Object-based Control/Data-flow Analysis

Berndt Farwer[1] **and Mauricio Varea**[2]
[1]farwer@informatik.uni-hamburg.de
[2]mv@ecs.soton.ac.uk

Declarative Systems and Software Engineering Group
Technical Report DSSE-TR-2005-1
March 2005

www.dsse.ecs.soton.ac.uk/techreports/

Department of Electronics and Computer Science
University of Southampton
Highfield, Southampton SO17 1BJ, United Kingdom

Object-based Control/Data-flow Analysis

Berndt Farwer
Fachbereich Informatik,
Universität Hamburg,
D-22527 Hamburg, Germany.
farwer@informatik.uni-hamburg.de

Mauricio Varea
School of Electronics and Computer Science,
University of Southampton,
SO17 1BJ, Southampton, UK.
m.varea@ecs.soton.ac.uk

(22^{nd} March, 2005)

Abstract

Not only does a clear distinction between control and data flow enhance the readability of models, but it also allows different tools to operate on the two distinct parts of the model. This paper shows how the modelling based on control/data-flow analysis can benefit from an object-based approach. We have developed a translation mechanism that is faithful and gives an extra dimension (hierarchy) to the existing paradigm of control and data flow interacting in a model. Our methodology provides a comprehensible separation of these two parts, which can be used to feed another analysis or synthesis tools, while still being able to reason about both parts through formal methods of verification.

1 Introduction

The analysis of systems with a combined control- and data-flow is not trivial. The interaction between these two unique parts usually obfuscates the analysis of such systems. Recently, an approach for modelling with a clear distinction between data and control has been proposed using Scade and Mode-Automata [11]. P/T nets are graphically more intuitive than the Scade models and have precise mathematical semantics. They have a tradition of successful applications in systems modelling [14], but they are not in their basic form capable of managing the additional complexity of analysing the interaction between control- and data-flow. Several extensions to P/T nets have been proposed [9, 8, 15, 1], in order to cope with expressibility and complexity of P/T nets, but they do not explicitly target the control/data-flow interaction problem. The DFN model [19] preserves the structure of a flat P/T net for the control domain, while using a modified semantics for the data domain. In recent research [7, 3, 4], the benefits of object-based modelling has been shown. In particular, object Petri nets (OPN) incorporate the notion of hierarchy by using standard nets as tokens of another net. This paper shows how the idea of control/data separation can

be modelled by an object-based paradigm, such as OPN, and summarises a translation from DFNs to OPNs.

2 Dual Flow Nets

The DFN model [19] assumes a structure where three type of elements (rather than two, as in standard P/T nets) comprise the entire system: *storage*, *reactive*, and *transformational* units. Given a weighted, directed, tripartite graph, its vertices $\mathcal{V} = P \cup T \cup Q$ can be used to map each of these elements as follows[1]: Storage elements ($p \in P$) relate to memory components in the system (e.g., registers, memory cells, latches, variables, etc.), reactive elements ($t \in T$) allude to components in the control part and transformational elements ($q \in Q$) refer to arithmetic operations performed among storage elements (i.e., components in the data path).

Definition 2.1 *A Dual Flow Structure is a seven-tuple* $\mathcal{S} = \langle P, T, Q, F, W, G, H \rangle$,

where: $P = \{p_1, p_2, \cdots, p_n\}$ *is a finite, non-empty set of places;*
$T = \{t_1, t_2, \cdots, t_m\}$ *is a finite set of transitions, with* $m \geqslant 0$*;*
$Q = \{q_1, q_2, \cdots, q_h\}$ *is a finite set of hulls, with* $h \geqslant 0$ *and* $m + h \geqslant 0$*;*
$F \subseteq (P \times T) \cup (T \times P) \cup (P \times Q) \cup (Q \times P) \cup (T \times Q) \cup (Q \times T)$
is a binary relation, called the flow relation*;*
$W : F \mapsto \mathbb{Z}^+ \cup \mathbb{Z}^-$ *is a* weight function*;*
$G : T \mapsto \sharp \cup \{\top\}$ *is a* guard function*,*
where $\sharp = \{=, \neq, >, <, \geqslant, \leqslant\}$*, and* $\top \in \mathbb{B}$*;*
$H : Q \mapsto \mathbb{Z}$ *is an* offset function*.*

Each transition $t \in T$ is either labelled with a symbol from the set $\sharp$ or $\top$, according to the guard function $G(t)$. Hulls $q \in Q$ are labelled with integers, corresponding to the offset function $H(q)$. The guard function $G(t)$ provides a mechanism for data flow to interfere in the control flow, i.e., to have conditional points in the control flow, where the condition is taken from the data flow. The offset function $H(q)$, on the other hand, enhances the functionality of the hull. The basic operation of a hull is to sum over the data domain, so the $H(q)$ function is provided in order to cover those situations where a constant is needed. In order to reduce notational clutter, symbols '$\top$' and numbers '0' (from $G(t)$ and $H(q)$ respectively) are not explicit across the net. Both control and data domains are formalised in Definitions 2.2, where the well known concepts of presets and postsets are extended to support a tripartite structure.

