PAPER

MOBnet: An Extended Petri Net Model for the Concurrent Object-Oriented System-Level Synthesis of Multiprocessor Systems

Pao-Ann HSIUNG[†], *Nonmember*, **Trong-Yen LEE**[††], *Member*, and **Sao-Jie CHEN**[††], *Nonmember*

SUMMARY A formal system-level synthesis model for the concurrent object-oriented design of parallel computer systems, called *Multi-token Object-oriented Bi-directional net* (MOBnet), is proposed. The MOBnet model extends the standard Petri net by defining (1) multiple tokens to represent different kinds of synthesis control information, (2) object-oriented nodes (places) to denote the system parts under synthesis, and (3) bi-directional arcs to model the *design completion check* and *synthesis rollback* operations. In this paper, we first show that MOBnet can serve as a pre-fabrication design methodology analysis tool in ways such as class hierarchy construction, design specification comparison, reachability analysis, and concurrent process management and analysis. We then formally prove MOBnet to be a valid model for concurrent synthesis and give experimental application examples to verify. Finally, solution schemes for the design completion check and synthesis rollback problems are formally validated by analyzing the dynamic behavior of MOBnet, and experimentally illustrated through examples.

key words: *concurrent object-oriented system-level synthesis, concurrent synthesis modeling, high-level Petri nets, design completion check, synthesis rollback*

1. Introduction

As proposed by the authors, the system-level synthesis of parallel computer systems has recently become a research topic in the design automation field [1], but theoretical modeling and analysis of the synthesis process is still lacking. Previously proposed models of synthesis process were mainly concerned with the interaction of design tools within CAD frameworks. There are practically little known researches devoted to the theoretical modeling of concurrent synthesis at the system-level of design. This lack of a formal model would be a great obstacle to the progress of system-level design automation researches. This paper takes a pioneer step in proposing such a model, called *Multi-token Object-oriented Bi-directional net* (MOBnet) model.

A brief introduction of related models proposed in CAD frameworks is given as follows. *Version Server* [2] supports a coarse-grain management of chip design data with versions and relationships modeling capability. *Pace* framework [3], [4] is an integration of *Spook* inter-process communication tool, *Ghost* user-interface, and *Dwarf* database system for electronic design automation. *Roadmap* model [5] represents a data-flow based model for CAD frameworks with task definition, flow definition, and tool execution. *Ulysses* system [6] supports a blackboard model using Artificial Intelligence techniques and a dependency graph called the task schema for the design tools execution. *Nelsis* [7], [8] is a CAD framework with design data management and design flow management services, advanced meta-data handling facility, versioning, support for multi-view hierarchical design, and consistency management.

The proposed MOBnet is basically a high-level extension of the standard *Petri net* model [9] and a modified version of the *Colored Petri net* (CP-net) model [10], [11], [12]. Several definition terms are adopted from the CP-nets. Since CP-net is a more general model for system design and analysis, we felt the need to have a modified version devoted specifically to the modeling of the process of concurrent system-level synthesis. Therefore, MOBnet extends the basic Petri net model in the following ways: (1) define multiple tokens to represent different synthesis control information, (2) use object-oriented nodes (places) to represent concurrently synthesizable system parts, and (3) use bi-directional arcs for the modeling of *design completion check* and *synthesis rollback*, the two problems induced by concurrent synthesis.

The paper first describes the concurrent object-oriented synthesis process and our motivating goals in Sect. 2 and then gives a formal definition of the MOBnet model in Sect. 3. Model validation is covered in Sect. 4. The modeling of the solution schemes for the two concurrent synthesis problems, namely, design completion check and synthesis rollback, are depicted in Sect. 5 and Sect. 6, respectively. Section 7 illustrates the use of the model in application examples. Finally, conclusion and future work are given in the last section.

Manuscript received May 9, 1996.
Manuscript revised September 20, 1996.

[†]The author is with the Institute of Information Science, Academia Sinica, Taipei, Taiwan, R.O.C.
[††]The authors are with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, R.O.C.

2. Concurrent Synthesis and Motivating Goals

2.1 Concurrent Object-Oriented System-Level Synthesis

System-level synthesis is defined as the design automation of a parallel computer hardware system satisfying user-given system-level specifications and design-specific constraints. *Object-oriented design* is the encapsulation and modeling of system parts or individual components as *object classes* with design characteristics, class functions, and inter-class relationships. A hierarchy of such component classes can be constructed prior to the synthesis process. When each of the component classes is permitted to be actively participating in its own synthesis, such classes are said to possess a certain degree of self-autonomy and the whole design process is known as *concurrent synthesis.*

An example of concurrent object-oriented system-level synthesis is the methodology given in [13], called the *Object-Oriented Concurrent Synthesis Methodology* (OOCSM). This methodology is an integration of four major techniques: object-oriented modeling, concurrent synthesis, fuzzy logic, and machine learning. As shown in Fig. 1, object-oriented modeling considered here is an extended version of the *Object-Modeling Technique* (OMT) proposed by Rumbaugh et al. [14]. Each system part is modeled as an individual component object; the objects are classified into aggregate nodes (*A-nodes*), generalized nodes (*G-nodes*), and physical or leaf nodes (*P-nodes*), depending on the relationship they have with their child classes. There are three types of relationships known as *aggregation*, *generalization*[14], and *dependence*[13]. *Aggregation* denotes the "whole/part" relationship in which classes of components are "part-of" the class representing the "whole" assembly. *Gener-*

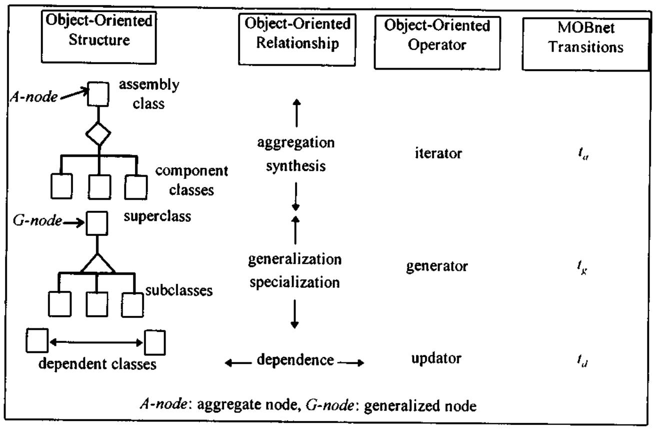


Fig. 1 Object-oriented relationships and operators.

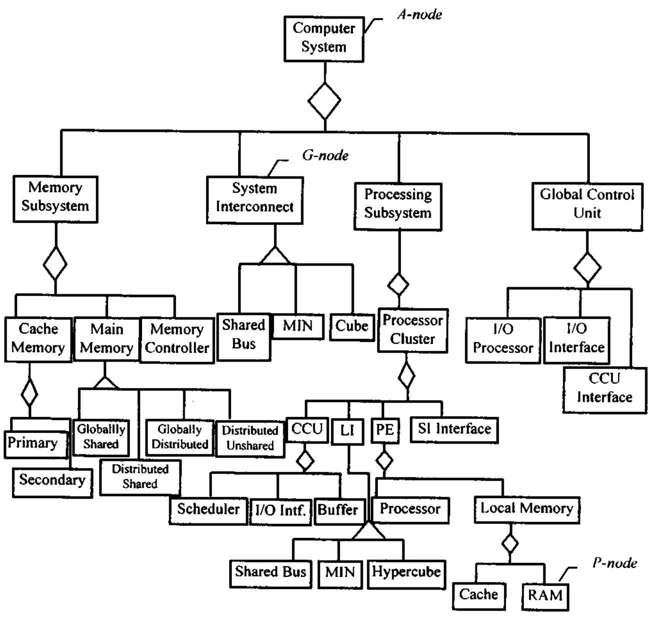


Fig. 2 Class hierarchy.

alization is the relationship between a class and its one or more refined versions. When two component classes are inter-related with respect to their design characteristics, they are said to have a *dependence* relationship.

