Concurrent semantics for structured design methods

P.A. Nixon and L. Shi
Department of Computer Science,
Trinity College, The University of Dublin, Dublin 2, Ireland.
Department of Computing,
Manchester Metropolitan University, Manchester, U.K.
Email: Paddy.Nixon@cs.tcd.ie, L.Shi@doc.mmu.ac.uk

Abstract

Design methods can be ambiguous due to different interpretations of symbols or concepts. This paper presents a formal semantics for the Ward/Mellor Structured Analysis Method for Real Time systems. These semantics ensures that an unambiguous meaning can be attributed to a particular design. Specifically, it ensures that concurrent and real-time properties of the design can be captured and analysed. This paper concentrates on the concurrent properties.

Keywords

Structured methods, Concurrency, Real-time systems

1 INTRODUCTION

Due to the inherent complexity of the task, the design of quality software is notoriously difficult. Worse still, the need for concurrent or real-time properties to be modelled further complicates the task, Birkinshaw et al (1994). To ease the process many design methods have been proposed which provide the software designer with notations and structures for software construction. In the real-time domain the most popular are due to Hatley/Pirbhai (1987), and Ward/Mellor (1986) , which are based on original work by DeMarco (1978). These methods aim to allow all the properties of the proposed system, including concurrency and timeliness, to be expressed in clear, unambiguous manner.

Nevertheless, for many reasons both technical and cultural, these diagramatic designs can be misinterpreted. To ensure rigorous definitions of the designers tools are made, formal semantics can be associated with the graphical methodology. This allows the designer to continue with a prefered method, whilst introducing the ability to analyse the design from a formal persepective and so removing ambiguity in the design. Work done by Elmstrøm et al, (1994) has shown the benefits of applying Petri Nets as a semantics for real-time design methods as part of the IPTES project. Yet they point out that a major

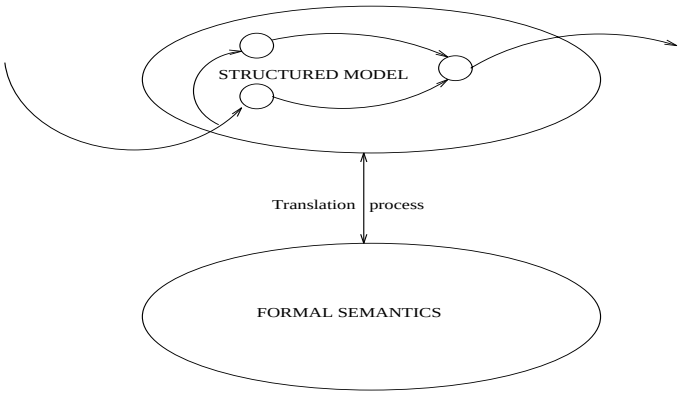

Figure 1 Notion of formal translation (Elmstrøm et al, 1994).

barrier to the widespread application of such formal semantics is that such processes can be very complex. A promising solution to the complexity problem is the use of compositional semantics. Namely, the translation process from design to formal representation is localised so that small parts of the design can be translated independently of other parts of the design. This independent translation can then lead to high speed translations and facilitate early analysis of parts of the design. This paper gives details of such a compositional approach. The paper concentrates on issues of concurrency, a complete description of the translation process is presented in Shi and Nixon (1995) and a detailed analysis of complexity for the translation is presented in Shi and Nixon (1995a)

2 BRIEF INTRODUCTION TO METHOD AND TIMED ER NETS

The particular method considered here is a the Extended Systems Modelling Language (ESML) of Rruyn et al (1988), which is a real-time extension of Tom DeMarco's structured analysis method based on data flow diagrams. Figure 2 gives the basic components of the model. The notations are defined as:

1. Transformations :
   A *data transformation* is an abstraction of a low-level system function, e.g. transforming data inputs into outputs, modifying stored data, or reporting some occurrence of event, etc..
   A *control transformation* controls the behaviour of other data/control transformations, e.g. deciding when or for how long the controlled transformations are active.
2. Data flows:
   A *discrete data flow (ddf)* is associated with a value only at discrete point of time, i.e. it is intermittently available. A *continuous data flow (cdf)* is associated with a value defined continuously over a time interval, i.e. it is continuously available.
   A *signal* is a non-value bearing data flow, it only reports that something(an event) has happened at discrete points of time.

