SAVE Project Report, Dept. of Computer and Information
Science, Linköping University, Linköping, February 2001.

Hierarchies for the Modeling and Verification of Embedded Systems

Luis Alejandro Cortés, Petru Eles, and Zebo Peng
Department of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden

Abstract

A flat representation of a realistic embedded system can be too big and complex to handle and understand. In order to represent efficiently large systems, a mechanism for hierarchical composition is needed so that the model may be constructed in a structured manner and composed of simpler units easily comprehensible by the designer at each description level. In this report we formally define the notion of hierarchy for a Petri net based representation used for modeling embedded systems. We show how small parts of a large system may be transformed by using the concept of hierarchy as well as the advantages of a transformational approach in the verification of embedded systems. A real-life example illustrates the feasibility of our approach on practical applications. This work has been done in the frame of the SAVE project, which aims to study the specification and verification of heterogeneous electronic systems.

1. Introduction

Embedded systems are typically constituted of heterogeneous components such as microcontrollers, digital signal processors, application specific instruction-set processors, and application specific integrated circuits, among others. Besides their heterogeneity, embedded systems are characterized by their dedicated function, real-time behavior, and high requirements on reliability and correctness [Cam96].

In order to devise systems with such features, the design process must be based upon a formal representation that captures the characteristics of embedded systems. Many computational models have been proposed in the literature to represent embedded systems [Lav99], including extensions to finite-state machines, data-flow graphs, and communicating processes. Particularly, Petri nets (PNs) are an interesting representation for this sort of systems: PNs, for instance, may represent parallel as well as sequential activities and easily capture non-deterministic behaviors. In embedded systems design, PNs have been extended in various ways to fit the most relevant traits of such systems, e.g. notion of time, and we can find several PN-based models with different flavors [Var01], [Mac99], [Sgr99], [Sto94], [Ess98]. We have recently introduced PRES+, a novel representation that extends PNs, in which tokens hold information, transitions perform transformation of data, and timing is captured by associating lower and upper limits to the duration of activities related to transitions [Cor00b].

However, the lack of hierarchical decomposition makes it difficult to specify and understand complex systems modeled as PNs. In this report we present an approach to the hierarchical modeling of embedded systems using PRES+. We formally define the concept of *hierarchical PRES+ model*, introducing *super-transitions* as hierarchical blocks, as well as the notions of *abstraction* and *refinement*. Thus hierarchical modeling can be applied throughout the whole design process of embedded systems. Since realistic systems tend to be complex and complicated, a flat representation may become too large to handle as well as error-prone. Hierarchy is a useful tool that allows the system to be constructed in a structured way by composing a number of fully understandable entities.

For a large class of embedded systems time-to-market is a very important issue. The use of hierarchical modeling during the design phases can help to shorten the time-to-market of embedded applications. Hierarchy permits systems to be designed in a modular way. Thus the system may be set up by reusing existing elements such as IP blocks and therefore reduce its design time.

There have been several approaches to the introduction of hierarchy into Petri nets. The method for stepwise refinement and abstraction of nets presented in [Suz83] is an elegant formulation to cope with the state explosion of PNs by transforming transitions and/or places into subnets and vice versa. Murata [Mur89] proposes a set of transformation rules used to refine and abstract PNs, which preserve liveness, safeness, and boundedness. Valette [Val79] defines the concept of *block,* which is a refinement net with one initial transition and one final transition, to represent divisible and non-instantaneous actions. These approaches, though dealing with the concept of hierarchy through sound formalisms, are not appropriate for embedded systems since the classical PN model lacks essential notions like timing. An important contribution of our work is the definition of hierarchy for a modeling formalism suitable for the design and verification of embedded systems. We define a semantic relation between *super-transitions* and their *refinements*. In our approach timing is explicitly handled in the hierarchy.

We also show how the hierarchical representation supports a transformation based concept and its advantages during the formal verification process. The notion of hierarchy defined in this report is the vehicle through which a portion of the entire system can be transformed. Such a transformational approach allows an important reduction in the verification cost. If a given model is modified using correctness-preserving transformations and then the resulting one is proved correct with respect to its specification, the initial model is guaranteed to be correct by construction and no intermediate steps need to be verified.

The rest of this report is organized as follows. A description of the design representation that we use to model embedded systems is presented in Section 2. The notions of hierarchy and abstraction/refinement are formally defined in Section 3. In Section 4 we illustrate the hierarchical modeling of a real-life application used in acoustic echo cancellation. Section 5 discusses transformations on PRES+ models and their benefits in reducing the verification effort. Finally, some conclusions are drawn in Section 6.

2. The Design Representation

The notation we use to model embedded systems is PRES+ (Petri net based Representation for Embedded Systems). PRES+ extends Petri nets to be used as representation in the design pro-

cess of such systems. When modeling embedded systems, PRES+ overcomes some of the drawbacks of the classical PN model: it captures explicitly timing information; it is more expressive since tokens might carry information; systems may be represented at different levels of granularity. Furthermore, both control and data information may be captured by a unified design representation.

In this section we briefly present, in a rather informal manner, the distinguishing features of PRES+. Figure 1 shows a simple example used to illustrate the main characteristics of this representation. A formal definition of the model can be found in [Cor00b].

