COMPOSITION IN MULTI-PARADIGM SPECIFICATION TECHNIQUES

Lynne Blair, Gordon Blair[1]

Abstract

This paper addresses the issue of composition in a multi-paradigm environment. Our work focuses on the application domain of distributed multimedia systems and, in addition to considering quality of service properties, we also explore dynamic quality of service management functions based on the concepts of monitors and controllers. We advocate the use of a multi-paradigm specification technique which, to suit our chosen application domain, consists of LOTOS, real-time temporal logic and timed automata specifications. We illustrate our approach by giving an example of a simple phone system, extended with dynamic QoS management capabilities.

Introduction

Multi-paradigm specification techniques are not a particularly new concept, but are referred to under different names depending on the audience. For example, Kiczales' *aspects* [17], Zave and Jackson's *multi-paradigm* approach [24], ODP's *viewpoints* [21][8], Finkelstein and Spanoudakis' *perspectives* [13] and our earlier work based on a *separation of concerns* [4] are all, at heart, addressing the same issue: the structuring of large and complex software systems. A key advantage of such an approach is that the overall system behaviour can be subdivided into different aspects (or viewpoints, perspectives, etc.), each of which is specified separately. Furthermore, different aspects may use different specification languages. As a result, a language particularly suited to the needs of a given aspect can be selected, rather than being forced to use the same language throughout the entire system specification.

However, although there are clear advantages to a multi-paradigm approach, there are also problems that do not arise in a more traditional single-language approach. In particular, the two issues of consistency and composition must be carefully addressed. Checks for *consistency* across the different aspects are necessary to establish that the

P. Ciancarini et al. (eds.), *Formal Methods for Open Object-Based Distributed Systems*

different specifications do not impose contradictory requirements. The mechanisms needed for this vary depending on the languages used. Many such mechanisms, particularly those for checking consistency between ODP viewpoint specifications. have been addressed elsewhere in the literature (e.g. [8], [9] and [11]). Consequently, in this paper, our focus will be on the problem of composition.

Composition is required in order to analyse the *overall* specification and perform any validation and/or performance analysis. There are various possible techniques for composing specifications. For example, Zave and Jackson's approach considers *conjunction* as composition [23]. In their approach, the necessary parts of the (aspect) specifications are translated into assertions in a first-order predicate logic. By considering the conjunction of these assertions, proofs of required properties can be performed. However, in our approach, we propose a different approach to composition, primarily because of the nature of the languages we intend to use. Each of our aspect specifications can be translated into a timed automaton, using timed labelled transition systems as our underlying semantic model. The parallel composition of two or more of these automata is then defined, and the resulting composition can be analysed as required. Note that we have developed a Java-based tool to support our multi-paradigm approach, including automation of the composition process, simulation and (simple) model-checking [16].

Background on distributed multimedia systems

Our work focuses on the application domain of distributed multimedia systems. In such systems, *quality of service* is a crucial concept. Any non-functional behaviour of the system is referred to as a quality of service property or constraint. There are many different categories of quality of service, e.g. timeliness, reliability, security, efficiency, etc, but it is the first two of these properties, timeliness and reliability, that we concentrate on in our work. However, simply being able to specify quality of service properties is not enough; it is also necessary to be able to *manage* such properties. Quality of service management ensures that the desired quality of service constraints are attained and, in the case of continuous media, sustained. Management policies may either be static (e.g. admission control, QoS negotiation and resource reservation) or dynamic (e.g. QoS monitoring, re-negotiation, policing and maintenance). It is the specification of these policies (particularly dynamic QoS management) which pose a new challenge to formal specification. Whilst most other work on the formal specification of distributed multimedia systems has addressed QoS properties, little work has been done on the more complex issues associated with (dynamic) QoS management. The example that we will present later in this paper illustrates dynamic QoS management, based on the concepts of monitors and controllers.

Description of our approach

Aspect-oriented specification

We refer to our multi-paradigm approach as *aspect-oriented* specification, that is, maintaining a separation between different aspects of a specification. As in Kiczales'

work [17], there are no absolute definitions of how many aspects should be considered, nor what they should describe. There are many possible interpretations of this for different applications. So, for example, whilst security and fault-tolerance may be crucial for one system, efficiency may be the key aspect of another system. This freedom is in contrast to ODP viewpoints where five viewpoints are prescribed and the content of each is (arguably) well-defined. However, our experience has pin-pointed three aspects which are central to the specification of distributed multimedia systems, namely the (functional) *abstract behaviour*, the (non-functional) *QoS parameters* and the (non-functional) *QoS management policies*. Whilst we acknowledge that other aspects are possible, perhaps even desirable, the chosen three aspects serve to explain and illustrate our approach. Further descriptions of the three aspects, along with our motivation for adopting an aspect-oriented approach, can be found in [5] and [6].

Choice of languages

There are a large number of different specification languages to choose from, many of which offer similar functionality or support a similar level of abstraction. However, the choice of which language to use is not crucial to our approach, although it will be noted later that the semantics will need to be interpreted in a particular form.

Our choice of languages has been guided (but is not exclusively defined) by our interest in distributed multimedia systems. Consequently, we adopt the use of the (standardised) formal description technique *LOTOS* [10] for an abstract specification of the functional behaviour. For the second aspect, QoS parameters, in previous work we have developed and evaluated a real-time *temporal logic*, QTL [4]. In this work, we were primarily concerned with *timeliness* properties such as latency, throughput and jitter. However, the need for another category of QoS, namely reliability properties, led us to develop a stochastic extension to QTL, called SQTL [18]. Consequently, we propose the use of a temporal logic such as QTL or SQTL for the specification of timeliness or reliability properties. Finally, for QoS management policies, our experience has shown that *timed automata* are a natural language in which to specify such behaviour. In particular, we adhere (with a few minor exceptions) to the description of timed automata supported by UPPAAL [3].

We refer the reader to the above-mentioned literature for formal descriptions of LOTOS and the temporal logics (QTL and SQTL). However, selected features of these languages will be explained, where necessary, with our example later in this paper. Timed automata will be discussed in some detail below.

Generating a global model

In the world of specification, the crucial feature which underpins everything else is the *semantics*: they must be clear, concise and, above all, unambiguous. For our aspect-oriented specification technique, we must deal with the semantics of the *composition* of two or more aspect specifications. This is an essential step if we are to perform any validation or performance modelling of the overall system behaviour. In our approach, the result of composition is a *global model* (as illustrated in figure 1).