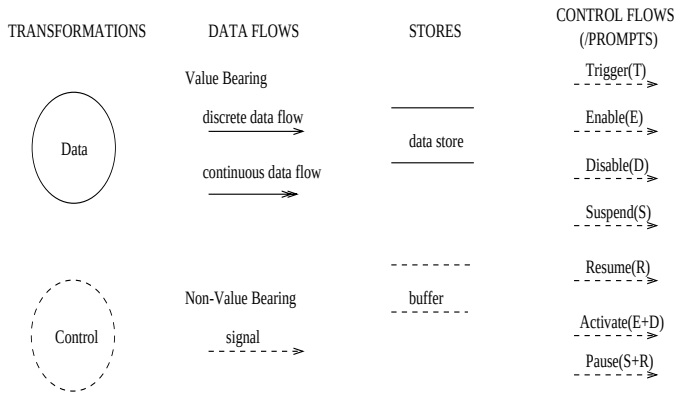

Figure 2 SA/RT notation

3. Stores:
   A *data store* acts as a repository for data that is subject to storage delay, and it is an abstraction of a file. A *buffer* actually is a special type of data store in which flows produced by transformations are subject to a storage delay before being *consumed* by other transformations. It is an abstraction of a stack or queue.
4. Control flows (or Prompts):
   Prompts represent control imposed by one control transformation on another data/control transformation, which include:

   - *Trigger*: A *Trigger* causes a flow transformation to perform a time-discrete action such as producing a data flow.
   - *Enable* and *Disable*: An *Enable/Disable* prompt initiates/terminates the activity of a transformation. When a transformation is disabled, it "forgets" any intermediate results and starts anew when enabled. *Activate* is a combination of *Enable* and *Disable*.
   - *Suspend* and *Resume*: *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 was left off when resumed. *Pause* is a combination of *Suspend* and *Resume*.

5. Flows from multiple sources and to multiple destinations may be represented by a splitting/merging notation.

These components are then used to construct a model, or specification, which captures the data flow through the proposed system. Control of the data flow is expressed by control transformations. Concurrency is not explicitly specified in the design but is often assumed. Equally, temporal characteristics are applied in the control aspects of the design. For complete details of the method the reader is refered to Rruyn et al (1988).

2.1 Timed ER Nets

A high level net model is used here and in Elmstrøm et al (1994) to give a semantic definition to the elements of ESML, in particular capuring the temporal and concurrent aspects of the design. The specific net model used is a Timed Entity Relation net. Entitity Relation (ER) nets are Petri Nets were tokens and transitions are given different interpretations. The tokens correspond to *environments*, essentially functions that can assocaite values with variables. Transistions have *actions* associated with them, which can effect the tokens. Timed ER (TER) Nets extend the ER notation by attaching a variable *chronos* to every token, the value of which is a timestamp. This variable is modified by transitions which produce the timestamps. The usual Petri Net conventions apply to TER Nets with some small differences; the interested reader is refered to Ghezzi er al (1991). As pointed out by Ghezzi, TER nets are general enough for the requirement specification of most complex real-time systems; yet, most of the usual temporal properties are undecidable in TER nets. But generally, the TER nets can assist the analysis of specifications in the following ways:

- to restrict the analysis to special decidable subcases corresponding to special classes of applications;
- to derive approximate solutions : by ignoring token values, we reduce TER nets 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 TER net;
- to provide interactive decision-support systems to assess them;
- to test specifications by simulation.

Timed ER nets have been chosen as the formal model for the semantics of ESML for these reasons.

3 TRANSLATION

A translation, or formal semantic definition, of a design must capture the intentions of the designer in an unambiguous manner. Consider the example, figure 3, to translate the simple construct of storing and retrieving data from a data store. Data transformation A places data in the store C and at some point in the future data transformation B retrieves it. The definition of ESML states that the data store can be viewed as a queue, so the order of arrival of the data is important. To ensure that the design has made the correct assumptions about the data store, and not overlooked the details of the ESML definition, a formal interpretation of the design can be extracted from the original and then animated using the petri net rules. Thus, the behaviour of the design can then be relayed to the designer.