Definition 2.2 *For every* $p \in P$*,* $t \in T$*, and* $q \in Q$*, the following presets and*

[1] where $P \cap T = \varnothing$, $P \cap Q = \varnothing$, $T \cap Q = \varnothing$, $P \neq \varnothing$, and $T \cup Q \neq \varnothing$.

postsets are defined:

$$
\begin{array}{lll}
{}^\bullet p = \{t \in T \mid (t,p) \in F\} & {}^\bullet t = \{p \in P \mid (p,t) \in F\} & {}^\bullet q = \{t \in T \mid (t,q) \in F\} \\
p^\bullet = \{t \in T \mid (p,t) \in F\} & t^\bullet = \{p \in P \mid (t,p) \in F\} & q^\bullet = \{t \in T \mid (q,t) \in F\} \\
{}^\circ p = \{q \in Q \mid (q,p) \in F\} & {}^\circ t = \{q \in Q \mid (q,t) \in F\} & {}^\circ q = \{p \in P \mid (p,q) \in F\} \\
p^\circ = \{q \in Q \mid (p,q) \in F\} & t^\circ = \{q \in Q \mid (t,q) \in F\} & q^\circ = \{p \in P \mid (q,p) \in F\}
\end{array}
$$

where:

$$
\begin{array}{lll}
{}^\bullet P = \bigcup_{\forall p \in P} {}^\bullet p; & {}^\bullet T = \bigcup_{\forall t \in T} {}^\bullet t; & {}^\bullet Q = \bigcup_{\forall q \in Q} {}^\bullet q; \\
P^\bullet = \bigcup_{\forall p \in P} p^\bullet; & T^\bullet = \bigcup_{\forall t \in T} t^\bullet; & Q^\bullet = \bigcup_{\forall q \in Q} q^\bullet; \\
{}^\circ P = \bigcup_{\forall p \in P} {}^\circ p; & {}^\circ T = \bigcup_{\forall t \in T} {}^\circ t; & {}^\circ Q = \bigcup_{\forall q \in Q} {}^\circ q; \\
P^\circ = \bigcup_{\forall p \in P} p^\circ; & T^\circ = \bigcup_{\forall t \in T} t^\circ; & Q^\circ = \bigcup_{\forall q \in Q} q^\circ;
\end{array}
$$

In classic Petri nets tokens are dynamically assigned to places using a marking function. Thus, they work out as indivisible quanta from the control flow. As the DFN model aims at a more complex analysis, where not only control but data information as well is taken into account, its marking function scheme incorporates an indivisible quantum for the data flow in addition to the classic quantum for the control flow. This means that the marking function of a DFN is defined in terms of a tuple, as it can be seen in Definition 2.3.

Definition 2.3 *The DFN marking function is defined as follows*

$$\mu : P \mapsto I\!N \times \mathbb{Z}_n$$

where the first element in the tuple ($\gamma \in I\!N$) is the number of control quanta *that reside inside a place p, while the second element ($\alpha \in \mathbb{Z}_n$) is the number of* data quanta[2]*. The following notation is used, in order to obtain each part in the tuple $\langle \gamma, \alpha \rangle$:*

$$\gamma = |\mu(p)| \qquad \alpha = \angle \mu(p)$$

The behaviour of a DFN model is described in terms of enabling and firing transitions, as in classic Petri nets, in addition to a synchronised data-flow operation scheme. The following two definitions introduce the rules that ensue from modifying the classic enabling and firing rules.

Definition 2.4 *A transition t is said to be* enabled*, for a given marking μ, if the following two conditions are met:*

i. all places in preset $p_i \in {}^\bullet t$ contain at least $W(p_i, t)$ tokens, that is:

$$\bigwedge_{p_i \in {}^\bullet t} (|\mu(p_i)| \geqslant W(p_i, t))$$

[2] $\mathbb{Z}_n$ stands for *"integer modulo 'n' "*

ii. all hulls in the preset $q_j \in {}^\circ t$ give a result that is comparable to 0, according to the Guard function. This is:

$$G(t) \in \sharp \implies \bigwedge_{q_j \in {}^\circ t} \left(\sum_{p_\ell \in {}^\circ q_j} \angle\mu(p_\ell) \cdot W(p_\ell, q_j) + H(q_j) \right) \sharp_g \, 0$$

where $\sharp_g \in \sharp$ is bound to the result of $G(t)$.

