Communicating Transaction Processes

Abhik Roychoudhury P.S. Thiagarajan
School of Computing
National University of Singapore
{abhik,thiagu}@comp.nus.edu.sg

Abstract

Message Sequence Charts (MSC) have been traditionally used to depict execution scenarios in the early stages of design cycle. MSCs portray inter-process (inter-object) interactions. Synthesizing intra-process (intra-object) executable specifications from an MSC-based description is a non-trivial task. Here we present a model called Communicating Transaction Processes *(CTP) based on MSCs from which an executable specification can be extracted in a straightforward manner. Our model describes a network of communicating processes as a collection of high-level labeled transition systems, where processes interact via common action labels. Each action is a non-atomic interaction which is described by a guarded choice of MSCs. Thus our model achieves a separation of concerns: the high-level transition systems depicting intra-process control flow, while the actions in the transition system capture inter-process interaction via MSCs. We show how to extract an ordinary Petri net from a CTP model thereby leading to a standard operational semantics. We also discuss the connection of our formalism to Live Sequence Charts, an extension of MSCs which also has an executable semantics.*

1 Introduction

Message Sequence Charts (MSCs) are an attractive visual formalism which are used in the early design stages of reactive systems. They portray scenarios that arise from component interactions and hence can be used to capture requirements and test cases. MSCs and a related mechanism called HMSCs (High-level Message Sequence Charts) have been standardized [15] for specifying telecommunication software. A version of MSC called Sequence Diagram is a behavioral diagram type used in UML [6].

In all these settings, MSCs are used to capture system requirements. To move towards an implementation, one must obtain an executable specification which is related in some fashion to the MSC-based requirements. The key difficulty here, as identified in [8], is that the inter-object interactions described in form of MSCs must be related to -or synthesized as- executable specifications given in terms of intra-object behaviors, say, one statechart for each object. This is a difficult problem and it has been studied in various limited contexts [1, 8, 10, 11].

In this paper, we propose using MSCs to construct executable specifications in a more direct fashion. The main idea is to use traditional methods to capture the control flow of the system components while using MSCs to describe the *non-atomic* component interactions. Among the various possibilities for describing the control flow in a multi-component system, we choose the well-known model of synchronized product of transition systems; a network of labeled transition systems that synchronize on common actions. We however impose two restrictions on the control flow; a minor technical one for convenience but also a major one in that the choice as to which interactions to take part in is made by the components in a *local* fashion. In Petri net terms, this is the so called free choice property [5]. We then refine each common abstract action γ involving a set of agents into a *transaction scheme* T_γ. Each such scheme is a guarded choice of MSCs. The life lines of the MSCs in T_γ will correspond to the set of agents participating in the common action γ. Each guarded MSC in T_γ, called a *transaction* will represent one possible interaction and will involve a complex flow of data and control signals. *When* a transaction scheme is to be executed is determined by the control flow in the high level product transition system. As to *which* transaction in T_γ will be chosen to be executed is determined by the guards which are propositional formulas built out of atomic propositions. The truth values of these atomic propositions (and hence those of the guards) will capture abstracted properties of the values of the variables and control states associated with the agents. A central feature of the model is that both the control flow and the evaluation of the guards (which then leads to the execution of a specific transaction within a transaction scheme) are done in a distributed and asynchronous manner. In broad terms, this is our Communicating Transaction Processes (CTP) model.

Our strategy of striking a balance between control flow and component interactions yields a model which is flexible, powerful and at the same time amenable to formal analysis and synthesis. Indeed, the problem of extracting an executable specification from the CTP model becomes very manageable and amenable to automation as we show in section 3. Our main point of reference for this work is the formalism of Live Sequence Charts [4] due to Damm and Harel in which the component interactions are elaborated in a powerful way using the LSC language while the control flow information is completely suppressed. On the other hand, in models such as Petri nets and distributed transitions systems, the focus is on a detailed presentation of control flow while the only mechanisms for capturing component interactions are the atomic notions of synchronizing transitions and shared buffers.

An alternative way to use MSCs to capture system behavior is via HMSCs. However, an HMSC is just a presentation of a *collection* of MSCs. The problem of extracting an executable specification from an HMSC is a non-trivial one. A detailed discussion as well results can be found in [2] regarding the possible relationships between HMSCs and executable specifications. There are a variety of choices available as to the executable specification mechanism such as state charts [11], Petri nets [2] and networks of automata communicating through FIFOs [10, 1]. Many versions of this synthesis problem are not even decidable [10, 1, 2]. In contrast, as we shall show, we can extract an executable specification in the form of a finite Petri net whose set of reachable markings may be infinite from a CTP model effectively and quite easily.

A number of the basic features of the CTP model are also exhibited by the model called Cyclic Transaction Processes considered in our earlier paper [13]. The control flow in [13] was restricted to be *cyclic* and the operational semantics was based on infinite state labeled transition systems with a complex mechanism for keeping track of the states of the partially executed charts. In the present paper, we do not require the control flow within a component to be cyclic. Further, we first collapse together the guarded set of MSCs into a single *event structure* which is a standard operational model [12] and then easily extract from the resulting model a conventional *finite* Petri net.

In the next section we introduce the CTP model while illustrating its main features with simple examples. In section 3 we provide the operational semantics of the CTP model in terms of ordinary Petri nets. The key step in this process is converting each transaction scheme into a finite labeled event structure. In section 4 we present some important behavioral properties of the model and the techniques that are currently available for verifying these properties. We also provide a more detailed comparison with the closely related formalism of LSCs. The concluding section sketches our current experimental efforts based on the CTP model and provides pointers to future research.

2 The CTP Model

Being based on MSCs, the CTP model captures non-atomic inter-process communications. However, in order to be amenable to efficient distributed implementation, this is combined with notations for describing intra-process control flow. As a starting example, consider the specification shown in Figure 1.[1] Each process interacts with the other process and then performs some internal action; this is done repeatedly. Note that the inter-process interaction and the internal actions have been separated into distinct units. A number of processes P will be involved in the execution of a chart. A process p which takes part in such an execution might next participate in an execution involving a different set of processes, say Q.

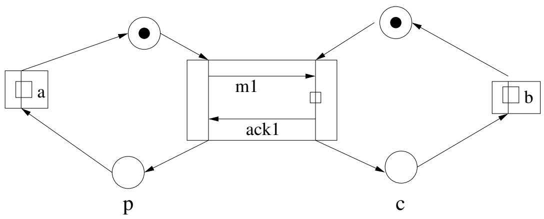

Figure 1. Inter-process communication and intra-process control flow

The organization of interactions into the various units is up to the convenience of the designer. For example, in Figure 1 we could have made the actions a and b also to be part of the chart involving processes p and c. Note also that the example shown in Figure 1 is essentially a Petri net where the local control states in each process are the places of the net (denoted by circles). Each (top-level) transition of this net is, in general, a collection of Message Sequence Charts at the refined level. A particular execution of the high-level transition, is an abstraction of the activity in which one of the charts associated with the high-level transition is chosen and executed. In the example of Figure 1 each net transition has a single chart associated with it. (An internal action is a degenerate chart involving just one process executing just one action). The choice as to which chart -in case more than one chart is associated with a transition- is based on the

[1]We adopt the usual MSC convention that horizontal and downward sloping arrows denote message send-receives between two processes. Further, a □ symbol on a single vertical line denotes an internal action (such as actions a and b in Figure 1). We denote a control state of a process as a circle.