A *Class Hierarchy* is shown in Fig. 2 for illustration. A component class is synthesized by traversing down the hierarchy of object classes and selecting the proper synthesis action (object-oriented operator) according to the object-oriented relationship between a parent class and one of its child classes. As described in OOCSM, the object-oriented operators used during the synthesis process are *iterator*, *generator*, and *updator*, which correspond to the *aggregation*, *generalization*, and *dependence* relationships, respectively.

2.2 Motivating Goals

Concurrent object-oriented synthesis induces two control-related problems, namely, identifying the completion point of the concurrent processes and clearing the deadlock-like situation caused by unsatisfiable specifications, which are solved by the design completion check and synthesis rollback mechanisms modeled by MOBnet in Sect. 5 and Sect. 6, respectively. In addition, MOBnet is also useful as a pre-fabrication design methodology analysis tool in the following ways:

(i) *Class Hierarchy Construction*: The choice of imposing a dependence relationship between two component classes impacts both the validity of the final synthesized design as well as the design process itself. Prior to the actual design, MOBnets can compare the effects before and after imposing any such dependence relationship using the two cases in the proof of Theorem 1 (Sect. 4).

(ii) *Design Specification Comparison*: Often two design alternatives may differ very little in their specifications, thus affecting only a small portion of the whole design. In such cases, MOBnets can be used to compare the differential effect of their specifications on the final design, by virtually synthesizing them using the guard functions and arc expressions of MOBnet in Eq. (11).

(iii) *Reachability Analysis*: It is quite time-consuming and resource-wasting to reach a stage where the almost-completed synthesis process has to be rolled back. Such undesirable situations can be avoided by using the reachability analysis mechanism provided by MOBnets (Theorem 2).

(iv) *Concurrent Process Management and Analysis*: In contrast to a serial design scheme, a concurrent one requires an overall analytic framework, within which process management can be discussed and analyzed. MOBnet is one such framework as each design thread starting from the root place represents a component synthesis process. Interactions among the threads or processes is accomplished through the *dependence* relationship and dependence checking mechanism as detailed in Case 2 of Theorem 1.

3. Multi-Token Object-Oriented Bi-Directional Net Model

Petri net is an effective modeling tool for the description and analysis of concurrency and synchronization in parallel systems [15]. A standard Petri net (PN) is defined as a 3-tuple, (P, T, A), where P is a set of places, T is a set of transitions, and $A \subseteq P \times T \cup T \times P$, the set of arcs. The PN depicted in Fig. 3 comprises five places and five transitions. The state of a Petri net is depicted by places having tokens and such a state is called a *marking*. A transition is *enabled* when all of its input places contain at least one token. An enabled transition can *fire* by removing one token from each input place and placing one token in each output place. Each firing of a transition modifies the distribution of tokens on places and thus produces a new marking for the PN.

Natural concurrency and synchronization in Petri nets can be used to accurately model the concurrency in the synthesis process. This is one of the main reasons for using Petri nets as our basic modeling approach, but Petri nets, in its standard form, is quite limited in its modeling capabilities. Though there are two high-level Petri nets in popular and wide use nowadays, namely, Predicate/Transition nets (PT-nets) and Colored Petri nets (CP-nets), yet both are too general to be applied directly in modeling system-level synthesis. We provide a modified version of Colored Petri nets, called *Multi-token Object-oriented Bi-directional net* (MOBnet).

MOBnet is a high-level Petri net, where places denote component classes, transitions represent synthesis actions, arcs specify the hierarchical relationships among classes, and tokens provide control information related to synthesis, dependence, design completion, and rollback. An example of MOBnet is shown in Fig. 4. Places in MOBnet are classified into three classes: p_a, p_g, and p_l corresponding to the aggregate,

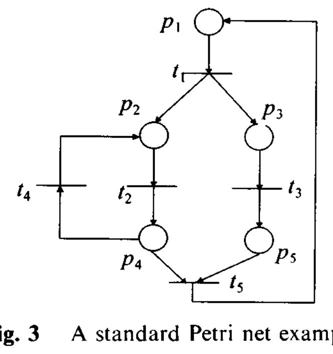
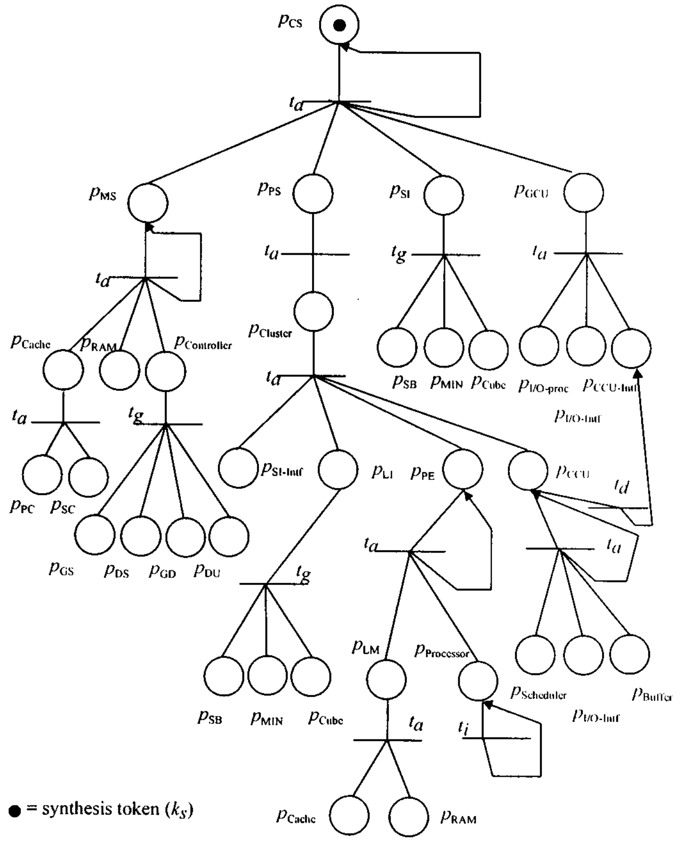


Fig. 3 A standard Petri net example.



Note: For ease of illustration, the arc from each transition to its parent place has been omitted, except for p_{CS}, p_{MS}, p_{PE}, p_{CCU}, and the instantiation transition connected to each leaf place with its surrounding arcs have also been omitted, except for $p_{Processor}$, and only one dependence has been illustrated in this example (t_d between p_{CCU} and $p_{CCU\text{-}Intf}$).

