An Improved Translation of SA/RT Specification Model to High-Level Timed Petri Nets

Lihua Shi[1] and Patrick Nixon[2]

[1] Dept. of Computing, Manchester Metropolitan University, M1 5GD, UK
[2] Dept. of Computer Science, Trinity College, Dublin 2, Ireland
Email:L.Shi@doc.mmu.ac.uk or Paddy.Nixon@cs.tcd.ie

Abstract. Structured analysis methods for real-time systems (SA/RT) are widely accepted by the industrial world as a mature approach to real-time systems design. These methods use highly expressive graphical specification languages to specify system requirements. Giving semantics to SA/RT specifications via selected formal models has the advantage of not only retaining their user-friendly and problem-oriented characteristics, but also making good use of the existing results of formal models for easier simulation and more powerful analysis. An automatic translation from SA/RT specification models to high-level timed Petri nets has recently been reported in [5]. But this translation suffers from some drawbacks, especially that it is not compositional, and the resulting subnets, in some cases, can be of at least exponential complexity. In this paper, we propose an improved translation, which is compositional and the resulting nets are of much lower complexity, e.g. the number of transitions is linear with respect to the scale of the original model. The efficient translation will benefit the simulation and analysis of specifications, and the compositionality of the translation process will support their incremental or modular development and compositional analysis.

1 Introduction

Tom DeMarco structured analysis and system specification (SASS) method with the data flow diagram has always been one of the most important methods used in the development of software systems since its emergence. But for real-time systems, it is still not powerful enough to describe the timing information and dynamic behaviour. In order to address this problem, different ways have been proposed to extend the data flow diagram to capture control and timing information, among which, Ward–Mellor[18, 19] and Hatley–Pirbhai[9] extensions are two of the most popular ones. Each is used by 1/6 of all real-time system analysts in USA according to [20]. Structured analysis for real-time systems (SA/RT), is usually used to refer to these kinds of extensions to SASS. In [14] the Extended Systems Modeling Language (ESML) was proposed by Rruyn et al., which was based on the above two techniques. The combined notation has a more comprehensive and flexible set of constructs for representing control logic than either of the original notations.

SA/RT methods use highly expressive graphical specification languages, allowing the developer to concentrate on the clear understanding of the nature of problems, rather than the handling of formalisms, and also supports communication among the people involved with the development, which is very important in early stages of the system development process. On the other hand, these specification languages are not formal in that they lack appropriate formal semantics. Different understandings to the same symbol or combination of symbols may occur, and it is hard to analyze the completeness and consistency of the specification, which may result in flaws in the design and implementation. This can be very harmful for the development of complex real-time systems, since those flaws remaining from specifications are the most difficult to detect and need more efforts to correct[15].

In order to make full use of the existing highly expressive graphical languages of SA/RT, some work has been done to give semantics to them, either directly or indirectly.

1. Direct way :
   At Deutsche System-Technik, a project has been undertaken to make the specification used in the Ward–Mellor structured method automatically analyzable and suitable for applications in the field of safety-critical systems[12]. By using techniques developed for defining the semantics of statecharts[8], a family of semantics (i.e. recursive casual-chain semantics, weakly-fair interleaving semantics, and full interleaving semantics) are given to *transformation schema* (TS), the specification language in Ward–Mellor method. And a number of ambiguities and inconsistencies in Ward–Mellor's original definition are resolved.
2. Indirect way, i.e. translating SA/RT specification into formal models :
   (a) In [12], CSP semantics for transformation schema is given by translating TS into CSP according to a set of rules.
   (b) In [13], a rigorous interpretation of Extended Systems Modeling Language (ESML) is given by translating ESML into Petri nets.
   (c) In IPTES (Incremental Prototyping Technology for Embedded Real-Time Systems) project, a tool is implemented to translate SA/RT models automatically into high-level timed Petri nets[5].

To give formal semantics directly to informal graphical languages can result in rather complicated semantics[11, 5] which are difficult to analyze; while translating them into a selected formal model has the advantage of not only retaining their user-friendly and problem-oriented characteristics, but also making good use of the existing results of the corresponding formal model for easier simulation and more powerful analysis.

The main formal models for real-time systems can be classed into three categories : temporal logics, process algebras, and Petri nets. Process algebras, eg. CSP, CCS, lead to methods for compositional verification, which is desirable for complex systems. But they are not appropriate for specifying inherently global properties, such as safety, liveness, fairness, and real-time response, which involve the global computation. While Petri nets and temporal logic are good at

describing properties that pertain to the complete systems, however, they are relatively unstructured and not ideal for compositional verification[11]. Recent work on the compositionality or modularity of Petri nets[6, 2, 3, 10] and the emergence of high-level timed Petri nets[7, 17] has made them more attractive to the specification and analysis of complex real-time systems.

Although some work has been reported to give rigorous interpretation to the SA/RT requirement model via Petri nets, *time* has been taken into consideration only by Elmstrom et al.[5]. This work suffers from the following drawbacks:

1. Lack of compositionality in the translation process: For example, the translation of a control transformation should depend on the type and sometimes even internal structure of all the controlled transformations. We believe that compositionality is essential for the translation to assist the incremental or modular development of SA/RT specifications and their compositional analysis.
2. High complexity of the resulting nets: In many cases the growth of transitions and arcs is intolerably fast. Especially in the case of state/transition diagrams, the complexity of the resulting subnets relating to one *state* becomes at least exponential with respect to the scale of the subdiagram. The high complexity of translation results is a severe problem to efficient simulation and analysis.

The general aim of our work is to solve the above problems. In addition, our work is based on the latest SA/RT model [14] which is more powerful than that used by [5] in that it allows the description of more control activities in a succinct way. We have proposed a translation which is compositional, and the resulting nets have much lower complexity, e.g. the number of transitions is linear with respect to the scale of the original model. Due to the space limit, this paper will only discuss the main improvement to the translations given in [5, 4]. A detailed description of our translation rules can be found in [16].

First, a brief description of the STER nets, i.e. the Petri net model we use, is given in the next section.

2 ER nets – high-level timed Petri nets

Environment/Relationship (ER) nets[7] are a kind of high-level timed Petri nets, where both time and functional aspects can be modelled in a semantically coherent way. It was shown in [7] that ER nets are the most powerful model among all the existing timed versions of Petri nets. On the one hand, the more general the model, the less amenable it is to analysis. While on the other hand, if a model lacks modeling power or flexibility, it forces the specifier to add new features informally, or specify things in an unnatural way, which inhibits the discovery of system properties of interest. For large real-time systems, due to the complexity of the applications, not many choices exist. Some control and timing properties in SA/RT cannot be specified in any of the existing timed models except ER nets[16].