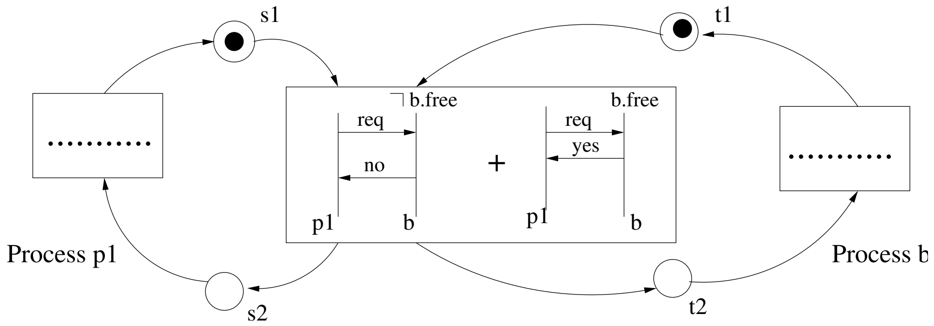

Figure 2. Choice of Inter-process communication

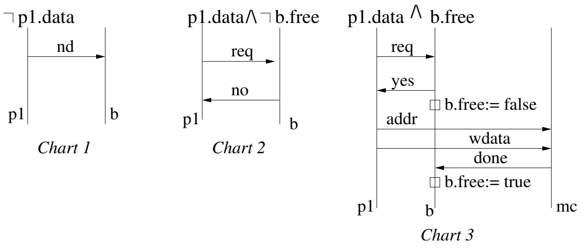

Figure 3. Distributed nature of choice in a net transition

value of the local variables of the processes. This is illustrated in Figure 2 (the choice is decided by the value of the variable $free$ belonging to process b). If $b.free$ holds once control reaches $s1$ and $t1$ respectively, we must execute the right-hand chart of Figure 2.

In general, the choice of which chart is executed at a particular net transition is a distributed one. Let the charts contained in a particular net transition be as shown in Figure 3. If $p1.data$ holds then chart 1 is ruled out. However, still we do not know whether chart 2 or chart 3 will be executed. This will depend on the value of variable $free$ in process b. As shown in Figure 3, each MSC associated with a net transition has a guard (which we will also refer to as a pre-condition). This guard is a distributed one in that it will in general involve propositions belonging to different processes participating in the MSC. At the end of the execution of a chart, the truth values of the various propositions will be set to new values in general.

2.1 The Definition of the CTP Model

A product transition system is a network of sequential transition systems that synchronize on common actions. The CTP model is obtained by taking a restricted class of product transition systems and refining the common actions into collections of guarded MSCs called *transaction schemes*.

Fix a finite set of process names $\mathcal{P}$ with p, q ranging over $\mathcal{P}$. Fix also a finite set of labels Γ and a family $\{\Gamma_p\}_{p \in \mathcal{P}}$ with each Γ_p a subset of Γ and $\bigcup \Gamma_p = \Gamma$. This induces the function loc which assigns to each label in Γ the set of agents that participate in the execution of that action. This function is given by: $loc(\gamma) = \{p \mid \gamma \in \Gamma_p\}$. If $loc(\gamma) = \{p\}$ then γ will be called p-local action. The members of Γ will be treated as abstract action labels in the first step where we define the control flow model. In the second step they will be interpreted as transaction schemes and further elaborated. Γ_p is the set of (abstract) actions that the process p will participate in.

Anticipating the need to build guards in the second step, we also fix AP_p a finite set of atomic propositions , one for each p and set $AP = \bigcup_{p \in \mathcal{P}} AP_p$. If $P \subseteq \mathcal{P}$ then we let $AP_P = \bigcup_{p \in P} AP_p$. By convention, we shall write $AP_{\mathcal{P}}$ as AP. Each subset of AP_P will be called P-valuation. If $P = \{p\}$ is a singleton we will write p-valuation.

For each p let $TS_p = \langle S_p, \Gamma_p, \longrightarrow_p, init_p, V_{p,in} \rangle$ be a finite-state transition system over Γ_p with an initial p-valuation. In other words, S_p is a finite set of states , $init_p \in S_p$ is the initial state, $\longrightarrow_p \subseteq S_p \times \Gamma_p \times S_p$ denotes the transition relation and $V_{p,in} \subseteq AP_p$ is the initial valuation of atomic propositions in AP_p. In this paper we will be only interested in control flows in which the choices as to which transaction scheme that p will take part in is decided locally by p (free choice). Further, to avoid notational clutter, we will require that each member of Γ_p is the label of at most one transition in TS_p. These two restrictions on TS_p can be formalized as follows.

(1) if $s \xrightarrow{\gamma}_p s_1$, $s \xrightarrow{\gamma'}_p s_2$ and $s_1 \neq s_2$, then γ and γ' are p-local actions. Thus, $loc(\gamma) = loc(\gamma') = \{p\}$.

(2) If $s_1 \xrightarrow{\gamma}_p s_2$ and $s_3 \xrightarrow{\gamma}_p s_4$ then $s_1 = s_3$ and $s_2 = s_4$.

Definition 1 (Product Transition System) *A product transition system over* $(\{\Gamma_p, AP_p\})_{p\in\mathcal{P}}$ *is denoted as* $\{TS_p\}_{p\in\mathcal{P}}$ *where each* $TS_p = \langle S_p, \Gamma_p, \longrightarrow_p, init_p, V_{p,in}\rangle$ *is as specified previously. This product transition system is defined to be the transition system* $\langle S, \Longrightarrow, init, V_{in}\rangle$ *where:*

- $S = \prod_{p\in\mathcal{P}} S_p$
- $init = \prod_{p\in\mathcal{P}} init_p$
- $s \stackrel{\gamma}{\Longrightarrow} s'$ *iff* $s(p) \stackrel{\gamma}{\longrightarrow}_p s'(p)$ *if* $p \in loc(\gamma)$ *and* $s(p) = s'(p)$ *otherwise. The notation* $s(p)$ *denotes local state of process* p *in global control state* s.
- $V_{in} = \bigcup_{p\in\mathcal{P}} V_{p,in}$

Next, we need to define transaction schemes. We begin with the standard notion of MSCs which we shall view, in the present context, as certain kinds of labeled partial orders. Their visual representation will be as shown in the various examples already. We shall use Σ_p to denote the set of actions executed by the process p. It consists of actions of the form $\langle p!q, m\rangle$, $\langle p?q, m\rangle$ and $\langle p, a\rangle$ where M is an alphabet of messages and Act is an alphabet of internal actions. The communication action $\langle p!q, m\rangle$ stands for p sending the message m to q and $\langle p?q, m\rangle$ stands for p receiving the message m from q. On the other hand, $\langle p, a\rangle$ is an internal action of p with a being the member of Act being executed. We set $\Sigma = \bigcup_{p\in\mathcal{P}} \Sigma_p$. We also denote the set of channels $Chan$ given by $Chan = \{(p,q) \mid p \neq q\}$.

