfigurability through a set of specific translation rules applied in different layers and scenarios for pseudo-code generation and by the possibility of installing the resultant functional parts on system nodes using well-defined communication protocol. The heart of described architecture lies within the specification of hosting platform called Petri Nets Operating System (PNOS) that includes the Reference Petri Nets interpreter.

Keywords: Dynamic Reconfigurability, Embedded Systems, Control Systems, Model-Driven Development, Model Transformation, Model Execution, Workflow Petri Nets, Reference Petri Nets

1 Introduction and Motivation

Control systems lie on the thin border between physical and information worlds. The process of control is usually described as a loop switching between reading data from sensors and triggering a number of actuators installed within the physical environment. Above all, the process should respect all user-defined rules. Control systems could be constructed as a set of programmable logic controllers with proprietary software installation, communicating with each other, thus forming distributed embedded control system. Our work considers a target platform for this type of systems implementation to be a set of minimalistic and

low energy consumption hardware devices, e.g. ATmega, PIC, or ARM microcontrollers, equipped with wireless transmission modules. Such devices are often used in the area of Wireless Sensor Networks (WSN) systems.

Usually a hardware part of any system implementation starts with selection of proper set of devices and their installation within the physical environment, including sensors and actuators attachment. The software part of system implementation complements the hardware one with the construction of appropriate application software, that controls each system unit and represents the whole system functionality. Dynamic reconfigurability features are necessary for the ability of the system to adapt itself to changes in environment and also to provide its maintainer with a possibility to change the system behaviour, while it is in runtime, i.e. without the necessity of complete destruction and further reconstruction, or even restart. Our main goal is to describe the software part of the process, that respects our focus on formal specification and dynamic reconfigurability.

In this paper, we are going to describe some recent results of the research in the field of dynamically reconfigurable distributed embedded control systems, and basic ideas of our research that aims to introduce complete methodology for control systems construction and administration, which uses formal and human readable notation as a system functionality specification, and provides the user of resulting system with the possibility to change its behaviour within the runtime. Introduced solution to the dynamic reconfigurability problem follows the model transformation and executable model paradigms - Workflow Petri Nets[1] are used as an abstract system specification modelling language, and the MULAN-like[5] multi-layered Reference Petri Nets[3] structure for modelling the resultant system implementation. The system run-time model is constructed from the work-flow one using graph transformations and then translated into the executable form, run by our specialized target platform.

2 Related Work

Related work could be divided into the following areas - embedded and operating systems, software engineering methods applied to the area of embedded systems, the usage of higher-level or visual languages for embedded systems specification and implementation, the dynamical reconfigurability within embedded systems, reconfigurable control systems (e.g. FMSs), multi-agent approach to the reconfigurable embedded systems development, system partitioning, code generation, and reconfigurable hardware.

The usage of formal modelling control system with dynamic reconfigurability features is not a new idea. Research activities in this topic are primarily focused on 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 reconfigurations. The main difference consists usually in 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 possible to change the system structure.

In our field of research the first group consists of formalisms based usually on some kind of Petri nets. Reconfigurable Petri Nets [11], presented by Guan and Lim, introduced a special place describing the reconfiguration behaviour. Net Rewriting System [12] extends the basic model of Petri Nets and offers a mechanism of dynamic changes description. This work has been improved [13] by the possibility to implement net blocks according to their interfaces. Intelligent Token Petri Nets [14] introduces tokens representing jobs. Each job reflects knowledge about the system states and changes, so that the dynamic change could be easily modelled. All the presented formalisms is able to describe the system reconfiguration behaviour, nevertheless only some of them define the modularity. Moreover, the study [15] shows, that the level of reconfigurability is dependent on the level of modularity and also that there are modular structures that are not reconfigurable.

The second group handles reconfigurations using extra mechanisms. Model-based control design method, presented by Ohashi and Shin [16], uses state transition diagrams and general graph representations. Discrete-event controller based on finite automata has been presented by Liu and Darabi [17]. 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 [18], 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 [19]. To develop embedded control system, the developer has to consider several areas. We can distinguish five areas [19] 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).

Presented approach is based on existing formalisms and architecture that are together used in the specific platform developed by our team. In relation to previously defined areas of software engineering for embedded control systems, we deal with *Process*, *Platform*, and *Middleware* areas in this paper. *Process* area focuses on work-flow modelling using Petri Nets, transformation of the work-flow models into the Reference Nets, and definition of the levels of abstraction. *Middleware* area focuses on multi-layered architecture inspired by the MULAN architecture. *Platform* area introduces Petri Nets Operating System (PNOS)

linked with Petri Net Virtual Machine (PNVM) that offer specific means for system reconfigurability. All mentioned elements will be described in details in next chapters.

3 Formalisms and Tools

3.1 Workflow Modelling

Work-flow modelling 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 and verifiable characteristics and large research background we adopted for the purposes of our research Wil van der Aalst's specification for system work-flow modelling, so called Extended Workflow Petri Nets[1]. Aalst's work is well-defined and resulting work-flow models could be used for the system processes verification and validation purposes. This way is very similar to the BPMN work-flow models, so it might be easily used by the business process modelling domain experts. For that reason we decided to use the Aalst's YAWL notation[2] and Workflow Petri Nets formalism[1] in the early beginning of system construction process. The main advantage of using Workflow Petri Nets is the possibility of system specification and its adaptation by the non-technically educated domain specialists.