ER nets: An ER net is a net where,

- Tokens are environments on ID and V, i.e. partial functions : $ID \rightarrow V$, where ID is a set of identifiers and V a set of values.
- Each transition t is associated with an action $\alpha(t) \subseteq ENV^{k(t)} \times ENV^{h(t)}$, where ENV is the set of all environments, $k(t)$ and $h(t)$ denote the cardinalities of the preset and the postset of transition t, respectively. The projection of $\alpha(t)$ on $ENV^{k(t)}$ is denoted by $\pi(t)$ and is called the preconditions of transition t.
- A marking m is an assignment of multisets of environments to places.
- A transition t is enabled in a marking m iff for every input place p_i of t, there exists at least one token env_i such that $< env_1, \ldots, env_{k(t)} > \in \pi(t)$.
- A firing is a triple $x =< enab, t, prod >$, such that $< enab, prod > \in \alpha(t)$; and the occurrence of the firing changes a marking m to $m' = m - enab + prod$.

Time ER (TER) nets: A TER net is an ER net where all tokens contain a variable *chronos*, which represents the timestamp, and for any firing $x =< enab, t, prod >$, the following axioms are satisfied: (1) constraint on timestamps: all elements of the tuple *prod* have the same value of *chronos*, called the time of the firing; (2) local monotonicity: the time of the firing cannot be less than the value of *chronos* of any token in *enab*.

Strong TER(STER) nets: ER nets allow both the *strong* and *weak* time semantics. With strong time semantics, if a transition is enabled and remains enabled for all possible time values at which it can fire, then it must fire; with weak time semantics, a continuously enabled transition may not fire during the specified timing points/duration. For the interpretation of SA/RT specifications, the STER model, i.e. the TER model with strong time semantics, applies. The formal definition of STER nets [7] is omitted here.

An example: Figure 1 illustrates a TER net, where $ID = \{a, b, chronos\}$, and $V = N$. According to the definition of the actions[3], $t1$ and $t2$ are enabled, and $t3$ is not. Given the tokens (of $P3$) $tok4 = \{(a, 1)(chronos, i)\}$, $tok5 = \{(a, 2)(chronos, j)\}$ where $10 \leq i \leq 15$, $3 \leq j \leq 6$, $< tok1, t1, tok4 >$ and $< tok2, t2, tok5 >$ are possible firings.

The data type of a place P can be defined as all the possible tokens in P. The data type of $P1$, for example, can be defined as:[4] $P1 :: \{a, chronos\} \rightarrow N$, or written as $P :: a, chronos : N$. Also, for the definition of an action to be more intuitive, we can express it in three parts as in [5], i.e.

time : predicates relating to chronos fields of all the preset/postset places
precondition : other pre-conditions
action : assignments to postset places to make other post-conditions hold

For example, $act1$ in figure 1 can be rewritten as:

$time : p1.chronos + 10 \leq p3.chronos \leq p1.chronos$
$precondition : p1.a < 10$
$action : p3.a := p1.a$

[3] Note for the Petri nets discussed in this paper, the weight of any arc is 1, so we can use P_i to denote the token in place P_i in describing the actions.

[4] It can be assumed that all the possible tokens in the same place have the same domain.

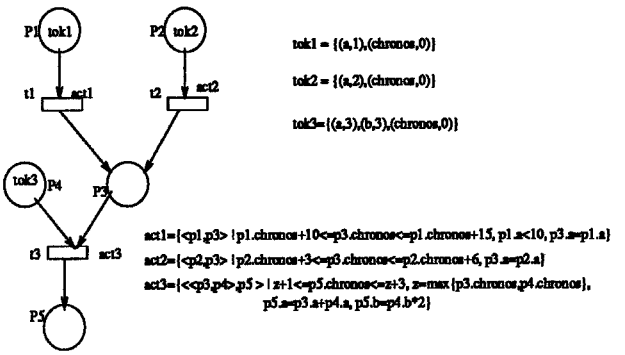

Fig. 1. an ER net considering time

3 Compositionality of the Translation

The development of an SA/RT model is a hierarchical process, and it is the flattened SA/RT model we are mainly concerned with, since an upper level of an SA/RT model is not considered in enough detail, and its information may be incomplete for it to be interpreted in a rigorous way. The most important principle of our translation with the flattened level is *compositionality* (or say, *locality*), i.e. the translation of each *component* is independent of other components.

3.1 Compositional Principle of the Translation

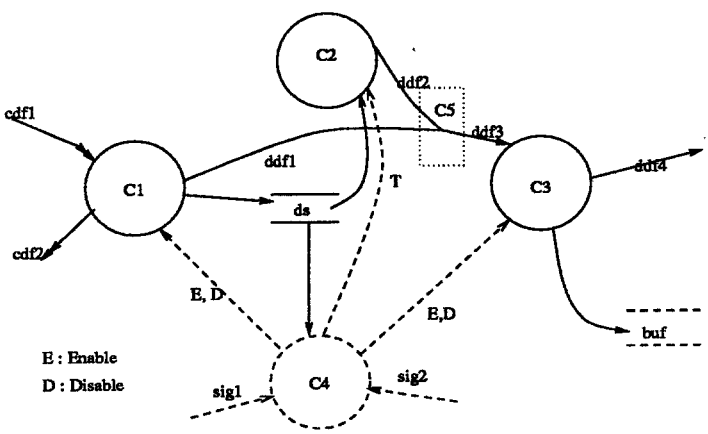

Fig. 2. The concept of *component* in transformation schema TS1

A *component* is either a data or control transformation together with all its inputs and outputs, or, a merging or splitting structure representing flows from

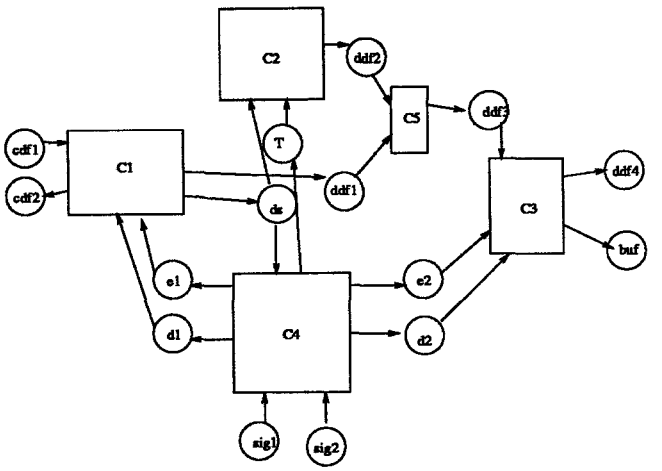

Fig. 3. The compositional principle of translation for TS1

multiple sources or to multiple destinations[18]. Figure 2 illustrates a transformation schema TS1 with five components $C1$-$C5$, where $C1$-$C3$ are data transformations, $C4$ is a control transformation, and $C5$ is a merging structure which merges $ddf1$ and $ddf2$ to $ddf3$. The interface of a component includes all the data/control flows to/from it. For example, $C3$ is a data transformation with $ddf3$, *Enable* and *Disable* as input interface, and with $ddf4$, buf as output interface; $C5$ is a merging structure with $ddf1$ and $ddf2$ as input interface, and with $ddf3$ as output interface.