There are many possible ways of describing a given meaning for a diagram in petri nets, some more complex than others. The complexity is the key issue which must be kept to a minimum. For instance, for the data store described above a possible petri net description could included at least n places, n transitions, and n arcs to capture the n element queue which defines the buffer (where n is the size of the buffer). Alternatively, a buffer could be described by a single place, two arcs, and two transitions (see figure 4). The TER

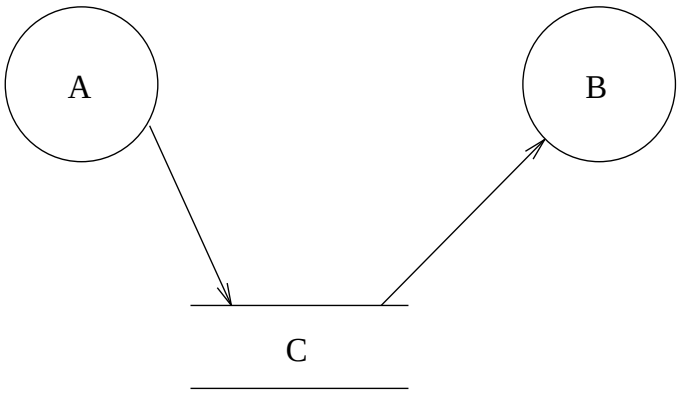

Figure 3 a simple construct

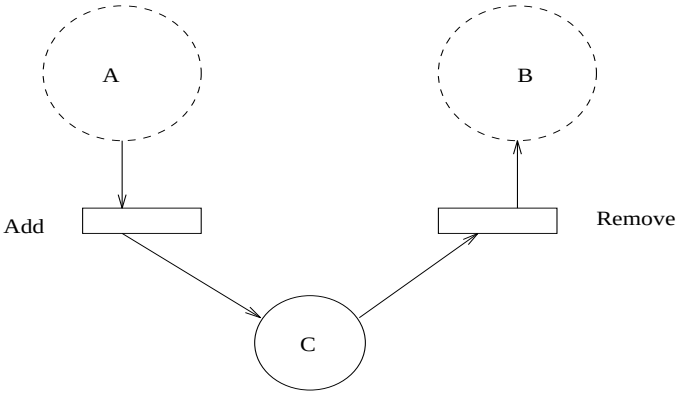

Figure 4 a simple semantics

net transitions correspond to functions. The transition for placing an element in the data store simple adds a timestamp, or chronos value, to the data stored. The transition to remove data ensures that the next data element removed is the elment with the lowest chronos value, i.e. the top of the queue.

A significant way of reducing complexity and improving efficiency is to use compositionality. This aims to modularise the semantics of the given structured method, ESML here, thus producing smaller nets to construct and analyse at a given time. The inteface between modules, or components, is well defined so that they can be combined without invalidating the work done on the indivdual components.

4 COMPOSITIONALITY

Some assumptions have to be made before any translation begins. Particularly, it has to be assumed that the SA/RT design is *complete*. By this it is meant that the usual top-down process of refining the abstract high level designs into more concrete low level refinements