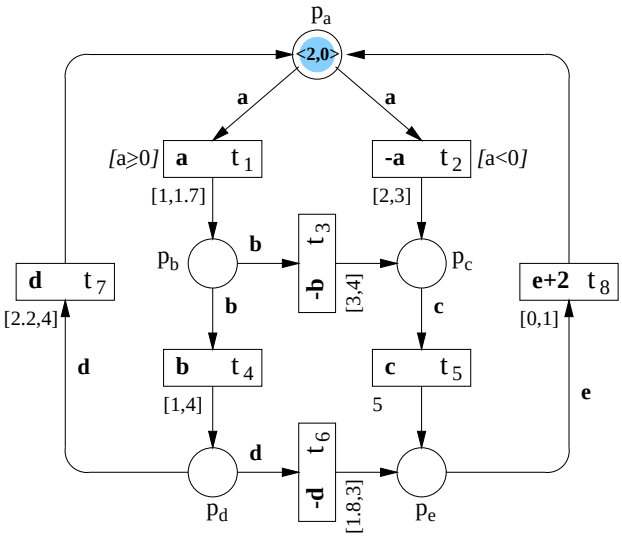

Figure 1. A PRES+ model

A PRES+ model is a five-tuple $N = (P, T, I, O, M_0)$ where P is a set of places, T is a set of transitions, I is a set of input (place-transition) arcs, O is a set of output (transition-place) arcs, and M_0 is the initial marking of the net. A marking is an assignment of tokens to the places of the net. A PRES+ net is 1-bounded[1], that is, a place $p \in P$ may hold at most one token for a certain marking M: $M(p) = 1$ when p is marked, otherwise $M(p) = 0$. A token is a pair $k = \langle v, r \rangle$ where v is the token value—may be of any type—and r is the token time—a non-negative real number. Thus tokens carry data and time information attached to them as stamps. The token type associated to a place p, denoted $\tau(p)$, is the type of value that a token may bear in p. For the initial marking M_0 shown in the model of Figure 1, p_a is the only marked place and its token $k_a = \langle v_a, r_a \rangle$ has token value $v_a = 2$ and token time $r_a = 0$.

Every transition $t \in T$ has one function, called transition function, associated to it. Such a function takes as arguments the token values of tokens in the pre-set of the transition[2]. In Figure 1 we inscribe transition functions inside transition boxes: the function associated to t_8, for example, is given by $f_8(e) = e + 2$ where e is the token value of the token in p_e when marked. We use inscriptions on the input arcs of a transition in order to denote the arguments

1. In order to handle multi-rate systems, the model could be easily extended by allowing unbounded nets. However, its analysis (for instance, formal verification) would become cumbersome. It is a trade-off between expressiveness and analysis power.
2. The pre-set $^\circ t$ of a transition $t \in T$ is the set of input places of t. The post-set t° is the set of output places of t. Correspondingly, the pre-set $^\circ p$ and the post-set p° of a place $p \in P$ are the sets of transitions for which p is output and input place respectively.

of its transition function and/or those of its guard.

A transition $t \in T$ may have a guard, a condition that must be satisfied in order to enable the transition when all its input places hold tokens. The guard of a transition is a function of the token values of tokens in the places of its pre-set. For instance, $a < 0$ represents the guard of t_2. Note that, for the initial marking, t_2 is not enabled even though its only input place is marked and its output place is not.

For every transition $t \in T$, there exist a minimum transition delay d^- and a maximum transition delay d^+. The non-negative real numbers $d^- \leq d^+$ represent the lower and upper bounds for the execution time (delay) of the function associated to the transition. Transition delays give the limits in time for the firing of a transition since it becomes enabled, unless it is disabled by the firing of another transition. Assuming in Figure 1, for instance, that t_1 fires at 1 time units and accordingly the token in p_a is removed and a new token $k_b = \langle 2, 1 \rangle$ is deposited in p_b, then t_3 and t_4 become enabled at 1 time units. Thus t_3 may not fire before 4 time units and must fire before or at 5 time units, unless it becomes disabled by the firing of t_4. When a transition fires, all tokens in its output places get the same token value and token time. The token time represents the instant at which the token was "created". The global time of the system, for a certain marking M, is given by the maximum token time of all tokens in the net.

3. Hierarchy

Embedded systems are complex structures which require models that allow a sound representation throughout their design cycle. PRES+ supports systems modeled at different levels of granularity with transitions representing simple arithmetic operations or complex algorithms. However, in order to handle efficiently the modeling of large systems, a mechanism of hierarchical composition is needed so that the model may be constructed in a structured manner, composing simple units fully understandable by the designer.

Hierarchical modeling can be conveniently applied along the design process of embedded systems. Sometimes the specification or requirements may not be complete or thoroughly understood. In a top-down approach, a designer may define the interface to each component and then gradually refine those components. On the other hand, a system may be constructed reusing existing elements such as IP blocks in a bottom-up approach. A hierarchical PRES+ model can be devised bottom-up, top-down, or by mixing both approaches. In this section we formalize the concept of hierarchy for PRES+ models. Some trivial examples are used in order to illustrate the definitions.

Definition 1. A place $p \in P$ is an *in-port* of the net $N = (P, T, I, O, M_0)$ iff $(t, p) \notin O$ for all $t \in T$. A place $p \in P$ is an *out-port* of N iff $(p, t) \notin I$ for all $t \in T$. ▪

We denote by inP and $outP$ the set of in-ports and out-ports respectively.

Definition 2. A transition $t \in T$ is an *in-transition* of $N = (P, T, I, O, M_0)$ iff

$$\bigcup_{p \in inP} p^\circ = \{t\}$$

A transition $t \in T$ is an *out-transition* of N iff

$$\bigcup_{p \in outP} {}^\circ p = \{t\} \quad \blacksquare$$

Note that the existence of non-empty sets inP and $outP$ is a necessary condition for the existence of in- and out-transitions. For the net N_1 shown in Figure 2, $inP_1 = \{p_a, p_b\}$, $outP_1 = \{p_d\}$, and t_{in} and t_{out} are in-transition and out-transition respectively.

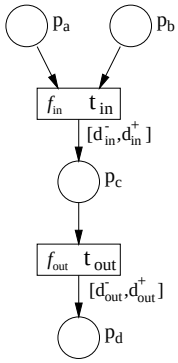

Figure 2. Simple subnet N_1

Definition 3. A *Hierarchical PRES+ Model* is a six-tuple $H = (P, T, \Lambda, I, O, M_0)$ where
$P = \{p_1, p_2, \ldots, p_m\}$ is a finite non-empty set of places;
$T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions;
$\Lambda = \{\boldsymbol{S}_1, \boldsymbol{S}_2, \ldots, \boldsymbol{S}_l\}$ is a finite set of *super-transitions*;
$I \subseteq P \times (\Lambda \cup T)$ is a finite set of input arcs;
$O \subseteq (\Lambda \cup T) \times P$ is a finite set of output arcs;
M_0 is the initial marking. ▪

Observe that a (non-hierarchical) PRES+ net is a particular case of a hierarchical PRES+ net with $\Lambda = \varnothing$. Figure 3 illustrates a hierarchical PRES+ net. Super-transitions are represented by thick-line boxes.