Each component corresponds to an STER subnet. Each flow or store connecting components in the transformation schema corresponds to a place (or some places) shared by STER components in the net. The translation rules are *compositional* (or *localized*) because each component in SA/RT model can be translated into an STER subnet independently, and the STER net corresponding to the SA/RT model as a whole can be obtained by combining these STER subnets via shared places.

Figure 3 illustrates the STER net structure corresponding to the transformation schema TS1. Rectangles $C_i (1 \leq i \leq 5)$ represent STER subnets for components C_i of TS1. Those flows in TS1 are all translated into shared places outside the rectangles. For example the discrete data flow $ddf1$ from component $C1$ to $C5$ in TS1 corresponds to a place $ddf1$ shared by subnets $C1$ and $C5$ here. So the STER net as a whole is just the composition of all the five STER subnets that share interface places.

To simplify the situation, it is assumed in the above example that each data/control flow corresponds to one place. The same principle follows if some flow corresponds to more than one place, or some group of flows share one place. The translation strategies of data flows and stores are illustrated in figure 4 (the translation of control prompts will be discussed in the next subsection). Each

flow f corresponds to place P_f, and when f is not connected with a data store, there is also a complementary, or say, an "empty" place P'_f (i.e. a token in P'_f represents that no value is attached to the flow f). Suppose the value of f is produced by component C1 and consumed by C2. Then the STER subnet of C1 should have transition *write* (and also *write'* if f is not a data store) that produce(s) a token to place P_f; similarly the STER subnet of C2 should have transition *read* (and also *read'* if f is from a buffer) that consume(s) a token from place P_f. Note that for buffer *buf* in figure 4(d), *read* and *read'* represent that after one item of *buf* is consumed by *C2*, *buf* is non-empty and empty respectively. Detailed explanations are given in [16].

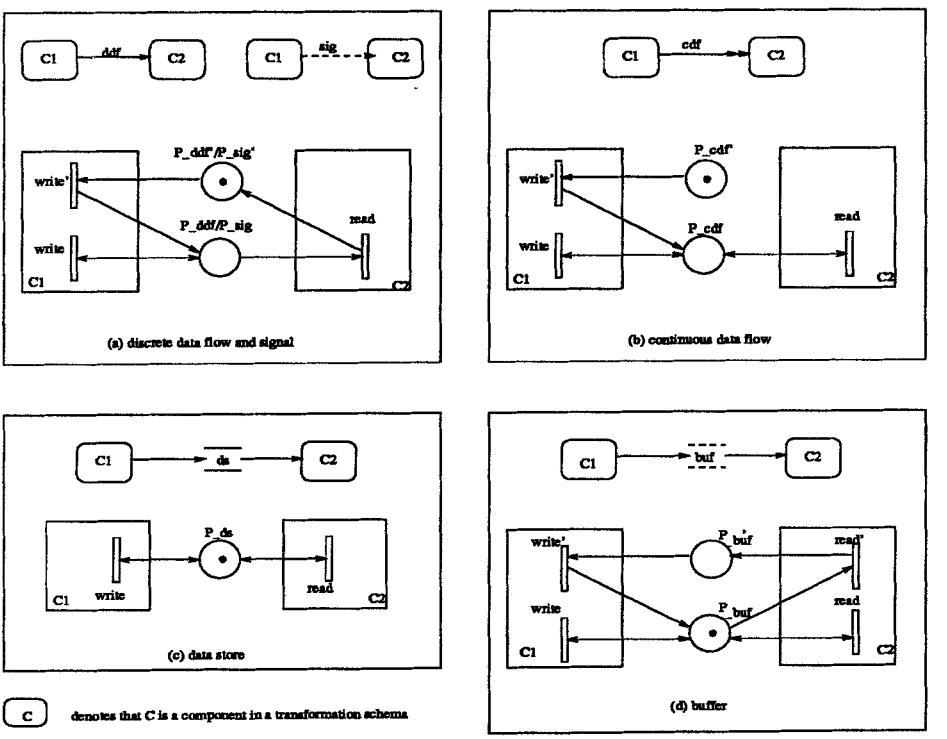

Fig. 4. The translation of data flows and stores

3.2 Localizing the Translation for Control Prompts

The translation in [5] is not compositional since some SA/RT constructs can not be translated independently. The main problem lies in the translation of control prompts, which are translated as transitions, and depend on the types, and even internal structures of all the transformations that receive them. We solve this

problem by translating all control prompts going to the same transformation as two complementary shared places, and making the translation of the controlling and controlled transformations independent of each other.

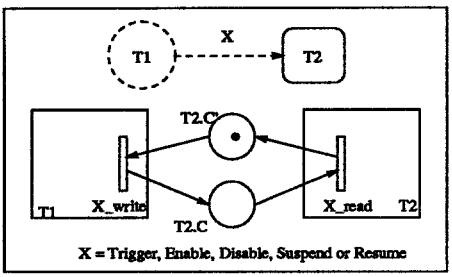

Fig. 5. The translation of control prompts

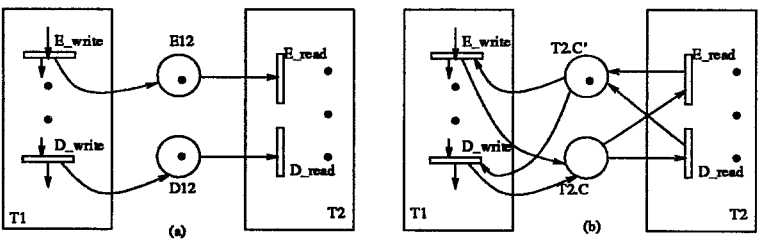

Fig. 6. Translation principles for control prompts : an example

Figure 5 illustrates translation principles for control prompts. T1 is a control transformation which controls a (data or control) transformation T2. The figure shows that all the control prompts of T2 share two complementary places $T2.C$ and $T2.C'$. STER subnet for T1 has a transition X_write to produce a token with value X to T2.C; while STER subnet for T2 has a transition X_read to consume a token with value X from T2.C. But the form and number of such X_write or X_read transitions may vary, and they depend *only* on transformation T1 or T2. Some definitions are given as follows (where $X \in \{Trigger, Enable, Disable, Suspend, Resume\}$):