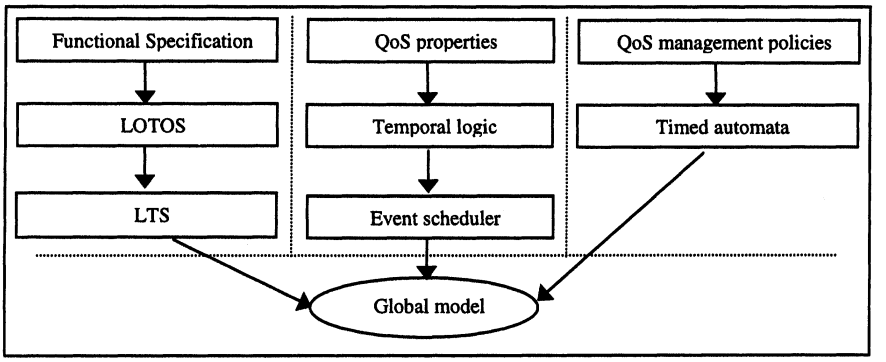

Figure 1. Generating our global model

Our global model is represented through timed automata using (timed) labelled transition systems to underpin these as our low-level semantic model. Consequently, any specification languages with operational semantics that can be associated with a LTS can be used within our framework. The availability of good LOTOS tools (e.g. [14], [20]) makes the generation of a labelled transition system relatively straightforward (assuming some restrictions on LOTOS operators and data type definitions are obeyed). It is also possible to map a temporal logic formula to timed automata (also referred to in this context as an event scheduler[2]). Earlier versions of this work have been presented in [4] for QTL and [18] for SQTL. The third strand in the above diagram shows the use of timed automata for QoS management policies. All mappings are formalised in the following section.

Semantic models

Timed labelled transition systems underpin the formal languages we use in our multi-paradigm specification environment. We start by formalising their untimed counterpart below.

Labelled transition systems

We define a labelled transition system over a set of (atomic) actions (Act) to have the following form:

$$\text{LTS} = \langle S, s_0, \longrightarrow \rangle$$

where S is a finite set of states,
$s_0 \in S$ is the initial state, and
$\longrightarrow \subseteq S \times Act \times S$ is a transition relation $\{ \xrightarrow{a} \mid a \in Act, \exists\, s,s' \in S,\ s \xrightarrow{a} s' \}$

Timed transition systems

To extend labelled transition systems with time, we permit two sorts of actions: atomic actions (Act) and delay actions (Δ) whose elements are denoted by $\varepsilon(d) \in \Delta$ (for $d \in \mathbb{R}^+$). We will denote the set of labels L as $Act \cup \Delta$. We also add time and data

variables to our labelled transition system. Let C be a finite set of (positive) real-valued clocks and let D be a finite set of data variables with Var=C∪D. With these modifications, each state in our *timed* transition system now consists of the pair:

$$s = (l, u)$$

where l is a node of the transition system, and
u is a variable assignment function s.t. for clocks $X \subseteq C$, $u : X \rightarrow \mathbb{R}^+$
and for data variables $Y \subseteq D$, $u : Y \rightarrow \mathbb{Z}$

Our initial state is now defined to be $s_0 = (l_0, u_0)$, where l_0 is the initial node of the transition system, and u_0 initialises all clocks $[C \mapsto 0]$ and all data variables $[D \mapsto 0]$.

The above timed transition system can be thought of as a low-level semantic model containing an infinite number of node/clock valuation pairs. Consequently, on top of this we define timed automata.

Timed automata

We use the timed automata of UPPAAL [19] as the basis for this work. We start with a standard finite state automaton and extend this with a set of clocks C and data variables D, as defined above, such that Var=C∪D. Let the set of constraints (guards) over Var be represented by G(Var). Note that the values of clocks and variables can be compared with constants and/or reset on transitions. Also, conditions on clocks or variables may guard a transition and invariants may be assigned to states (which must remain true whilst in that state). We define a timed automaton formally as:

$$TA = \langle S, s_0, \longrightarrow, I \rangle$$

where S is a finite set of states,
$s_0 \in S$ is the initial state,
$\longrightarrow$ is a transition relation,
$I : S \rightarrow G(Var)$ is an invariant assignment function for each state
(this must be satisfied by all variables whilst operating in that state).

Our transitions are more complex than for the timed transition systems above:

$$\longrightarrow = \langle l, g, a, r, l' \rangle$$

where l, l' are nodes of the automaton and
$a \in L$ (as above),
but we also have:
g is a constraint (guard) s.t. g::=x~c where x∈ Var, ~::=<|=|>, c is a constant
r is a reset (or reassignment) function s.t. for $r \subseteq Var$, $[r \mapsto r']u$
i.e. all variables r are updated; other variables are unchanged (still satisfy u)
The transitions should also now satisfy the following rules:

$(l, u) \xrightarrow{g,a,r} (l', u')$ if g is satisfied by u, and $u' = [r \mapsto r']u$

$(l, u) \xrightarrow{\varepsilon(d)} (l', u')$ if $(l = l')$, $u' = u + d$, and u' satisfies I(l').

Hence time can pass whilst in a given state.

Semantic model of LOTOS

Derivable processes. The operational semantics of LOTOS define a set of axioms and inference rules for each possible behaviour description of a specification. From

these we can derive a set of transitions that the LOTOS specification may perform, and subsequently associate a labelled transition system (LTS) with the specification.

Let Act denote the set of atomic LOTOS actions and let L denote Act $\cup$ {i}. Also, let D_P denote the (smallest) set of processes *derivable* from a given process P such that:

$P \in D_P$, and

if $P' \in D_P$ and $\exists\, a \in L, P' \xrightarrow{a} P''$ then $P'' \in D_P$

The labelled transition system for the *specification* P can now be defined as:

$$LTS = \langle S, s_0, \longrightarrow \rangle$$

where $S = D_P$,

$s_0 = P$, and

$\longrightarrow = \{ \xrightarrow{a} \mid a \in L, \exists\, s, s' \in D_P, s \xrightarrow{a} s' \}$.