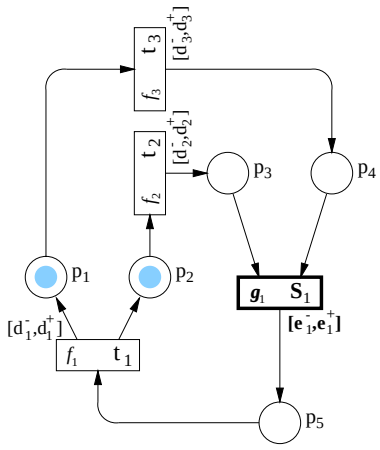

Figure 3. A hierarchical PRES+ model

Definition 4. The *pre-set* ${}^\circ\boldsymbol{S}$ and *post-set* $\boldsymbol{S}^\circ$ of a super-transition $\boldsymbol{S} \in \Lambda$ are given by ${}^\circ\boldsymbol{S} = \{p \in P | (p, \boldsymbol{S}) \in I\}$ and $\boldsymbol{S}^\circ = \{p \in P | (\boldsymbol{S}, p) \in O\}$ respectively. ▪

Similar to transitions, the pre(post)-set of a super-transition $\boldsymbol{S} \in \Lambda$ is the set of input(output) places of $\boldsymbol{S}$.

Definition 5. For every super-transition $\boldsymbol{S} \in \Lambda$ there exists a *high-level function*

$$\boldsymbol{g} : \tau(p_1) \times \tau(p_2) \times \ldots \times \tau(p_a) \rightarrow \tau(q)$$

associated to $\boldsymbol{S}$, where ${}^\circ\boldsymbol{S} = \{p_1, p_2, \ldots, p_a\}$ and $q \in \boldsymbol{S}^\circ$. ▪

Recall that $\tau(p)$ denotes the *type* associated with the place $p \in P$, i.e. the type of value that a token may bear in that place. Observe the usefulness of high-level functions associated to super-transitions in, for instance, a top-down approach: for a certain component of the system, the designer may define its interface and a high-level description of its functionality through a super-transition, and in a later design phase refine the component. In current design methodologies it is also very common to reuse predefined elements such as IP blocks. In such cases, the internal structure of the component is unknown to the designer and therefore the block is best modeled by a super-transition and its high-level function.

Definition 6. For every super-transition $\boldsymbol{S} \in \Lambda$ there exist a *minimum estimated delay* $\boldsymbol{e}^-$ and a *maximum estimated delay* $\boldsymbol{e}^+$, where $\boldsymbol{e}^- \leq \boldsymbol{e}^+$ are non-negative real numbers that represent the estimated lower and upper limits for the execution time of the high-level function associated to $\boldsymbol{S}$. ▪

Definition 7. A super-transition may not be in *conflict* with other transitions or super-transitions, that is:
(i) ${}^\circ\boldsymbol{S}_1 \cap {}^\circ\boldsymbol{S}_2 = \varnothing$ and $\boldsymbol{S}_1^\circ \cap \boldsymbol{S}_2^\circ = \varnothing$ for all $\boldsymbol{S}_1, \boldsymbol{S}_2 \in \Lambda$ such that $\boldsymbol{S}_1 \neq \boldsymbol{S}_2$;
(ii) ${}^\circ\boldsymbol{S} \cap {}^\circ t = \varnothing$ and $\boldsymbol{S}^\circ \cap t^\circ = \varnothing$ for all $\boldsymbol{S} \in \Lambda$, $t \in T$ ▪

In other words, a super-transition may not "share" input places with other transitions/super-transitions, nor output places. In what follows, the input and output places of a super-transition will be called *surrounding* places.

Definition 8. A super-transition $\boldsymbol{S}_i \in \Lambda$ together with its surrounding places in the hierarchical net $H = (P, T, \Lambda, I, O, M_0)$ is a *semi-abstraction* of the (hierarchical) subnet $N_i = (P_i, T_i, \Lambda_i, I_i, O_i, M_{i,0})$ (or conversely, N_i is a *semi-refinement* of $\boldsymbol{S}_i$ and its surrounding places) iff:
(i) There exists a unique in-transition $t_{in} \in T_i$;
(ii) There exists a unique out-transition $t_{out} \in T_i$;
(iii) There exists a bijection $h_{in} : {}^\circ\boldsymbol{S}_i \rightarrow inP_i$ that maps the input places of $\boldsymbol{S}_i$ onto the in-ports of N_i;
(iv) There exists a bijection $h_{out} : \boldsymbol{S}_i^\circ \rightarrow outP_i$ that maps the output places of $\boldsymbol{S}_i$ onto the out-ports of N_i;
(v) $M_0(p) = M_{i,0}(h_{in}(p))$ and $\tau(p) = \tau(h_{in}(p))$ for all $p \in {}^\circ\boldsymbol{S}_i$;
(vi) $M_0(p) = M_{i,0}(h_{out}(p))$ and $\tau(p) = \tau(h_{out}(p))$ for all $p \in \boldsymbol{S}_i^\circ$;
(vii) t is disabled in the initial marking $M_{i,0}$ for all $t \in (T_i - t_{in})$ ▪

Note that in the above definition the concept of super-transition is similar to the notion of *block* defined by Valette [Val79], where the refinement net has one initial transition and one final transition. A subnet may, in turn, contain super-transitions. It is straightforward to prove that the net N_1 of Figure 2 is indeed a semi-refinement of $\boldsymbol{S}_1$ in the hierarchical net of Figure 3.

