Imprecise Synthesis

Bill Mitchell
Department of Computing, University of Surrey
Guilford, Surrey GU2 7XH, UK
w.mitchell@surrey.ac.uk

Robert Thomson, Paul Bristow
Motorola UK Research Lab, RG22 4PD, UK
{brt007, paul.c.bristow}@motorola.com

Abstract

Around a third of significant faults in embedded systems are caused by defective requirements specifications. Indstrial case studies using existing techniques to construct analytical requirements models have been frustrated because of imprecise compositional semantics in the industrial specifications. The paper describes a method for synthesising analytical models from imprecise MSC requirements specifications that ensures the models reflect the intended compositions, and not those that result from the imprecision within the requirements.

1. Introduction

Scenario based specifications are common in telecommunications and automotive concurrent embedded systems. Many of these scenarios are described as Message Sequence Charts (MSCs). MSC is mandated by standards bodies such as the ITU and ETSI [4] for defining scenarios within standards. MSC 2000 [13] is currently being incorporated within UML 2.0. Studies have shown that around 30% of faults in embedded systems are caused by errors and omissions of requirements specifications [12]. Hence the study of analytical models for MSC based specifications is a worthwhile exercise.

The paper describes a method for synthesizing requirements MSC specifications into a particular type of finite state machine that characterises permissible compositions of the requirements scenarios. The method was formulated from an industrial case study of around three hundred MSCs used within Motorola to define TETRA [11] requirements. Industrial MSC scenarios are commonly annotated with global state like information, which should make composition straightforward. However, these states are often used imprecisely across different requirements scenarios. Therefore not all the compositions that result from treating the states as if they are precisely defined will be valid. We will refer to these state like constructs as phases.

Intuitively we regard a phase as a composite global state with imprecise compositional semantics. A composite state within a UML statechart can have several entry and exit states (as well as a start and stop state). Where a phase occurs in an MSC scenario it does not specify which entry or exit states are used. This makes composition based on state semantics ambiguous. When the same phase occurs in two MSCs it is not clear whether the entry and exit states of the two phase occurrences are the same. The case study showed that it is only safe to assume they are the same when the phase occurrences are reached in a consistent manner in both scenarios. The paper defines a phase semantics that formalizes when two occurrences of a phase are consistently reached and define the same entry and exit states. The paper then defines a phase transition process that captures the valid compositions of process behaviours with respect to the phase semantics. The phase transition process is canonical up to a particular kind of simulation equivalence that respects phase transitions (Definition 6.1). This leads to a process algebra that generates the phase transition process from the MSC requirements, (Theorem 7.1). The phase transition process is always finite (Theorem 7.3) and can therefore be represented by a finite state machine.

There has been much work on the area of synthesising scenario based specifications into analytical models. Alur and Yannakakis [2] described a method for realizing MSC scenarios as processes that capture the implied behaviour of the specifications. Bontemps and Schobbens [7] have described methods for synthesising automata from Live Sequence Charts (LSCs, [9]), and have used game theory [3] to automatically construct strategies for constructing programs from LSCs. Uchitel, Kramer and Magee [10] describe methods to automatically construct UML statecharts from MSC scenarios that are annotated with state information.

This work has been applied to industrial requirements specifications in Motorola case studies. These were frustrated by the inconsistent manner that state like annotations were introduced into scenarios, which invalidated the resultant models. This motivated the phase semantics given here, which circumvented the issues identified by the case studies. The method described here has been implemented by Motorola UK Research Labs, and is being used in a pilot study for a new telecommunications mobile 3G handset.

The phase semantics is a weakening of the state semantics from [10], whilst permitting some additional semantics in the spirit of LSCs. The semantics here differs from that of LSCs in that mandatory behaviour is defined dynamically within the domain of possible scenarios. This permits a semantics which uses domain knowledge to define valid compositions.

2. Partial Order Semantics for MSCs

An MSC defines a partial order semantics on the order that system events can be observed to occur. Such a partial order is referred to as the causal order of the MSC. The causal order restricts how events can occur within any trace of the system. The semantics are well known in the literature ([1], [5], [13]), so we only provide a terse exposition here. For arbitrary MSCs that include iteration the set of traces generated by the MSC for the whole system is not always regular [1], however the traces of each individual system process in an MSC are always regular which is sufficient for our purposes.