Definition 2.4 states whether a transition of a DFN model is enabled or not. The influence of both control and data flow aspects in this evaluation can be observed from the combined form of the enabling condition. Thus, from the control flow point of view, the enabling of a transition depends on the token distribution throughout the DFN model, i.e., subpart (*i.*) of the definition. From the data flow point of view, the dependence is established by the conjunction in subpart (*ii.*), where the data quanta in ${}^\circ q, \forall q \in {}^\circ t$ is used as an argument of the condition stated by $G(t)$. The summation (over ℓ) is further explained in Definition 2.6. An enabled transition *may* fire, following the traditional firing rule approach as described in Definition 2.5.

Definition 2.5 *The firing of an enabled transition t_j changes a marking μ_k into μ_{k+1} by means of the following rules:*

i. $|\mu_{k+1}(p_i)| = |\mu_k(p_i)| - W(p_i, t_j)$, $\forall p_i \in {}^\bullet t_j$

ii. $|\mu_{k+1}(p_i)| = |\mu_k(p_i)| + W(t_j, p_i)$, $\forall p_i \in t_j{}^\bullet$

iii. Each hull $q \in t_j{}^\circ$ is executed (cf. Definition 2.6).

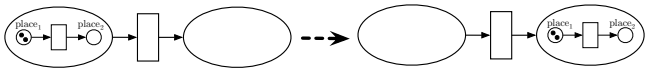
Definition 2.6 shows, in simple terms, that the hull performs a summation of data quanta over the data domain. From the behavioural point of view, the execution of hulls $q \in Q$ are synchronised with transitions $t \in T$ in the net, i.e., no hull q can fire nondeterministically.

Definition 2.6 *The firing of any transition $t \in {}^\bullet q$ produces the* execution *of the hull q, which changes a marking μ_k into μ_{k+1} as follows:*

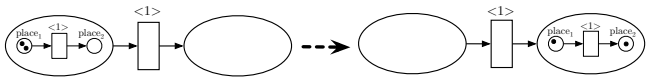
$$\angle\mu_{k+1}(p_j) = W(q, p_j) \cdot \left(\sum_{p_i \in {}^\circ q} \angle\mu_k(p_i) \cdot W(p_i, q) + H(q) \right) \; \forall p_j \in q^\circ$$

3 Object Petri Nets

Object Petri nets are a hierarchical approach to Petri nets. When using nets as tokens of another net, the nets in the lower level of hierarchy are called *object nets* and the net in the higher level is referred to as *system net*. To describe the dynamic behaviour of object Petri nets, a distinction is needed between two fundamentally different kinds of transitions. *Autonomous transitions* can occur in the system net or in an object net and locally change the marking of the



(a) Autonomous transition firing



(b) Synchronised transition firing

Figure 1: Firing modes in object Petri nets

respective net only. *Synchronous transitions* have *synchronisation requirements* preventing them from occurring autonomously.

A simple example of an autonomous system net transition firing is shown in Figure 1(a). Here the object net is simply moved to a different place of the system net. In contrast, a synchronous firing of the system net transition and the object net transition is shown in Figure 1(b). The synchronisation requirement is shown in the Figure 1(b) by the use of the label $<1>$ for the respective transitions. These transitions cannot fire autonomously, and the pair of them are in the synchronisation relation ϱ (which we define formally below).

Another important attribute of object Petri nets is the existence of various semantics that differ mainly in the treatment of the token nets (e.g.,in [2, 17]). *Reference semantics* treat a token net (name) in a place of the system net as a reference to a net instance. Hence, if the same token net (name) appears in different places of the net, they point to the same object net instance. In particular, the marking of each object net instance is applicable for all tokens referring to it. When using *value semantics*, each token net is treated as an individual copy with its own marking.

Object Petri net formalisms have been studied in [12, 16, 13, 2, 6]. Our generic approach in this paper is adequate for all of these formalisms. We use $\wr X \int$ to denote the set of multisets over the set X. An object Petri net consists of a system net (Def. 3.1), a set of object nets (Def. 3.2), and a synchronisation relation (Def. 3.3). For simplicity of the presentation, we only give a formal definition of 2-level OPNs. Allowing arbitrary OPNs as object net tokens leads to multi-level nesting, which our translation to Prolog and toolset can cope with.

Definition 3.1 (system net) *A* system net *is a tuple* $SN = \langle \Sigma, P, T, F, C, V, E \rangle$ *where the following hold:*

(i) Σ *is a set of types (colours) with reflexive and transitive subtype relation* $\sqsubseteq$.