Fig. 4 A MOBnet example.

generalized, and leaf (physical) component classes, respectively. MOBnet transitions are classified into four types: t_a, t_g, and t_d corresponding to the aggregation, generalization, and dependence relationships among the component objects, respectively, and t_i is used for the instantiation and the possible synthesis rollback of leaf places. Arcs in MOBnets connected to t_a or t_g are bi-directional in nature comprising the top-down synthesis direction (the forward direction) and the bottom-up completion checking direction (the reverse direction); arcs connected to t_d are also bi-directional in nature comprising the dependence checking direction (the forward direction) and the dependence response direction (the reverse direction). There is an additional arc from each transition to its parent place, which is used for synthesis rollback. Four types of tokens are defined, namely, synthesis (k_s), dependence (k_d), completion (k_c), and rollback tokens (k_r). A synthesis token enables synthesis transitions (t_a and t_g). A dependence token is used for checking the synthesis status of the class (place), which has a dependence relationship (transition) with the class (place) containing a synthesis token. A completion token notifies parent classes (places) of the completion of child class syntheses, and a rollback token requests parent or dependent classes (places) to relax design specifications so that the synthesis of a child or a dependent class (place) is possible. Note that synthesis tokens traverse only in the forward (synthesis) arc direction, completion tokens and rollback tokens traverse only in the reverse arc direction, and dependence tokens traverse in either arc directions.

The above informal presentation gives a brief idea of what MOBnet is and how it models synthesis. The rest of this section will give a more formal definition of MOBnet (Definition 2) and some related definitions.

Definition 1 (Multi-Set): A **multi-set** m, over a non-empty and finite set S, is a function $m \in [S \to \mathcal{N}]$, where $\mathcal{N}$ is the set of non-negative integers and $w(s) \in \mathcal{N}$ is the number of appearances of the element $s \in S$ in the multi-set m. The multi-set m can be represented by a formal sum: $\sum_{s \in S} w(s)s$, and let S_{MS} denote the set of all multi-sets over S. For example, given $S = \{a, b, c, d, e\}$, $m_1 = a+2c+e$ and $m_2 = a+2b+3c+e$ are both members of S_{MS}. □

Some other notations used are: the type of a variable v is denoted by Type (v) and the set of variables in an expression *expr* is denoted by Var (*expr*).

Definition 2 (MOBnet): A **MOBnet** is a tuple $MOBN = (K, P, T, A, N, C, G, E, F, S)$ where:

(i) K is a finite set of tokens, $K = \{k_s, k_c, k_d, k_r\}$.

(ii) P is a finite set of places, $P \in P'_{MS}$, where $P' = \{p_a, p_g, p_l\}$.

(iii) T is a finite set of transitions, $T \in T'_{MS}$, where $T' = \{t_a, t_g, t_d, t_i\}$.

(iv) A is a set of arcs, $A \subseteqq P \times T \cup T \times P$ and $\forall (a_1, a_2) \in A \Rightarrow (a_2, a_1) \in A$.

(v) N is a node function, $N : A \rightarrow P$, i.e., each arc is mapped to its connected place.

(vi) C is a token function, $C : P \rightarrow K_{MS}$, mapping each place to the multi-set of tokens that are allowed at that place.

(vii) G is a guard function, $G : T \rightarrow \{true, false\}$ and Var($G(t)$) $\in K_{MS}$ for any $t \in T$.

(viii) E is an arc expression function. It is defined from A into expressions such that: $\forall a \in A : [\text{Type}(\text{Var}(E(a))) \in C(N(a))]$.

(ix) F is a direction function, $F : A \rightarrow D$, where $D = \{d_f, d_r\}$ and d_f and d_r are the forward and reverse directions, respectively. Let $a \in A$, $a = (a_1, a_2)$, if a_1 is a parent of a_2 in the MOBnet hierarchy or ($a_1 = N(a)$ and $a_2 = t_d$), then $F(a) = d_f$, else if a_1 is a child of a_2 or ($a_1 = t_d$ and $a_2 = N(a)$), then $F(a) = d_r$.

(x) S is a status function, $S : P \rightarrow \{-1, 0, 1\}$, where $\forall p \in P$,

$$S(p) = \begin{cases} -1, & \text{if synthesis not yet started} \\ 0, & \text{if still under synthesis} \\ 1, & \text{if synthesis completed} \end{cases}$$ □

Besides specializing K, P, and T from CP-nets, MOBnet introduces two new functions, the direction function, F, and the status function, S, for modeling synthesis actions and component status, respectively. An example of MOBnet is shown graphically in Fig. 4 with $P = 9p_a + 3p_g + 23p_l$ and $T = 9t_a + 3t_g + 23t_i + 1t_d$.

Definition 3 (Binding Type and Bindings): For a transition $t \in T$ with variables Var($G(t)$) $= \{v_1, v_2, \ldots, v_n\}$, the **binding type** $BT(t)$ is defined as: $BT(t)$ = Type(v_1) $\times$ Type(v_2) $\times \ldots \times$ Type(v_n). The set of all **bindings** $B(t)$ is defined as: $B(t) = \left\{ \begin{array}{l} (c_1, c_2, \ldots, c_n) \in BT(t) \mid \\ G(t)\langle v_1 = c_1, v_2 = c_2, \ldots, v_n = c_n \rangle \end{array} \right\}$, where $G(t)\langle v_1 = c_1, v_2 = c_2, \ldots, v_n = c_n \rangle$ denotes the evaluation of the guard expressions $G(t)$ in the binding $\langle c_1, c_2, \ldots, c_n \rangle$. □

For example, if $G(t) = xk_s + yk_d$, $x = 1$, $y \geqq 0$, where x, y are non-negative integers and r is an arc connected to $t \in T$, then (1, 0) is a binding.

Definition 4 (Token Distribution): A **token distribution** is a function M, defined on P such that $M(p) \in C(p)_{MS}$ for all $p \in P$. A **binding distribution** is a function Y, defined on T such that $Y(t) \in B(t)_{MS}$ for all $t \in T$. A **marking** of a MOBnet is a token distribution and a **step** is a *non-empty* binding distribution. For example, the initial marking of MOBnet is $M_{IN} : M(p_{CS}) = \{k_s\}$. □

For any two markings, M_1 and M_2, relations $\mp$ and $\leqq$ are defined as follows:

(i) $M_1 \mp M_2$ iff $\exists p \in P : M_1(p) \mp M_2(p)$.

(ii) $M_1 \leqq M_2$ iff $\forall p \in P : M_1(p) \leqq M_2(p)$.

The other relations $<$, $\geqq$, $>$, and $=$ are defined analogously to $\leqq$.

Definition 5 (Step Enabling): A step Y is **enabled** in a marking M iff the following property is satisfied:

$$\forall p \in P : \left[\sum_{(t,b) \in Y} E(p,t)\langle b \rangle \leqq M(p) \right] \quad (1)$$

where the relation $\leqq$ is defined analogously to that for M. □

For example, if $p = p_{\text{CCU}}$ in Fig. 4 and $M(p_{\text{CCU}}) = k_s + 5k_d$, $E(p_{\text{CCU}}, t_a) = xk_s + yk_d$, $x = 1$, $y \geqq 1$ (ref. Eq. 10), then $Y(t_a) = (1,1), (1,2), \ldots, (1,5)$ are all enabled in $M(p_{\text{CCU}})$.