T2.C' :: *chronos* : *data_type_of_time*
T2.C :: *chronos* : *data_type_of_time*
data : $\{Trigger, Enable, Disable, Suspend, Resume\}$
X_write :: *time* :
precondition :
action : $T2.C.data = X$

X_read :: *time* : $time_0(X\_read)$
precondition : $T2.C.data = X$
action :

Where for transition t, $time_0(t)$ is defined as :

$$\forall p \in t.postset,\ p.chronos := max\{pre.chronos | pre \in t.preset\}$$

We omit the parameter t when there is no ambiguity. So the transition specified with $time_0$ should fire immediately when its *precondition* holds.

Note here we do not use separate places for different control prompts as for data flows. Consider the simple example as illustrated in figure 6(a), where two places $E12$ and $D12$ are used to represent the two prompts *Enable* and *Disable* sent by T1 to T2 respectively. When *Enable* and *Disable* are generated by the automaton of T1 sequentially, yet with the same timestamp; or when *time* is simply not considered in the analysis of some properties, usually it is required that the prompts be consumed in the same order as they are produced. But with transitions *E_read* and *D_read*, it is indeterminate which one fires first when their preset places have the same value in *chronos*, or when *time* is not considered. Thus the control prompts may not be consumed in the same order as they are produced. This problem can be solved in our method by using shared places for all control prompts, as in figure 6(b), where $T2.C$ is $safe$ by initially putting one token in its complementary place $T2.C'$. So we can guarantee that all the control prompts are accepted and consumed in an orderly manner.

3.3 Benefits of Compositional Translation

The compositionality of the translation process benefits the development of SA/RT specifications in the following aspects:

- Assisting the interactivity of the development process of SA/RT specifications. The development of SA/RT specification is quite an interactive process. The users modify the specification, and expect a *responsive* change in the corresponding animation and analysis. The compositional translation localizes the modification of the underlying subnets, thus improving the efficiency and interactivity.
- Assisting the incremental development of specifications. In many occasions, the development process of a specification can be incremental. For example, a critical part of the model may be developed and its critical properties need to be analyzed first. Compositional translation allows the translation and analysis of part of the model, thus supporting the incremental development of specifications.
- Assisting modular development and analysis of specifications. Any module in transformation schema can be translated independently into a Petri net module; Petri net modules can be combined just by shared places. Modifying of any part of the specification only results in *localized* modification of the underlying net and other parts, including their properties, will be kept intact. Thus compositional translation is essential to the compositional/modular development and analysis of specifications.

4 Improved Efficiency to the Resulting Nets

It is mainly for the benefit of analysis that formal models are used to interpret SA/RT specifications. As pointed out in [7], STER nets are general enough for the requirement specification of most complex real-time systems; on the other hand, most of the usual temporal properties are undecidable in STER nets. Generally, the STER nets can be analyzed in the following ways[7]:

- to restrict the analysis to special decidable subcases corresponding to special classes of applications;
- to derive approximate solutions : by ignoring token values, STER nets are reduced into low-level (timed) Petri nets. So in general, all known techniques for analyzing (timed) Petri nets can be used as approximate analysis aids in the case of STER net;
- to provide interactive decision-support systems to assess them;
- to test specifications by simulation.

Whatever method is used, the complexity of simulation and analysis of Petri nets grows with their sizes, especially the numbers of transitions and arcs, that is, the efficiency of translations in our case.

In [5], the growth of transitions and arcs in the resulting nets is very fast in some cases. Our strategies can greatly decrease the complexity. In this section we illustrate this improvement via state/transition diagrams, where the size of the nets grows the fastest in [5].

4.1 An Efficient Translation for State/Transition Diagrams

Figure 7 shows a typical example of a state/transition diagram STD, with two states and three transitions. In order to distinguish between a transition in Petri nets and a (state) transition in state/transition diagrams, in the following we use *Stransition* to denote the latter. The translation for state/transition diagrams is localized in the sense that the subnet for each state is decided only on this state and Stransitions from it. For the ease of description, we consider the input and output part of Stransitions separately.

The translation for inputs of Stransitions. In this part we ignore the Stransition outputs in order to concentrate on the rules of the inputs.

Consider first a specious solution in figure 8(a)(note that only Stransitions from *state1* to *state2* of STD are considered at this moment), where each state corresponds to one place, and each Stransition corresponds to one transition. It has problems when *sig1* and *sig2* arrive at *state*1 simultaneously, and both the input conditions of Stransition 1 and 2 are satisfied. In this situation, only one transition will fire, thus only one of the two signals is consumed. So the other signal remains, and will be consumed by some transition later. But this violates the requirement that an Stransition occurs only if its signal input comes *when* or *after* the origin state is reached[14].

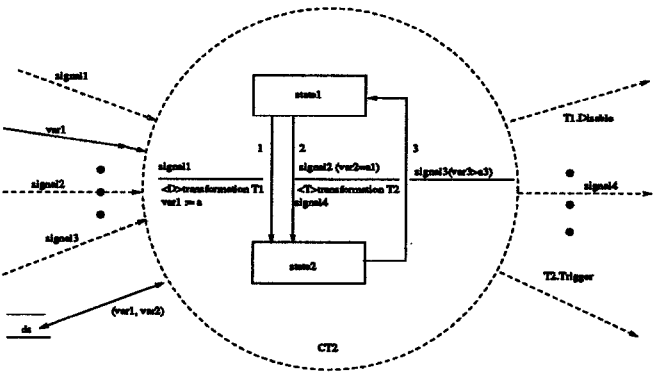

Fig. 7. STD : a state/transition diagram

Another possible method which is used in [5] is illustrated in figure 8(b). It considers all the combinations of input signals, and if more than one signal arrive at the same time, only one of the Stransitions whose input conditions are satisfied is selected (nondeterministically) to occur, but all the signals are consumed. So four transitions are used to represent two Stransitions:

- t12-1a, t12-2a : Stransition1 or 2 occurs when $sig2$ or $sig1$ is empty.
- t12-1b, t12-2b : Stransition1 or 2 occurs when both $sig1$ and $sig2$ are non-empty, and both are consumed after the firing.