Mapping to timed automata. In order to map an (untimed) LOTOS specification onto timed automata, we adopt the following terminology:

variables:	C={ }, D={ } $\therefore$ Var={ }
actions:	$a \in Act \cup \{i\} \cup \Delta$
states:	$\forall i, s_i = l_i$ and u is irrelevant since Var={ }
transitions:	$\longrightarrow = \langle l, g, a, r, l' \rangle$ where $\forall g$, g=true and the set of variables r={ }
	Also, $(l, u) \xrightarrow{\varepsilon(d)} (l', u')$ if $(l = l')$
invariants:	$I : S \rightarrow$ true

Semantic model of QTL

Syntax of QTL. In this section, we restrict our attention to QTL [4], rather than consider the stochastic extensions of SQTL. We plan to return to stochastic issues in future work. We formally define the syntax of QTL[3] as follows:

$$\phi ::= \text{false} \mid \phi_1 \rightarrow \phi_2 \mid \bigcirc_{\sim d} \phi \mid \phi_1 \, U_{\sim d} \, \phi_2 \mid a \mid \pi_1 \sim \pi_2$$

where $\sim ::= < \mid = \mid >$

$\pi \in \mathcal{P}$ and $\pi ::= x \mid c \mid x + c$ (i.e. propositions with addition by constant only)

$d \in \mathbb{R}^+$

$a \in$ Act (the set of all possible events)

and $\rightarrow$, $\bigcirc$ and U denote *implies*, *next-state* and *until* respectively.

Note that other operators can be defined from these in the usual way (e.g. see [4]).

Clocks. We now let C be a set of clocks, one for each bounded temporal operator in ϕ, D be a set of data variables and, as before, Var = C $\cup$ D. Let $u:C \rightarrow \mathbb{R}^+$ and $u:D \rightarrow \mathbb{Z}$. We allocate the set of clocks as follows:

$C[\![\text{false}]\!] = \{\}$

$C[\![a]\!] = \{\}$

$C[\![\phi_1 \rightarrow \phi_2]\!] = C[\![\phi_1]\!] \cup C[\![\phi_2]\!]$

$C[\![\bigcirc_Q \phi]\!] = \{C_Q\} \cup C[\![\phi]\!]$

$C[\![\phi_1 U_Q \phi_2]\!] = \{C_Q\} \cup C[\![\phi_1]\!] \cup C[\![\phi_2]\!]$

To handle the resetting of these clocks, let res(ϕ) be a function which resets all clocks r in ϕ to zero; all other clocks keep their value according to u (i.e. for $r \subseteq C$,

$[r \mapsto 0]u$). Now, for any occurrence of the timed operators O_Q or U_Q in a formula, we must reset the associated clock:

$[\ Q \mapsto 0\ ] \in res(O_Q \phi)$, and

$[\ Q \mapsto 0\ ] \in res(\phi_1 U_Q \phi_2)$

Derivation Rules. In order to interpret a temporal logic formula as an automaton, we first define a set of derivation rules, in much the same way as for LOTOS above. The *derivative* of formula ϕ with respect to action α, denoted $D_\alpha[[\phi]]$, is:

$D_\alpha[[false]] = false$

$D_\alpha[[a]] = true$ if $a=\alpha$, otherwise false

$D_\alpha[[\phi_1 \rightarrow \phi_2]] = D_\alpha[[\phi_1]] \rightarrow D_\alpha[[\phi_2]]$

$D_{\alpha,C}[[O_{\sim d}\ \phi]] = \phi$ where $u(C)=d$

$D_{\alpha,C'}[[\phi_1 U_{\sim d} \phi_2]] = (\ D_{\alpha,C'}[[\phi_2]]$ where $u(C') = d\ ) \vee$

$(\ D_{\alpha,C'}[[\phi_1]] \wedge (\phi_1 U_{=d-u(C')} \phi_2)$ for all $u(C') \leq d\ )$

In order to get our nodes for the labelled transition system, we define the transitive closure of D, denoted D* inductively as follows:

$\phi \in D^*[[\phi]]$, and

if $\varphi \in D^*[[\phi]]$ then for all $a \in$ Act, $D_a[[\varphi]] \in D^*[[\phi]]$

Example. To illustrate this process, we consider the simple untimed formula: $\phi = \Diamond p$. Note that $\Diamond$ is a derived operator which is equivalent to TRUE U p; hence we use the derivation rules for until, as given above. Our formula has the possible actions p, or something other than p (we denote this by the action *). Consequently, $\propto$ in the derivation rules above can be p or *. Following the rules above, it can easily be shown that $D_p[[\Diamond p]] = D_p[[p]] \vee \Diamond p$, and $D_*[[\Diamond p]] = D_*[[p]] \vee \Diamond p$. We now evaluate the nested derivatives to get $D_p[[p]] =$ true and $D_*[[p]] =$ false. Hence, we now have:

$D_p[[\Diamond p]] =$ true$\vee \Diamond p =$ true, and

$D_*[[\Diamond p]] =$ false$\vee \Diamond p = \Diamond p$

This shows that the derivative of $\Diamond p$ yields two formulae: true and $\Diamond p$. We have already evaluated $D[[\Diamond p]]$ hence we need only consider D[[true]]. The above rules state D[[true]]=true with respect to *any* action. We will return to this example below.

Mapping to timed automata. We are now in a position to associate a formula with a timed automata. *States* are composed of nodes and clock assignment functions. The conditions for the clock assignment functions are embedded in the definition of D*. *Transitions* have the form:

$$\longrightarrow = \langle\ l, g, a, r, l'\ \rangle$$

where l, l' are nodes of D*,

g is a constraint (guard),

a $\in$ Act, and

r = res(φ) where res is the reset function described above and $\varphi \in l$

Also $(l,u) \xrightarrow{g,a,r} (l',u')$ if $l' = D_{a,g}[[\phi]]$, and $u' = [\ r \mapsto 0\ ]u$

$(l,u) \xrightarrow{\varepsilon(d)} (l',u')$ if $(l=l')$, $u'=u+d$, and u' satisfies $I(l')$

Consequently, we can define our timed transition system for a formula ϕ as:

$$LTS = \langle\ S, s_0, \longrightarrow, I\ \rangle$$

where $S = D^*[\![\phi]\!]$ and $s_0 = \phi$,
$\longrightarrow = \langle l, g, a, r, l' \rangle$, and
$I : S \rightarrow G(Var)$

Example revisited. Returning to our example, $D^*[\![\phi]\!]=\{\Diamond p , true\}$, i.e. we have two nodes with $s_0=\Diamond p$. By examining the results of the derivation rules with respect to the actions p and * we can see that the following automaton is obtained.