If a net N_i is the semi-refinement of some super-transition $\boldsymbol{S}_i$, it is possible to characterize N_i in terms of both function and time by putting tokens in its in-ports and then observing the value and time stamp of tokens in its out-ports after a certain firing sequence. If the time stamp

of all tokens deposited in the in-ports of N_i is zero, the token time of tokens obtained in the out-ports is called the *execution time* of N_i. For example, the net N_1 of Figure 2 is characterized by putting tokens $k_a = \langle v_a, 0 \rangle$ and $k_b = \langle v_b, 0 \rangle$ in its in-ports and observing the token $k_d = \langle v_d, r_d \rangle$ after firing t_{in} and t_{out}. Thus the execution time of N_1 is equal to the token time r_d, bounded by $d_{in}^- + d_{out}^- \le r_d \le d_{in}^+ + d_{out}^+$. Note that the token value $v_d = f_{out}(f_{in}(v_a, v_b))$ where f_{in} and f_{out} are the transition functions of t_{in} and t_{out} respectively.

The definition of semi-abstraction/refinement is just "syntactic sugar" that allows a complex design to be constructed in a structured way by composing simpler entities. We have not defined, so far, a semantic relation between the functionality of super-transitions and their refinements. Below we define the concepts of *strong* and *weak refinement* of a super-transition.

Definition 9. A (hierarchical) subnet $N_i = (P_i, T_i, \Lambda_i, I_i, O_i, M_{i,0})$ is a *strong refinement* of the super-transition $\boldsymbol{S}_i \in \Lambda$ together with its surrounding places in the hierarchical net $H = (P, T, \Lambda, I, O, M_0)$ (or $\boldsymbol{S}_i$ and its surrounding places is a *strong abstraction* of N_i) iff:
(i) N_i is a semi-refinement of $\boldsymbol{S}_i$;
(ii) N_i "implements" $\boldsymbol{S}_i$, i.e. N_i and $\boldsymbol{S}_i$ are *function-equivalent*[3];
(iii) The minimum estimated delay $\boldsymbol{e}_i^-$ of $\boldsymbol{S}_i$ is equal to the lower bound of the execution time of N_i;
(iv) The maximum estimated delay $\boldsymbol{e}_i^+$ of $\boldsymbol{S}_i$ is equal to the upper bound of the execution time of N_i. ▪

The net N_1 shown in Figure 2 is a semi-refinement of $\boldsymbol{S}_1$ in the hierarchical net of Figure 3. N_1 is a strong refinement of $\boldsymbol{S}_1$ if, in addition, (Definitions 9(ii), 9(iii), and 9(iv) respectively): (a) $\boldsymbol{g}_1 = f_{out} \circ f_{in}$; (b) $\boldsymbol{e}_i^- = d_{in}^- + d_{out}^-$; (c) $\boldsymbol{e}_i^+ = d_{in}^+ + d_{out}^+$.

Observe that the concept of strong refinement requires the super-transition and its strong refinement to have the very same time limits. Such a concept could have limited practical use since the high-level description and the implementation have typically different timings and therefore their bounds for the execution time do not coincide. We relax the requirement of exact correspondence of lower and upper bounds on time; this yields to a weaker notion of refinement, yet more practical.

Definition 10. A (hierarchical) subnet $N_i = (P_i, T_i, \Lambda_i, I_i, O_i, M_{i,0})$ is a *weak refinement* of the super-transition $\boldsymbol{S}_i \in \Lambda$ together with its surrounding places in the hierarchical net $H = (P, T, \Lambda, I, O, M_0)$ ($\boldsymbol{S}_i$ and its surrounding places is a *weak abstraction* of N_i) iff:
(i) N_i is a semi-refinement of $\boldsymbol{S}_i$;
(ii) N_i "implements" $\boldsymbol{S}_i$, i.e. N_i and $\boldsymbol{S}_i$ are *function-equivalent*;
(iii) The minimum estimated delay $\boldsymbol{e}_i^-$ of $\boldsymbol{S}_i$ is less than or equal to the lower bound of the execution time of N_i;
(iv) The maximum estimated delay $\boldsymbol{e}_i^+$ of $\boldsymbol{S}_i$ is greater than or equal to the upper bound of the execution time of N_i. ▪

In the sequel whenever we refer to *refinement* it will mean *weak refinement*.

3. Several notions of equivalence have been defined for PRES+ models in [Cor00a]. Intuitively, two nets are *function-equivalent* if they perform the same function, that is to say, whenever tokens in corresponding in-ports have the same value, there exists a firing sequence that leads to the same token values in corresponding out-ports.

Given a hierarchical PRES+ net $H = (P, T, \Lambda, I, O, M_0)$ and refinements of its super-transitions, it is possible to construct an equivalent non-hierarchical net. For the sake of clarity, in the following discussion we will consider nets with a single super-transition, nonetheless these concepts can be easily extended to the general case.

Definition 11. Let us consider the hierarchical net $H = (P, T, \Lambda, I, O, M_0)$ where $\Lambda = \{\boldsymbol{S}_1\}$, and let the subnet $N_1 = (P_1, T_1, \Lambda_1, I_1, O_1, M_{1,0})$ be a refinement of $\boldsymbol{S}_1$ and its surrounding places. Let $t_{in}, t_{out} \in T_1$ be unique in-transition and out-transition respectively. Let inP_1 and $outP_1$ be respectively the sets of in-ports and out-ports of N_1. The equivalent net $H' = (P', T', \Lambda', I', O', M_0')$, *one level lower in the hierarchy tree,* is defined as follows:
(i) $\Lambda' = \Lambda_1$
(ii) $P' = P \cup (P_1 - inP_1 - outP_1)$
(iii) $T' = T \cup T_1$
(iv) $(p, \boldsymbol{S}) \in I'$ if $(p, \boldsymbol{S}) \in I_1$;
$(p, t) \in I'$ if $(p, t) \in I$, or
$(p, t) \in I_1$ and $p \notin inP_1$;
$(p, t_{in}) \in I'$ if $(p, \boldsymbol{S}_1) \in I$
(v) $(\boldsymbol{S}, p) \in O'$ if $(\boldsymbol{S}, p) \in O_1$;
$(t, p) \in O'$ if $(t, p) \in O$, or
$(t, p) \in O_1$ and $p \notin outP_1$;
$(t_{out}, p) \in O'$ if $(\boldsymbol{S}_1, p) \in O$
(vi) $M_0'(p) = M_0(p)$ for all $p \in P$;
$M_0'(p) = M_{1,0}(p)$ for all $p \in (P_1 - inP_1 - outP_1)$
■