Let $<$ be a causal order on a set of events E for some MSC. A message m defines two events, a send event $!m$ and a receive event $?m$. A causal ordering defines the partial order semantics for a set of concurrent events in an MSC scenario. For a set $S \subseteq E$ define

$$\begin{aligned} \mathsf{n}(S,<) &= \{x \in E \mid \exists\, y \in S : y < x, \\ &\qquad \text{and } \neg\exists\, z \in E : y < z < x\} \\ \mathsf{m}(S,<) &= \{x \in S \mid \neg\exists\, y \in S : y < x\} \\ \mathsf{af}(a,S,<) &= \mathsf{m}((S - \{a\}) \cup \mathsf{n}(\{a\},<),<) \end{aligned}$$

The set $\mathsf{af}(a,S,<)$ characterises how events may follow a in an execution trace, where S describes the set of all events that are eligible to occur concurrently with a. Suppose we have an execution trace t that is a total extension of $<$. Let a be some event in t, so that t is of the form $t_0 \cdot a \cdot t_1$ (where $\cdot$ denotes concatenation). Let S be the set of minimal events from the set of all events in t_1. Then t_1 must be of the form $b \cdot t_2$ where $b \in \mathsf{af}(a,S,<)$.

For a set $S \subseteq E$ recursively define a process term on S from $<$ by

$$\mathsf{Pr}(S,<) = \sum_{\{a \in S\}} a \cdot \mathsf{Pr}(\mathsf{af}(a,S,<),<)$$

and $\mathsf{Pr}(\emptyset,<) = 0$.

For a partial order $<$ on events E, the term $\mathsf{Pr}(\mathsf{m}(E,<),<)$ defines the partial order semantic process for $<$. Given an MSC M, and its partial order semantics $<_M$, let $\mathsf{Pr}(M) = \mathsf{P}(\mathsf{m}(E,<_M),<_M)$. Then $\mathsf{Pr}(M)$ defines the externally observable behaviour of the system defined by M. Similarly for an MSC M, for each process P in M we can extract the set of events $E(P)$ that occur in P and the causal ordering $<_P$ imposed by M on the events in P. The process $\mathsf{Pr}(P) = \mathsf{Pr}(\mathsf{m}(E(P),<_P),<_P)$ then defines the observable behaviour of P in M. In [5] they define a particular type of asynchronous parallel composition $|_A$, together with a special traffic channel process T. Roughly the idea is that when a process transmits a send message $!m$ it is stored by the traffic channel until the relevant process wishes to consume the receive message $?m$. Then T delivers the message. T is designed so that it acts rather like an unbounded buffer with random access.

One can prove [5] that when M consists of processes P_i for $1 \le i \le n$, then $\mathsf{Pr}(M)$ is strong bisimulation equivalent to

$$T \mid_A \mathsf{Pr}(P_1) \mid_A \mathsf{Pr}(P_2) \mid_A \cdots \mid_A \mathsf{Pr}(P_n)$$

In general it is also provable that $\mathsf{Pr}(S,<_1)$ is strong bisimulation equivalent to $\mathsf{Pr}(S,<_2)$ if and only if the set of traces of $\mathsf{Pr}(S,<_1)$ are the same as $\mathsf{Pr}(S,<_2)$. This is proved by showing that the set of traces of $\mathsf{Pr}(S,<)$ uniquely define $<$ with respect to S.

We may therefore identify each process in an MSC first with a process algebra term defined by the causal ordering in the MSC, and second by a set of traces that are defined by that process.

Definition 2.1 *Define $T(P)$ to be the set of traces generated by the process $\mathsf{Pr}(P)$.*

The definition here for $\mathsf{Pr}(P)$ has to be extended for MSC scenarios that include alternatives, process creation, iteration and so on. For further details refer to [5].

3. Informal Compositional Semantics

Informally we can give requirements scenarios the following semantics, which will allow us to construct phase transition processes from them. Suppose we have a scenario M that defines message exchanges between processes, including the process Q.

Consider figure 3, which is a requirements scenario for a wireless mobile handset. This describes how a WAP 'Browser' process downloads a Java application iteratively from the 'Air Interface' process until it receives the 'EOF' message, or it detects that the file is corrupted.

The extended hexagonals are MSC condition symbols that describe which operational phase is active at any time. We will refer to them as phase symbols from now on. A phase should be regarded as a state, but with imprecise compositional semantics which are context dependent.

An event x in an MSC M occurs in the scope of phase symbol u if the first phase symbol prior to x within the process it belongs to is u. We refer to a phase that x is in the scope of as an active phase for x.

Given a trace of events for a process, we can annotate each event with the phases that were active before and after that event occurred. When these phases are different the event is called a phase transition event. Such an annotated trace is defined to be a phase trace, which we formally define later (see definition 4.1). Consider two phase traces t_1 and t_2, where t_2 is a suffix of t_1 and terminates with a phase transition. Hence we may write $t_1 = t_3 \cdot t_2$ for some t_3, and let the end event of t_2 be an event e that causes a transition to some phase p. I.e. $t_2 = t_2' \cdot e$ where the phase prior to e is not p, and the phase after e is p. In this case we will say t_1 and t_2 match and that p is reached consistently. Hence we can suppose the two occurrences of p refer to the same entry and exit states in both traces. This leads us to the idea of phase transition simulation between processes.