Definition 6 (Direct Reachability): When a step Y is enabled in a marking M_1 it may **occur**, thus changing the marking M_1 to another marking M_2 defined by:

$$\forall p \in P : \left[M_2(p) = \left\{ \begin{array}{l} (M_1(p) - \sum_{(t,b) \in Y} E(p,t)\langle b \rangle) \\ + \sum_{(t,b) \in Y} E(t,p)\langle b \rangle \end{array} \right\} \right] \quad (2)$$

M_2 is said to be **directly reachable** from M_1 by the occurrence of the step Y, which can be denoted by $M_1 Y M_2$. For example, referring to Fig. 4, the marking M_2 (a k_s in each of p_{MS}, p_{PS}, p_{SI}, and p_{GCU}) is directly reachable from M_1 (a k_s in p_{CS}). □

Definition 7 (Finite Occurrence Sequence): A **finite occurrence sequence** is a sequence of markings and steps: $M_1 Y_1 M_2 Y_2 \ldots M_n Y_n M_{n+1}$, such that $n \in \mathcal{N}$, and $M_i Y_i M_{i+1}$ for all $i \in \mathcal{N}^+$, the set of all positive integers. □

Definition 8 (Reachability): A marking M_2 is **reachable** from a marking M_1 iff there exists a finite occurrence sequence having M_1 as start marking and M_2 as end marking, i.e., iff there exists for some $n \in \mathcal{N}$ a finite sequence of steps such that: $M_1 Y_1 Y_2 \ldots Y_n M_2$. □

For example, let $M_1 = M_{IN}$, the initial marking, through a sequence of steps $Y_1 = Y_2 = (1, 0)$, the marking $M(p_{\text{SB}}) = k_s$ is reached, where p_{SB} is the place representing `Shared Bus`.

4. Model Validation

Having known the static structure of the MOBnet model, its dynamic behavior must be investigated in order to prove that it is a valid model for synthesis. This section shows how MOBnet models the dynamic behavior of concurrent object-oriented synthesis.

As described in [16], conceptual models can be validated using five different analysis techniques: face validity analysis, historical analysis, intended use and requirements analysis, model concepts and fidelity analysis, and logic trace analysis. The MOBnet model, being a conceptual model of the synthesis process, needs to be validated before any further use. We base our validation of the MOBnet model on two techniques: the *Model Derivative Analysis*, a kind of historical analysis, and the *Requirements Analysis*.

Definition 9 (Model Derivative Analysis): **Model Derivative Analysis** is the process where the information from any assessments of a previous model, upon which the current model is based, is reviewed, changes between the two models compared, and areas needing reassessments determined. □

Definition 10 (Requirements Analysis): **Requirements Analysis** involves review of the intended use to derive critical requirements for the effective use of resources. This method familiarizes the analyst with the capabilities of the model, verifies that the requirements of the intended use are supported by the model, and provides a mechanism to focus the efforts during the application phase on the critical portions of the model. This method of analysis involves criticality analysis and system analysis. □

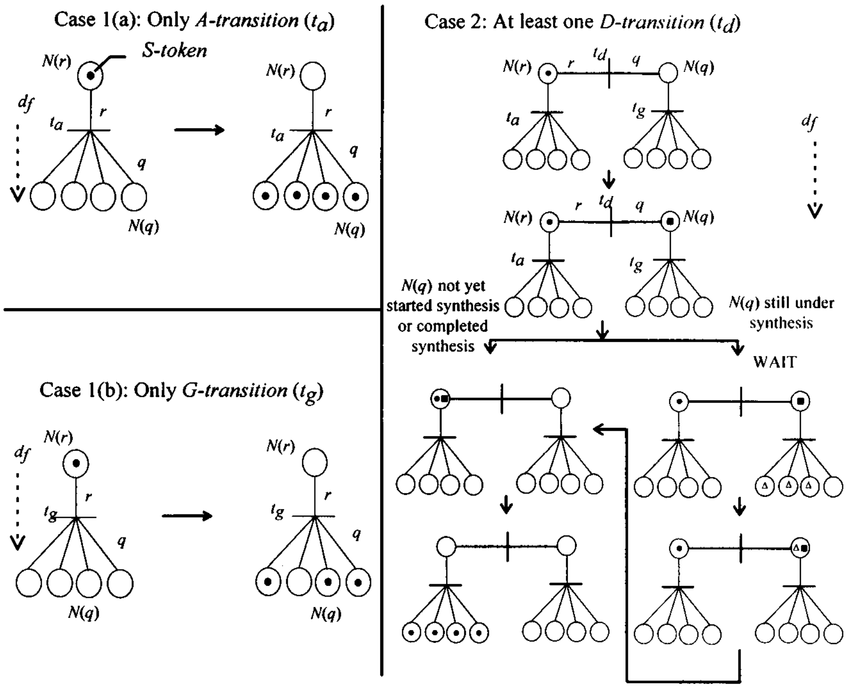
MOBnet is a modified version of the Colored Petri nets, therefore based on Model Derivative Analysis, we are allowed to use all the results of analysis performed on the CP-nets, which are unaffected by our modifications. The analysis specific to our modifications is performed using the method of Requirements Analysis.

Definition 11 (Model Validity): A synthesis model is said to be a **valid model** if it can correctly and consistently model all synthesis actions, their concurrency, and the synchronization among the actions. □

The following theorem shows that the MOBnet model is a valid one.

Theorem 1: The MOBnet model is a valid model for concurrent object-oriented synthesis.

Proof: There are mainly two synthesis operators in an object-oriented synthesis environment, namely, iterator and generator [1], [13], corresponding to the aggregation and generalization relationships between objects [14]. The actions of these operators are modeled by the transitions, t_a and t_g, respectively. The concurrency of synthesis actions is modeled by the concurrent enabling and firing of the corresponding transitions. As shown in Fig. 5, two cases are distinguished: synthesis without dependence and synthesis with dependence. In the proof below, unless otherwise specified, r and q are


Case 1(a): Only *A-transition* (t_a)
S-token
N(r)
df
t_a
r
q
N(q)
Case 1(b): Only *G-transition* (t_g)
t_g
Case 2: At least one *D-transition* (t_d)
t_d
t_g
N(q) not yet started synthesis or completed synthesis
N(q) still under synthesis
WAIT

• = *synthesis token* ▪ = *dependence token* ▵ = *completion token*
N(*r*) = *aggregate node* *N*(*q*) = *generalized node*

Fig. 5 MOBnet: Model validation.

arcs in A such that $F(r) = d_f$, $F(q) = d_r$, and $N(r)$ and $N(q)$ are the places connected hierarchically above and below the transition under consideration, respectively.
Case 1: *Synthesis without dependence*
The synthesis-specific guard functions for $t \in \{t_a, t_g, t_i\}$ are defined as follows:

$$G_1(t) : [M(N(r)) = k_s \wedge F(r) = d_f] \quad (3)$$

where r is an arc in A connecting a place, $N(r)$, to t in the forward (synthesis) direction.

As seen from the above guard functions, $t \in \{t_a, t_g, t_i\}$ is enabled as soon as there is a synthesis token in the place connected hierarchically above t. Hence, more than one transition may be enabled simultaneously for modeling the concurrent nature of synthesis.