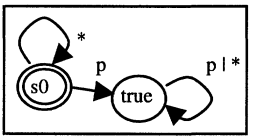

Figure 2. An automaton representing $\phi=\Diamond p$

Composition

Given our multi-paradigm approach, we now have several timed automata that we must compose together. Let SA be the set of synchronising actions, i.e. those actions on which the execution of each automaton should synchronise. We will denote the synchronisation function by $f(a_1,a_2)$[4]; this defines that two actions a_1 and a_2 synchronise (provided $a_1 \neq i$ and $a_2 \neq i$). Note that our algorithm applies to intra-language composition (e.g. composing two automata) as well as inter-language composition (e.g. composing LOTOS with an automaton).

Let $s_1=(l_1,u_1) \in S_1$ and $s_2=(l_2,u_2) \in S_2$ where S_1 and S_2 are timed automata. Also, for LOTOS let $L=Act_{LOTOS} \cup \{i\}$, for QTL let $L=Act_{QTL} \cup \Delta$, and for TA, let $L=Act_{TA} \cup \Delta$.

The composition rules are given in the table below.

Table 1. Rules for the composition $S_1 \|_{SA} S_2$ (where $a \in L$)

$\dfrac{s_1 \xrightarrow{g,a,r} s_1',\ a \notin SA}{s_1 \|_{SA} s_2 \xrightarrow{g,a,r} s_1' \|_{SA} s_2}$	1
$\dfrac{s_2 \xrightarrow{g,a,r} s_2',\ a \notin SA}{s_1 \|_{SA} s_2 \xrightarrow{g,a,r} s_1 \|_{SA} s_2'}$	2
$\dfrac{s_1 \xrightarrow{g1,a1,r1} s_1', s_2 \xrightarrow{g2,a2,r2} s_2',\ a \in SA}{s_1 \|_{SA} s_2 \xrightarrow{g,a,r} s_1' \|_{SA} s_2'}$ where $f(a1,a2)=a$, $g = g1 \wedge g2$ and $r = r1 \cup r2$	3
$\dfrac{s_1 \xrightarrow{\varepsilon(d)} s_1', s_2 \xrightarrow{\varepsilon(d)} s_2'}{s_1 \|_{SA} s_2 \xrightarrow{\varepsilon(d)} s_1' \|_{SA} s_2'}$ where $s_i = (l_i,u_i)$ and $s_i' = (l_i',u_i')$ and $l_i' = l_i, u_i' = u_i + d$	4

The resulting composition can be defined as:

$$Comp = \langle S, s_0, \longrightarrow, I \rangle$$

where S is a finite set of states s.t. $s_1 \|_{SA} s_2 \in S \Leftrightarrow s_1 \in S_1$ and $s_2 \in S_2$,
$s_0 \in S$ is the initial state s.t. $s_0 = s_{0,1} \|_{SA} s_{0,2}$,
$\longrightarrow = \langle l, g, a, r, l' \rangle$ is a transition relation derived from rules 1-4 above,
$I(s_1 \|_{SA} s_2) = I(s_1) \wedge I(s_2)$

Validation and performance modelling

Once we have composed our specifications together, a number of validation techniques can be performed. At this stage, since we have a global model, there is no difference between our multi-paradigm approach and a single-language (timed) approach. The following figure (3) illustrates several possible validation strategies.

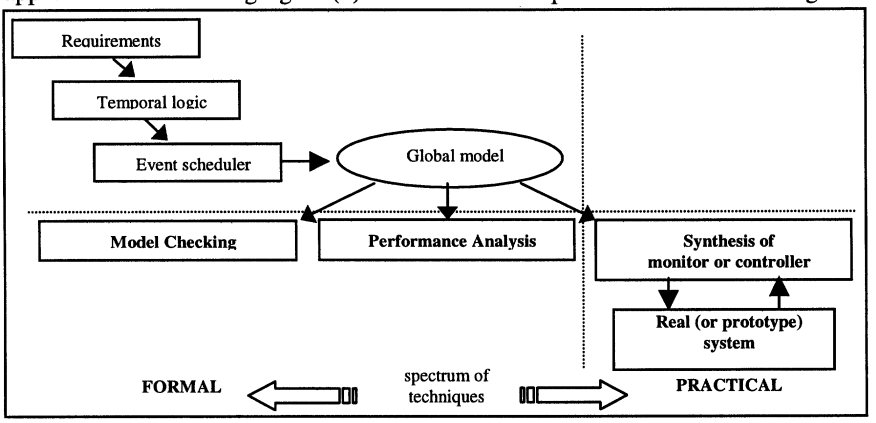

Figure 3. Validation and synthesis possibilities from the global model

Firstly, model checking involves considering a set of user requirements, and proving that these requirements satisfy our (combined) specification. Secondly, performance analysis techniques such as reachability analysis can be applied to explore the state space of our global model. Thirdly, "real-world" monitors and controllers can be synthesised from our formal specification and can be used to manage a real system. A more detailed description of these techniques is beyond the scope of this paper. Consequently, for model checking techniques we refer the reader to [1] and [2] and to our previous work in [4] and [18], for details of reachability analysis strategies see [15], and for details of the synthesis of monitors and controllers see [6] and [7].

Example: a multi-way phone system

Informal description

The example we present in this paper is that of a simple multi-way phone system. Although this is not an example of a multimedia system, we feel that it nicely illustrates our approach and the process of composition. Multimedia examples can be found in our other work (e.g. [4] and [6]). The LOTOS specification for this multi-

way phone system was previously presented as part of a LOTOS tutorial [12]. An informal description of the system is provided in table 2, and Appendix A contains an abbreviated listing of the LOTOS specification.

Table 2. An informal description of a multi-way phone system

Basic behaviour:
• When a phone (A) is lifted off-hook, a dial tone occurs. • If phone B is off-hook, A gets an engaged tone. In this case, phone A must be placed on-hook or the dial request cancelled. • If B does not answer, either phone A must be placed on-hook or the dial request cancelled (both of which cause phone B to stop ringing) • A phone can be either on or off-hook. • A may now dial a second phone (B). • If phone B is on-hook, A gets a ring tone and phone B starts ringing. • If B answers (with an off-hook whilst ringing), a connection is made between A and B and a conversation may occur. Either phone may terminate the conversation by placing their phone on-hook.