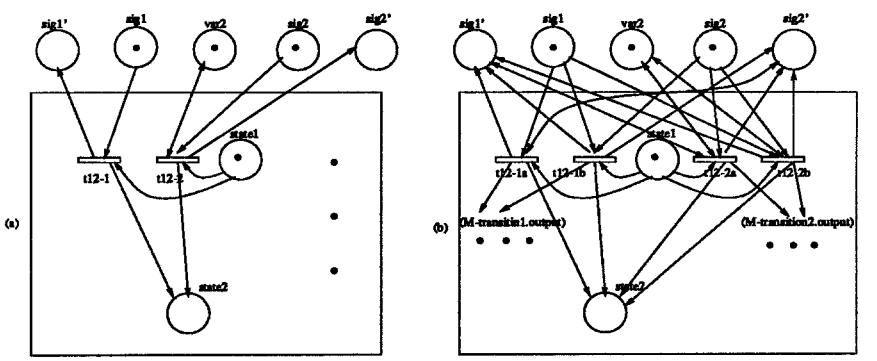

Fig. 8. Possible STER subnets for STD

With this translation, any signal that comes at *state1* will not remain and thus will not be used later, but the number of transitions needed increases rapidly with the number of Stransitions. Suppose there are n Stransitions from $state_i$ to $state_j$, each with a signal event, the number of corresponding transitions (when outputs are not considered) will be $n \times 2^{n-1}$(A proof can be found in [16]).

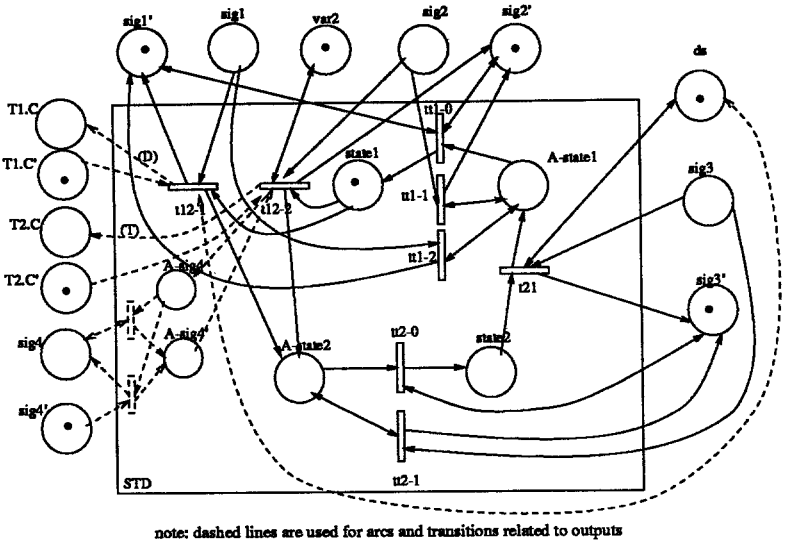

Fig. 9. STER subnet for STD

To solve this problem of "*transition explosion*", we use an auxiliary place for each state to consume all the possible signal inputs of that state before it is reached, so no state will consume any signal that comes earlier. Figure 9 illustrates our translation of STD. *A_state*1, 2 are auxiliary places for *state*1, 2 respectively. Transitions t12-1 and t12-2 represent *Stransition*1, 2 from *state*1 to *state*2, and t21 represents *Stransition*3 from *state*2 to *state*1. But these transitions do not lead to the next legal state directly. For example with t21, a token goes to the auxiliary state *A-state*1, and when the token goes from *A-state*1 to *state*1 by tt1-0, it uses $sig1'$ and $sig2'$ as *test places*, to make sure that any input signal of *state*1(i.e. $sig1$ or $sig2$) that comes earlier is consumed by tt1-1 or tt1-2. For *A-state*2, the situation is similar[5].

The translation for outputs of Stransition. According to the formation rules, several signals can appear in the output part of one Stransition. Since each signal, whether it goes to a buffer or not, corresponds to two complementary places, a possible translation as in [5] is to use one transition for each combination of output signal places. Suppose an Stransition *Stran*1 sends three signals, $sig1$, $buf2$ and $sig3$, then it corresponds to 8 transitions as in figure 10(a), where e.g. transition t_{011} corresponds to the situation when $sig1$ is empty, $buf2$ and $sig3$ are non-empty, etc. Generally if the number of output signals in an Stransition is n, then 2^n transitions are needed, and the number of related arcs is $n(2^n + 2^{n-1})$.

[5] There are some differences if the input signal of a Stransition comes from a buffer. Firstly a data item in a buffer will not be removed unless it is used as the event of some Stransition. Secondly, two transitions are needed for each Stransition with a buffer input, which correspond to *read* and *read'* in figure 4(d) respectively. An example can be found in [16]

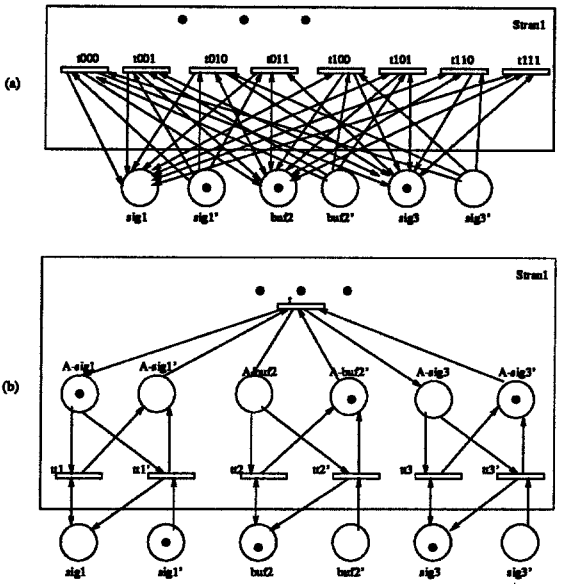

Fig. 10. Possible translations for a Stransition *Stran*1 with 3 active outputs

Our translation for *Stran*1 is illustrated in figure 10(b). Two complementary auxiliary places are used for each of the signals, e.g. *A-sig1* and *A-sig1'* are auxiliary places for *sig*1. Transition t corresponds to *Stran*1, consuming its inputs and producing outputs, but the tokens for signal outputs are sent to *auxiliary* places. With transitions tt_i and $tt'_i (1 \leq i \leq 3)$, tokens in auxiliary places are sent to actual output shared places. Note that the *time* part of each transition tt_i or tt'_i is defined as $time_0$, i.e. they fire immediately when they are enabled. This means the outputs to places *sig1*, *buf2* and *sig3* will be produced with the same timestamp, although may be sequentially.