The synthesis actions are modeled through the arc expressions as follows:

$$\left.\begin{array}{lcll} E_1(N(r), t_a) & = & k_s & \text{and} \\ E_1(t_a, N(q)) & = & k_s & \forall \text{ child place } N(q); \\ E_1(N(r), t_g) & = & k_s & \text{and} \\ E_1(t_g, N(q)) & = & k_s & \text{if } N(q) \text{ selected;} \\ E_1(N(r), t_i) & = & k_s & \text{and} \\ E_1(t_i, N(r)) & = & k_c & \text{if } N(r) \text{ instantiated;} \end{array}\right. \quad (4)$$

The above arc expressions mean that when a transition $t \in \{t_a, t_g, t_i\}$ is fired, the synthesis token is removed from $N(r)$ and a synthesis token added to each $N(q)$ connected to t_a, to only those $N(q)$ connected to t_g that have been selected for further synthesis by the design methodology, and a completion token added to the leaf place $N(r)$ if it can be instantiated under the given specifications.
Case 2: *Synthesis with dependence*
MOBnet uses the dependence transition, t_d, to model the dependence relationship between objects. Whenever a place, $N(r)$, connected to a t_d transition, receives a synthesis token, k_s, it must first verify whether its dependence on other object classes will affect its own synthesis. This verification comprises two firings of t_d: a check-request firing ($d1$) and a check-response firing ($d2$). The check-request firing of t_d occurs as soon as the place $N(r)$ receives a k_s token.

$$G_{d1}(t_d) : [M(N(r)) = k_s] \quad (5)$$

On the first firing of t_d, the synthesis token k_s is not removed, but a dependence token, k_d, is added to all the places $N(q)$, which are connected to t_d and have a dependence relationship with $N(r)$.

$$E_{d1}(N(r), t_d) = 0 \quad \text{and} \quad E_{d1}(t_d, N(q)) = k_d \quad (6)$$

The second firing of t_d occurs when the place, $N(q)$, receives a dependence token, k_d, (as a result of the first firing of t_d) and if the synthesis status of $N(q)$ is -1 (synthesis not yet started) or 1 (synthesis completed).

$$G_{d2}(t_d) : \left[\begin{array}{l} M(N(r)) = k_s \wedge M(N(q)) = k_d \\ \wedge S(N(q)) \neq 0 \end{array}\right] \quad (7)$$

On the check-response firing, the dependence token, k_d, is removed from $N(q)$ and a dependence token, k_d, added to $N(r)$.

$$E_{d2}(N(q), t_d) = k_d \quad \text{and} \quad E_{d2}(t_d, N(r)) = k_d \quad (8)$$

The synthesis-related transition (t_a, t_g, or t_i) connected to the place $N(r)$ is fired only when $N(r)$ has a synthesis token and x dependence tokens, where x is the degree of dependence of $N(r)$ (i.e., the number of places that $N(r)$ depends on), and $x \geqq 1$.

$$G_2(t) : [M(N(r)) = k_s + xk_d \wedge F(r) = d_f], x \geqq 1 \quad (9)$$

$$E_2(N(r), t) = k_s + xk_d, x \geqq 1 \quad (10)$$

Hence, in general, summarizing the above two cases, Eq. (3) to Eq. (10) can be combined into the following Eq. (11):

$$\left.\begin{array}{l} \forall x \geqq 0, t \in \{t_a, t_g, t_i\}, \\ \quad G(t) : [M(N(r)) = k_s + xk_d \wedge (F(r) = d_f)]; \\ G(t_d) : \ [M(N(r)) = k_s] \vee [M(N(r)) = k_s \\ \qquad \wedge M(N(q)) = k_d \wedge S(N(q)) \neq 0]; \\ \quad E_s(N(r), t) = k_s + xk_d, x \geqq 0 \quad \text{and} \\ \quad E_s(t, N(q)) = k_s; \\ \quad E_{d1}(N(r), t_d) = 0 \text{ and} \\ \quad \forall\ N(q) \text{ dependent with } N(r), \\ \quad E_{d1}(t_d, N(q)) = k_d; \\ \quad E_{d2}(N(q), t_d) = k_d \text{ and } E_{d2}(t_d, N(r)) = k_d \end{array}\right. \quad (11)$$

In summary, the concurrency of synthesis actions in a concurrent object-oriented design environment can be modeled by the concurrent enabling of guard functions and firing of transitions in MOBnets (Eqs. (3) and (5)). Object-oriented synthesis actions, such as iteration, generation, and instantiation, are modeled by the arc expressions in Eq. (4). Synchronization among the synthesis actions is modeled by the check-request firings (Eqs. (5) and (6)) and the check-response firings (Eqs. (7) and (8)) of MOBnets. From the above discussion, we conclude that MOBnet can accurately model all concurrent synthesis actions, their concurrency, and their synchronization, which makes MOBnet a valid model by Definition 11. □

5. Modeling Design Completion Check

Concurrent design poses two problems that must be handled by a synthesis kernel, and thus needs modeling and analysis. One of the problems is the determination of exactly when the design will be completed and the other is the probable necessity for rollback of synthesis steps. Since system parts undergo synthesis concurrently, it becomes quite difficult to grasp the exact synthesis status of the full system under design. One solution is for the individual system part to send an acknowledgment of completion to its parent class once it has undergone complete synthesis. Once the root of the hierarchy receives a message of design completion, we are sure that the design is complete, otherwise there exist some unsatisfiable specifications that have hindered design completion. This blocking of design completion will be dealt with in Sect. 6.

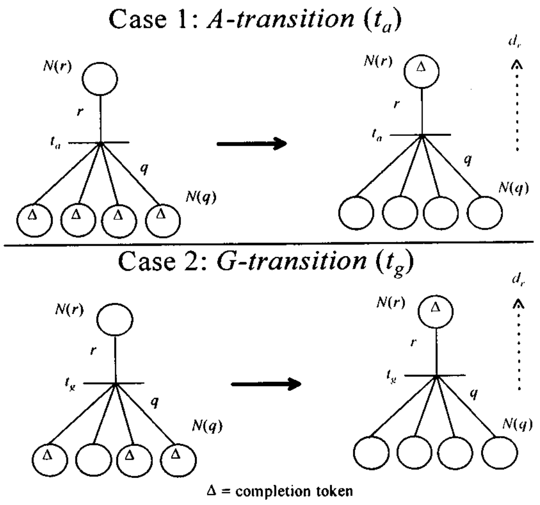
As shown in Fig. 6, design completion check is modeled using completion tokens in MOBnet. This checking process propagates in the direction exactly opposite to that of synthesis. Synthesis is top-down whereas completion check is bottom-up. As soon as a leaf (physical) place has undergone object instantiation (firing of t_i), a completion token is added to it. An *aggregation* transition, t_a, is fired in the reverse (bottom-up) direction as soon as there is a completion token in each of the child places connected to t_a.

$$G_c(t_a) : [\forall q, F(q) = d_r \wedge M(N(q)) = k_c] \tag{12}$$

A generalization transition, t_g, is fired in the reverse (bottom-up) direction as soon as there is a completion token in at least one child place connected to t_g.

$$G_c(t_g) : [\exists q, F(q) = d_r \wedge M(N(q)) = k_c] \tag{13}$$

For the t_i transition connected by an arc r to a leaf



Case 1: *A-transition* (t_a)
$N(r)$
r
t_a
q
$N(q)$
d_r
Case 2: *G-transition* (t_g)
t_g
Δ = completion token

Fig. 6 MOBnet: Modeling design completion check.

place, $N(r)$, if the specifications of $N(r)$ can be met (i.e., meet(spec($N(r)$))), then t_i will be fired.