3.2 Reference Nets

Second step of the system construction process consists of the transformation of Workflow Petri Nets model into the multi-layered Reference Nets model complying with the nets-within-nets concept defined by Rüdiger Valk [3] and formalized as Reference Nets by Olaf Kummer [4]. In our proposed system development methodology, Reference Nets are translated into the interpretable form, that is transferred through the network to the specific nodes, responsible of its execution. The problem of generating the code from formal specification to its runnable form is mainly based on the decomposition of the whole system model to a set of sub-models, that is usually called the partitioning problem. 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 also uses four layers - infrastructure, platform, processes and sub-process. Each of these layers is mapped from the formal specification to the target platform specification.

4 Reconfigurable Architecture

Reference Nets allows to construct the system hierarchically, in several layers of abstraction. Each element of layer at any level of abstraction could be changed by change in nets marking. Nets representing system functionality are migrating

over nets of other layers changing the system functionality. The multi-layered nature of the system and responsibilities of particular levels of system decomposition is described in more detail in our previous work[10].

The core characteristics of resulting system, its dynamic reconfigurability, is based in our solution on the ability of Reference Petri Nets interpretable representations to migrate among places of the system as tokens, similarly as in reference Nets. The new or modified Petri Net, that represents the system partial behaviour change could be sent over other Petri Nets to its destination place to change the whole system functionality. In our solution, these Petri Nets parts are maintained by the Petri Nets Operating System (PNOS) and interpreted by the Petri Nets Virtual Machine (PNVM) engine[8]. System decomposition is inspired by MULAN architecture [5].The PNOS contains PNVM (Petri Net Virtual Machine) engine that interprets Petri Nets which are installed within the system in the form of a interpretable byte-code called Petri Nets ByteCode (PNBC). PNOS also provides the installed processes with the access to input and output of the underlying hardware that is connected 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], or Ethernet interface.

The important net (lying above processes nets) interpreted in PNOS is so called Platform net. Platform net is responsible for the interpretation of commands which are read from buffered serial line, or Ethernet. 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 of functionality is implemented in each level of resulting system. Next section describes the Reference nets formalism that is used as an intermediate language for the target implementation.

5 The Development Process

System development process is described in Fig. 1. It starts with the specification of the whole system work-flow, in an hierarchical way. Work-flow model is transformed to the Reference Nets layered architecture and might be further simulated and debugged using the Renew Reference Nets tools [6]. After this stage, the final set of Reference Nets is then translated into the Petri Nets ByteCode (PNBC) that is used either for the target prototype simulation using SmallDEVS tools [7] and also to be transferred to the nodes of the system infrastructure. More detailed description of the whole PNOS architecture and functionality could be found in [8].

5.1 Model Transformations

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

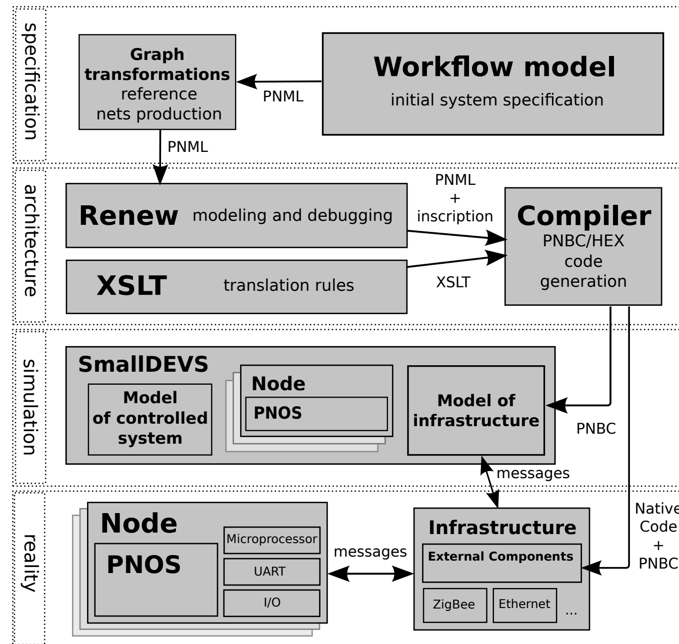

Fig. 1. System construction process

account the set of work-flow specifications described within the work-flow 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, that 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 work-flow model, also other specific rules for the communication protocol could be derived.

5.2 Basic Definitions

Let us introduce some basic definitions of formalisms used during the system development. As a basement the classical Petri nets definition describes the main rules of the specification formalism.

Definition 1 (Petri Net). *A Petri net is a triple* $PN = (P, T, F)$ *where:*

- *P and T are disjoint finite sets of places and transitions, respectively and*
- *$F \subseteq (P \times T) \cup (T \times P)$ is a binary relation called the flow relation representing arcs of the net.*
- *${}^{\bullet}x = y | yFx$ is called input set (preset) of the element x and*
- *$x^{\bullet} = y | xFy$ is called output set (postset) of the element x, where $x \in P \cup T$.*

Van der Aalst's extensions to Petri Nets add two basic conditions to the nets construction. Our modelling approach is very similar, so we can use his definition, but for the further transformation of models we need some more rules to be added. First let us introduce the simple work-flow net definition.