According to this strategy, we use auxiliary places for the output signal *sig*4 in STD. The transitions and arcs related to the outputs in STD are represented by dashed lines in figure 9.

The complexity of the resulting nets. Suppose for a state/transition diagram, the total number of Stransitions is *Strans*, and the number of those with signal inputs is $Strans_{sig}$, of which the number of those with signals coming from buffers is $Strans_{bsig}$; the number of all the output signals is *out.sig*, and the number of states is *states*, then all the transitions in the corresponding STER subnet consists of:

1. transitions from place $state_i$ to A_state_j(or $state_j$) corresponding to the Stransition from $state_i$ to $state_j$, e.g. *t12-1* in figure 9. The number is $Strans + Strans_{bsig}$, since two transitions are used for an Stransition with an input signal coming from a buffer.
2. transitions that make sure all input signals of the next legal state are empty

before it is reached, e.g. *tt1-1* in figure 9. The maximum number is $\underline{Strans_{sig} - Strans_{bsig}}$ (i.e. the number of Stransitions with signal inputs which are not coming from buffers). It is *maximum* since different Stransitions to the same states may share same input signals, and in this case only one transition for each signal is needed.

3. transitions from place A_state_i to $state_i$, e.g. *tt1-0* in figure 9. The maximum number of these transitions is $\underline{states}$. Note that there is no such transition for states that do not have signal inputs and hence need no auxiliary place.
4. transitions that produce tokens to signal, or (signal) buffer places from their auxiliary places, where two are used for each signal output, as transitions with dashed lines in figure 9. The number is $\underline{2out.sig}$.

So the total number of transitions corresponding to the state/transition diagram in the worst case is :
$(Strans + Strans_{bsig}) + (Strans_{sig} - Strans_{bsig}) + states + 2out.sig$
$= Strans + Strans_{sig} + states + 2out.sig$
$\leq 2Strans + states + 2out.sig$
This result is a big improvement to the corresponding translation in [5], where the number of transitions for a *single* state in a state/transition diagram is analyzed in [5] as follows :

> ...(Suppose) n denotes the number of input signals (not coming from buffers) and B denotes the number of input signals (coming from buffers) in the condition of the state transitions from a specific state, (then the number of transitions corresponding to this state is[6]:)
>
> $$\sum_{i=1}^{2^n}(\sum_{j=1}^{n} g(i,j)) + \sum_{b=1}^{B}(\sum_{st\_tr=1}^{l}(\prod_{st\_tr\_out=1}^{N} h(st\_tr\_out))$$
>
> where g is an auxiliary function defined as :
>
> $$g(i,j) = \begin{cases} \sum_{st\_tr=1}^{l}(\prod_{st\_tr\_out=1}^{N} h(st\_tr\_out)) & if\ odd(i - 1\ div\ 2^{j-1}) \\ 0 & if\ even(i - 1\ div\ 2^{j-1}) \end{cases}$$
>
> where l denotes the number of state transitions which have signal j or b as incoming signal, N denotes the number of output flows in state transition st_tr, and st_tr_out denotes a specific output flow in state transition st_tr. h is an auxiliary function ...

The definition of h is omitted here, but we must point out that $\forall i, h(i) \geq 2$.

[6] The result the authors get is:

$$\sum_{i=1}^{2^n}(\sum_{j=1}^{n} g(i,j)) + (\sum_{b=1}^{B}(\sum_{st\_tr=1}^{l}(\prod_{st\_tr\_out=1}^{N} h(st\_tr\_out))) \times 2^n$$

which is different to the above result in the number of transitions for buffers.

5 An Example

Figure 11(a) illustrates a simple case of a control transformation CT with a state/transition diagram given in [5]. The translation for CT in [5] depends on the information of transformation $T1$ that it disables: (1)$T1$ is a discrete data

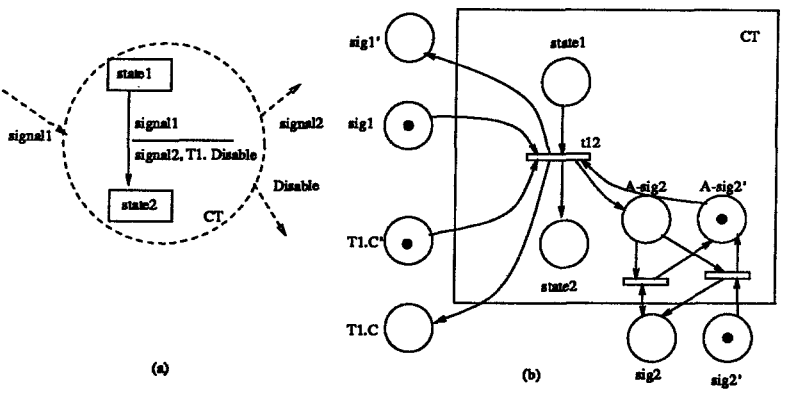

Fig. 11. CT: a simple example of control transformation

transformation. The High-level timed Petri net(HLTPN) for CT has 7 transitions and 46 arcs(figure 29 of [5]); (2)$T1$ is a continuous data transformation. The HLTPN for CT has 11 transitions and 76 arcs(figure 30 of [5]); (3)$T1$ is a control transformation. In this case, the HLTPN for CT will depend on the internal information of T1, e.g. the number of states in the case of state/transition diagram. The HLTPN for CT has 7 transitions and 46 arcs (figure 31 of [5]) when T1 has *two* states.

But with our strategies, the STER subnet for CT with only 3 transitions and 15 arcs can be derived independently. It only shares those places outside the rectangle with other components, e.g. $T1.C$ and $T1.C'$ are places it shares with transformation T1. Note that t12.action includes $T1.C.data := Disable$.

6 Other Improvements

6.1 Moore-type automata

Moore-type automata are not used in [18, 19], and they are not considered in [5]. But since they succinctly describe the combinational control logic, they are used in [9], and adopted in [14].

A Moore-type automaton is used if the logic of a control transformation is purely combinatorial – the control exerted during a time period depends only on a combination of continuous input or stored variable values that hold during the period. It can be represented as an activation table as M1 in figure 12. For each

row no.	INPUT		TRANSFORMATIONS		OUTPUT		
	ds1.var1	var2	T1	T2	sig1	var3	var4
1	ON	1	D	*	Y	5	9.5
2	ON	2	D	*	*	10	8
3	ON	3	E	T	*	15	7.6
4	OFF	1	E	*	Y	20	5
5	OFF	2	E	*	Y	25	6.8
6	OFF	3	E	*	*	1000	2

Fig. 12. M1 : A Moore-type automaton

row of the table, whenever the condition in the input part is true, some actions as indicated in the other part of the row should be undertaken, i.e. sending control prompts to transformations, sending signals, and modifying variables, etc.