Turning now to the definition of MSCs, we define Σ-labeled poset to be a structure $Ch = (E, \leq, \lambda)$ where $(E, \leq)$ is a poset and $\lambda : E \to \Sigma$ is a labeling function. For $X \subseteq E$ we define $\downarrow(X) = \{e' \mid e' \leq e \text{ for some } e \in X\}$. When $X = \{e\}$ is a singleton we shall write $\downarrow(e)$ instead of $\downarrow(\{e\})$. We say that X is *downclosed* in case $X = \downarrow(X)$. For $p \in \mathcal{P}$ and $a \in \Sigma$, we set $E_p = \{e \mid \lambda(e) \in \Sigma_p\}$. These are the events that p takes part in. Further, $E_{p!q} = \{e \mid e \in E_p \text{ and } \lambda(e) = \langle p!q, m\rangle \text{ for some } m \in M\}$. Similarly, $E_{p?q} = \{e \mid e \in E_p \text{ and } \lambda(e) = \langle p?q, m\rangle \text{for some } m \in M\}$. We define for any channel $c = (p,q)$, the communication relation R_c as: $(e, e') \in R_c$ iff $\mid \downarrow(e) \cap E_{p!q} \mid = \mid \downarrow(e) \cap E_{q?p} \mid$ and $\lambda(e) = \langle p!q, m\rangle$ and $\lambda(e') = \langle q?p, m\rangle$ for some message m.

An MSC (over $(\mathcal{P}, M, Act)$) is a Σ-labeled poset $Ch = (E, \leq, \lambda)$ which satisfies:

(1) $\leq_p$ is a linear order for each p where $\leq_p$ is $\leq$ restricted to $E_p \times E_p$.

(2) Suppose $\lambda(e) = \langle p?q, m\rangle$. Then $\mid \downarrow(e) \cap E_{p?q} \mid = \mid \downarrow(e) \cap E_{q!p} \mid$ and there exists $e' \in \downarrow(e)$ such that $\lambda(e') = \langle q!p, m\rangle$ and $\mid \downarrow(e) \cap E_{q!p} \mid = \mid \downarrow(e') \cap E_{q!p} \mid$.

(3) For every p, q with $p \neq q$, $\mid E_{p?q} \mid = \mid E_{q!p} \mid$.

(4) $\leq = (\leq_{\mathcal{P}} \cup R_{Chan})^\star$ where $\leq_{\mathcal{P}} = \bigcup_{p\in\mathcal{P}} \leq_p$ and $R_{Chan} = \bigcup_{c\in Chan} R_c$.

This definition assumes a FIFO discipline for each channel. Other variations can also be dealt with easily. In what follows, we let $agents(Ch)$ denote the set of agents participating in the MSC $Ch = (E, \leq, \lambda)$ and define it as $agents(Ch) = \{p \mid E_p \neq \emptyset\}$.

Definition 2 (Transaction Scheme) *A Transaction Scheme* γ *is a finite collection of guarded Message Sequence Charts* $\{[I^i : Ch^i]\}_{i=1}^k$. *Each* Ch^i *is an MSC over* $(\mathcal{P}, M, Act)$. *Each* I^i *is of the form* $\bigwedge_{p\in agents(Ch^i)} I_p^i$ *where* I_p^i *is a propositional logic formula built from the propositions in* AP_p.

For each chart Ch^i in a transaction scheme, we have only mentioned a pre-condition. We have not specified the valuations of atomic propositions upon exiting from a chart. However, send and receive actions have a well-defined meaning. We can also assume that the internal actions are expressed in a standard imperative language. The operational semantics of this imperative language then lends a meaning to the internal actions. Consequently each event in a chart will have a well-defined effect on the truth-values of the local atomic propositions and as a sum total of these effects, we can associate with each chart an output valuation O^i. If more than one output valuation is possible, we can consider them as different transactions. Hence in what follows, we will assume that a transaction scheme is of the form $\{[I^i : Ch^i : O^i]\}_{i=1}^k$ over $(\mathcal{P}, M, Act)$.

Finally, we can now define a Communicating Transaction Processes (CTP) system model as follows.

Definition 3 (CTP System Model) *A CTP is a product transition system* $\{TS_p\}_{p\in\mathcal{P}}$ *over* $(\Gamma, \mathcal{P})$ *where* Γ *is a finite set of transaction schemes over* $(\mathcal{P}, M, Act)$. *Further, for each* $\gamma \in \Gamma$, $agents(\gamma) = loc(\gamma)$.

Here $loc(\gamma)$ is as before where γ is viewed as an abstract action label in high level product transition system; $agents(\gamma)$ is the set of agents participating in some transaction associated with the transaction scheme γ. Let $\gamma = \{[I^i : Ch^i : O^i]\}_{i=1}^k$. Then $agents(\gamma) = \bigcup_{i=1,2,\ldots n} agents(Ch^i)$. Thus the restriction in the above says that the processes taking part in a high level transition in the control flow model are the same as the processes taking part in the transaction scheme associated with this high level transition. This restriction still allows the designer to reorganize the distribution of transactions across the various transaction schemes. Indeed, in the extreme case can one collapse the whole model into a single messy transaction scheme with just one control state for each process! A subclass of CTPs can be obtained by requiring $loc(\gamma) = agents(Ch^i)$ for each i above. In such CTPs one cannot arbitrarily rearrange the transactions.

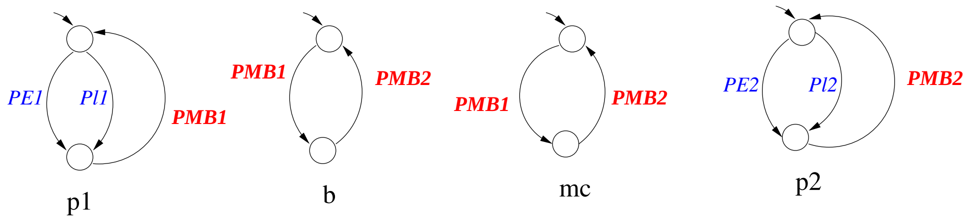

Figure 4. CTP system model of Multiprocessor example (common actions are shown in bold)

2.2 A Simple Example

Consider two processors communicating with a shared memory via a bus. The bus controller serves as an arbiter for bus access and serializes the bus access requests by the two processors. The memory controller provides data to the processors for read requests and commits data for write requests. Two of the simple schemes of this system are shown in Figure 5 whereas the high-level control flow is as shown in Figure 4. The two processors are denoted by processes $p1$ and $p2$; b is the bus controller and mc is the memory controller.