Definition 2 (Workflow Net). *A Petri net $PN = (P, T, F)$ is a WF-net (WorkFlow net) if and only if [1]:*

- *PN has two special places: $i \in P$ and $o \in P$. Place i is a source place: ${}^{\bullet}i = \emptyset$. Place o is a sink place: $o^{\bullet} = \emptyset$.*
- *If we add a transition $t*$ to PN which connects place o with i (i.e. ${}^{\bullet}t^{*} = \{o\}$ and $t^{*\bullet} = \{i\}$), then the resulting Petri net is strongly connected.*

Some other simplification rules added by Aalst and Hofstede extended workflow models to provide for better human-readability. Some special types of transitions representing logical operators and some special operations for manipulation with tokens were added. Transitions and places are considered to be tasks and conditions. Each EWF-net consists of tasks (either composite or atomic) and conditions which can be interpreted as places. Tasks in elementary form are atomic units of work, and in compound form modularize an execution order of a set of tasks. In contrast to Petri nets, it is possible to connect "transition-like objects" like composite and atomic tasks directly to each other without using a "place-like object" (i.e., conditions) in-between[2].

Definition 3 (Extended Workflow Net). *An extended work-flow net (EWF-net) is a tuple EWF = (C, i, o, T, F, S, name, split, join, rem, nofi) such that [2]:*

- *C is a set of conditions,*
- *$i \in C$ is the input condition,*
- *$o \in C$ is the output condition,*
- *T is set of tasks,*
- *$F \subseteq (C \setminus \{o\} \times T) \cup (T \times C \setminus \{i\}) \cup (T \times T)$ is the flow relation,*
- *every node in net graph $(C \cup T, F)$ is on a directed path from i to o,*
- *$split : T \to \{AND, XOR, OR\}$ specifies the split behaviour of each task,*
- *$join : T \to \{AND, XOR, OR\}$ specifies the join behaviour of each task,*
- *$rem : T \not\to \mathbb{P}(T \cup C \setminus \{i, o\})$ specifies the additional tokens to be removed by emptying a part of the work-flow, and*
- *$nofi : T \not\to \mathbb{N} \times \mathbb{N}^{inf} \times \mathbb{N}^{inf} \times \{dynamic, static\}$ specifies the multiplicity of each task (minimum, maximum, threshold for continuation, and dynamic/static creation of instances).*

Our approach follows the previous definitions and adds some more rules to enable the extended work-flow models with communication features to satisfy the developer ability to combine multiple work-flow specifications.

Definition 4 (Extended Communicating Workflow Net). *We call Extended Communicating Workflow net $ECWF$ = (EWF,I,O,F^C) a EWF net that has following properties:*

- EWF *is an extended work-flow net,*
- I *is a set of* $ECWF$ *input places, where* $\forall p_I \in I : {}^\bullet p_I = \emptyset \wedge p_I \neq i$,
- O *is a set of* $ECWF$ *output places, where* $\forall p_O \in O : p_O^\bullet = \emptyset \wedge p_O \neq o$,
- F^C *is a communication flow* $F^C \subseteq (I \times T) \cup (T \times O)$,
- $I \cup P_{EWF} = \emptyset \wedge O \cup P_{EWF} = \emptyset$.

To specify complete work-flow model a definition of Workflow Specification was introduced by Aalst and Hofstede. We adopted this definition and added some slight change to one of the rules.

Definition 5 (Workflow Specification). *A Workflow Specification S is a n-tuple (Q, top, $T^\diamond$,map) such that:*

- *Q is a set of ECWF-nets,*
- $top \in Q$ *is the top level work-flow[2],*
- $T^\diamond = \cup_{N \in Q} T_N$ *is the set of all tasks[2],*
- $\forall_{N_1,N_2 \in Q} N_1 \neq N_2 \Rightarrow (C_{N_1} \cup T_{N_1}) \cap (C_{N_2} \cup T_{N_2}) = \emptyset$, *i.e., no name clashes[2],*
- $map : T^\diamond \not\twoheadrightarrow Q \setminus \{top\}$ *is a surjective injective (bijective) function which maps each composite task onto a EWF net[2], and*
- *the relation* $\{(N_1, N_2) \in Q \times Q \mid \exists_{t \in dom(map_{N_1})} map_{N_1}(t) = N_2\}$ *is a tree[2].*

And also some special types of tasks representing composite and multi-instance tasks were added by Aalst and Hofstede.

Definition 6. *Whenever we introduce a work-flow specification S = (Q, top, $T^\diamond$, map), we assume T^A, T^C, T^{SI}, T^{MI}, $C^\diamond$ to be defined as follows [2]:*

- $T^A = \{t \in T^\diamond | t \notin dom(map)\}$ *is the set of atomic tasks,*
- $T^C = \{t \in T^\diamond | t \in dom(map)\}$ *is the set of composite tasks,*
- $T^{SI} = \{t \in T^\diamond | \forall_{N \in Q} t \in dom(nofi_N)\}$ *is the set of single instance tasks,*
- $T^{MI} = \{t \in T^\diamond | \exists_{N \in Q}\ t \in dom(nofi_N)\}$ *is the set of (potentially) multiple instance tasks, and*
- $C^\diamond = \cup_{N \in Q} C_N^{ext}$ *is the extended set of all conditions.*

Final definition describes the Workflow System consisting of set of Extended Communicating Workflow Specifications and communication transitions.