$$G_c(t_i) : [\text{meet}(\text{spec}(N(r)))] \tag{14}$$

On firing of a transition $t \in \{t_a, t_g\}$ in the reverse direction, all completion tokens are removed from child places and one added to the parent place.

$$\begin{aligned} &\forall q, F(q) = d_r, E(N(q), t) = k_c \quad \text{and} \\ &E(t, N(r)) = k_c, \quad \forall t \in \{t_a, t_g\} \end{aligned} \tag{15}$$

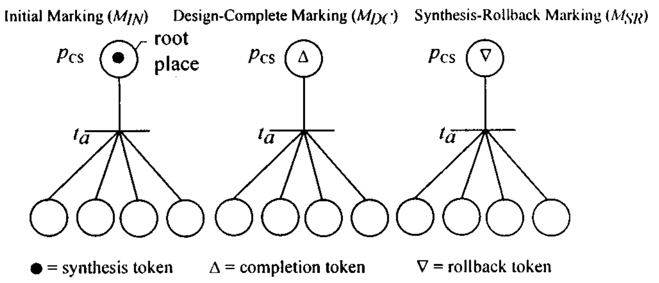
Three special markings of MOBnet: the initial marking, M_{IN} ($M(p_{CS}) = k_s$), the design-complete marking, $M_{DC}(M(p_{CS}) = k_c)$, and the synthesis-rollback marking, $M_{SR}(M(p_{CS}) = k_r)$, are shown in Fig. 7, where p_{CS} is the root place representing the `Computer System` (CS).

Lemma 1: There exists a marking $M(p) = k_c$, for some $p \in P$, that is unreachable from M_{IN} iff M_{DC} is unreachable from M_{IN}.

Proof: Suppose $\exists p \in P$ such that the marking $M(p) = k_c$ is unreachable from M_{IN}. Since $M(p)$ is unreachable from M_{IN}, the corresponding guard function (Eq. (12), (13), or (14)) of the transition $t \in \{t_a, t_g, t_i\}$ connected hierarchically below p is never enabled in any **step** $Y(t)$. Hence, by Eq. (15), no completion token, k_c, is placed in any ancestor place of p, this means that the guard functions (Eqs. (12)–(14)) of all transitions connected hierarchically above p will never be enabled, among which one of the transitions is connected to the root place, p_{CS}. Thus, there does not exist any step in which the transition connected below p_{CS} is enabled, hence, M_{DC} is unreachable.

Now suppose M_{DC} is unreachable from M_{IN}, that is, the guard function (Eq. (12)) of the transition, t_a, connected below p_{CS}, cannot be enabled in any step $Y(t_a)$. That means, there exists at least one child place, p, connected to t_a which does not own a completion token, k_c. This, in turn, means depending on the transition, t_a, t_g, or t_i connected below that child place, the transition's guard function (Eqs. (12)–(14)) cannot be enabled. Hence, there exists at least one place, p, such that $M(p)$ is unreachable from M_{IN}. □

Theorem 2: M_{DC}, is reachable from M_{IN}, iff the design can be completed under the given specifications and constraints.



Initial Marking (M_{IN})
Design-Complete Marking (M_{DC})
Synthesis-Rollback Marking (M_{SR})
p_{CS}
root place
t_a
● = synthesis token
Δ = completion token
∇ = rollback token

Fig. 7 MOBnet Markings: M_{IN}, M_{DC}, M_{SR}.

Proof: We prove by contradiction. Suppose M_{DC} is reachable from M_{IN} and the design cannot be completed under the given specifications and constraints, which means there exists at least one physical object, represented by a leaf place, p, whose specifications cannot be satisfied. Hence, the marking $M(p) = k_c$ is unreachable from M_{IN}. From Lemma 1, if $M(p)$ is unreachable, then M_{DC} is also unreachable, but this contradicts the statement that M_{DC} is reachable from M_{IN}. Hence, our assumption is wrong, there is no such unsynthesizable leaf place, p. Now, suppose the design can be completed, but M_{DC} is not reachable from M_{IN}. From Lemma 1, since M_{DC} is not reachable, there exists a leaf place, p, such that $M(p)$ is not reachable. Hence, there is at least one physical object, represented by p, that cannot be synthesized; this means the design cannot be completed, in contradiction with our assumption. □

6. Modeling Synthesis Rollback

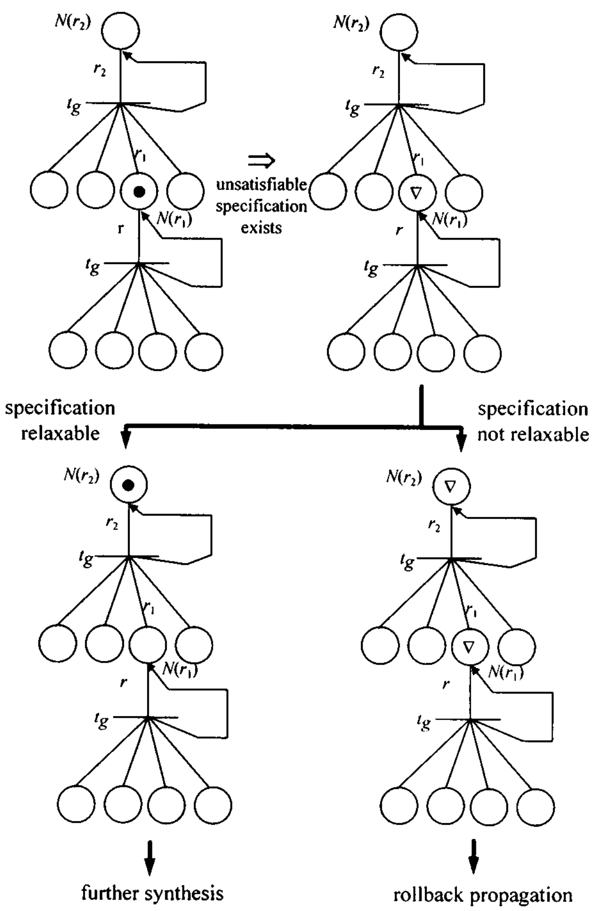
When M_{DC} is not reachable from M_{IN}, the synthesis process is *blocked* due to the existence of object places with unsatisfiable specifications such that synthesis cannot be completed. To overcome such a *deadlock-like* situation, a rollback of synthesis actions, called *synthesis rollback*, is performed starting at the unsynthesizable object place, and propagating in the reverse (bottom-up) direction until a place with relaxable specifications is encountered.

As illustrated in Fig. 8, MOBnet models synthesis rollback using the rollback token whose function is similar to the *antimessages* used in the simulation time-warp mechanism [17]. As mentioned in Sect. 3, there is an arc from each transition to its parent place, which is used for synthesis rollback. When a place $N(r)$ is found unsynthesizable, the connected synthesis transition $t \in \{t_a, t_g, t_i\}$ is enabled (Eq. (16)) and fires, thus removing the synthesis token and adding a rollback token to itself (Eq. (17)).

$$G_r(t) : \begin{bmatrix} & (M(N(r)) = k_s + xk_d) \\ \wedge & (\neg\text{meet}(\text{specs}(N(r)))) \end{bmatrix} \tag{16}$$

$$\begin{aligned} &E_r(N(r), t) = k_s + xk_d \quad \text{and} \\ &E_r(t, N(r)) = k_r \end{aligned} \tag{17}$$

Rollback is propagated upwards as long as the unsatisfiable specifications are not relaxed. Suppose a place $N(r_1)$ has a rollback token which contains an unsatisfiable specification inherited from $N(r_2)$. If $N(r_1)$ cannot relax the specification, then the transition $t \in \{t_a, t_g, t_d\}$ connected between $N(r_1)$ and $N(r_2)$ is fired in the reverse direction and a rollback token placed in $N(r_2)$, thus propagating rollback information. If $N(r_2)$ can relax the specification, then t is fired and a synthesis token placed in $N(r_2)$, thus allowing the resynthesis of $N(r_2)$ and all concerned child places.