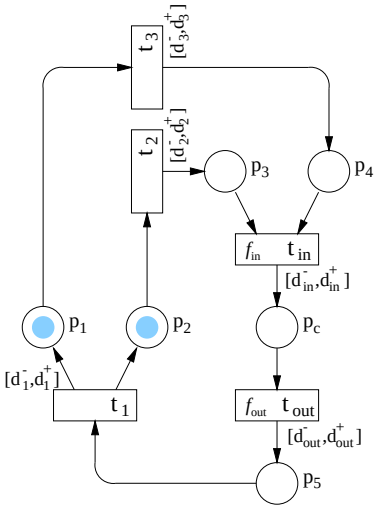

Figure 4. A non-hierarchical PRES+ model

Given the hierarchical net of Figure 3 and being N_1 (Figure 2) a refinement of $\boldsymbol{S}_1$, we can construct the equivalent non-hierarchical net as illustrated in Figure 4.

4. Hierarchical Modeling of a GMDFα

In this section we model a GMDFα (Generalized Multi-Delay frequency-domain Filter)

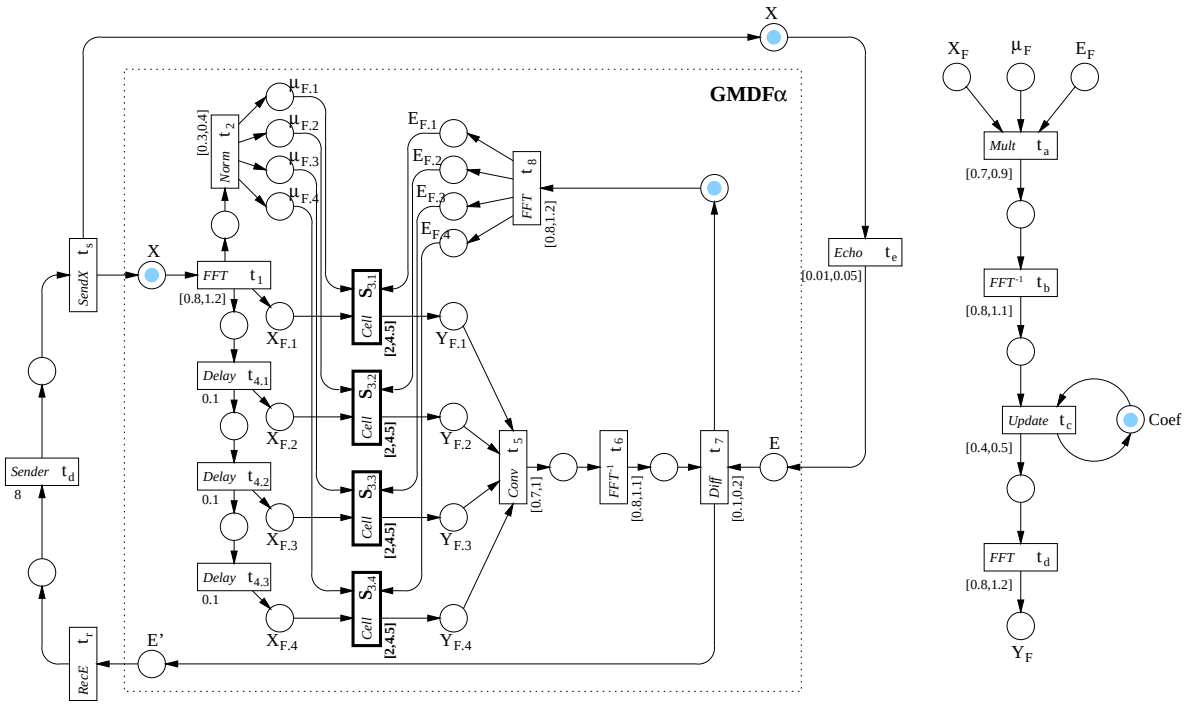

Figure 5. Hierarchical modeling of a GMDFα
(super-transitions are represented by thick-line boxes)

[Fre97], [ElH95] using PRES+. GMDFα has been used in acoustic echo cancellation for improving the quality of hand-free phone and teleconference applications. The GMDFα algorithm is a frequency-domain block adaptive algorithm: a block of input data is processed at one time, producing a block of output data. The impulse response of length L is segmented into K smaller blocks of size N (K=L/N), thus leading to better performance. R new samples are processed at each iteration and the filter is adapted α times per block (R=N/α).

The filter inputs are the signal X and its echo E, and the output is the reduced or cancelled echo E'. In Figure 5 we show the hierarchical PRES+ model of a GMDFα with K=4. The transition t_1 transforms the input signal X into the frequency domain by a FFT (Fast Fourier Transform). t_2 corresponds to the normalization block. In each one of the basic cells $\boldsymbol{S}_{3.i}$ the filter coefficients are updated. Transitions $t_{4.i}$ serve as delay blocks. t_5 computes the estimated echo in the frequency domain by a convolution product and then it is converted into the time domain by t_6. The difference between the estimated echo and the actual one (signal E) is calculated by t_7 and output as E'. Such a cancelled echo is also transformed into the frequency domain by t_8 to be used in the next iteration when updating the filter coefficients. In Figure 5 we also model the environment with which the GMDFα interacts: t_e models the echoing of signal X, t_s and t_r represent, respectively, the sending of the signal and the reception of the cancelled echo, and t_d is the entity that emits X.