Definition 7 (Workflow System). *Let us call Workflow System the triple WS = ($\widehat{S}$, T^{WS}, F^{WS}), where:*

- $\widehat{S}$ *is non-empty finite set of extended communicating work-flow specifications,*

- T^{WS} *is a finite set of communication transitions,*
- $F^{WS} \subseteq (O^{WS} \times T^{WS}) \times (T^{WS} \times I^{WS})$ *is a system communication flow relation, where* $O^{WS} = \bigcup_{O_{Si} i \in <1,\dots,n>}$ *is a set of all extended communicating work-flow specifications output places and,* $I^{WS} = \bigcup_{I_{Si} i \in <1,\dots,n>}$ *is a set of all extended communicating work-flow specifications input places.*

Target system representation for the first phase of system model transformation is constructed as a set of Reference Nets based on Valk's nets-within-nets paradigm that is formalized as an Elementary Object System which consists of elementary net systems (EN System) $EN = (B,E,F,C)$, which is defined as finite set of places B, finite set of transitions E, disjoint from B, a flow relation $F \subseteq (B \times E) \cup (E \times B)$ and an initial marking $C \subseteq B$ [3].

Definition 8 (Elementary Object System). *An elementary object system is a n-tuple* $EOS = (SN, \widehat{ON}, Rho, type, \widehat{M})$ *where [3]:*

- $SN = (P, T, W)$ *is a Petri net, called system net of EOS,*
- $\widehat{ON} = \{ON_1, \dots, ON_n\}(n \geq 1)$ *is a finite set of EN systems, called object systems of EOS, denoted by* $ON_i = (B_i, E_i, F_i, m_{0i})$*, which is either elementary net system or a system net of embedded EOS,*
- $Rho = (\rho, \sigma)$ *is the interaction relation, consisting of a system/object interaction relation* $\rho \subseteq T \times E$ *where* $E := \bigcup\{E_i | 1 \leq i \leq n\}$ *and symmetric object/object interaction relation* $\sigma \subseteq (E \times E) \setminus id_E$*,*
- $type : W \to 2^{\{1,\dots,n\}} \cup \mathbb{N}$ *is the arc type function, and*
- $\widehat{M}$ *is a marking defined in following definition.*

Definition 9 (System Marking). *The set* $Obj := \{(ON_i, m_i) | 1 \leq i \leq n, m_i \in R(ON_i)\}$ *is the set of objects of the elementary object system. An object-marking (O-marking) is a mapping* $\widehat{M} : P \to 2^{Obj} \cup \mathbb{N}$ *such that* $\widehat{M}(p) \cap Obj \neq \emptyset \Rightarrow \widehat{M}(p) \cap \mathbb{N} = \emptyset$ *for all* $p \in P$.

Next paragraphs are going to describe both transformation process phases. The first one is the transformation of the work-flow model into the operational nets-within-nets model, second one the transformation of the nets-within-nets model into its interpretable form, reflecting the target PNOS platform.

5.3 From Workflow Nets to Reference Nets

We decided to describe our methods on the sample home automation example. The whole system functionality is described in the form of work-flow model in our approach represented by the Workflow System depicted in Fig. 2. There are following elements within the work-flow models - places, transitions, and logical transitions[1], sub-process transitions[1], connecting arcs, and system nodes borders. Places could be named, when there is a name on the place it is further considered as an variable name. Transitions could be also named. The named transition represents calling some particular atomic function of the underlying PNOS.