More than 2 connections:
• During a conversation, either party may dial a third party, e.g. B dials C. • If C is off-hook, B gets an engaged tone. B must cancel the dial request before dialling again. A and B remain connected throughout. • If C is on-hook, B gets a ring tone and C starts ringing • If C does not answer, B must cancel the dial request before dialling again. Again, A and B remain connected. • If C answers, a connection is made between B and C and, since A is currently connected to B, a connection is also made between A and C. • If 3 (or more) phones are connected and one of them puts their phone on-hook, the other phones remain connected.

Real-time behaviour

Whilst the LOTOS specification has provided us with a good description of abstract behaviour, it does not permit the specification of real-time behaviour. Timed extensions to LOTOS have been developed, but we believe that such languages lead to a *tangling* of aspects (c.f. [17]). Our belief is that, by maintaining a separation of aspects at a specification level, large and complex specifications can be developed that are simpler to understand and maintain. To illustrate this idea for real-time behaviour, we consider two constraints over the abstract LOTOS specification, using the real-time temporal logic, QTL. We assume that our unit of time is seconds.

Constraint 1. If one user dials another user whose phone is on-hook, then the remote phone starts ringing within 1 second.

$$\phi_1: \quad \Box\,(\text{ dial !ringtone ?p:id\_sort ?to:id\_sort} \rightarrow \Diamond_{\leq 1} \text{ ring !to })$$

This formula states that it is always the case that whenever a "dial !ringtone" event occurs (with parameters p and to) then, in some state in the future bounded by one second, a "ring !to" event will occur, i.e. the phone with id "to" will start ringing. Intuitively the symbol ! represents the output of some data on the occurrence of the associated action, whilst ? represents the input of some data. It is the job of the synchronisation function (mentioned earlier) to handle how different input/output events behave when composed in parallel.

We could similarly place constraints on the time between "on !after_ringtone" and "stop_ring" events and between "cancel !after_ringtone" and "stop_ring" events.

Constraint 2. A phone can ring unanswered for a maximum of 30 seconds before being automatically cancelled.

ϕ_2: $\Box$ (dial !ringtone ?p:id_sort ?to:id_sort $\rightarrow$ ($\Diamond_{<30}$ (on !after_ringtone !p !to $\vee$ connect !p !to) $\vee$ $\Diamond_{\leq 30}$ cancel !after_ringtone !p !to)

With this formula, whenever a "dial !ringtone" occurs then, in some state less than 30 seconds into the future, either an "on !after_ringtone" or a "connect" will occur, or at a future state ≤30 seconds from now, a "cancel !after_ringtone" will occur.

Monitors and controllers

A third aspect involves the behaviour associated with quality of service management. Although this is usually associated with multimedia examples, we can illustrate the principle of monitors and controllers using our phone example. We will consider two scenarios below and formalise the required behaviour using timed automata. For convenience, we will assume that the unit of time here is minutes.

Scenario 1. To prevent overloading the exchange, management imposes a temporary restriction on the number of phones off-hook at a given time (limited to 3) until a more permanent solution is found.

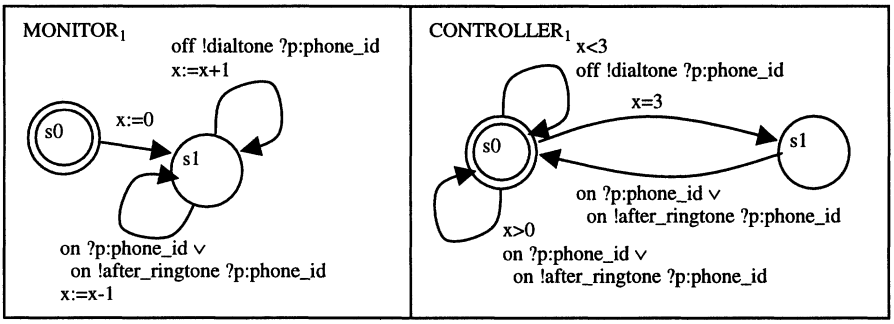

Figure 4. Automata to monitor and control the number of phones off_hook

Figure 4 illustrates two automata, a monitor and a controller, that together achieve the desired behaviour of scenario 1. The first automaton shows a data variable x being used as a count for the number of off-hooks at any given time. The variable is incremented whenever an off-hook occurs and decremented whenever an on-hook occurs. This variable is then used in the second automaton to implement the policy that x should always be less than, or equal to, 3. Note that the two automata synchronise on the LOTOS events "off !dialtone", "on" and "on !after_ringtone". The values of p:phone_id are instantiated through synchronising with the LOTOS specification (see Appendix A).

Also note that, although it would be possible to combine the monitor and controller as one automaton, we choose to keep them separate. Whilst not significant for small examples, as the behaviour becomes more complex, this separation means that control policies are immediately identifiable and easily changed.

Scenario 2. Management is alerted to the amount of time that the user of phone 1 spends talking on the phone. They decide to monitor the time carefully and set the user a quota of 20 minutes per day (but, at least initially, without the drastic action of disconnecting any connection in progress on reaching the 20 minute quota). If the quota is exceeded, the user receives a quota violation signal and is barred from getting a dial tone (off_hook event) until the following day when the quota is reset.

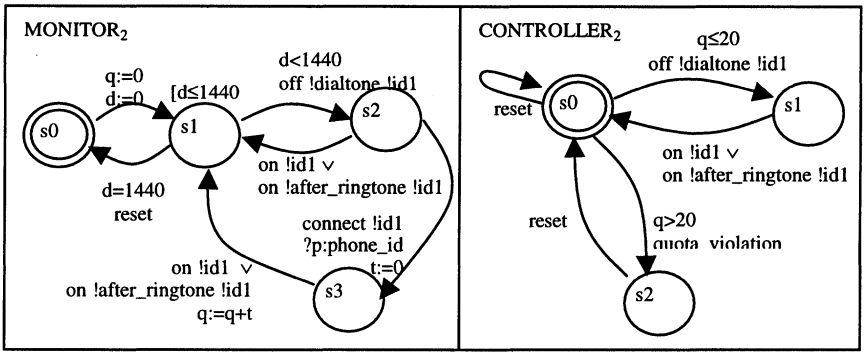

Figure 5. Automata to monitor and control the connection time of phone 1

Figure 5 illustrates the monitor and controller aspects of the second scenario. In the first automaton, a clock variable d is used to count up to 1440 (the number of minutes in a day) before being reset. In this time, if an off-hook of phone 1 occurs followed by a connection with another phone, then another clock variable t is initialised. When phone 1 is returned on-hook, q is incremented by the current value of t. In this way, q keeps track of the cumulative connection time of phone 1. In the controller automaton, phone 1 is only allowed to perform an off-hook if the value of q is ≤ 20. Otherwise, a quota violation is reported and the user of phone 1 must wait for the quota to be reset.