Figure 13 illustrates the STER net for M1. Each transition t_i $(1 \leq i \leq 6)$ corresponds to the ith row of the activation table. *P_mode* is an auxiliary place that records the latest fired transition, so it can be used to keep a transition from firing continuously when the input conditions remain the same. In translating the signal outputs, the same method as for state/transition diagrams is used, i.e. two auxiliary complementary places are used for each signal output. For continuous data flow outputs, i.e. *var3* and *var4*, they are all empty or non-empty at the same time, so they can share one *empty* place *cdf'*, and also share two auxiliary places *A-cdf* and *A-cdf'*. Note that for each transition $t_i(1 \leq i \leq 6)$, there are an arc from $T1.C'$ and an arc to $T1.C$ sending either *Enable* or *Disable* prompt to transformation T1. We do not draw these arcs in detail for the net to be more readable. Some of the definitions are:

P_mode :: *chronos* : *data_type_of_time*
data : 0..6 *0 is used when M1 is just enabled *\
A-cdf :: *chronos* : *data_type_of_time*
var3 : *data_type_of_var3*;
var4 : *data_type_of_var4*)
$t1$ :: *time* : $time_0$
precondition : $(ds1.data.var1 = ON) and (var2.data = 1)$
$and(P\_mode.data \neq 1)$
action : $T1.C.data := Disable, P\_mode.data := 1,$
$A\_cdf.var3 := 5, A\_cdf.var4 := 9.5$

Other transitions $t2$-$t6$ can be similarly defined.

Suppose the number of signal outputs is n, and the number of rows in the activation table is *rows*, then the number of transitions is : $rows + 2(n + c)$, where $c = 1$ when there is continuous output, and $c = 0$ otherstate/transition diagramControl Prompts *Suspend* and *Resume Suspend* and *Resume* prompts are not incorporated in [18] and [19], and they are not considered in [5]. But they permit more detailed modeling of certain implementation situations, and are useful for the design of real-time systems.

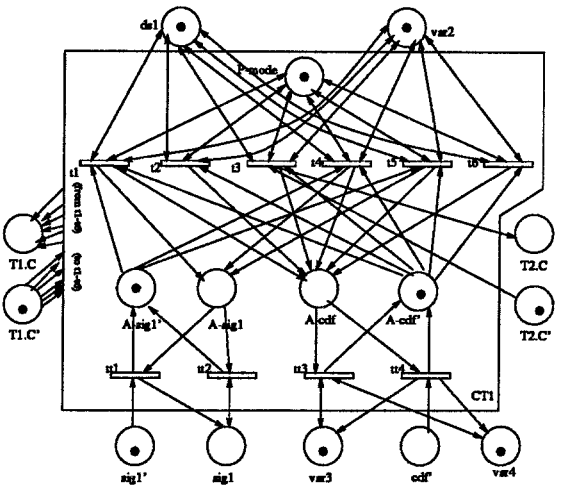

Fig. 13. STER net for M1

Suspend and *Resume* prompts are similar to *Enable* and *Disable*, except that a suspended transformation remembers its intermediate results and the system context and picks up where it left off when resumed. For a control transformation that has *Pause* (i.e. *Suspend* and *Resume*) as outputs, the related translation is similar to that for CT with $T1.Disable$ as output in figure 11(b). For a transformation with *Pause* as inputs, the translation rule varies with its type. We illustrate them by the following two cases.

1. *Pause* for a control transformation CT1, which is described by STD, a state/transition diagram without terminal states
   Suppose STD has n states, denoted by $state_i (1 \leq i \leq n)$. The corresponding net is shown in figure 14. Places *suspended* and *enabled* represent the two complementary states of CT1, and when a token is in place *enabled*, another token should also be in one of places $state_i$ (or their auxiliary state $A\text{-}state_i$, which we omit in this figure, since they are irrelevant for the translation here). We have the following definitions:

   suspended :: $chronos : data\_type\_of\_time$
   $data : 1..n$
   ts_0 :: *Suspend comes at suspended state, nothing happens*\
   $ts_i (1 \leq i \leq n)$:: *transitions consume Suspend prompt at $state_i$ *\
   $time : time_0$
   $precondition : CT1.C.data = Suspend$
   $action : suspended.data := i$
   tr_0 :: *Resume comes when CT1 is in enabled state, nothing happens*\
   $tr_i (1 \leq i \leq n)$ :: *transitions consume Resume prompt at $state_i$ *\
   $time : time_0$
   $precondition : CT1.C.data = Resume, suspended.data = i$

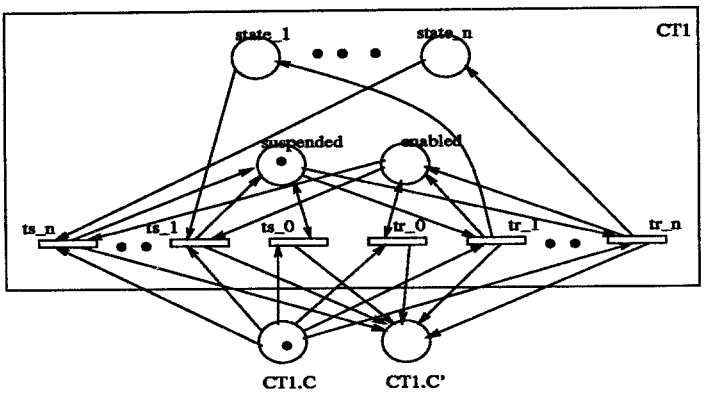

Fig. 14. STER net for CT1

action :

2. *Pause* for data transformations
   For a data transformation, say DT, there are two internal *enabled* states, i.e. *idle* or *executing*. For place *executing*, the *start_time* field is used to record the time when DT begins to process the inputs, and the temporal requirements of DT (e.g. output delay) are usually calculated with respect to its value. When *Pause* is used, the temporal requirements depend not only on *start_time*, but also on the suspended time period. So to obtain correct temporal properties of DT, the suspended time should be counted to *start_time* of the place *executing*. The corresponding subnet is the same as figure 14 except that (1) $n = 2$, and $state_1 = idle$, $state_2 = executing$; (2) the data type of *suspended* should include the field *start_time* : *data_type_of_time*; and (3) the action part of ts_2, which accepts a *Suspend* prompt at *executing* state, should include: $suspended.start\_time := CT1.C.chronos$; and the action part of tr_2, which accepts a *Resume* prompt at (*executing_*)*suspended* state, should include the following:
   $executing.start\_time := executing.start\_time +$
   $(CT1.C.chronos - suspended.start\_time)$
   * ($CT1.C.chronos - suspended.start\_time$) is the suspended time*\