(ii) P *is a set of* system net places *and* T *is the set of* system net transitions *such that* $P \cap T = \varnothing$.

(iii) $F \subseteq (P \times T) \cup (T \times P)$ *is the* flow relation*, also called the* set of arcs*.*

(iv) $C : P \to \Sigma$ *is a total function, called the* typing function *or* colouring function *of the system places.*

(v) V *is a set of typed variable symbols with* $\text{type}(v) \in \Sigma$ *for all* $v \in V$*.*

(vi) $E : F \to \wr V \int$ *is the arc labelling function.*

(vii) The set of variables on the incoming arcs of transition t *is denoted by* V_t *(i.e,* $V_t = \{E((p,t))|(p,t) \in F\}$*) and, for every variable* v *on an outgoing arc it is required that* $v \in V_t$ *holds. Define* $V := \bigcup_{t \in T} V_t$*.*

A transition is *enabled* if there is a binding, assigning to the variables of the arc inscriptions nets from the current marking in the respective places.

For the two-level case we define object nets as P/T nets.

Definition 3.2 (object net) *An* object net $ON = \langle P, T, F, W \rangle$ *is a P/T net, where* P *is a set of places,* T *with* $T \cap P = \varnothing$ *is a set of transitions,* $F \subseteq (P \times T) \cup (T \times P)$ *is the flow relation, and* $W : P \to I\!N$ *is the arc-weight function.*

As with the system net from Definition 3.1 we have omitted the marking from the net structure. Markings are introduced in the same canonical way for OPNs as for ordinary Petri nets. The crucial addition to the system net and the object nets is a synchronisation relation, which is used to allow communication between the different nets.

Definition 3.3 (synchronisation relation) *Let* $SN = \langle \Sigma, P, T, F, C, V, E \rangle$ *be a system net and let* $\{ON_i\}_{i \in I}$ *be a set of object nets* $ON_i = \langle P_i, T_i, F_i, W_i \rangle$ *such that* T *and all* T_i *are disjoint. Let* $\widetilde{T} = T \cup \bigcup_{i \in I} T_i$ *denote the set of all transitions.*

Then a synchronisation relation is a tree-like relation $\varrho \in \widetilde{T} \times \widetilde{T}$*, such that its reflexive and transitive closure* ϱ^* *is asymmetric and* $(t', t) \in \varrho \wedge (t'', t) \in \varrho \;\Rightarrow\; t' = t''$*.*

The intention of the synchronisation relation is, that a transition $t \in \widetilde{T}$ in an object Petri net is enabled, if it is enabled in its net (in the sense of P/T-nets enablement) and the following condition holds: If there is a transition $t' \in \widetilde{T}$ such that $(t, t') \in \varrho$ or $(t', t) \in \varrho$, then t and t' can only fire together synchronously. A transition t that has no partner in ϱ is *autonomously* enabled whenever it is enabled in the usual sense of P/T nets, i.e., whenever all places in ${}^\bullet t$ hold at least the amount of tokens specified by the weight function. [3]

A (2-level) OPN is essentially a system net with an associated set of object net tokens and a synchronisation relation between transitions of the system net and object nets.

[3] The synchronisation relation will usually underly some locality constraints. For instance, in the multi-level case, synchronisations may be restricted to take place only between an ON transition t and a SN transition t', if the ON resides in an input place of t'.

Definition 3.4 (object Petri net) *An* object Petri net *("OPN") is a quadruple* $\langle SN, \{ON_i\}_{i \in I}, I, \varrho \rangle$ *where* SN *is a system net, the* ON_i *are object nets,* I *is a finite indexing set and* ϱ *is a synchronisation relation.*

W.l.o.g. we require that all transition sets (from SN and ON_i) are disjoint. The definitions given above apply to the two level case. Arbitrary levels of nesting can be introduced into the model by allowing the object nets to be OPNs.

In this paper we use a slightly extended OPN formalism, in which we allow the object net to be a simple kind of coloured net. In particular, we use integers as the only types or colours in our object nets. By introducing colours, we also need to introduce expressions over variables as arc inscriptions instead of the arc weights introduced for the P/T nets and we introduce as transition guards simple relations on integers as used in the DFN.

The treatment of object nets depends to a great extent upon the semantic paradigm used. This shows particularly in the events of fork and join transitions. A fork transition in *value semantics* produces multiple copies of the token nets removed from its input places. Each of the copies can evolve independently. Considering *reference semantics*, on the other hand, would produce multiple references to the same net, so that any evolution of the object net is reflected in all 'copies'. Similar effects apply for join transitions.