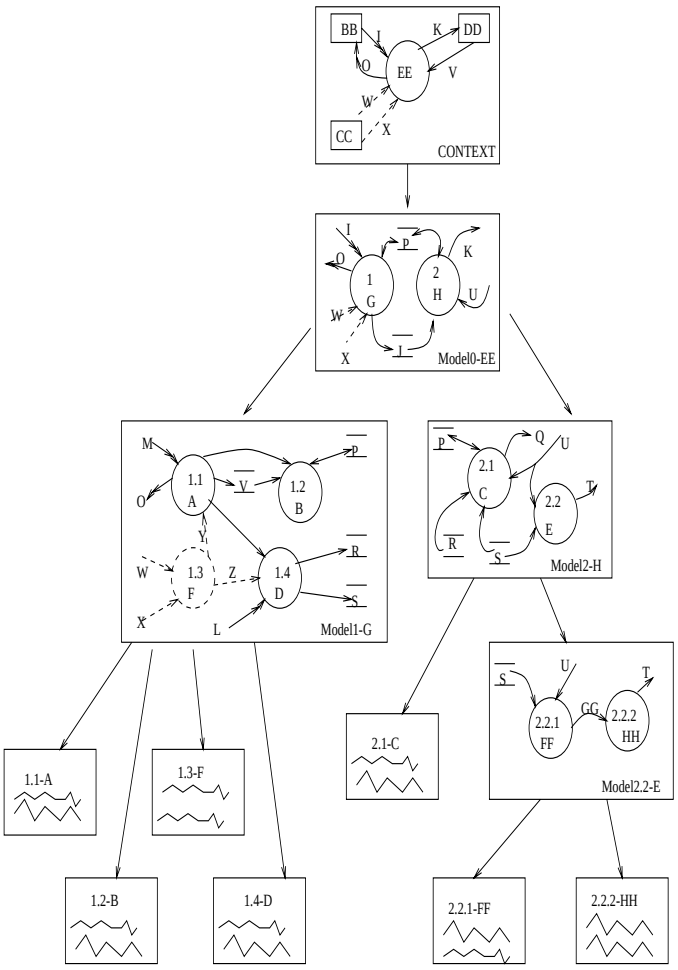

Figure 5 a hierarchical transformation schema

has taken place. Thus the translation process is dealing with a meaningful design, what ever that means. Also it is assumed that the complete specification is *flattened*. This means that the hierarchies have been removed in such a way as to leave a complete design. Figure 6 is a flattened version of Figure 5 taken from Ward and Mellor (1986).

The most important principle of the proposed translation of the is *compositionality* (or *locality*), i.e. the translation of each *component* is independent of other components. The concept of component is defined first and then the principle of compositionality is illustrated.

A *component* is either a data or control transformation together with all its inputs and outputs, or a merging or splitting structure representing a flow from multiple sources or to multiple destinations. Figure 7 illustrates a transformation schema 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.

In the translation, each *component* corresponds to a TER subnet, or simply a *TER*

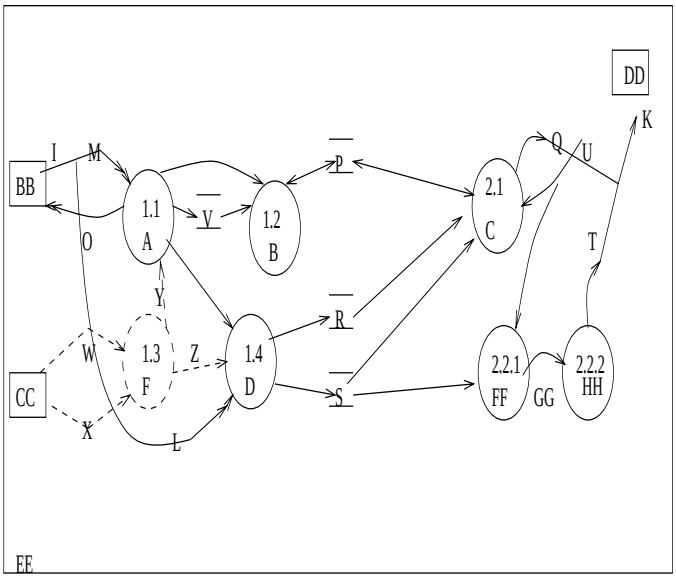

Figure 6 flattened transformation schema

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

Figure 8 illustrates a TER net structure corresponding to the transformation schema TS1 in figure 7. Rectangles $C_i (1 \leq i \leq 5)$ represent TER subnets for components C_i in figure 7. 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 shared by subnets $C1$ and $C5$ in the subnet for TS1. To simplify the situation, it is assumed in this example that each data/control flow corresponds to one place, the same the principle is followed when some flow corresponds to more than one place, or some group of flows share one place. Those places with no input or no output arcs, e.g. $cdf1$ and $ddf4$, correspond to flows to or from outside the schema. So the TER net as a whole is just the composition of all the five TER subnets that share interface places.

The assumption above that each flow corresponds to one place is not always true. For data flows other than data stores, two places are used for each flow as in figure 9(a,b,d); and for a control transformation, two places are used to represent all its input prompts, as illustrated in figure 9(e).

The mapping rules for flows and stores are illustrated in figure 9. For any place p, $P - cdf'$ is the complement (or empty) place of p, e.g. the token in p' in figure 9(b) denotes that no data is attached with cdf, in other words cdf is empty.