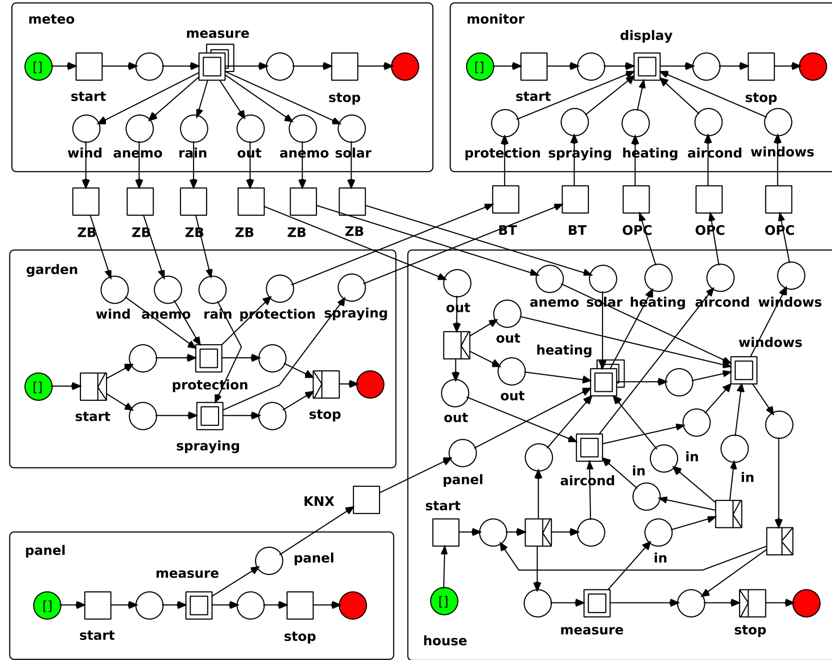

Fig. 2. Workflow System net

Logical transitions are: AND-split, AND-join, OR-split, OR-join, and AND/OR-split, they simplify the model to be easily readable for the non-technically educated domain experts. Sub-process transitions represent condensed parts of the system, that are described in another diagram, e.g. in Fig. 3.

Generating the Infrastructure layer Work-flow model of the intended system is translated into multi-layered Reference Nets model. Each layer of the Reference Nets model is generated separately using different production rules. First part of the system, that should be generated from the original model is the top level Infrastructure layer net, that describes the communication among all nodes of the system and could be used as a sort of deployment diagram. Infrastructure layer is a basic layer of the Reference Nets model and serves for the validation purposes and also as a description of the distribution of target system structure. Basically the main purpose of Infrastructure layer lies in description of the system nodes and their communication.

Within the Infrastructure layer, each node is represented as a place in which the particular Platform layer net is located. If there is any communication between nodes, this communication is represented as a transition between corresponding nodes. For example model described in Fig. 2 should be translated into the Infrastructure net described in Fig. 4. This layer is produced by the following set of rules.

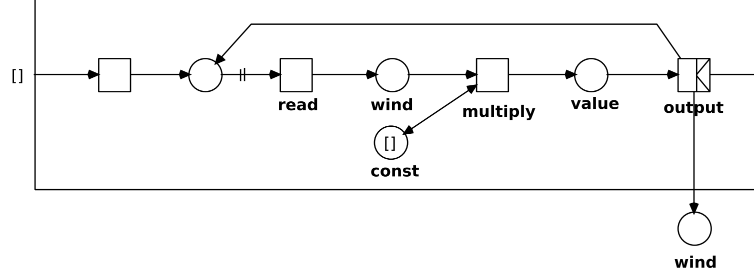

Fig. 3. Measure subprocess

Let $WS = (\widehat{S},\ P^{WS},\ T^{WS},\ F^{WS})$ be a Workflow System which has to be transformed and $SN = (P^I, T^I, W^I)$ a system net representing the Infrastructure layer of the target elementary object system should be generated using Algorithm 1.

Algorithm 1
(∗ demonstrates infrastructure net construction ∗)
1. $P^I = T^I = W^I \leftarrow \emptyset$
2. for each work-flow specification produce place in the system net, $\forall s \in \widehat{S}$: $P^I = P^I \cup \{p_{name(s)}\}$
3. for every set of the communication transitions with the same name, place one transition to the system net, $\forall \xi(t) \in \chi(T^{WS}) = [\chi(T_i^{WS})]_{i \in <1,\dots,n>}$: $T^I = T^I \cup \{t_{name(\xi(t))}\}$, where $name(\xi(t_i)) = name(\xi(t_j))(i \neq j)$
4. connect all communication transitions to the corresponding places with double-sided arcs, $\forall p_I \in P^I, \forall t_I \in T^I : p_i^I \in {}^\bullet t_i^I \wedge p_i^I \in t_i^{I\bullet}$, where $t^{WS} \in T^{WS} : \forall p_i^{WS} in C_i : p_i^{WS} \in {}^\bullet t^{WS} \vee p_i^{WS} \in t^{WS\bullet}$
5. annotate all arcs with arbitrary names
6. place inscriptions to the transitions that invoke the : *output* up-link in the source node and places the result to the : *input* up-link of all the target nodes

Each node of the system, placed logically within the Infrastructure net place is considered to run on some piece of hardware installed with the PNOS. Because PNOS also consists of the PNVM it is able to interpret Reference Nets translated into the PNBC pseudo-code. Basic layer of the system, that must be installed on all nodes of the system is Platform layer, that brings a set of basic meta-operations that enables the node with other Reference Nets manipulation means - like loading, unloading nets, passing values, etc. This layer is described in Fig. 5. After the Platform layer was installed on the basic PNOS and become interpreted by the PNVM kernel, it is possible to send to it some other nets to define or modify the node behaviour. Basic types of such nets are Processes and Sub-processes of the target system.