For the translation carried out in the next section we use reference semantics and use the reference (object) Petri net editor/simulator Renew [10] for our examples. The synchronisation relation translates into synchronous channels attached to the system an object net transitions as up- and down-links, thus the inscriptions change from the general representation in the preceding figures (e.g. $< 1 >$) to invocations of a channel ch from a system net transition to an object net transition in the object net referenced by x denoted by x:ch(). Transitions that can accept the invocation carry the corresponding inscription, for instance :ch().

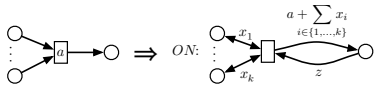
4 Translation

Our translation takes the idea behind the DFN model one step further by not only separating data flow from control flow, but making this even better visible in the model. The visualisation relies solely upon well-known concepts from the theory of Petri nets and the usual firing rule together with a synchronisation relation that was introduced for object-based Petri nets. The data are represented by an object net while the system net represents the control flow. Note that we use a coloured Petri net as the object net in our translation, as mentioned in the previous section.

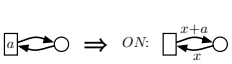
4.1 Procedure

Before giving a formal translation procedure, we sketch the basic idea: Fig. 2 shows the transformation from a DFN to an OPN for pure data transfers (2(a)) and for triggering (2(c)). The latter has to do with the control flow and is achieved by using synchronisation between the system net of the OPN and the

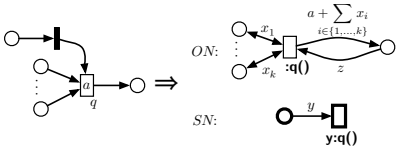
object net that represents the data, i.e. arcs from transitions to hulls in the DFN are represented by synchronisation requirements in the OPN.



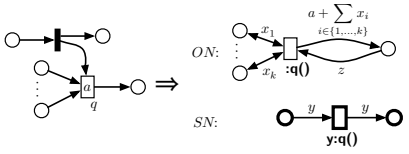
(a) Data translation



(b) Place in both preset and postset



(c) Control/synchronisation with sink



(d) Control/synchronisation

Figure 2: Transating DFNs into OPNs

Hulls in DFNs require a triggering transition to fire. Thus, the sub-nets from Figure 2(a) and 2(b) will never be enabled. They would require an (enabled) trigger such as those of Figures 2(c) and 2(d) to be enabled.

We give a formal translation as Transformation 1 that carries out the transformation in Figure 2. Note that the guards of the DFN's transitions are translated to guards in the object net, since they represent conditions related to values in the data part of the marking.

Transformation 1 Given a dual flow structure $\mathcal{S} = \langle P, T, Q, F, W, G, H\rangle$ with an initial marking μ we construct an equivalent object Petri net $\langle SN, \{ON\}, \{\mathcal{S}\}, \varrho\rangle$ with system net $SN = \langle \bar{\Sigma}, \bar{P}, \bar{T}, \bar{F}, \bar{C}, \bar{V}, \bar{E}\rangle$, object net $ON = \langle \hat{\Sigma}, \hat{P}, \hat{T}, \hat{F}, \hat{C}, \hat{V}, \hat{W}\rangle$, and synchronisation relation $\varrho \in \widetilde{T} \times \widetilde{T}$ defined by:

i. The system net SN is given by:

(a) $\bar{\Sigma} := \{\mathsf{otok}, \{\bullet\}\} = \{\{ON\}\}$, i.e., there is only one type of tokens apart from the black token.

(b) $\bar{P} := \{\bar{p} \,\big|\, p \in ({}^\bullet T \cup T^\bullet)\} \cup \{\mathsf{start}\}$ is the set of control places together with a distinguished start place

(c) $\bar{T} := \{\bar{t} \,\big|\, t \in T\} \cup \{t_{\mathsf{start}}\}$ is the set of control transitions together with a distinguished start transition

(d) $(\bar{a}, \bar{b}) \in \bar{F} :\iff b \in a^\bullet$ specifies the control flow relation, $(\mathsf{start}, t_{\mathsf{start}}) \in \bar{F}$ and $\{(t_{\mathsf{start}}, p) \,\big|\, |\mu(p)| \geq 1\} \subset \bar{F}$

(e) $\forall \bar{p} \in \bar{P} \,.\, \bar{C}(\bar{p}) := \mathsf{otok}$, i.e. all places apart from the initial place

(f) $\bar{V} := \{x\}$

(g) $\forall (a,b) \in F \,.\, \bar{E}(a,b) := \begin{cases} \epsilon & \text{if } a \text{ or } b \text{ is } t_{\mathsf{start}} \\ x & \text{otherwise} \end{cases}$, since apart from the initialisation where only black tokens are involved, we only move the object net representing the data within the system net, so the same reference is involved in any transition (input and output).

ii. The object net ON with the natural numbers as the only colour ($\hat{\Sigma} := \{I\!N\}$) is given by:

(a) $\hat{P} := \{\hat{p} \,\big|\, p \in (Q^\circ \cup {}^\circ Q)\}$, the relevant data places

(b) $\forall \hat{p} \in \hat{P} \,.\, \hat{C}(\hat{p}) := I\!N$, i.e., the data places can hold natural numbers

(c) $\hat{T} := \{\hat{t} \,\big|\, t \in (P^\circ \cap {}^\circ P)\}$, the relevant data transitions, corresponding to those hulls that actually do data manipulations and do not only adjust values for use with guards

(d) $\hat{V} := \{x_p\}_{p \in P} \cup \{z_p\}_{p \in P}$ introduces variables named after the places

(e) $\forall (p,q) \in F \cap (P \times Q)$:

- $\{(\hat{q},\hat{p}),(\hat{p},\hat{q})\} \subseteq: \hat{F}$ restores the data flow relation from places to hulls
- $\hat{W}(\hat{q},\hat{p}) := \hat{W}(\hat{p},\hat{q}) := x_p$

supplies individual variables for the different input arcs

(f) $\forall (q,p) \in F \cap (Q \times P)$:

- $\{(\hat{q},\hat{p}),(\hat{p},\hat{q})\} \subseteq: \hat{F}$ restores the data flow relation from hulls to places
- $\hat{W}(\hat{p},\hat{q}) := \begin{cases} z_p & \text{if } p \notin ({}^\circ q \cap q^\circ) \\ x_p & \text{otherwise} \end{cases}$ ensures that original values are kept in the input places unless the input place is also an output place of the same hull
- $\hat{W}(\hat{q},\hat{p}) := \left(\sum_{r \in {}^\bullet q \cap P} W(r,q) \cdot x_r \right) + H(q)$ calculates the result of the hull operation

(g) $\hat{\mathrm{G}}(t) := \begin{cases} \top & \text{if } \mathrm{G}(t) = \top \\ \bigwedge\limits_{q_j \in {}^\circ t} \left(\sum\limits_{p_\ell \in {}^\circ q_j} \angle \mu(p_\ell) \cdot W(p_\ell, q_j) + H(q_j) \right) \sharp_g 0 & \text{if } G(t) = \sharp_g \in \sharp \end{cases}$

for all $t \in \hat{T}$ sets up the guards according to the transition guards of the DFN and the offsets of the hulls

iii. synchronisation relation:

$$\varrho := \{(\bar{t},\hat{t}') \in \bar{T} \times \hat{T} \,\big|\, (t,t') \in F \cap (T \times Q)\}$$

The initial marking of the OPN is determined by the tokens in the initial marking of the DFN system in such a way that the corresponding object net places have a reference to the object net with appropriate integers in its corresponding places.

Lemma 4.1 *A hull q can be executed iff there is an object net transition that is enabled together with a system net transition that synchronises with it.*

Proof
Hulls in DFNs require a triggering transition to fire. The translation enables the corresponding object net transition only by a synchronisation introduced for the arc between a transition and a hull in the DFN. □

Lemma 4.2 *Data quanta across places of the net changes, following the execution of a hull. This changes are reflected in the marking of the object net of an OPN.*

Proof
The special meaning of the arc-weight function between hulls and places, i.e. a weighted sum, is reflected in the corresponding operation of the object net. The translation also reproduces another particularity of DFNs: a value that is read from a place is never consumed, unless another hull explicitly replaces it for something else. □

Theorem 4.3 *The translation of DFNs into OPNs from Transformation 1 is faithful, i.e., the OPN terminates iff the DFN terminates, and in the case of termination the results are the same.*

Proof
Using Lemmas 4.1 and 4.2, it can be shown that for every firing sequence in the control part and taking into account the data part, there exists a corresponding firing sequence in the system net and the object net, respectively. Together we can state that every firing sequence of the DFN leads to an equivalent firing sequence of the constructed OPN. Furthermore it is clear from the construction that no firing sequences exist for the OPN, for which there is no corresponding firing sequence of the DFN. □

4.2 Example

In this section, we provide an example to clarify the translation methodology. Figure 3 shows a DFN model of the Fibonacci algorithm, where a token and the value of "n" are placed in p_5, and the Fibonacci result "$f(n)$" is expected in p_6, after n iterations. The recursive part of the algorithm is reflected in the cycle containing $\{p_1, p_2, p_3\}$ and its associated transitions and hulls. Place p_4