The refinement of the basic cells $\boldsymbol{S}_{3.i}$ is shown in Figure 5(b) where the filter coefficients are computed and thus the filter is adapted by using FFT^{-1} and FFT operations. It is worth noticing that instances of the same net (Figure 5(b)) are used as refinements of the different cells $\boldsymbol{S}_{3.i}$. Transition delays as well as estimated delays of super-transitions in Figure 5 are given in milliseconds.

5. Design Verification

For the levels of complexity typical to modern electronic systems, traditional validation techniques like simulation and testing are neither sufficient nor viable to verify their correctness. Formal methods are becoming an alternative to ensure the correctness of designs. In [Cor00b] we have proposed a method to formally verify embedded systems represented in PRES+ by using model checking. There are several types of analysis that can be performed on PRES+ models: the absence/presence of tokens in places of the net, time stamps of such tokens, and their token values. These analyses have been called reachability, time, and functional analysis respectively. Our approach to verification focuses on the first two, that is, reachability and time analyses. We do consider transition functions whenever they affect the marking or time stamps associated to tokens.

Model checking is an approach to formal verification used to determine whether the model of a system satisfies its *specification*, that is, certain required properties. The two inputs to the model checking problem are the system model and the properties that such a system must satisfy, usually expressed as temporal logic formulas. Our approach allows to determine the truth of CTL (Computation Tree Logic) [Cla86] and TCTL (Timed CTL) [Alu90] formulas with respect to a PRES+ model. Formulas in CTL are composed of atomic propositions, boolean connectors, and temporal operators. Temporal operators consist of forward-time operators (**G** globally, **F** in the future, **X** next time, and **U** until) preceded by a path quantifier (**A** all computation paths, and **E** some computation path). TCTL is a real-time extension of CTL that allows to inscribe subscripts on the temporal operators to limit their scope in time. For instance, $\mathbf{AF}_{<\mathbf{n}}\ p$ expresses that, along all computation paths, the property *p* becomes true within n time units.

In [Cor00b] we have proposed a systematic procedure to translate PRES+ models into timed automata. This allows us to use various model checking tools, namely HyTech [HyT], KRONOS [Kro], UPPAAL [Upp], in order to verify properties of embedded systems modeled in PRES+. The verification of hierarchical PRES+ models is done by constructing the equivalent non-hierarchical net as stated in Definition 11, and then using the verification approach discussed in [Cor00b]. Note that obtaining the non-hierarchical PRES+ model can be done automatically so that the designer is not concerned with flattening the net: he inputs to the model checker a hierarchical PRES+ model as well as the properties he is interested in, and then he obtains an answer to the question “does the specification hold in the model of the system?”. In case of a negative answer, diagnostic information is generated in order to explore the cases that make the specification fail.

The verification technique may be improved by transforming the system model into a simpler one, yet semantically equivalent. In the next section we will show how transformations can be conveniently used to simplify the system model for the sake of verification. Thus the verification effort along the design process may be reduced significantly.

5.1. Transformations on PRES+ Models

A flat representation of a real-life embedded system can be too big and complex to handle and understand. The concept of hierarchy defined above allows systems to be modeled in a structured way. Thus, using the notion of abstraction/refinement, the system may be broken down

into a set of comprehensible nets structured in a hierarchy. Each one of these nets may represent a sub-block of the current design. Such a sub-block can be a pre-designed IP component as well as a design alternative corresponding to a subsystem of the system under design.

Transformations performed on large and flat systems are, in general, difficult to prove. Hierarchical modeling permits a structural representation of the system in such a way that the composing (sub)nets are simple enough to be transformed efficiently. The concept of hierarchy defined above is the vehicle through which a portion of the whole system may be transformed. Consider a net $H = (P, T, \Lambda, I, O, M_0)$ and let $\boldsymbol{S} \in \Lambda$ be a super-transition; suppose that N' and N'' are refinements of $\boldsymbol{S}$, that is, both fulfill the conditions stated in Definition 10. If N' and N'' are "equivalent", then either N' or N'' may be used as *the* refinement of $\boldsymbol{S}$ without changing the overall system at all.

Moreover, if (a) N' is a refinement of $\boldsymbol{S}$; (b) there exists a transformation rule that allows to transform N' into N''; (c) the transformation conserves the conditions of refinement (Definition 10) for N''; then such a transformation rule may be stored in a library of transformations and used during verification in order to alleviate the computational effort of the model checking process.

Therefore we can define a set of transformation rules that make it possible to transform only a part of the system model. A simple transformation is shown in Figure 6. We do not intend to provide here a comprehensive set of transformations but rather illustrate the transformation of just a portion of the model taking advantage of the definition of hierarchy for PRES+ models. Assume that N' and N'' are *total-equivalent* in the sense defined in [Cor00a]. The intuitive idea behind *total-equivalence* is as follows (the reader is referred to [Cor00a] for a formal definition): (a) there exist bijections that define one-to-one correspondence between in(out)-ports of N' and N''; (b) having initially tokens with the same token value/time in corresponding in-ports, there exists a firing sequence which leads to the same marking and the very same token values and times in corresponding out-ports. The nets N' and N'' are *total-equivalent* if the above requirements hold. It is not difficult to formally prove that N' and N'' are total-equivalent, provided the conditions given in Figure 6 are satisfied. The most interesting part for this particular transformation is that if N' is a refinement of a given super-transition $\boldsymbol{S}$ (see Definition 10), then N'' is also a refinement of $\boldsymbol{S}$, and consequently such a transformation rule can be used to ease the verification of PRES+ models.

The kind of transformations worth using during verification are those that transform a net into another total-equivalent. Since an external observer could not distinguish between two total-equivalent nets (for the same tokens in corresponding in-ports, the observer would get in both cases the very same tokens in corresponding out-ports), the global system properties remain in terms of reachability, time, and functionality. Therefore such transformations are *correctness-preserving*: if a property *p* holds in a net, it does in the other (a total-equivalent one); if *p* does not hold in the first net, it does not either in the second.