Lets consider the translation rules for *control flows* and $data flows$ in detail. Rules for other data flows in figure 9(a-c) are derived similarly and full details can be found in Shi and Nixon (1995).

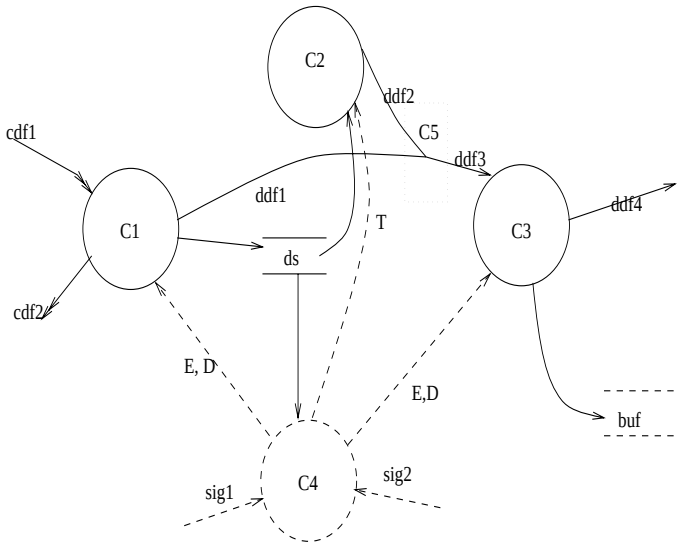

Figure 7 The concept of *component* in transformation schema TS1

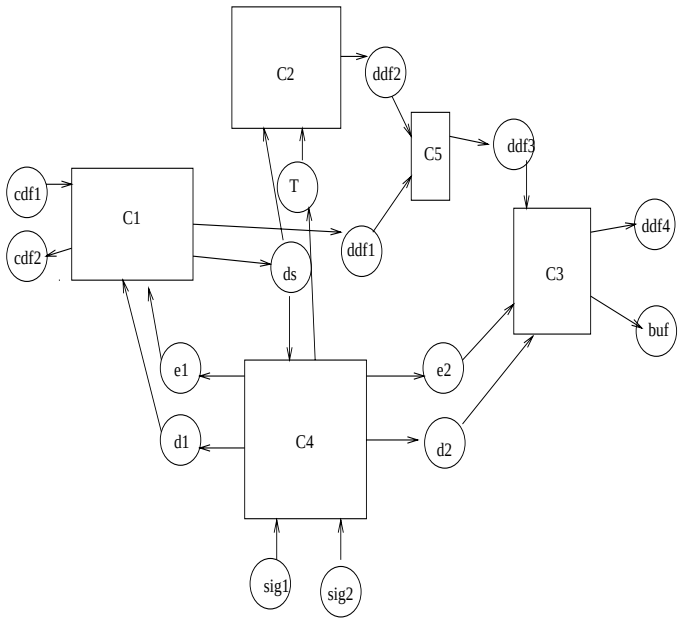

Figure 8 The compositional principle of translation for TS1

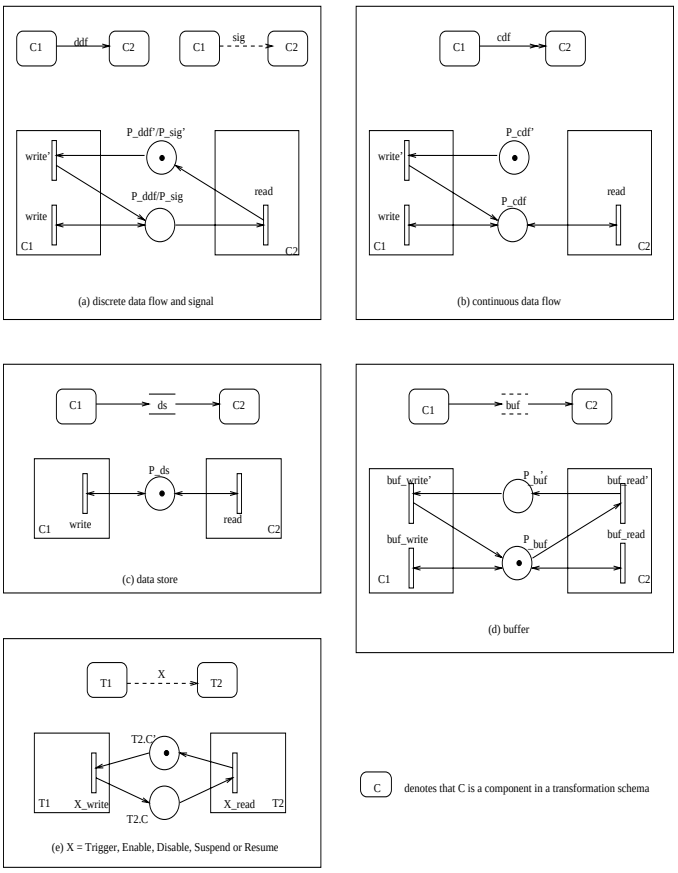

Figure 9 The translation principle of flows and stores

4.1 Translation of control flows

The translation in Elmstrøm et al (1994) is not compositional in that some SA/RT constructs are not 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 structurals of all the transformations that receive them. This problem is solved 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.

Figure 9(e) illustrates the translation method for control prompts. T1 is a control transformation which controls a (data or control) transformation T2. In the TER net, all the control prompts of T2 share two complementary places $T2.C$ and $T2.C'$. The token in $T2.C'$ indicates that no control prompt is present, and a token in $T2.C$ would carry the information of the control prompt to T2. The TER subnet for T1 has a transition X_write to produce the control prompt X; while the TER subnet for T2 has a transition X_read to consume the prompt X. But the form and number of such X_write or X_read transitions may vary, and they depend *only* on transformation T1 or T2.

Note separate places are not used here for different control prompts as for data flows.

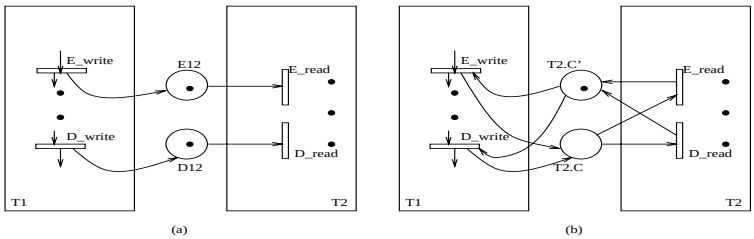

Figure 10 Translation principles for control prompts : an example