Composition

The above examples have illustrated the principle of our multi-paradigm specification approach. We now use these examples to illustrate the composition process. Whilst a shortage of space precludes the presentation of the complete composition process, we start with the composition of the monitors and controllers for the two scenarios presented above. Note that a worked example of the complete composition process can be found in a companion paper currently under development.

To simplify the presentation of this example below, we adopt the following shorthand: we consider "on" and "on !after_ringtone" events as the same event, we use "on1/2/3" to denote "on !id1 ∨ on !id2 ∨ on !id3" and "off1/2/3" to denote "off !dialtone !id1 ∨ off !dialtone !id2 ∨ off !dialtone !id3". We note that the monitor and controller for scenario 1 synchronise on the events "on1/2/3" and "off1/2/3"; for scenario 2, they synchronise on "on1", "off1" and "reset". The results of the two compositions are shown in figure 6.

The next stage is to compose C_1 and C_2 together, synchronising on events "on1" and "off1". This is presented in appendix B. As can be seen, the composed behaviour soon becomes quite complex, but can be calculated using the tool reported in [16]. By combining this automaton with the labelled transition system generated from the

LOTOS specification (from Appendix A), we get an even larger automaton (with 143 states). Clearly, this is too large to be (meaningfully) presented in this paper. Finally, we can impose our real-time constraints over the system by composing formulae ϕ_1 and ϕ_2 with the rest of the behaviour.

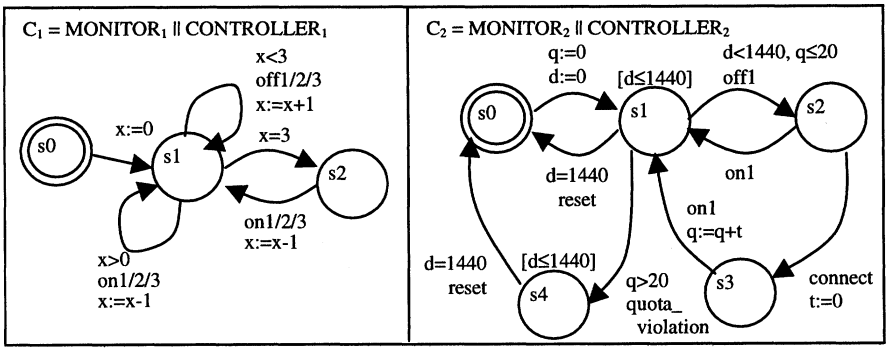

Figure 6: Composition of monitors and controllers for scenarios 1 (C1) and 2 (C2)

Conclusions

In the introduction to this paper, we gave a number of advantages for multi-paradigm specification techniques. However, before such claims can be realised, it is important to provide tractable solutions to the problems of determining consistency and achieving composition between partial specifications. Problems of consistency have been considered in detail by collaborators in our current EPSRC project e.g. [11]. We therefore focus on composition in multi-paradigm techniques. Our approach to composition is based on a common global model using timed labelled transition systems and timed automata. It should be noted that the Kronos tool [22], supports the composition of automata (to produce a product automaton). However, to our knowledge, ours is the first piece of work (and tool) to support composition and the verification of specifications developed in a multi-paradigm setting. Crucially, such an approach enables a wide range of specification techniques to be employed. Finally, our approach also gives us the advantage that we can map to realisable dynamic QoS management components.

We have illustrated the expressiveness of multi-paradigm specification techniques through a simple example of a multi-way phone system. The basic behaviour is written in LOTOS, a language ideally suited to the abstract specification of such systems. The resultant specification is clean and uncluttered. We then imposed real-time constraints on the phone system, expressing such constraints in the real-time temporal logic, QTL. Again, this proved to be a good notation to express the required properties. Finally, we introduced additional management concerns into the system using timed automata. Importantly, we have also given an illustration of composition in this example (focusing on timed automata).

Ongoing research is considering further the specification and verification of adaptive multimedia behaviour, extending the composition framework (and associated tool) to enable the expression of probabilistic and stochastic behaviour and

incorporating more advanced model checking techniques. We are also currently extending our work to use branching time temporal logic.

Acknowledgements

We would like to acknowledge the financial support of EPSRC (grant reference number GR/L28890), and also the contribution of our collaborators on the V-QoS project, namely Howard Bowman, John Derrick and Jeremy Bryans (University of Kent at Canterbury). Our thanks also go to Abderrahmane Lakas for his work on algorithms and tool support for event schedulers (whilst working on an earlier EPSRC project), and to Anders Andersen for his work on the realisation of QoS management monitors and controllers. Finally, thanks to Trevor Jones for his recent work on the development of our tool, and ongoing work relating to branching time temporal logic.

Endnotes

1. Computing Dept., Lancaster University, Lancaster LA1 4YR, e-mail: {lb, gordon}@comp.lancs.ac.uk
2. Note that in [18], we call the labelled transition system derived from the temporal logic formulae an *event scheduler*. Intuitively, the aim of this event scheduler is to schedule the functional behaviour (events) according to the specified QoS properties. To achieve this, every enabled action in the LOTOS derived labelled transition system is submitted to the event scheduler. The event scheduler then decides, according to the timing constraints (and/or stochastic constraints for the case of SQTL), if this action is allowed to happen, and, if so, when it will happen.
3. Note that, for simplicity, we have not considered past tense operators here.
4. Also related to the concept of a synchronisation function are correspondence relations (see [9]).

References

[1] R. Alur, C. Courcoubetis, D.L. Dill, "Model Checking for Real-time Systems", Proceedings of the Fifth Annual Symposium on Logic in Computer Science, pages 414-425, IEEE Computer Society Press, 1990.

[2] R. Alur, C. Courcoubetis, D.L. Dill, "Model Checking for Probabilistic Real-time Systems", In Proceedings of the 18th International Conference on Automata, Languages and Programming (ICALP'91), LNCS 510, pp 115-136, Berlin: Springer-Verlag, 1991.