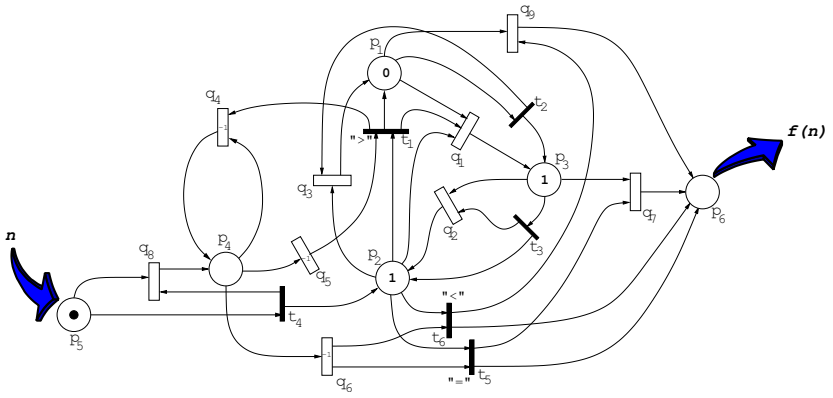
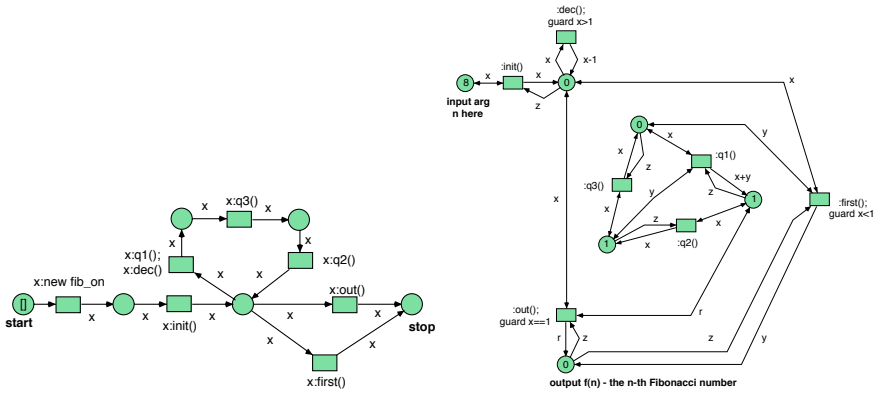


Figure 3: DFN model of Fibonacci



(a) System net (b) Object net

Figure 4: OPN representation of the Fibonacci example

serves as a decremental counter that governs the loop execution. Some additional transitions (t_5 and t_6) and hulls (q_6, q_7 and q_9) take care of both initial conditions, i.e. when $n \in \{0, 1\}$.

Applying the methodology described in Section 4.1, the OPN model shown in Figure 4 is obtained, where the same computation is executed on the input value in the object net. In both models there are three places (in the centre of the respective nets) that represent the cached values of the current value and the previous two values of the Fibonacci function. There are corresponding places for the control flow in the system net.

The arc inscriptions in the OPN are simplified to x instead of x_p whenever there is just one place in the preset of a transition. When there are two places, we use x and y as variables. To simplify the notation, we have also omitted the annotations $(\bar{}, \hat{})$ that were necessary in the formal construction.

In the OPN model the control flow is visualised separately by the system net (Fig. 4(a)). This has some advantages which become apparent when commu-

nicating the model to a person who is familiar with Petri nets but not DFNs, since OPNs use mainly the same firing rule as ordinary Petri nets.

Although control and data flows are also separated in the DFN model, they are still represented on the same level of the net representation. Note that in this example there is just one control token present at any stage of the calculation. This is not a restriction that applies to DFN systems in general so that concurrency is possible in the original model and in its translations into OPNs. The construction given in this paper takes care of the general case and is not restricted in this respect.

5 Verification

Both DFN and OPN models have a well established verification methodologies [18, 5]. In an attempt to corroborate our translation methodology, we have successfully verified the termination and partial correctness of the the Fibonacci computations using both approaches.

Model checking of the DFN required the specification of the net in the SMV model checker input language. We specified the following LTL formulae for termination (eq. 1) and correctness (eq. 2):

$$\Diamond |\mu(p_6)| = 1 \tag{1}$$

$$\Box \left(\bigcirc\bigcirc \angle\mu(p_3) = \bigcirc \angle\mu(p_3) + \angle\mu(p_3) \right) \tag{2}$$

By application of the translation methodology introduced in Section 4.1, we have obtained the OPN representation detailed in Section 4.2. This lead to an automatically generatable Prolog program which, used in conjunction with the XTL model checker, uses a XSB-based tabled Prolog system to infer the correctness of the above formulae.

6 Conclusions

We have developed a translation from dual flow nets to object Petri nets that retains the separation of control flow and data flow. The object net representation has proved to be more accessible to those familiar with Petri nets, even though DFNs are also based on Petri nets. The principle of locality is adhered to in OPNs, while the locality has to be somewhat extended for DFNs.