Consider the simple example as illustrated in figure 10(a), where two places $E12$ and $D12$ are used to represent the two prompts *Enable* and *Disable* sent by T1 to T2 respectively. The control prompts *Enable* and *Disable* are generated by the automaton of T1 in an orderly manner, but with the same timestamp. When *time* is not considered in the analysis, the control prompts may not be consumed in the same order as they are produced, since for 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. But usually it is required that the prompts be consumed in the same order as they are produced. This problem can be solved by our method by using shared places for all control prompts, as in figure 10(b), where $T2.C$ is $safe$ by initially putting one token in its complementary place $T2.C'$. So all the control prompts of T2 are accepted and consumed in an orderly manner by this translation.

The names, e_i, d_i, s_i, r_i and g_i (where $i = 1, 2, \ldots$) are used to indicate transitions that consume the *Enable*, *Disable*, *Suspend*, *Resume* and *Trigger* prompts respectively; arcs connected to these places can be omitted to ease reading.

4.2 Translation of Data Transformations

The next step is to consider the translation of data transformations. The concepts of *active data* and *active input* are important for understanding a data transformation. An active input arrives independently of any action of the receiving transformation, and activates the transformation when it is available (i.e. in *idle* state). An active data output is created by the activity of a data transformation, and can be an active data input for another transformation. The following definitions are assumed:

- Active data : a *ddf* not connected with data store or a signal
- Nonactive data : a *cdf* or a *ddf* connected with data store
- Active input : an active data input, or a *Trigger* prompt input *

According to the definitions above, data transformations can be placed into two classes:

1. Data transformations with an active input
   inputs ::
   $active\_data\{nonactive\_data\}[Activate][Pause]$

*Note that the definition of active input in Ward (1986) does not include *Trigger*. Here we include *Trigger* to make the following description simpler.