[3] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, W. Yi: "UPPAAL: a Tool Suite for Automatic Verification of Real-time Systems", In Hybrid Systems III (Verification and Control), Alur, Henzinger, Sontag (eds), LNCS 1066, pp 232-243, Berlin: Springer, 1996.

[4] G.S. Blair, L. Blair, H. Bowman, A. G. Chetwynd, "Formal Specification of Distributed Multimedia Systems", London: UCL Press, 1998.

[5] L. Blair, G.S. Blair, "The Impact of Aspect-Oriented Programming on Formal Methods (Position Paper)", presented at the Aspect-Oriented Programming Workshop at ECOOP'98, Brussels, July 1998.

[6] L. Blair, G.S. Blair, A. Andersen, "Separating Functional Behaviour and Performance Constraints: Aspect-Oriented Specification", Internal Report No. MPG-98-07, see link on http://www.comp.lancs.ac.uk/computing/users/lb/vqos.html, May 1998.

[7] G.S. Blair, G. Coulson, M. Papathomas, P. Robin, "An Architecture for Next Generation Middleware", To appear in Middleware'98, The Lake District, U.K., September 1998.

[8] E. Boiten, H. Bowman, J. Derrick, M. Steen, "Issues in Multi-paradigm Viewpoint Specification", Proceedings of Viewpoints'96, SIGSOFT FSE4, 1996.

[9] E. Boiten, H. Bowman, J. Derrick, M. Steen, "Viewpoint Consistency in Z and LOTOS: A Case Study", Proceedings of Formal Methods Europe (FME'97), 1997.

[10] T. Bolognesi, E. Brinksma, "Introduction to the ISO Specification Language LOTOS", Computer Networks and ISDN Systems, Vol. 14, No. 1, pp 25-59, North-Holland, Amsterdam, 1988.

[11] H. Bowman, E.A. Boiten, J. Derrick, M.W.A. Steen, "Strategies for Consistency Checking based on Unification", to appear in Science of Computer Programming, Dec 1998.

[12] L. Drayton, A.G. Chetwynd, G.S. Blair, "An Introduction to LOTOS through a Worked Example", Computer Communications (Special Issue on FDTs in Communications and Distributed Systems), Vol. 15, No. 2, pages 70-85, Butterworth-Heinemann, March 1992.

[13] A. Finkelstein, G. Spanoudakis (eds), SIGSOFT '96 International Workshop on Multiple Perspectives in Software Development (Viewpoints '96), ACM Press, 1996.

[14] H. Garavel, "An Overview of the Eucalyptus Toolbox", Proceedings of the International Workshop on Applied Formal Methods in System Design (Maribor, Slovenia), pp 76-88, June 1996, see http://www.inrialpes.fr/vasy/Publications/Garavel-96.html.

[15] G.J. Holzmann, "Design and Validation of Computer Protocols", Englewood Cliffs (NJ): Prentice-Hall, 1991.

[16] T. Jones, L. Blair, G. Blair, "A Tool Suite for Multi-paradigm Specification", Internal Report No. MPG-98-Ed**, see http://www.comp.lancs.ac.uk/computing/users/lb/vqos.html, September 1998.

[17] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin, "Aspect-Oriented Programming", PARC Technical Report, SPL97-008 P9710042, see http://www.parc.xerox.com/spl/projects/aop/reports.html, February 1997.

[18] A. Lakas, G. S. Blair, A. Chetwynd, "Specification and Verification of Real-Time Properties Using LOTOS and SQTL", Proceedings of the 8th International Workshop on Software Specification and Design, pp 75-84, Paderborn, Germany, March 1996.

[19] K.G. Larsen, P. Pettersson, W. Yi, "Diagnostic Model-Checking for Real-Time Systems", Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, 22-24 October, 1995.

[20] Lite: LOTOS Integrated Tool Environment, Tele-Informatics and Open Systems (TIOS) Group, University of Twente, The Netherlands, http://wwwtios.cs.utwente.nl/lotos/lite/

[21] ITU Recommendation X.901-904, ISO/IEC 10746 1-4, "Open Distributed Processing – Reference Model", Parts 1-4, July 1995.

[22] S.Yovine, "Kronos: A Verification Tool for Real-time Systems", In Springer International Journal of Software Tools for Technology Transfer, 1(1/2), October 1997.

[23] P. Zave, M.A. Jackson, "Conjunction as Composition", ACM Transactions on Software Engineering and Methodology", II(4), pp 379-411, ACM Press, October 1993.

[24] P. Zave, M.A. Jackson, "Where Do Operations Come From? A Multi-paradigm Specification Technique", IEEE Transactions on Software Engineering, XXII(7), pp 508-528, IEEE, July 1996.