We will illustrate the benefits of using transformations, which are in turn possible due to the concept of hierarchy, in the verification of the GMDFα discussed in Section 4. Considering a filter of length 1024 and an overlapping factor of 4, we have the following parameters: L=1024, α=4, K=4, N=256, and R=64. Having a sampling rate of 8 kHz, the maximum execution time for one iteration is 8 ms (64 new samples must be processed at each iteration). The completion of one iteration is determined by the marking of the place *E'*.

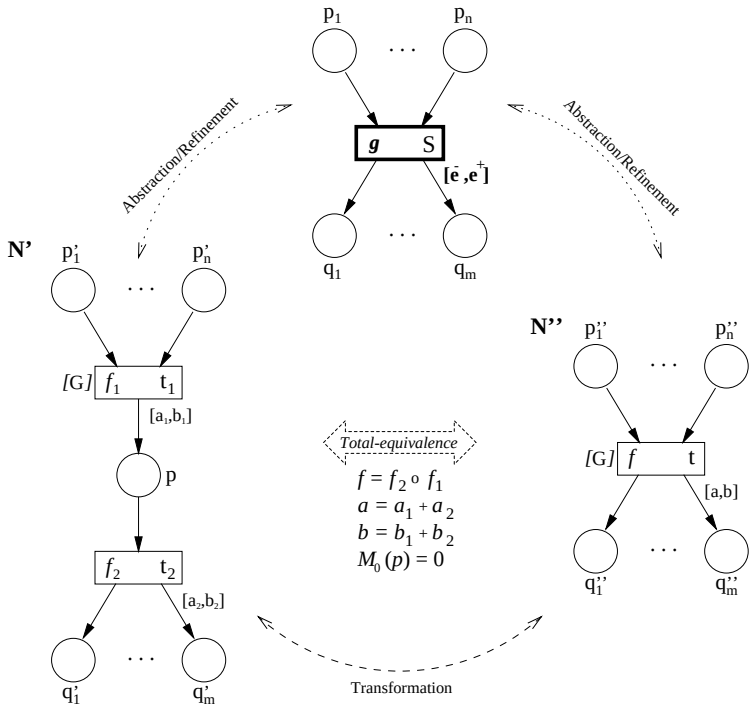

Figure 6. A simple transformation rule

First we want to prove that the system will eventually complete its functionality which can be expressed as a CTL formula $\mathbf{AF}\ E'$. Second, according to the time constraint of the system, it is not sufficient to finish the filtering iteration but also to do so with a bound on time. This aspect of the specification is captured by the TCTL formula $\mathbf{AF}_{<8}\ E'$. At this point, our task is to formally verify that the model of the GMDFα shown in Figure 5 satisfies the formulas $\mathbf{AF}\ E'$ and $\mathbf{AF}_{<8}\ E'$.

A straightforward way could be flattening the system model and applying directly the verification technique discussed in [Cor00b]. However, a wiser approach would be trying to simplify first the system model by transforming it into a total-equivalent one, through transformations from a library already proved to be correctness-preserving. Such transformations are a mathematical tool that allows a significant improvement in the verification cost. The improvement is possible because of a simple observation: the smaller the model is, the lower the verification cost becomes, in terms of both time and memory. Therefore we try to reduce the model aiming to obtain a simpler one, still semantically equivalent, so that the correctness is preserved.

We start by using the transformation rule illustrated in Figure 6 on the refinement of the basic cell, so that we obtain the subnet of Figure 7(b). Note that in this transformation step, no time is spent in proving the transformation itself because it is part of a library. Since these two subnets (Figures 7(a) and 7(b)) are total-equivalent, the functionality of the entire GMDFα remains unchanged. Using other simple transformation rules (not discussed in this report), it is possible to obtain a simpler, still total-equivalent, representation of the basic cell as shown in Figure 7(c). Applying again the transformation rule of Figure 6, the basic cell refinement is further simplified into the single-transition net of Figure 7(d). Finally we check the specification against the simplest model of the system, that is, the one in which the refinement of the basic cells $\boldsymbol{S}_{3.i}$ is the net shown in Figure 7(d). We have verified the above two formulas and such a model of the GMDFα indeed satisfies its specification. The verification using UPPAAL

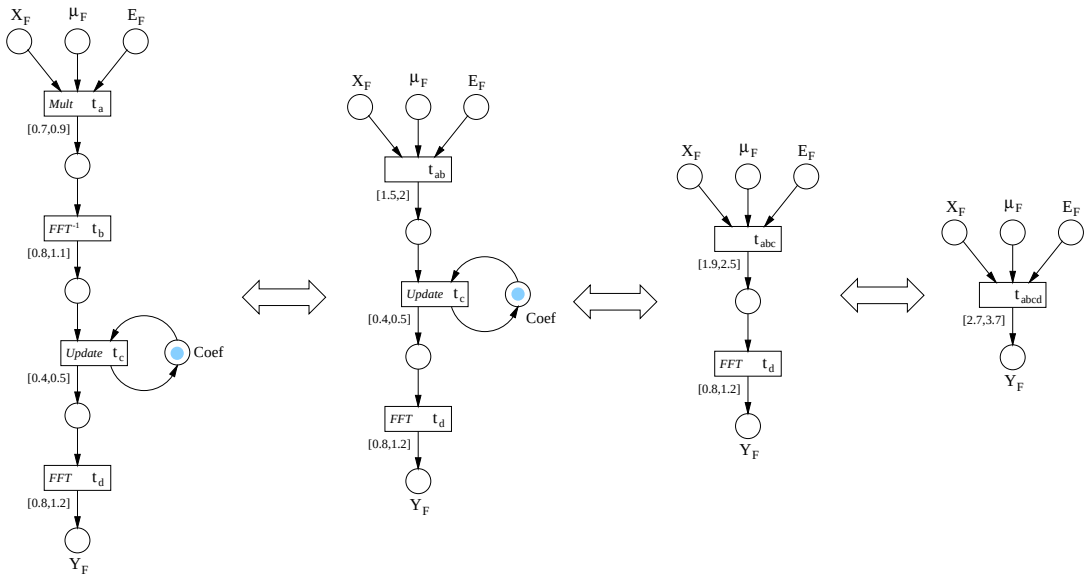

Figure 7. Transformations of the basic cell

[Upp] on a Sun Ultra 10 workstation takes less than 1 second (see last row of Table 1).