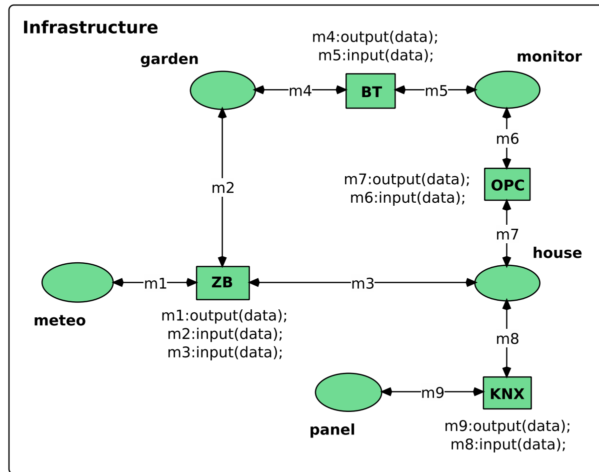

Fig. 4. System Infrastructure net

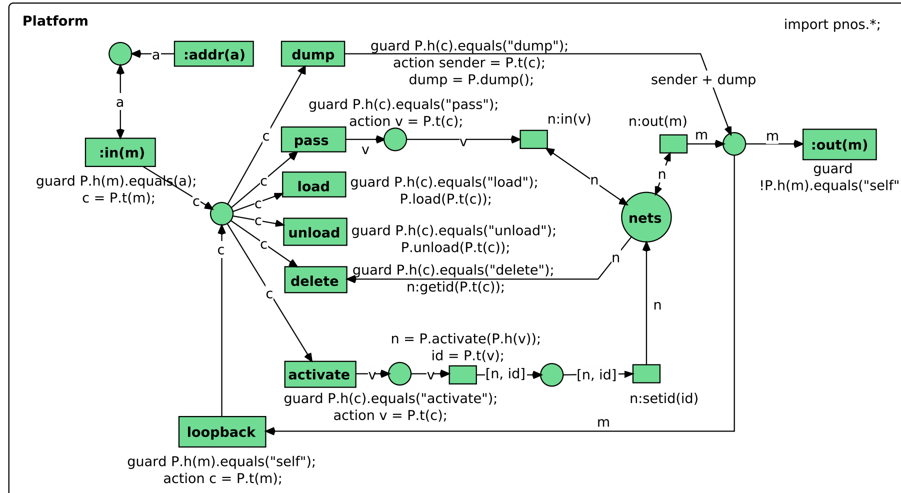

Fig. 5. Platform net

Generating the Process layer The translation of Processes layer also has its own set of production rules. When translating the work-flow model, there is at least one process net generated for each Workflow Specification within the the system model. Main process net consists of the set of meta-operations, that enable the main process to receive and run new nets definitions, and to pass the received values to running subnets. Input place is used for receiving the data by : *input* up-link. Output place serves as an buffer for the : *output* up-link. Nets place then stores all sub-process nets. During the main process life-cycle, each sub-process net is taken from the nets place, it is started, or served with parameters and started. Started net is then put back to nets place, where it resides, until the result is produced. When the result is ready, the net is taken from the temporary place again, the output result is taken, and the net is then stored again back to the nets place, or it could be stopped. The result of the net is then propagated according to the logic specified in the main process net. The example of translating the *garden* node main process net is shown in Fig. 6.

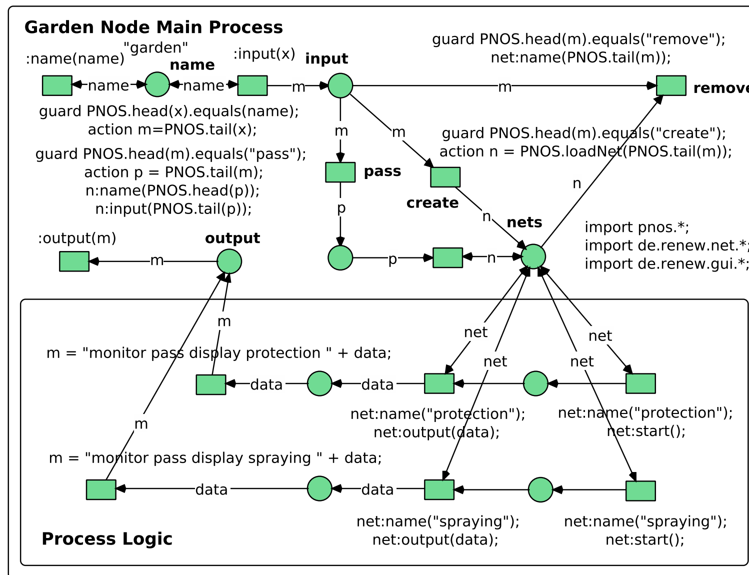

Fig. 6. Garden Main Process net

All the process nets should be produced according to the following rules. Let $S_i = (Q, top, T^{\diamond}, map)$ be a Workflow Specification to be transformed and $ON_i = (P_i^P, T_i^P, W_i^P)$ a net of the Processes layer of the target system. For the translation following Algorithm 2 should be used.