Pause can also interact with other control prompts, i.e. *Trigger* or *Activate*. See [16] for detailed descrstate/transition diagramThe Inheritance of "Disabling" Control According to [18], [19] and [14], when a control transformation is disabled, all the transformations it controls should be disabled as well. An example of controlling structure of a transformation schema is given in figure 15(a), where CT3 and DT1 should be disabled at the moment CT2 is disabled by CT1. But this requirement is not considered in [5], because the translation of a control transformation depends on the type and the internal structure of the disabled transformations, and that means in this example, the translation of CT1 would depend on all its *decending* transformations, i.e. CT2, CT3 and DT1, to meet

the "*disabling transitivity*" requirement above, which would be impractical.

This requirement can be easily implemented in our translation, as illustrated in figure 15(b). Note that transition t is the abstract form of all the transitions in the STER subnet of CT2 that make CT2 go to *disabled* state on receiving a *Disable* prompt. It can be defined as follows:

time : $time_0$
precondition : $CT2.C.data = Disable$
action : $CT3.C.data := Disable$, $DT1.C.data := Disable$

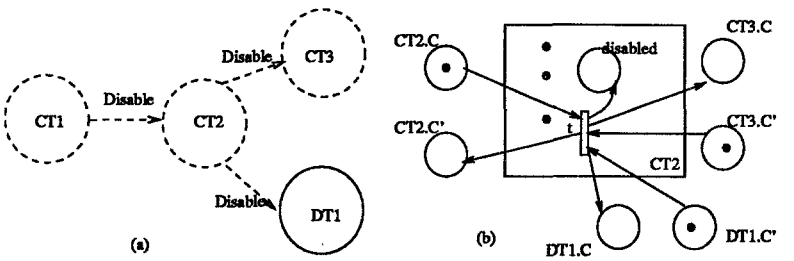

Fig. 15. Translation of disabling control

7 Conclusions

This paper has presented a new method for translating SA/RT models to high-level timed Petri nets; this can greatly improve the efficiency of the resulting nets with respect to their numbers of transitions and arcs, and thus facilitate the simulation and analysis of specifications. Also the introduction of compositional principles into the translation process supports the incremental or modular development of specifications and their compositional analysis. The resulting nets can be analyzed at different levels of abstraction as described in §4. Despite the previous efforts on such analysis[7, 6, 1], more work needs to be done for efficient analysis methods of high-level timed Petri nets and especially their modular/compositional analysis.

References

1. C. Bellettini, M. Felder, and M. Pezzé. A Tool for Analysing High-Level Timed Petri Nets. Technical Report IPTES-PDM-41-V2.0, Politecnico di Milano, 1993.
2. L. Bernardinello and F. De Cindio. A Survey of Basic Net Models and Modular Net Classes. In *LNCS 609: Advances in Petri nets*, pages 304–351. 1992.
3. Wilfried Brauer, Robert Gold, and Walter Vogler. A survey of behaviour and equivalence preserving refinements of Petri Nets. *LNCS*, 483:1–46, 1990.

4. R. Elmstrøm, R. Lintulampi, and M. Pezzé. Giving semantics to SA/RT by means of high-level timed Petri nets. *Real-Time Systems Journal*, 5(2-3):249–271, May 1993.
5. R. Elmstrøm, R. Lintulampi, and M. Pezzé. Automatic translation of SA/RT to high-level timed Petri nets. Technical Report IPTES-PDM-17-V2.3, Jan 1994.
6. M. Felder, C. Ghezzi, and M. Pezzé. Hierarchical Decomposition of High Level Timed Petri Nets. Technical Report IPTES-PDM-54-V2.0, Politecnico di Milano, Dec 1993.
7. C. Ghezzi, D. Mandrioli, S. Marasca, and M. Pezzé. A unified high-level Petri net formalism for time-critical systems. *IEEE SE*, 17(2):160–172, Feb. 1991.
8. D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, and M. Trachtenbrot. STATEMATE : A working environment for the development of complex reactive systems. *IEEE SE*, 16(4):403–414, Apr. 1990.
9. D. J. Hatley and I. A. Pirbhai. *Strategies for Real Time Specifications*. New York, Dorset House, 1987.
10. P. Huber, K. Jensen, and R. M. Shapiro. Hierarchies in coloured Petri nets. *LNCS*, 483:313–341, 1990.
11. J. S. Ostroff. Formal methods for the specification and design of real-time safety critical systems. *The journal of Systems and Software*, pages 33–60, April 1992.
12. J. Peleska, C. Huizing, and C. Petersohn. A Comparison of Ward and Mellor's Transformation Schema with STATE-and ACTIVITYCHARTS. Technical report, Christian-Albrechts-Univerdity Kiel, 1993.
13. Gernot Richter and Bruno Maffeo. Toward a Rigorous Interpretation of ESML–Extended Systems Modeling Language. *IEEE-SE*, 19(2):165–180, Feb 1993.
14. W. Rruyn, R. Jensen, D. Keskar, and P. Ward. ESML : An extended systems modeling language based on the data flow diagram. *ACM SIGSOFT, Software Engineering Notes*, 13(1):58–67, Jan. 1988.
15. J. Rushby. Formal Methods and the Certification of Critical Systems. Technical report, ACAA, Dec. 1993.
16. L. Shi. Uniting formal and structured design methods for real-time systems. Transfer report, Department of Computing, Manchester Metropolitan University, 1995.
17. W. M. P. van der Aalst. Interval Timed Coloured Petri Nets and their Analysis. In *LNCS 691: Application and Theory of Petri nets*, pages 453–472. 1993.
18. P. Ward and S. Mellor. *Structured Development for Real-Time Systems*, volume 1-3. New Jersey: Prence Hall, 1985.
19. Paul T. Ward. The transformation schema: an extension of the data flow diagram to represent control and timing. *IEEE SE*, 12(2):198–210, Feb 1986.
20. D. P. Wood and W. G. Wood. Comparative Evaluations of Specification Methods for Real-Time Systems. Technical report, SEI, CMU, Dec 1989.