Since the transformations used along the simplification of the GMDFα model are total-equivalence transformations, the initial model of Figure 5 is correct by construction, i.e. satisfies the system specification, and therefore need not be verified. However, in order to illustrate the verification cost (time) at different stages, we have verified the intermediate steps (models in which the refinements of the basic cells $\boldsymbol{S}_{3.i}$ are given by the nets shown in Figures 7(b) and 7(c)) as well as the initial model. The results are shown in Table 1. Recall, however, that this is not needed as long as the transformation rules are correctness-preserving. Observe how much effort is saved when the basic cells $\boldsymbol{S}_{3.i}$ are refined by the simplest net compared to the original model.

Table 1: Verification times of the GMDFα (obtained on a Sun Ultra 10 workstation)

Refinement of the basic cell	Verification time [s]		
	HyTech	KRONOS	UPPAAL
Fig. 7(a)	1953	NA†	108
Fig. 7(b)	241	NA†	61
Fig. 7(c)	27	NA†	9
Fig. 7(d)	8	19	<1

†. Not available: out of memory

Thus verification is carried out at low cost (short time) by first using correctness-preserving transformations aiming to simplify the system representation. If the simpler model is correct (its specification holds), the initial one is guaranteed to be correct by construction and intermediate steps need not be verified.

6. Conclusions

We have formally defined the concepts of hierarchy and abstraction/refinement for a Petri net based representation aimed to model embedded systems. In our approach it is feasible to represent large systems as a set of comprehensible nets structured in a hierarchy and, at the same time, the essential characteristics of the system may be captured by the model. Our notion of hierarchy handles explicitly timing.

We showed how hierarchy can be used as the vehicle that permits the transformation of parts (blocks) of the system and we illustrated how such a transformational approach may be extremely useful to reduce the verification cost. By using correctness-preserving transformations, a system model is guaranteed to be correct by construction and verification is performed only on a simplified, still semantically equivalent, one.

A GMDFα (Generalized Multi-Delay frequency-domain Filter) has been studied in order to illustrate the hierarchical modeling of a practical system. This application has also been used during the experiments we carried out to show the worthiness of transformations to reduce the time spent in verification.

References

[Alu90] R. Alur, C. Courcoubetis and D. L. Dill, "Model Checking for Real-Time Systems," in *Proc. Symposium on Logic in Computer Science*, 1990, pp. 414-425.

[Cam96] R. Camposano and J. Wilberg, "Embedded System Design," in *Design Automation for Embedded Systems*, vol. 1, pp. 5-50, Jan. 1996.

[Cla86] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications," in *ACM Trans. on Programming Languages and Systems*, vol. 8, pp. 244-263, April 1986.

[Cor00a] L. A. Cortés, P. Eles, and Z. Peng, "Definitions of Equivalence for Transformational Synthesis of Embedded Systems," in *Proc. ICECCS*, 2000.

[Cor00b] L. A. Cortés, P. Eles, and Z. Peng, "Verification of Embedded Systems using a Petri Net based Representation," in *Proc. ISSS*, 2000.

[ElH95] A. El Helwani and P. Le Scan, "VLSI Architecture of the Generalized Multi Delay Frequency-Domain Algorithm for Acoustic Echo Cancellation," in *Proc. ICASSP*, 1995, pp. 3195-3198.

[Ess98] R. Esser, J. Teich, and L. Thiele, "CodeSign: An embedded system design environment," in *IEE Proc. Computers and Digital Techniques*, vol. 145, pp. 171-180, May 1998.

[Fre97] L. Freund, M. Israel, F. Rousseau, J. M. Bergé, M. Auguin, C. Belleudy, and G. Gogniat, "A Codesign Experiment in Acoustic Echo Cancellation: GMDFα," in *ACM Trans. on Design Automation of Electronic Systems*, vol. 4, pp. 365-383, Oct. 1997.

[HyT] HyTech: The HYbrid TECHnology Tool, `http://www-cad.eecs.berkeley.edu/~tah/HyTech/`

[Kro] KRONOS, `http://www-verimag.imag.fr/TEMPORISE/kronos/`

[Lav99] L. Lavagno, A. Sangiovanni-Vincentelli, and E. Sentovich, "Models of Computation for Embedded System Design," in *System-Level Synthesis*, A. A. Jerraya and J. Mermet, Eds. Dordrecht: Kluwer, 1999, pp. 45-102.

[Mac99] P. Maciel, E. Barros, and W. Rosenstiel, "A Petri Net Model for Hardware/Software Codesign," in *Design Automation for Embedded Systems*, vol. 4, pp. 243-310, Oct. 1999.

[Mur89] T. Murata, "Petri Nets: Analysis and Applications," in *Proc. IEEE*, vol. 77, pp. 541-580, April 1989.

[Sgr99] M. Sgroi, L. Lavagno, Y. Watanabe, and A. Sangiovanni-Vincentelli, "Synthesis of Embedded Software Using Free-Choice Petri Nets," in *Proc. DAC*, 1999, pp. 805-810.

[Sto94] E. Stoy and Z. Peng, "An Integrated Modelling Technique for Hardware/Software Systems," in *Proc. ISCAS*, 1994, pp. 399-402.

[Suz83] I. Suzuki and T. Murata, "A Method for Stepwise Refinement and Abstraction of Petri Nets," in *Journal of Computer and System Sciences*, vol. 27, pp. 51-76, Aug. 1983.

[Upp] UPPAAL, http://www.uppaal.com/
[Val79] R. Valette, “Analysis of Petri Nets by Stepwise Refinement,” in *Journal of Computer and System Sciences*, vol. 18, pp. 35-46, Feb. 1979.
[Var01] M. Varea and B. Al-Hashimi, “Dual Transitions Petri Net based Modelling Technique for Embedded Systems Specification,” in *Proc. DATE Conference*, 2001.