The schemes *Pl1* and *PE1* are local schemes in which only $p1$ participates as shown Figure 5. They represent local choices. Scheme *Pl1* is executed when processor $p1$ has data to transfer ($p1.data$ is true). Thus, this scheme consists of a single degenerate MSC. The MSC consists of a single internal action which is a no-op. If processor $p1$ has no data to transfer, then scheme *PE1* is executed. This scheme consists of two MSCs. The choice of which chart is executed is made by the environment. If the environment (*i.e.* the application running on the processor) has data to transfer then $p1.data$ is set; otherwise it remains reset. In this simple example, whether *Pl1* or *PE1* is executed, the process $p1$ next participates in the same transaction scheme, namely, *PMB1*. In general however, this branching in the control flow could lead to different transaction schemes being chosen.

Since the processors have similar behavior, the scheme *PMB1* is identical to *PMB2* except that process $p1$ is replaced by $p2$. (Similar remarks hold for *PE1* and *PE2* as well as *Pl1* and *Pl2*) The scheme *PMB1* is the one shown earlier in Figure 3. This scheme involves a decision by the bus controller b about granting bus access to $p1$. In Figure 3, $p1.data$ holds when $p1$ has data to transfer; $b.free$ holds when the bus is free for transfer. After the transfer the bus is set free. In this simple example, we have assumed that the bus is released after every access, and only write transfers are shown. We have also used our formalism to model more complex interactions such as burst transfers and split transfers [13].

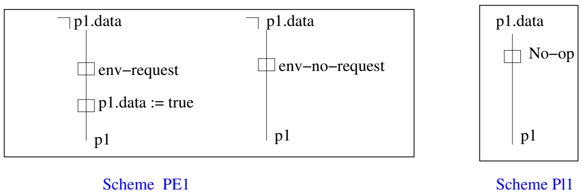

Figure 5. Local Choices and Environment Interaction in Transaction Schemes of Fig. 4

3 The Petri Net Semantics

In this section, we provide an operational semantics for the CTP model in terms of Petri nets. A key step in our semantics is to combine the different guarded transactions within a transaction scheme into a single entity. This entity will consist of a parallel composition of computation trees; one computation tree for each process that participates in the transaction scheme. Finite labeled *event structures* can be conveniently used for representing such a parallel composition. We define:

Definition 4 (Event Structure) *An* event structure *is a triple $ES = (E, \leq, \#)$ where E is a set of events, $\leq \subseteq E \times E$ is a partial ordering* causality *relation and $\# \subseteq E \times E$ is a* conflict *relation which is required to satisfy the following conditions: (a) $\#$ is irreflexive and symmetric, and (b) conflict is inherited via causality, that is $(e_1 \# e_2 \wedge e_2 \leq e_3) \Rightarrow e_1 \# e_3$.*

As a related notion, we define a Σ-labeled event structure to be a structure $ES = (E, \leq, \#, \Lambda)$ where $(E, \leq, \#)$ is an event structure and $\Lambda : E \rightarrow \Sigma$ is a labeling function. In diagrams, as illustrated in Figure 6, it will be convenient to represent the causality and conflict relation in a minimal fashion. To this end, we define the minimal causality relation $\lessdot$ via:

- $e \lessdot e'$ iff $e < e'$ and for every e'', if $e \leq e'' \leq e'$ then $e = e''$ or $e'' = e'$.

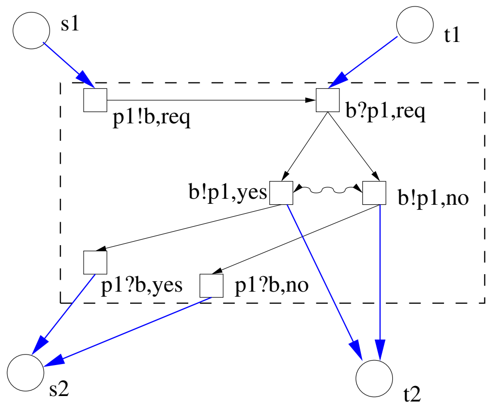

Figure 6. Event Structure for Transaction scheme in Figure 2 (shown in dashed box)

The event structure corresponding to the transaction scheme in Figure 2 is shown in Figure 6. The minimal causal relationship is captured by unidirectional arrows. The minimal conflict relation (conflicts which are not inherited via causality) is captured by curved bidirectional arrows. The way the minimal and maximal events of this event structure are connected to the input and output control states of the transaction scheme are also shown.

3.1 Constructing Event structures

In order to define our operational semantics, we first recall that $AP = \bigcup_{p \in \mathcal{P}} AP_p$ is the set of atomic propositions. Let T be a *transaction scheme* (refer Definition 2) of the form $T = \{I^i : Ch^i : O^i\}_{i=1}^{n}$ where each I^i is a propositional formula built out of AP, each $Ch^i = (E^i, \leq^i, \lambda^i)$ is a chart over $(\mathcal{P}, M, Act)$ and each O^i is a subset of AP. We let $T^i = [I^i : Ch^i : O^i]$ for each i and call I^i, the *input guard*, Ch^i the *body* and O^i the *output valuation* of the transaction T^i. We will assume without loss of generality that the sets $\{E^i\}_{i=1,\ldots,n}$ are pairwise disjoint.

We construct the labeled event structure $ES_{\mathrm{T}} = (E, \leq \#, \lambda)$ to be associated with a transaction scheme T as follows. The set of events E is obtained from the event sets E^i $(i = 1, \ldots, n)$ but after identifying events that have isomorphic pasts. Consequently we start with a set X whose elements will be of the form (e, i, P, V_P) where $e \in E^i$, $P = \{p \mid \exists e' \in E^i_p \text{ and } e' \leq^i e\}$ and V_P is a P-valuation such that $V_P \models \bigwedge_{p \in P} I^i_p$. Note that E^i_p is the set of events in E^i in which p participates, that is, for any event $e \in E^i_p$, the label $\lambda^i(e)$ is of the form $\langle p!q, m\rangle$ or $\langle p?q, m\rangle$ or $\langle p, a\rangle$. Actually, the second and third components in (e, i, P, V_P) are redundant but we will carry them for convenience. Next let $x = (e, i, P, V_P)$ and $y = (d, j, Q, V_Q)$ be in X. Then $x \equiv y$ iff $\downarrow(e)$ in Ch^i is isomorphic to $\downarrow(d)$ in Ch^j in the obvious sense. We shall denote the $\equiv$-equivalence class containing x as $[x]$.

Set of Events: We define E, the set of events of ES_{T} to be the $\equiv$-equivalence classes of X. Thus, $E = \{[x] \mid x \in X\}$.

Causality Relation: Let $[x], [y]$ be in E. Then $[x] \leq [y]$ iff there exists (e, i, P, V_P) in $[x]$ and (d, j, Q, V_Q) in $[y]$ such that $i = j$, $e \leq^i d$ and $V_Q \cap AP_P = V_P \cap AP_P$.