Algorithm 2
(∗ demonstrates process nets construction ∗)

1. $P_i^P = T_i^P = W_i^P \leftarrow \emptyset$
2. add nets, input and output places, $P_i^P = P_i^P \cup \{p_{nets}, p_{in}, p_{out}\}$
3. add the platform meta-operations, $T_i^P = T_i^P \cup \{t_{name}, t_{pass}, t_{create}, t_{remove}\}$
4. for each sub-process in swim-lane construct the first transition that takes the subnet from the nets place and invokes the : *start* up-link and a transition that triggers the : *output* up-link, $\forall t_{S_i} \in T^{top} : T_i^P = T_i^P \cup \{t^P_{i(start)}, t_{i(out)}\}$, where ${}^\bullet t_{i(start)} = p_{nets} = t^\bullet_{i(start)} \wedge {}^\bullet t_{i(out)} = p_{nets} = t^\bullet_{i(out)}$
5. connect both transitions with synchronization place and corresponding arcs, $\forall t^C \in T_i^C$, where $\exists p \in P, t \in T_i \subset T \setminus \bigcup\{T_i\} : p \in t^{C\bullet} \wedge\ p \in t^\bullet : P_i^P = P_i^P \cup \{p_i^P\}$, where $t^{P\bullet}_{i(start)} = p_i^P = {}^\bullet t^P_{i(out)}$
6. add one more place for each output communication to store the results of the sub-process, $P_i^P = P_i^P \cup \{p_i^P\} : t^{P\bullet}_{i(start)} = p_i^P$
7. if the output is to be sent to another node add the transition that constructs the message and puts the resulting message into the output sink, $T_i^P = T_i^P \cup \{t_i^P\} : {}^\bullet t_i^P = p_i^P \wedge\ t_i^{P\bullet} = p_{out}$
8. translate special transitions according to the rules defined by Aalst [1]
9. omit input places
10. copy left places, $\forall c \in C : P_i^P = P_i^P \cup c_c^P$
11. copy left transitions, $\forall t \in T : T_i^P = T_i^P \cup t_t^P$

Generating the Sub-process layer Within the house work-flow model, there is a measure sub-process used in *meteo* and *house* modules. This sub-process should be translated to the Sub-process layer using Algorithm 3.

Algorithm 3
($*$ demonstrates sub-process nets construction $*$)
1. for all sub-process places produce corresponding places, $\forall c \in C : P_i^P = P_i^P \cup c_c^P$
2. for all sub-process transitions produce corresponding transitions, $\forall t \in T : T_i^P = T_i^P \cup t_t^P$
3. translate special transitions according to rules defined by Aalst [1]
4. if there's a loop, switch the do-while-do loop to the while-loop and add the while condition place to the beginning of loop and add the : *stop* transition to enable removing the condition, search for the transitions inscriptions within the dictionary - transition producing the values and transitions consuming the values

Resulting sub-process net is described in Fig. 7.

5.4 From Reference Nets to Petri Nets Byte Code

Following part of the development process comprises of target system code generation. In our approach, each layer of the system should be compiled to target code independently. All generated levels communicate with each other using up-links and down-links.

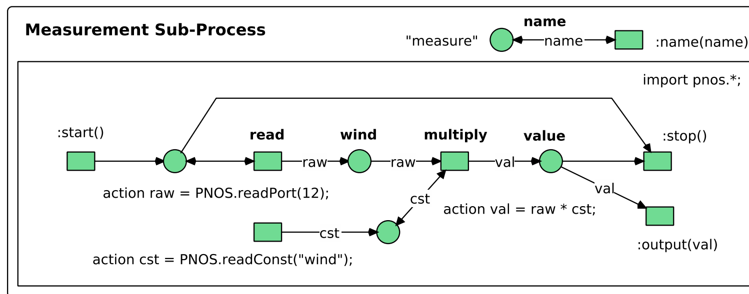

Fig. 7. Measure Sub-process net

The only part of the system, which is implemented natively, is the PNOS kernel, including PNVM [8]. The example of byte-code follows. It represents the measure net (depicted in Fig. 7). In fact, it is a human-readable version of the byte-code. In this representation, numbers are represented as 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/wind)
 (cond/wind/cst/value/name)
 (Ustart()()(P1(B1)(V1)))
 (Ustop()()(O1(B1)(V1)))
 (Uoutput(val)()(P4(B1)(V1)))
 (Uname(name)()(P5(B1)(V1)))
 (I(O5(B1)(S1)))
 (Tread(cond/raw)
  (P1(B1)(V1))
  (A(:(V2)(r(S2))))
  (O2(B1)(V2)))
 (Tconst(cst)
  (A(:(V1)(r(S2))))
  (O2(B1)(V1)))
 (Tmultiply(raw/cst/val)
  (P2(B1)(V1))
  (P3(B1)(V2))
  (A(:(V2)(/(*(V1)(V2))(I10000))))
  (O4(B1)(V2))))
```

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 Fig. 5. The PNOS Platform functionality is described in more detail in [8], [9], [10].

6 Installation and Reconfiguration

The main operating principle of resulting system could be described on the tasks of system construction - installation, and its reconfiguration. The installation of the system starts with placing proper nodes to the target environment. Each node should be installed with the PNOS, PNVM and basic platform layer. The physical communication between nodes using different wired or wireless communication technologies should be established. In our running example the scenario should start with installing the processes for each Workflow Specification and then sending particular sub-processes nets to relevant nodes.

```
meteo load measure-wind
meteo create mw1 measure-wind
meteo load measure-anemo
meteo create ma1 measure-anemo
...
meteo start
meteo pass mw1 start
meteo pass ma1 start
...
```

The other important part of system functionality is its reconfiguration. It should be performed on each defined level of the system architecture. Basically, the node firmware including the PNOS and PNVM could be reprogrammed and rebuilt and then sent over the air to the particular node. The Platform net could be modified and also sent to the particular node, but usually we do not expect this layer to be modified often. The next level of reconfiguration is the processes layer. All processes of the node could be changed and then passed to its platform to change the behaviour of the node. Finally all the sub-processes nets could be modified and sent to particular nodes processes that reinstall them within the nets place. The example of the reconfiguration process follows.