● = synthesis token Δ = completion token ∇ = rollback token

Fig. 8 MOBnet: Modeling synthesis rollback.

$$\begin{aligned} &\forall t \in \{t_a, t_g, t_d\}, \\ &\quad G_r(t) : [\exists r_1, (F(r_1) = d_r \wedge M(N(r_1)) = k_r)] \end{aligned} \tag{18}$$

$$\begin{aligned} &E_r(N(r_1), t) = k_r \quad \text{and} \\ &E_r(t, N(r_2)) = \begin{cases} k_r, & \text{if spec. of } N(r_2) \\ & \text{is not relaxable} \\ k_s, & \text{otherwise.} \end{cases} \end{aligned} \tag{19}$$

Synthesis rollback of the complete system becomes necessary if the marking, M_{SR}, is reachable from the initial marking, M_{IN}. When such a situation occurs, the designer of the system is requested to relax the system specifications if possible.

Inter-relating design completion check (Sect. 5) and synthesis rollback (Sect. 6), we have the following theorem.

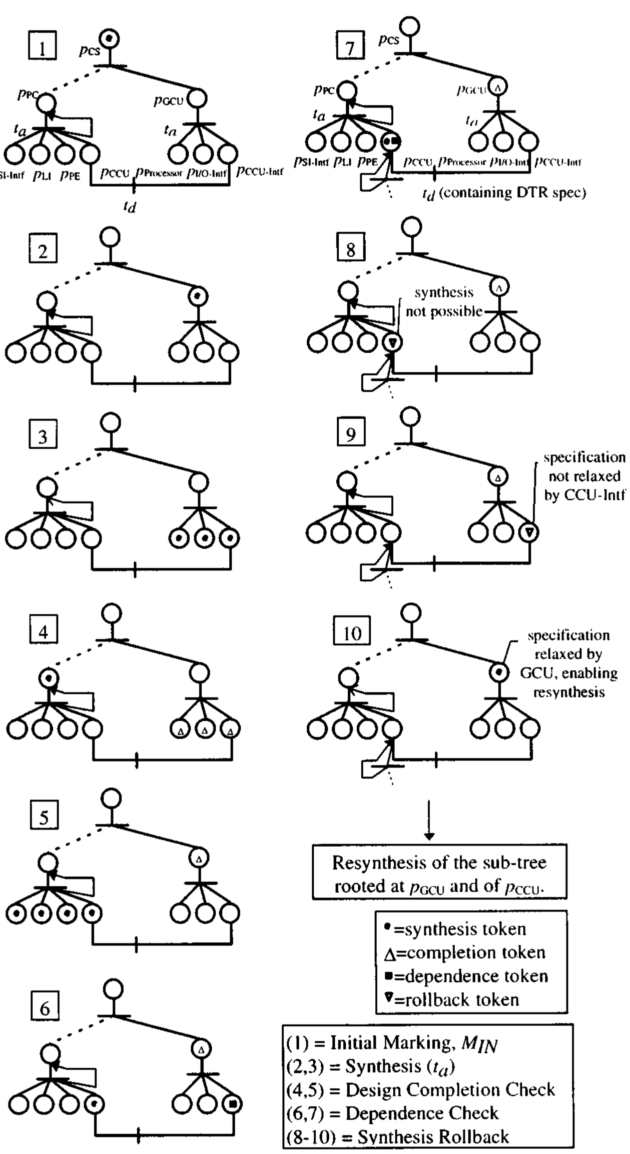
Theorem 3: The design-complete marking, M_{DC}, in a MOBnet is reachable (or unreachable) from the initial marking, M_{IN}, iff the synthesis-rollback marking, M_{SR}, is unreachable (or reachable) from M_{IN}.

7. Application Example

The MOBnet model for the Class Hierarchy of Fig. 2 is shown in Fig. 4. An example is given in this section to illustrate the modeling of synthesis rollback. The case of a possible relaxation of unsatisfiable specification and resynthesis after two steps of rollback is illustrated.

The target multiprocessor system, as synthesized by OOCSM [13], is an Alpha-21064 chip-based SIMD Shared Memory Architecture (SMA) with Globally Shared (GS) memory and a Multistage Interconnection Network (MIN) as the system interconnection.

1. *Synthesis*: As shown in Fig. 9, starting with the initial marking, M_{IN} : $M(p_{CS}) = k_s$, the transition, t_a, connected to p_{CS} is enabled (Eq. (3)) and fires by removing the synthesis token, k_s, from p_{CS} and by adding a synthesis token to each place hierarchically connected below t_a (Eq. (4)), that is, p_{MS}, p_{PS}, p_{SI}, and p_{GCU}. Let us consider only the synthesis of p_{GCU} in this example. The addition of a synthesis token to p_{GCU} enables



p_{CS}
p_{PC}
p_{GCU}
t_a
$p_{SI\text{-}Intf}$ p_{LI} p_{PE}
p_{CCU} $p_{Processor}$ $p_{I/O\text{-}Intf}$ $p_{CCU\text{-}Intf}$
t_d
t_d (containing DTR spec)
synthesis not possible
specification not relaxed by CCU-Intf
specification relaxed by GCU, enabling resynthesis
Resynthesis of the sub-tree rooted at p_{GCU} and of p_{CCU}.
•=synthesis token
Δ=completion token
▪=dependence token
▾=rollback token
(1) = Initial Marking, M_{IN}
(2,3) = Synthesis (t_a)
(4,5) = Design Completion Check
(6,7) = Dependence Check
(8-10) = Synthesis Rollback

Fig. 9 MOBnet: Application example.

the transition, t_a, connected below p_{GCU} to fire such that the token is removed from p_{GCU} and one added to $p_{CCU\text{-}Intf}$. The enabling and firing of transitions repeat and we reach a stage where p_{CCU} has a synthesis token. Since CCU and CCU-Intf are dependent objects, the dependence transition t_d connecting p_{CCU} and $p_{CCU\text{-}Intf}$ is enabled (Eq. (5)) and fires (Eq. (6)). Here, CCU-Intf is assumed to be already synthesized, hence t_d is enabled again (Eq. (7)) and fires in the opposite direction (Eq. (8)), resulting in a synthesis token and a dependence token (containing information on the *Data Transfer Rate* (DTR) specification of CCU) in p_{CCU}.

2. *Synthesis Rollback*: Suppose the specifications of the Cluster Control Unit (CCU) are: DTR $\geq$ 256 KB/s, *Throughput* $\geq$ 40 MB/s, and *Cost* $\leq$ \$500

Suppose the object, CCU, begins self-synthesis (i.e., a synthesis token is placed in p_{CCU}) and after dependence checking, the t_a connected to p_{CCU} fires and finds that if it is synthesized, its DTR is at most 250 KB/s; hence, synthesis is impossible. Now, the transition, t_a, connected below p_{CCU} is enabled (Eq. (16)) and fires (Eq. (17)) by removing k_s and k_d from p_{CCU} and adding a rollback token to p_{CCU}. According to Eq. (18), this enables the rollback of $p_{CCU\text{-}Intf}$, thus requesting it to relax its DTR lower bound, if possible.