Conflict Relation: First we define the relation $\widehat{\#}$ to be the least subset of $E \times E$ which satisfies the following. Suppose $[x], [y] \in E$ are such that $[x] \not\leq [y]$ and $[y] \not\leq [x]$. Furthermore, there exist (e, i, P, V_P) in $[x]$ and (d, j, Q, V_Q) in $[y]$ such that $e \in E^i_p$ and $d \in E^j_p$ for some p but $i \neq j$. Then $[x]\widehat{\#}[y]$. We now define the conflict relation $\#$ as the least subset of $E \times E$ which (a) contains $\widehat{\#}$, (b) is a symmetric relation, and (c) inherits through causality, that is, $[x]\#[y]$ and $[y] \leq [z]$ implies $[x]\#[z]$.

Labeling Function: Finally, the labeling function Λ is given by: $\Lambda([(e, i, P, V_P)]) = \lambda^i(e)$.

Lemma 3.1 *$ES_{\mathrm{T}} = (E, \leq, \#, \Lambda)$ is a labeled event structure.*

Proof: Due to the isomorphism condition imposed in the definition of $\equiv$, it is easy to observe that $\leq$ is a partial ordering relation. From the definition of the relation $\#$ it is symmetric and is inherited via $\leq$. We need to show that it is irreflexive. Assume for contradiction that there exists $[x]$ such that $[x]\#[x]$. In this case it is not difficult to see there exist $[y]$ and $[z]$ such that $[y]\widehat{\#}[z]$ and $[y] \leq [x]$ and $[z] \leq [x]$. This implies there exist (e, i, P, V_P) in $[y]$ and $(e1, i, P1, V_{P1})$ in $[x]$ such that $e \leq^i e1$. Further, there exist (d, j, Q, V_Q) in $[z]$ and $(d1, j, Q1, V_{Q1})$ in $[x]$ such that $d \leq^j d1$. Then by the definition of the $\equiv$ relation, it follows that there exists $(d', i, Q', V_{Q'})$ in $[z]$ such that $d' \leq^i e1$. But then from the definition of $\widehat{\#}$ it follows that there exists p such that $e \in E^i_p$ and $d' \in E^i_p$. This leads to $e \leq^i d'$ or $d' \leq^i e$ which in turn leads to $[y] \leq [z]$ or $[z] \leq [y]$ contradicting $[y]\widehat{\#}[z]$. The fact that the labeling function is well-defined is obvious. □

3.2 The Labeled Petri Net Semantics

We construct the Petri net semantics for the CTP model in three steps. First we convert each labeled event structure yielded by a transaction scheme into an acyclic net (without an initial marking). We then merge these nets with the high level control flow net. As a last step we refine local control states and the transitions to expose information about the valuations of the atomic propositions.

From Event Structure to Acyclic Net First, let T be a transaction scheme and $ES_T = (E, \preceq, \#, \lambda)$ be its event structure representation. For $e \in E$ we set $proc(e) = p$ if there exists (x, i, P, V_P) in e such that $x \in E_p^i$. It is easy to see that $proc$ is a well-defined function, and it is also easy to check that if $e \widehat{\#} e'$, $proc(e) = proc(e')$.

Next, we define the relation $EQ_\#^T \subseteq E \times E$ as follows (where the context is clear, we will write EQ instead of $EQ_\#^T$): $e\, EQ\, e'$ if $e \,\widehat{\#}\, e'$; furthermore, if $e_1 \leq e$, $e'_1 \leq e'$, and $e_1 \widehat{\#} e'_1$, then $e_1 = e$ and $e'_1 = e'$. The role of $EQ_\#^T$ will be clear once we define the net associated with our event structure. Before doing so, note that the *minimal* causality relation of event structure ES_T is denoted as $\lessdot$. We define the net representation N_T of $ES_T = (E, \preceq, \#, \lambda)$ as the acyclic net (B_T, E_T, F_T) where:

(1) The set of transitions $E_T = E$.

(2) The set of places B_T and the flow relation F_T are the least sets which satisfy:

   (i) Suppose $e \lessdot e'$ and $proc(e) \neq proc(e')$. Then $(e, e') \in B_T$, $(e, (e, e')) \in F_T$, and $((e, e'), e') \in F_T$.

   (ii) Suppose $e \lessdot e'$ and $proc(e) = proc(e')$. Then $(e, [e']_{EQ}) \in B_T$ and $(e, (e, [e']_{EQ})) \in F_T$ and $((e, [e']_{EQ}), e'') \in F_T$ for every $e'' \in [e']_{EQ}$.

Merging the Control Flow Let $\mathrm{TP} = \{TS_p\}_{p \in \mathcal{P}}$ be a CTP over $(\Gamma, \mathcal{P})$ where Γ is a finite set of transaction schemes over $(\mathcal{P}, M, Act)$. Let $TS_p = (S_p, \Gamma_p, \longrightarrow_p, init_p, V_{p,in})$ be the transition system associated with transaction process p (note that $V_{p,in}$ is the initial p-valuation). For each transaction scheme T in Γ let ES_T be its event structure representation and $N_\mathrm{T} = (B_\mathrm{T}, E_\mathrm{T}, F_\mathrm{T})$, the net associated with ES_T. For convenience we will denote the set of pre and post control states of the transaction scheme T as ${}^\bullet\mathrm{T}$ and $\mathrm{T}^\bullet$ respectively and define these sets as:

${}^\bullet\mathrm{T} = \{s \mid \mathrm{T} \in \Gamma_p \text{ and } s \xrightarrow{\mathrm{T}}_p s' \text{ for some } s, s' \in S_p\}$.

$\mathrm{T}^\bullet = \{s' \mid \mathrm{T} \in \Gamma_p \text{ and } s \xrightarrow{\mathrm{T}}_p s' \text{ for some } s, s' \in S_p\}$.

We can now carry out the second step in providing the operational semantics. The *control flow Petri net* of TP is the Petri net $\mathrm{CFN_{TP}} = (S_\mathrm{TP}, T_\mathrm{TP}, F_\mathrm{TP}, M_{in,\mathrm{TP}})$ where:

- $S_\mathrm{TP} = \bigcup\{S_p \mid p \in \mathcal{P}\} \cup \bigcup\{B_\mathrm{T} \mid \mathrm{T} \in \Gamma\}$.
- $T_\mathrm{TP} = \bigcup\{E_\mathrm{T} \mid \mathrm{T} \in \Gamma\}$
- $F_\mathrm{TP} = \bigcup_{\mathrm{T} \in \Gamma}(F_\mathrm{T}) \cup$
  $\{(s, e) \mid e \in min(E_\mathrm{T}), s \in {}^\bullet\mathrm{T} \cap S_p, proc(e) = p\} \cup$
  $\{(e, s') \mid e \in max(E_\mathrm{T}), s' \in \mathrm{T}^\bullet \cap S_p, proc(e) = p\}$)
- $M_{in,\mathrm{TP}}(z) = 1$ if there exists p s.t. $z = init_p$ (the initial state of some p). Otherwise $M_{in,\mathrm{TP}}(z) = 0$.