$| Trigger\{nonactive\_data\}[Pause]$
$outputs :: \{active\_data\}^+\{data\_store\}$

2. Data transformations without active input
   $inputs :: \{nonactive\_data\}^+[Activate][Pause]$
   $outputs :: \{nonactive\_data\}^+\{active\_data\}$

When there is more than one active data in the output part of a data transformation, they are interpreted as alternatives; only one of them can be produced at a given time. We will consider the translation of the first of these here, translations for the other data transformations can be found in Shi and Nixon (1995).

5 CONCLUSIONS

Whatever method is used, the complexity of simulation and analysis of Petri nets grows with their size, especially the numbers of transitions and arcs. So the efficiency of simulation and analysis crucially depends on the efficiency of the translation.

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

- Assisting the interactivity of the development process of SA/RT specifications
  The development of an 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 specification may be developed and its critical properties need to be analyzed first. The 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 any part of the specification only results in *localized* modifications of the underlying net and other parts, including their properties, will be kept intact. Thus the compositional translation is essential to the compositional/modular development and analysis of specifications.

In summary, the compositional and efficient translation given here can benefit the analysis and the development of SA/RT specifications. The work has also highlighted the importance of the modularity/compositionality of TER nets.

Future work will include investigation into the compositionality of temporal properties and their transformation into HLTPNs. Work is also beginning on a prototype tool for automating the process presented. On a more theoretical note, our research would benefit from more work on the efficient analysis of high-level timed Petri nets and especially their modular/compositional analysis. finally, the authors believe the translation is not tied to the SA/RT described in the paper and work is proceeding to substantiate this hypothesis.

6 REFERENCES

W. Rruyn, R. Jensen, D. Keskar, and P. Ward. (1988) *ESML : An extended systems modeling language based on the data flow diagram.* ACM SIGSOFT, Software Engineering Notes, 13(1):58–67, Jan. 1988.

C. Ghezzi, D. Mandrioli, S. Marasca, and M. Pezzé. (1991) *A unified high-level Petri net formalism for time-critical systems.* IEEE SE, 17(2):160–172, Feb. 1991.

C I Birkinshaw P R Croll D G Marriott and P A Nixon, (1994) *Engineering safety-critical parallel systems* , In Information and Software Technology, Vol. 36, No. 7, pp. 449-456, 1994.

R. Elmstrøm, R. Lintulampi, and M. Pezzé. (1994) *Automatic translation of SA/RT to high-level timed Petri nets.* Technical Report IPTES-PDM-17-V2.3, Odense, Jan 1994.

T. DeMarco. (1978) *Structured Analysis and System Specification* . New Jersey, Prence-Hall.

D. J. Hatley and I. A. Pirbhai. (1987) *Strategies for Real Time Specifications.* New York, Dorset House.

P. Ward and S. Mellor. (1985) *Structured Development for Real-Time Systems*, volume 1-3. New Jersey: Prence Hall.

Paul T. Ward. (1986) *The transformation schema: an extension of the data flow diagram to represent control and timing.* IEEE SE, 12(2):198–210, Feb 1986.

L. Shi and P. Nixon. (1995a) *An improved translation from SA/RT model to high-level timed Petri nets.* In the proceedings of FME'96, to appear in LNCS, Springer Verlag.

L. Shi and P. Nixon. (1995) *Uniting formal and structured design methods for real-time systems.* Technical report, Dept. of Computing, Manchester Metropolitan University, 1995.

7 BIOGRAPHY

Patrick Nixon recieved his B.Sc in Computer Science from Liverpool University in 1990 and his Ph.D in Computer Science from Sheffield University in 1994. He was a lecturer in Computing at Manchester Metropolitan University until 1995, when he took up a post of lecturer at Trinity College Dublin. His research interests include software engineering, parallel and distributed computing, distributed object systems, and applied formal methods.

Lihua Shi is currently a Ph.D student in Computing at Manchester Metropolitan University. Prior to this she recieved her B.Sc in Computer Science and a Masters in Software Engineering from East China Normal University, and was a lecturer there until 1994.

ACKNOWLEDGEMENTS

This research is supported by a research scholarship from Manchester Metropolitan University for Lihua Shi.