Since the design of the CCU-Intf might affect the other components within the GCU, a rollback token is added to $p_{CCU\text{-}Intf}$, *enabling* the connected transition, t_a, and thus propagating the rollback to GCU. Now, according to Eq. (19), GCU relaxes DTR and resynthesizes itself by adding a synthesis token to p_{GCU}, thus the CCU-Intf is resynthesized using the relaxed specification. CCU can now synthesize itself, under the given specifications.

3. *Design Completion Check*: Suppose that after the firing of each instantiation transition t_i connected to the leaf places $p_{Scheduler}$, $p_{I/O\text{-}Intf}$, and p_{Buffer} and adding a completion token to each of them; the t_a connected above these leaf places is now enabled (Eq. (12)) and fires in the reverse direction by removing all completion tokens from the leaf places and adding a completion token to p_{CCU} (Eq. (15)). This design completion information is propagated upward until the root place (p_{CS}) receives a completion token, thus reaching M_{DC} from M_{IN}.

8. Conclusion and Future Work

A formal model called *Multi-token Object-oriented Bi-directional net* (MOBnet) model was proposed for the concurrent object-oriented system-level synthesis. It was first proved to be a valid model in modeling synthesis actions, their concurrency, and the synchronization among them. Then, the analysis of the dynamic behavior of the model was performed by modeling solution schemes for the design completion check and synthesis rollback problems in a concurrent synthesis environment. The validation and dynamic behavior analysis of MOBnets presented in this paper show that it can be quite a helpful model for synthesis in ways such as class hierarchy construction, design specification comparison, reachability analysis, and concurrent process management and analysis. Application examples have shown how MOBnet can be successfully applied to real synthesis problems.

Acknowledgment

This research was supported by the National Science Council, Taipei, Taiwan under grant NSC 85-2623-D002-015.

References

[1] P.A. Hsiung, S.J. Chen, T.C. Hu, and S.C. Wang, "PSM: An object-oriented synthesis approach to multiprocessor system design," IEEE Trans. VLSI Systems, vol.4, no.1, pp.83–97, March 1996.

[2] R.H. Katz, "Managing the chip design database," IEEE Computer, vol.16, no.12, pp.26–35, Dec. 1983.

[3] P.R. dos Santos, H. Sarmento, and L. Vidigal, "Ghost/Spook, user interface and process management in the Pace framework," Proc. European Design Automation Conference, pp.501–505, March 1990.

[4] H. Sarmento and P.R. dos Santos, "Pace—a framework for electronic design automation," Proc. IFIP WG 10.2 Workshop on Electronic Design Automation Frameworks, pp.85–97, Nov. 1990.

[5] P. van den Hamer and M.A. Treffers, "A data flow based architecture for CAD frameworks," Proc. International Conference on Computer-Aided Design, pp.482–485, 1990.

[6] M. Bushnell and S.W. Director, "Automated design tool execution in the Ulysses design environment," IEEE Trans. CAD of Integrated Circuits and Systems, vol.8, no.3, pp.279–287, March 1989.

[7] P. van der Wolf and T.G.R. van Leuken, "Object type oriented data modeling for VLSI data management," Proc. 25th ACM/IEEE Design Automation Conference, pp.351–356, June 1988.

[8] P. van der Wolf, P. Bingley, and P. Dewilde, "On the architecture of a CAD framework: The Nelsis approach," Proc. European Design Automation Conference, pp.29–33, March 1990.

[9] C.A. Petri, "Kommunikation mit automaten," Schriften des IIM Nr. 2, Institut fur Instrumentelle Mathematik, Bonn, 1962. English Translation: Technical Report RADC-TR-65-377, Griffiss Air Forse Bas, New York, vol.1, suppl. 1, 1966.

[10] K. Jensen, "Colored Petri nets and the invariant method," Theoretical Computer Science, vol.14, pp.317–336, 1981.

[11] K. Jensen, "High-level Petri nets," Applications and Theory of Petri Nets, ed. G. Rozenberg Informatik-Fachberichte, vol.66, Springer-Verlag, pp.166–180, 1983.

[12] K. Jensen, "Colored Petri nets," Petri Nets: Central Models and Their Properties, in Advances in Petri Nets 1986 Part I, Lecture Notes in Computer Science, vol.254, Springer-Verlag, pp.248–299, 1987.

[13] P.A. Hsiung, "System Level Synthesis for Parallel Computers," Ph.D. Dissertation, Graduate Institute of Electrical Engineering, National Taiwan University, June 1996.

[14] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, "Object-Oriented Modeling and Design," Prentice-Hall, Englewood Cliffs, 1991.

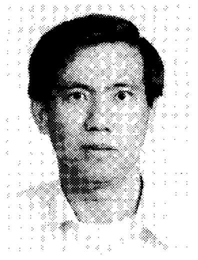
[15] K. Jensen and G. Rozenberg, eds., "High-level Petri Nets Theory and Application," Springer-Verlag, Berlin, Heidelberg, 1991.

[16] P.L. Knepell and D.C. Arangno, Simulation Validation: A Confidence Assessment Methodology, IEEE Computer Society Press, Los Alamitos, CA., 1993.

[17] D.R. Jefferson, "Virtual time," ACM Trans. Programming Languages and Systems, vol.7, no.3, pp.404–425, July 1985.

Pao-Ann Hsiung received the B.S. degree in mathematics and the Ph.D. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, ROC, in 1991 and 1996, respectively. From 1993 to 1996, he was a Teaching Assistant and System Administrator in the Department of Mathematics, National Taiwan University. Currently, he is a post-doctoral researcher at the Institute of Information Science, Academia Sinica, Taipei, Taiwan, ROC. His main research interests include: system-level design automation of multiprocessor systems, formal specification, modeling, and verification, parallel architecture design and simulation, and object-oriented design techniques in system syntheses.

Trong-Yen Lee received the B.S. and M.S. degree from National Taiwan Normal University, Taiwan, Republic of China, in 1981 and 1988, respectively. Currently he is a Ph.D. candidate in the Department of Electrical Engineering, National Taiwan University. His current research interests include parallel architectures, simulation, and design automation systems.



Sao-Jie Chen received the B.S. and M.S. degrees in electrical engineering from the National Taiwan University, Taipei, Taiwan, ROC, in 1977 and 1982 respectively, and the Ph.D. degree in electrical engineering from the Southern Methodist University, Dallas, USA, in 1988. Since 1982, he has been a member of the faculty in the Department of Electrical Engineering, National Taiwan University, where he is currently a full professor. From 1985 to 1988, he was on leave from the National Taiwan University and working toward his Ph.D. at the Southern Methodist University. During the fall of 1987, he held a visiting appointment at the Department of Electrical and Computer Engineering, University of Wisconsin, Madison. His current research interests include: VLSI physical design automation, object-oriented software engineering, and supercomputer architecture design and simulation. Dr. Chen is a member of the Chinese Institute of Engineers, the Association for Computing Machinery, the IEEE, and the IEEE Computer Society.