By $min(E_\mathrm{T})$ ($max(E_\mathrm{T})$) we mean the set of minimal (maximal) elements under $\leq_\mathrm{T}$, the causality relation of the event structure ES_T. The last step in our construction is to refine the places and the transitions of the control flow net to expose the valuations.

Constructing the Petri Net Let $\mathrm{CFN} = (S_\mathrm{TP}, T_\mathrm{TP}, F_\mathrm{TP}, M_{in,\mathrm{TP}})$ be the control flow net of TP, a CTP. Then $\mathrm{PN_{TP}}$ is the Petri net representation of TP and it is the Petri net $\mathrm{PN_{TP}} = (S, T, F, M_{in})$ where S, T and F are the least set of elements satisfying the following conditions:

- Suppose s is in S_p. Then (s, V_p) is in S where V_p is a p-valuation. Next let γ be in Γ and $N_\gamma = (B_\gamma, E_\gamma, F_\gamma)$ be the net associated with ES_γ and $(x, y) \in B_\gamma$. Now suppose $(e, i, P, V_P) \in x$. Then $((x, y), V_P)$ is in S.
- Let γ be in Γ and $N_\gamma = (B_\gamma, E_\gamma, F_\gamma)$ be the net associated with ES_γ and $x \in E_\gamma$. Suppose $(e, i, P, V_P) \in x$. Then (x, V_P) is in T.
- Suppose $(s, x) \in F_\mathrm{TP}$ with $s \in S_p$ for some p and $(e, i, \{p\}, V_p) \in x$. Then $((s, V_p), (x, V_p))$ is in F. Also, suppose $(x, s') \in F_\mathrm{TP}$ with $s' \in S_p$ for some p and $(e, j, Q, V_Q) \in x$ and O^j is the output valuation of the transaction $[I^j : Ch^j : O^j]$. Then $((x, V_Q), (s', V_p))$ is in F where $V_p = O^j \cap AP_p$. Finally, let $((x, y), V_P) \in S$. Then $((x, V_P), ((x, y), V_P))$ is in F. Furthermore, $(((x, y), V_P), (y, V_Q)) \in F$ provided $proc(x) \neq proc(y)$ and (y, V_Q) is in T and $V_Q \cap AP_P = V_P$. In case $proc(x) = proc(y)$ then $(((x, y), V_P), (z, V_Q)) \in F$ provided $(z, V_Q) \in y$ and $V_Q \cap AP_P = V_P$.
- $M_{in}(z) = 1$ if $z = (init_p, V_{p,in})$ for some p. Otherwise $M_{in}(z) = 0$

This concludes the construction of the Petri net to be associated with a CTP.

4 Analysis and Verification Aspects

Here we introduce some important behavioral properties of the CTP model and the techniques currently available for determining these properties. We also discuss the relationship between the CTP model and an *asynchronous* version of the Live Sequence Charts (LSC) formalism [4].

4.1 Well-formed Transaction Schemes

For pragmatic reasons, our definition of the CTP model imposes almost no syntactic restrictions. For instance, con-

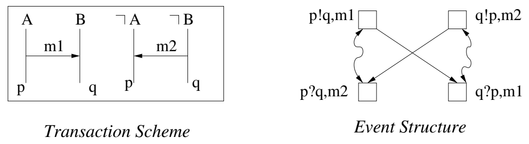

Figure 7. A Transaction Scheme which is not well-formed

sider the transaction scheme shown in Figure 7 and its associated event structure. If the control flow enables this transaction scheme and the valuation is $\{\neg A, B\}$, there will be a deadlock and no event in the associated event structure will execute. On the other hand if the valuation is $\{A, \neg B\}$ then the send events $\langle p!q, m1\rangle$ and $\langle q!p, m2\rangle$ can execute in any order after which there will be a deadlock. Thus local deadlocks can arise due to incomplete specification of the transaction schemes. As a method for detecting and eliminating such undesirable behaviors, we propose the notion of *well-formed* transaction schemes. Intuitively, this notion says that in the locality of a transaction scheme, the maximal executions of the event structure associated with the transaction scheme are precisely the executions of the transactions mentioned in the transaction scheme.

Definition 5 (Well-formed Transaction Scheme) *Let* $\mathrm{T} = \{T^i = [I^i : Ch^i : O^i]\}_{i=1,2,\ldots,n}$ *be a transaction scheme and* $ES_{\mathrm{T}} = (E, \leq, \#, \Lambda)$ *be its event structure representation. For a configuration (a downclosed conflict-free subset of events) c of* ES_{T}*, we let* $ES_{c,\mathrm{T}}$ *be the sub-event structure induced by c; it is the event structure* $(c, \leq_c, \#_c)$ *where* $\leq_c(\#_c)$ *is* $\leq(\#)$ *restricted to c. Let* $MAXC_{\mathrm{T}}$ *be the set of maximal configurations of* T*. Then, transaction scheme* T *is said to be* well-formed *iff there exists a bijection* $f : \{1, 2, \ldots, n\} \to MAXC_{\mathrm{T}}$ *such that* Ch^i *is isomorphic to* $f(i)$ *for each i in* $\{1, 2, \ldots, n\}$.

Each transaction scheme can be effectively analyzed to determine if it is well formed. Clearly the transaction scheme shown in Figure 7 is not well-formed. We are *not* advocating the notion of well-formedness as mandatory but we believe it is a useful criterion using which certain types of incomplete and inconsistent specifications at the level of transaction schemes can be caught.

4.2 Behavioral Properties

Let TP be a CTP and $\mathrm{N_{TP}}$ be its Petri net representation. We shall assume the standard behavioral notions for Petri nets here [5]. We will say that that TP is *transaction-deterministic* if for every reachable marking M of $\mathrm{PN_{TP}}$, if x and y are events belonging to the event structure associated with a transaction scheme in TP and both x and y are enabled at M then $proc(x) \neq proc(y)$. Consequently x and y can occur causally independent of each other at M. Thus transaction-determinism guarantees during the course of executing events taken from a transaction scheme, there will be no conflict. We will also say that TP is *bounded* in case its Petri net is bounded (has only a finite set of reachable markings).

We say that a transaction scheme is *anchored* in case each of its transactions is anchored. A transaction is anchored if its associated MSC, say, $Ch = (E, \leq, \lambda)$ has a least element e_{in} and greatest element e_{fin} and further more,there exists p such that $e_{in}, e_{fin} \in E_p$. Thus the transaction is initiated and terminated by a single agent. An interesting observation here is that if each transaction scheme in a CTP is anchored, then the CTP is bounded.

Via the Petri net semantics, the notions of TP being *live* and *dead-lock free* can also be defined. Clearly, all these properties are decidable since the corresponding problems for Petri nets are decidable. We are currently studying how efficient decision procedures can be developed by exploiting the additional structure provided by the CTP model.