Appendix A. An abbreviated (uncommented) spec. of a multi-way phone system

```
specification phone_system[off, on, dial,
   cancel, ring, stop_ring, connect]: noexit
behaviour
    Phones[off, on, dial, cancel, ring,
          stop_ring, connect]
  || Exchange[off, on, dial, cancel, ring,
          stop_ring, connect]
          ({} of ListIds, {} of ListPairs)
where

process Phones[off, on, dial, cancel, ring,
          stop_ring, connect] : noexit :=
   Phone[off, on, dial, cancel, ring, stop_ring,
             connect](id1)
||| Phone[off, on, dial, cancel, ring, stop_ring,
             connect](id2)
||| Phone[off, on, dial, cancel, ring, stop_ring,
             connect](id3)
endproc

process Phone[off, on, dial, cancel, ring,
       stop_ring, connect](p:idsort) : noexit :=
  ( MakeCall[off, on, dial, cancel, ring,
       stop_ring, connect](p)
  [] ReceiveCall[off, on, dial, cancel, ring,
       stop_ring, connect](p) )
  >> Phone[off, on, dial, cancel, ring,
       stop_ring, connect](p)
endproc

process MakeCall[off, on, dial, cancel, ring,
       stop_ring, connect](p:idsort) : exit :=
   off !dialtone !p;
   DialSomeone[off, on, dial, cancel, ring,
       stop_ring, connect](p)
endproc

process ReceiveCall[off, on, dial, cancel, ring,
       stop_ring, connect](p:idsort) : exit :=
   ring !p;
   (  stop_ring !p; exit
    [] off !connect !p;
       DialSomeone[off, on, dial, cancel, ring,
          stop_ring, connect](p) )
endproc

process DialSomeone[off, on, dial, cancel,
   ring, stop_ring,connect](p:idsort) : exit :=
      ( dial !ringtone !p ?callee:idsort;
        NotEngaged[off, on, dial, cancel, ring,
          stop_ring, connect](p, callee) )
   [] ( dial !engagedtone !p ?callee:idsort;
        Engaged[off, on, dial, cancel, ring,
          stop_ring, connect](p, callee) )
   [] ( on !p; exit )
endproc

process NotEngaged[off, on, dial, cancel, ring,
          stop_ring, connect]
          (p, callee:idsort) : exit :=
    ( on !after_ringtone !p !callee; exit )
  [] ( cancel !after_ringtone !p !callee;
       DialSomeone[off, on, dial, cancel, ring,
          stop_ring, connect](p) )
  [] ( connect !p !callee;
       DialSomeone[off, on, dial, cancel, ring,
          stop_ring, connect](p) )
endproc

process Engaged[off, on, dial, cancel, ring,
          stop_ring, connect]
          (p, callee:idsort) : exit :=
    ( cancel !after_engagedtone !p;
      DialSomeone[off, on, dial, cancel, ring,
          stop_ring, connect](p) )
  [] ( on !p; exit )
endproc

process Exchange[off, on, dial, cancel, ring,
       stop_ring, connect]
       (eng:ListIds, conn:ListPairs) : noexit :=
   SubExchange[off, on, dial, cancel, ring,
          stop_ring, connect](eng, conn)
   >> accept eng:ListIds, conn:ListPairs in
   Exchange[off, on, dial, cancel, ring,
stop_ring,
          connect](eng, conn)
endproc

process SubExchange[off, on, dial, cancel, ring,
       stop_ring, connect](eng:ListIds,
       conn:ListPairs) : exit(ListIds,ListPairs) :=
  ( off !dialtone ?p:idsort;
    exit(Insert(p,eng),conn)))
[] ( on ?p:idsort;
    exit(Remove(p, eng), fully_remove(p,
conn)) )
[] ( on !after_ringtone ?from:idsort ?to:idsort;
    stop_ring !to;   exit(Remove(from,eng),
          fully_remove(from,conn)))
[] ( dial !ringtone ?p:idsort ?to:idsort
       [SystemPhone(to) and (to NotIn eng)];
    ring !to; exit(eng, conn) )
[] ( dial !engagedtone ?from:idsort ?to:idsort
       [SystemPhone(to) and (to IsIn eng)];
    exit(eng, conn) )
[] ( cancel !after_ringtone ?from:idsort
       ?to:idsort;
    stop_ring !to; exit(eng, conn) )
[] ( connect ?from:idsort ?to:idsort;
    off !connect !to; exit(Insert(to, eng),
       fully_connect(from,to, conn)))
[] ( cancel !after_engagedtone ?p:idsort;
    exit(eng, conn) )
endproc

endspec
```

Appendix B. Composition of timed automata C_1 and C_2

Table B1. Labels for the timed automata representing the composition of C_1 and C_2

Labels for states:	
s0	= s0 || s0
s1	= s1 || s1 (Inv)
s2	= s1 || s0
s3	= s2 || s0
s4	= s1 || s4 (Inv)
s5	= s2 || s4 (Inv)
s6	= s2 || s1 (Inv)
s7	= s1 || s2
s8	= s2 || s2
s9	= s2 || s3
s10	= s1 || s3
s11	= s0 || s1 (Inv)
s12	= s0 || s4 (Inv)
States marked (Inv) must satisfy the invariant [d ≤ 1440]	

Labels for behaviour (of the form {g;a;v}):	
b1:	{ _ ; _ ; x:=0 }
b2 :	{ x<3; off2/3; x:=x+1 }
b3 :	{ x>0; on2/3; x:=x-1 }
b4 :	{ d<1440,q≤20,x<3; off1; x:=x+1 }
b5 :	{ d=1440; reset; _ }
b6 :	{ _ ; _ ; d:=0,q:=0 }
b7 :	{ x=3; _ ; _ }
b8 :	{ _ ; on2/3; x:=x-1 }
b9 :	{ q>20; quota_violation; _ }
b10 :	{ x>0; on1; x:=x-1 }
b11 :	{ _ ; on1; x:=x-1 }
b12 :	{ _ ; connect ; t:=0 }
b13 :	{ _ ; on1; q:=q+t,x:=x-1 }
b14 :	{ x>0; on1; q:=q+t,x:=x-1 }

These labels serve to simplify the presentation of the graph below. Note that the composition of C_1 and C_2 synchronises on events "on1" and "off1". The result is presented below in figure B1.

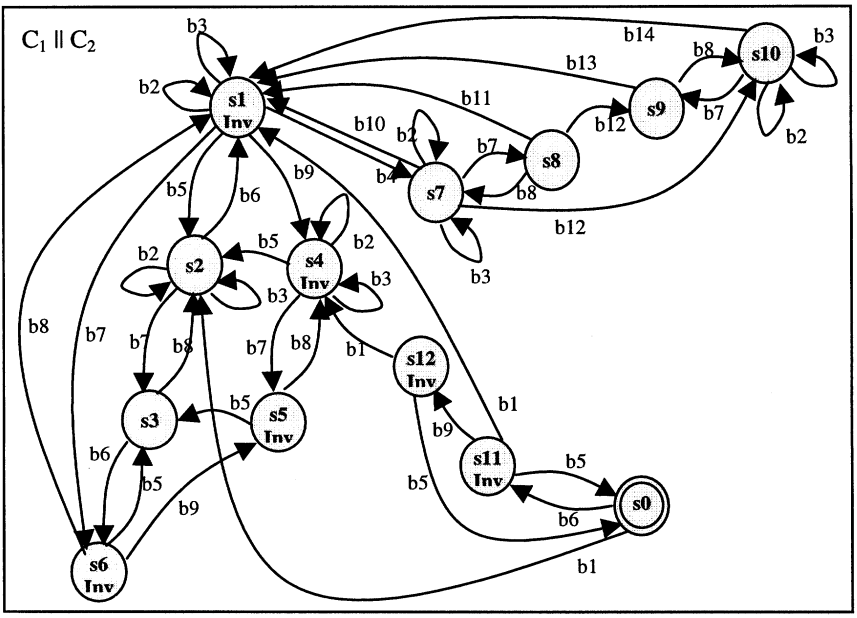

Figure B1: The timed automata representing the composition of C_1 and C_2