A major benefit of using OPNs is the tool support we find in the RENEW tool [10], which allows graphical editing and animated execution of reference nets. RENEW is a Petri net editor and simulator that is based on a plug-in architecture allowing easy and flexible addition of functionality. Some basic structural analysis tools have recently been developed for ordinary Petri nets and analysis of workflow nets is currently under development. It should be fairly straightforward to build upon these developments to add functionality for DFNs. The plug-in architecture makes it easily extensible, such that export modules, e.g. for model checkers or other analysis tools, are easy to integrate.

There are some interesting open questions and problems, that should be addressed in further research, some of which we briefly state hereafter:

- How can the use of further level(s) of the OPN be exploited to hide irrelevant information from the data flow and simplify further the verification?
- Define a (greatest possible) class of OPNs that can be translated into DFNs.
- Integration of the DFN formalism into the Renew tool.
  - Import of DFNs with translation to OPNs
  - Inclusion of a DFN mode
  - Export of DFN-like OPNs to SMV for model checking
- Translation to Maude for simulation and model checking

References

[1] L. A. Cortés, P. Eles, and Z. Peng. Verification of Embedded Systems using a Petri Net based Representation. In *Proc. of the 13th International Symposium on System Level Synthesis (ISSS)*, pages 149–155, Madrid, Spain, 20-22 2000.

[2] B. Farwer. Comparing concepts of object Petri net formalisms. *Fundamenta Informaticae*, 47(3–4):247–258, 2001.

[3] B. Farwer, S. Kalvala, and K. Misra. Controller synthesis for object Petri nets. In *Formal Methods and Software Engineering: 5th ICFEM 2003*, pages 432–451. LNCS 2885. Springer-Verlag, 2003.

[4] B. Farwer and M. Köhler. Mobile object-net systems and their processes. *Fundamenta Informaticae*, 60(1–4):113–129, 2004.

[5] B. Farwer and M. Leuschel. Model checking object Petri nets in Prolog. In *Proc. of the 6th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 20–31. ACM Press, 2004.

[6] B. Farwer and K. Misra. Modelling with hierarchical object Petri nets. *Fundamenta Informaticae*, 55(2):129–147, 2003.

[7] B. Farwer, D. Moldt, and F. García-Vallés. An approach to modelling FMS with dynamic object Petri nets. In *Proc. of the 2002 IEEE International Conference on Systems, Man and Cybernetics.*, pages 1–6.

[8] K. Jensen. Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use. *EATCS Monographs in Theoretical Computer Science*, 3(Practical Use), 1997.

[9] K. Jensen and G. Rozenberg, editors. *High-level Petri Nets: Theory and Application*, 1991.

[10] O. Kummer, F. Wienberg, and M. Duvigneau. RENEW – The reference net workshop http://www.renew.de.

[11] O. Labbani, J.-L. Dekeyser, and P. Boulet. Mode-automata based methodology for scade. *Hybrid Systems: Computation and Control: 8th International Workshop, HSCC 2005*, volume 3414 of *LNCS*, pages 386–401. Springer-Verlag, 2005.

[12] C. A. Lakos. Object Petri nets—definition and relationship to coloured nets. Technical report, TR94-3, Computer Science Department, University of Tasmania, 1994.

[13] I. A. Lomazova. Nested Petri nets — a formalism for specification of multi-agent distributed systems. *Proc. Concurrency Specification and Programming (CSP)*, pages 127–140. 1999.

[14] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings IEEE*, 77(4):541–580, 1989.

[15] Z. Peng and K. Kuchcinski. Automated Transformation of Algorithms into Register-Transfer Level Implementations. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, 13(2):150–166, 1994.

[16] R. Valk. Petri nets as token objects. An introduction to elementary object nets. In J. Desel and M. Silva, editors, *Proc. of Applications and Theory of Petri Nets*, volume 1420, pages 1–25. Springer-Verlag, 1998.

[17] R. Valk. Relating different semantics for object Petri nets. Technical report, FBI-HH-B-266/00, Fachbereich Informatik, Universität Hamburg, 2000.

[18] M. Varea, B. M. Al-Hashimi, L. A. Cortés, P. Eles, and Z. Peng. Symbolic Model Checking of Dual Transitions Petri Nets. In *10th International Symposium on Hardware/Software Codesign (CODES)*, pages 43–48, may 2002.

[19] M. Varea, B. M. Al-Hashimi, L. A. Cortés, P. Eles, and Z. Peng. Dual Flow Nets: Modelling the Control/Data-Flow Relationship in Embedded Systems. *ACM Transactions on Embedded Computing Systems*, 2005. (accepted for publication).