4.3 Connection to Live Sequence Charts (LSC)

Our CTP formalism serves as a high level executable specification language based on Message Sequence Charts. Recently, Damm and Harel have developed the Live Sequence Charts (LSC) formalism which is also a MSC based modeling language. A powerful execution framework for LSCs based on the so called Play-in/Play-out approach is also being developed by Harel and his collaborators [7, 9]. In this section, we explore the connections between the CTP and LSC formalisms, namely (a) how to interpret LSCs over the CTP model , and (b) how to translate CTP models to the LSC language.

For basic features of LSCs such as universal/existential charts and hot/cold conditions the reader is referred to [4]. We define a basic *universal* LSC (over $(\mathcal{P}, M, Act)$) with a *pre-chart* as a structure $[PCh, BCh]$ where:

(1) $PCh = (E_{PCh}, \leq_{PCh}, \lambda_{PCh})$ is a MSC called the *pre-chart*.

(2) $BCh = (E, \leq, \lambda)$ is a MSC called the *body* with $E_{PCh} \cap E = \emptyset$.

(3) $[PCh, BCh]$ denotes $PCh \circ BCh$, the asynchronous concatenation of PCh with BCh.[2]

(4) $agents(min(BCh)) \subseteq agents(PCh)$.

In order to explain the last condition given above, recall that $min(BCh)$ is the set of minimal events in BCh.

[2]This is in keeping with the asynchronous nature of our CTP model; [4] mentions a variant involving synchronous concatenation.

The last condition is intended to ensure that in the asynchronous concatenation of PCh followed by BCh, every event of BCh will have a causal predecessor in PCh. As might be expected, we define the asynchronous concatenation $Ch^1 \circ Ch^2$ of two MSCs $Ch^1 = (E^1, \leq^1, \lambda^1)$ and $Ch^2 = (E^2, \leq^2, \lambda^2)$ with $E^1 \cap E^2 = \emptyset$ as the MSC $Ch = (E, \leq, \lambda)$ where $E = E^1 \cup E^2$ and $\lambda(e) = \lambda^1(e)$ ($\lambda^2(e)$) if e is in E^1 (E^2). Finally $\leq$ is the least partial ordering relation over E which contains $\leq^1$ and $\leq^2$ and satisfies: if $e \in E^1_p$ and $e' \in E^2_p$ for some p then $e \leq e'$.

Next we define a basic *universal* chart with a *pre-condition* as a structure $[P, \varphi, BCh]$ where φ is a propositional formula built out of AP_P called the pre-condition and BCh is a chart called the body such that $agents(min(BCh)) \subseteq P$. Basic *existential* charts denoted $\langle PCh, BCh\rangle$ with a pre-chart as well as basic existential charts with pre-conditions denoted $\langle P, \varphi, BCh\rangle$ can be defined in a similar fashion. Neither existential nor universal charts with post-charts are interesting. We however define a basic universal (existential) chart with a post-condition as the structure $[BCh, P, \varphi]$ ($\langle BCh, P, \varphi\rangle$) where, as before, φ is a propositional formula built out of AP_P and $agents(max(BCh)) \subseteq P$.

A cold condition is a basic existential chart with a pre-condition whose body is empty. A hot condition is a basic universal chart with a post-condition whose body is empty. LSCs can now be inductively obtained by starting with the basic charts and allowing the body itself to be an LSC. Viewing the resulting class of LSCs as atomic assertions, one can obtain LSC specifications by forming boolean combinations of these atomic assertions.

When does a CTP model satisfy a LSC specification ? Due to space considerations we will not spell out the full LSC semantics in a general setting. Instead, we will just interpret the basic LSC specifications over CTPs. It will then be easy to extend this interpretation to more complicated LSC specifications. Let $\mathrm{TP} = \{TS_p\}_{p\in\mathcal{P}}$ be a CTP over $(\Gamma, \mathcal{P})$ where Γ is a finite set of transaction schemes over $(\mathcal{P}, M, Act)$. Let $\mathrm{PN_{TP}}$ be the Petri net representation of TP, constructed from Σ-labeled event structures representing the transaction schemes. Let $\Longrightarrow$ be the labeled transition relation defined over the reachable markings of $\mathrm{PN_{TP}}$ given by: $M \stackrel{\alpha}{\Longrightarrow} M'$ iff there exists a transition t of $\mathrm{PN_{TP}}$ such that t is enabled at M and M' is the resulting marking when t occurs at M. Furthermore, $\Lambda(t) = \alpha$ where Λ is the obvious labeling function that assigns to each transition of $\mathrm{PN_{TP}}$, a label in Σ (the set of labels of events appearing in the transaction schemes). This transition relation $\Longrightarrow$ is extended to Σ-sequences in the obvious way and this extension will be also be denoted as $\Longrightarrow$. Next let $Ch = (E, \leq, \lambda)$ be an MSC. Then λ applied pointwise to a linearization of $(E, \leq)$ yields a member of Σ^*. We let $lin(Ch)$ the subset of Σ^* obtained this way, and refer to it also, by abuse of terminology, as the linearizations of Ch. We define Σ_{Ch} to be the subset of Σ given by $\Sigma_{Ch} = \{\lambda(e) \mid e \in E\}$. Finally, if $\sigma \in \Sigma^*$ and $\Sigma' \subseteq \Sigma$ then $\lceil_{\Sigma'} (\sigma)$ is the Σ' projection of σ. For an MSC Ch we will often write $\lceil_{Ch}$ instead of $\lceil_{\Sigma_{Ch}}$. We are now prepared to interpret the basic LSC specifications over CTPs.

1. Let $[PCh, BCh]$ be basic universal chat with a pre-chart. Then TP satisfies $[PCh, BCh]$ iff *every* reachable marking (s_0, V_0) of $\mathrm{PN_{TP}}$ satisfies the following condition. Suppose $M_0 \stackrel{\sigma_0}{\Longrightarrow} M_1$ such that $\lceil_{PCh} (\sigma_0)$ is in $lin(PCh)$. Then for *every* $M_1 \stackrel{\sigma_1}{\Longrightarrow} M_2$, there exists $M_2 \stackrel{\sigma_2}{\Longrightarrow} M_3$ such that a prefix of $\lceil_{\Sigma'} (\sigma_0\sigma_1\sigma_2)$ corresponds to a member of $lin(PCh \circ BCh)$ where $\Sigma' = \Sigma_{PCh} \cup \Sigma_{BCh}$. Thus, this universal requirement demands that whenever (a linearization of) PCh has been executed then this *must* be followed by an execution of (a linearization of) BCh.

2. Next suppose $[P, \varphi, BCh]$ is a basic universal charts with a pre-condition. Then TP satisfies $[P, \varphi, BCh]$ iff *every* reachable marking (s_0, V_0) of $\mathrm{PN_{TP}}$ satisfies the following condition. Suppose $V_0 \models \varphi$. Then for every $M_0 \stackrel{\sigma_0}{\Longrightarrow} M_1$ there exists $M_1 \stackrel{\sigma_1}{\Longrightarrow} M_2$ such that a prefix of $\lceil_{BCh} (\sigma_0\sigma_1)$ is a member of $lin(BCh)$. Hence this universal requirement demands that whenever φ holds then this *must* be followed by an execution of BCh.