```
meteo pass mw1 stop
meteo destroy mw1
meteo unload measure-wind
meteo load measure-wind
meteo create mw1 measure-wind
meteo pass mw1 start
...
```

There is a plan in future to add the pause and resume operations to the platform, to be able to pause any particular net instance, change its template and resume then. For that it is necessary to invent, how to represent the pausing and resuming conditions in Petri Nets, that is not part of this material.

7 Conclusion

We described the basics of model transformation and execution-based methodology of distributed embedded control system development. Among the main methods it uses Petri Nets models transformations and target system prototype code generation. Development process starts with the work-flow model of the system specification defined according to the rules of Van der Aalst's Workflow Specifications. Work-flow model of the system describes the functionality from user's or domain specialist's point of view. Using our methods, the work-flow model is further transformed to the multi-layered architecture based set of Reference Petri Nets. Each layer of the system is then translated to the specific target representation called Petri Nets ByteCode (PNBC), which is interpreted by the Petri Nets Virtual Machine (PNVM), that is a part of the Petri Nets Operating System (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 its 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.

Acknowledgement

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), p. 245-275.
3. Valk, R. 1998. Petri Nets as Token Objects: An Introduction to Elementary Object Nets. Proceedings of the 19th International Conference on Application and Theory of Petri Nets (ICATPN '98), Jörgen Desel and Manuel Silva (Eds.). Springer-Verlag, London, UK, p. 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. 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, p. 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. Janoušek, V., Kironský, E. 2009. Interactive evolutionary modelling and simulation of discrete-event systems using prototypical objects. International Journal of Autonomic Computing. London: Inderscience Publishers, 2009, Vol. 1, Iss. 2, p. 104-120. ISSN 1741-8569.
8. Richta, T., Janoušek, V. 2013. Operating System for Petri Nets-Specified Reconfigurable Embedded Systems. Proceedings of Computer Aided Systems Theory - EUROCAST 2013, published as LNCS 8111. Berlin Heidelberg: Springer Verlag, 2013, p. 444-451. ISBN 978-3-642-53855-1.
9. Richta, T., Janoušek, V., Kočí, R. 2012. Code Generation For Petri Nets-Specified Reconfigurable Distributed Control Systems, Proceedings of 15th International Conference on Mechatronics - Mechatronika 2012, Prague, CZ, FEL Č, 2012, p. 263-269. ISBN 978-80-01-04985-3.
10. Richta, T., Janoušek, V., Kočí, R. 2013.Petri nets-based development of dynamically reconfigurable embedded systems. In Daniel Moldt and Heiko Rölke, editors, Petri Nets and Software Engineering. International Workshop, PNSE'13, Milano, Italy, June 24-25, 2013. Proceedings, volume 989 of CEUR Workshop Proceedings, pages 203–217. CEUR-WS.org, 2013.
11. Guan, S., U., Lim, S., S. 2004. Modeling adaptable multimedia and self-modifying protocol execution. Future Gener. Comput. Syst., Vol. 20, Iss. 1, p. 123-143.
12. Llorens, M., Oliver, J. 2004. Structural and dynamic changes in concurrent systems: Reconfigurable Petri Nets. IEEE Transactions on Automation Science and Engineering, Vol. 53, Iss. 9, p. 1147-1158.
13. Li, J., Dai, X., Meng, Z. 2009. Automatic reconfiguration of Petri net controllers for reconfigurable manufacturing systems with an improved net rewriting system based approach. IEEE Transactions on Automation Science and Engineering, Vol. 6, Iss. 1, p. 156-167.
14. Wu, N., Q., Zhou, M., C. 2011. Intelligent token Petri nets for modelling and control of reconfigurable automated manufacturing systems with dynamic changes. Transactions of the Institute of Measurement and Control, Vol. 33, Iss. 1, p. 9-29.
15. Almeida, E., E., Luntz, J., E., Tibury, D., M. 2007. Event-condition-action systems for reconfigurable logic control. IEEE Transactions on Automation Science and Engineering, Vol. 4, Iss. 2, p. 167-181.
16. Ohashi, K., Shin, K., G. 2011. Model-based control for reconfigurable manufacturing systems. Proceedings of IEEE International Conference on Robotics and Automation, p. 553-558.
17. Liu, J., Darabi, H. 2004. Control reconfiguration of discrete event systems controllers with partial observation. IEEE Transactions on Systems, Man, and Cybernetics, Part B, Cybernetics, Vol. 34, Iss. 6, p. 2262-2272.
18. Dumitrache, I., Caramihai, S, I., Stanescu, A., M. 2000. Intelligent agent-based control systems in manufacturing. Proceedings of IEEE International Symposium on Intelligent Control, p. 369-374.
19. Oshana, R., Kraelig, M. 2013. Software Engineering for Embedded Systems: Methods, Practical Techniques, and Applications, Newnes.