3. Next suppose $\langle PCh, BCh\rangle$ is a basic existential chart with pre-chart. Then TP satisfies $\langle PCh, BCh\rangle$ iff there *exists* a reachable marking M_0 and $M_0 \stackrel{\sigma_0}{\Longrightarrow} M_1$ such that a prefix of $\lceil_{\Sigma'} (\sigma_0)$ contains a member of $lin(PCh \circ BCh)$. As before $\Sigma' = \Sigma_{PCh} \cup \Sigma_{BCh}$.

4. Now suppose $\langle P, \varphi, BCh\rangle$ is a basic existential chart with a pre-condition. then TP satisfies $\langle P, \varphi, BCh\rangle$ iff there exists a reachable marking M_0 and $M_0 \stackrel{\sigma_0}{\Longrightarrow} M_1$ such that $V_0 \models \varphi$ and a prefix of σ_0 corresponds to a member of $lin(BCh)$.

5. Basic charts with post-conditions are dealt with similarly.

One can effectively decide whether or not a bounded CTP TP satisfies an LSC requirement lsc. This is so because from $\mathrm{PN_{TP}}$ we can extract a finite Kripke structure. Moreover, it is known that lsc can be effectively transformed into a CTL^* formula [8]. As a result we can apply the known model checking procedure for CTL^* to solve this problem [3]. This however will involve high complexity and more efficient decision procedures are needed to solve this problem.

Translating CTP models to LSC We can also translate a CTP model into an LSC specification. Consequently the play engine mechanism developed in the LSC framework [9] becomes readily accessible. Furthermore, this translation also makes it clear that the CTP model is a restricted version of the LSC formalism in which only universal charts are used but the intra-object control flow is explicitly specified using traditional mechanisms.

Let $\mathrm{TP} = \{TS_p\}_{p\in\mathcal{P}}$ be a CTP over $(\Gamma, \mathcal{P})$ where Γ is a finite set of transaction schemes over $(\mathcal{P}, M, Act)$. Assume as before that each transaction process p is associated with $TS_p = (S_p, \Gamma_p, \longrightarrow_p, init_p, V_{p,in})$. Recall also that the set of pre and post control states of the transaction scheme T denoted as ${}^\bullet\mathrm{T}$ and $\mathrm{T}^\bullet$. To construct the LSC specification of TP, we will deploy $\bigcup\{S_p \mid p \in \mathrm{P}\}$ also as atomic propositions. Now let T be a transaction scheme of TP with $\mathrm{T} = \{[I^i : Ch^i : O^i] \mid i = 1, 2, \ldots, n\}$. We recall that each I^i is a propositional formula built out of AP, each $Ch^i = (E^i, \leq^i, \lambda^i)$ is a chart over $(\mathcal{P}, M, Act)$ and each O^i is a subset of AP. With T, we associate the LSC specification lsc_{T} given by $lsc_{\mathrm{T}} = lsc_1 \wedge lsc_2 \ldots \wedge lsc_n$ where for each i we have $lsc_i = [BCh_i, post_i]$ with $BCh_i = [pre_i, Ch^i]$. Also, pre_i and $post_i$ are given by $pre_i = \bigwedge_{s\in{}^\bullet\mathrm{T}} s \wedge I^i$ and $post_i = \bigwedge_{s\in\mathrm{T}^\bullet} s \wedge \bigwedge_{A\in O^i} A$.

Intuitively, T translates to the universal requirement: *whenever the pre-control states of* T *hold and the guard for the i-th transaction holds, then the i-th transaction* must *execute followed by the holding of the post-valuation O^i and the post-control states of* T. However, the actual semantics is given in an asynchronous execution framework.

5 Discussion

In this paper we have presented CTP, a high level executable specification language for modeling reactive systems. Our model is based on Message Sequence Charts (which emphasize inter-process communication) and explicit description of intra-process control flow. The main questions to be pursued in this context involves well-formedness checking, formal verification as well synthesizing implementations from such models.

We have constructed a translator that transforms a CTP program into an internal representation of the Petri net representing the behavior of the CTP model. Using this translator we are currently automating the analysis of transaction schemes for well-formedness. Work is also underway to devise a more efficient and direct procedure for determining the boundedness property. An interesting related problem is the issue of quasi-static schedulability as formulated in [14]. We conjecture that the schedulability criterion via T-allocations used for free choice nets [14] extends smoothly to the setting of CTPs.

We are also currently exploring the means for extending our model along a number of dimensions, namely: parameterizing each component as an instance of a class together with the parameterization of the transaction schemes; further relaxing the control flow restrictions; and, adding timing constraints.

6 Acknowledgments

This research was partially supported by an A*STAR Pilot Project R-252-000-113-304 and a NUS Research Project R-252-000-103-112.

References

[1] R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2001.

[2] B. Caillaud, P. Darondeau, L. Helouet, and G. Lesventes. Hmscs as partial specifications ... with pns as completions. In *Modeling and Verification of Parallel Processes 4th Summer School, MOVEP 2000, LNCS 2067*, 2001.

[3] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[4] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1), 2001.

[5] J. Desel and J. Esparza. *Free Choice Petri Nets*. Cambridge University Press, 1995.

[6] B. Douglass. *Doing Hard Time: Developing Real-time Systems using UML, Objects, Frameworks and Patterns*. Addison-Wesley, 1999.

[7] D. Harel and H. Kugler. From play-in scenarios to code: An achievable dream. In *Fundamental Approaches to Software Engineering (FASE), LNCS 1783*, 2000.

[8] D. Harel and H. Kugler. Synthesizing state-based object systems from LSC specifications. *International Journal on Foundations of Computer Science*, 13(1), 2002.

[9] D. Harel, H. Kugler, R. Marelly, and A. Pnueli. Smart play-out of behavioral requirements. In *International Conference on Formal Methods in Computer Aided Design (FMCAD)*, 2002.

[10] J. Hendriksen, M. Mukund, K. Kumar, and P.S. Thiagarajan. Message sequence graphs and finitely generated regular MSC languages. In *International Colloquium on Automata, Languages and Programming (ICALP), LNCS 1853*, 2000.

[11] I. Krueger, R. Grosu, P. Scholz, and M. Broy. From MSCs to statecharts. In *International Workshop on Distributed and Parallel Embedded Systesms (DIPES)*, 1998.

[12] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains. *Theoretical Computer Science (TCS)*, 13, 1981.

[13] A. Roychoudhury and P.S. Thiagarajan. An executable specification language based on message sequence charts. In *Formal Methods at the Crossroads: from Panacea to Foundational Support*. Springer Verlag, To be published as LNCS volume, 2002.

[14] M. Sgroi and L. Lavagno. Synthesis of embedded software using free-choice Petri nets. In *ACM Design Automation Conference (DAC)*, 1999.

[15] Z.120. Message Sequence Charts (MSC'96), 1996.