A process P simulates the phase transitions of Q when the following holds. If we observe a trace of annotated events of P that leads to a phase transition, with some suffix equal to a phase trace of Q, then P must be able to simulate the (annotated) behavior of Q from then on. We define this equivalence formally in Definition 6.1. Given a number of specification processes Q_i it is possible to define a canonical process that simulates the phase transitions of them all as will be defined in section 7. This is the phase transition process mentioned in the introduction.

This informal semantics is similar to LSC semantics where a scenario can express some behaviour as universal, which must occur once a particular guard in the form of a prechart has been witnessed. That is if ever the system behaviour matches the prechart part of the LSC it must then match the rest of the behaviour in the main chart. The difference here is that when certain initial behaviour is matched then the subsequent phase symbol(s) take on state like semantics, which is similar to but not the same as the state semantics of [10].

Another difference is that an MSC is not explicitly divided into distinct sections consisting of prechart and main chart. If P matches any initial behaviour leading to a transition, then it matches all the remaining behaviour. Hence which parts of the scenario act like a prechart guard, and which like the main chart are determined dynamically. In section 5 we will define context semantics in order to include some domain knowledge that allows matching to be applied consistently across large specification repositories.

4. Annotated Events

Let $\mathcal{P}$ be the set of phase symbols associated with an MSC M, and let ψ be a map that defines the set of phase names for each symbol. I.e. $\psi : \mathcal{P} \longrightarrow 2^{\mathsf{Ph}}$, where Ph is the set of phase names.

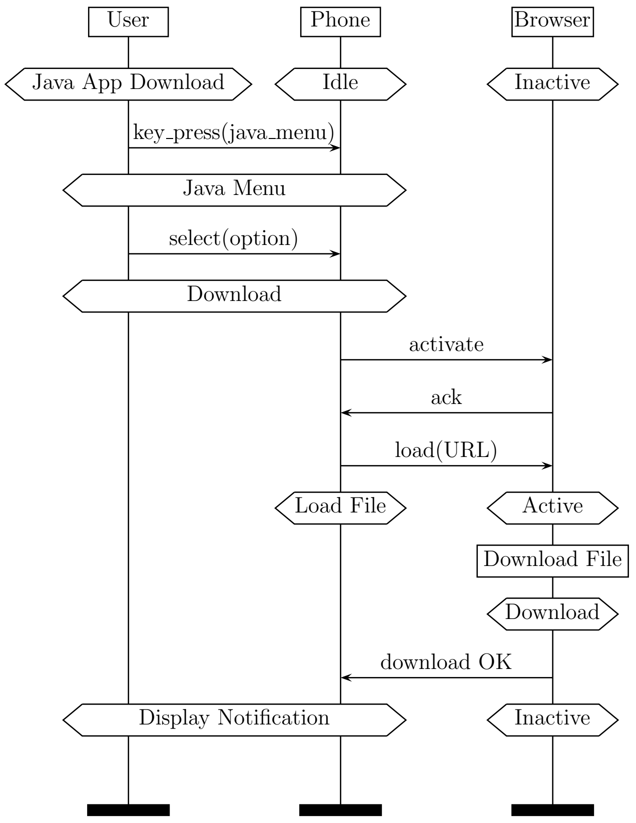

Figure 1. Java App Download Scenario

Where E is the set of events for an MSC M, let $\phi : E \longrightarrow \mathsf{Ph}$ be the function that maps each event e to the set of phases it belongs to, that is the set $\psi(u)$, where e is in the scope of u.

Definition 4.1 *Define the phase traces for a process P in an MSC scenario M to be sequences of triples:*

$$(S_0, e_0, S_1)\,(S_1, e_1, S_2)\cdots(S_n,\, e_n,\, S_{n+1})$$

where $e_0, \ldots, e_n$ is an event trace of P, $S_i \subseteq \mathsf{Ph}$, $\phi(e_i) = S_i$, and S_{n+1} is the last phase for process P in the scenario M.

In figure 1 each process generates a single phase trace. For example the phase trace t_0 for the 'Browser' process is:

({Inactive},	?activate,	{Inactive})
({Inactive},	!ack,	{Inactive})
({Inactive},	?load(URL),	{Active})
({Active},	Download File,	{Download})
({Download},	!download OK,	{Inactive})

A phase trace is deliberately chosen to look like a sequence of transitions within a finite state automaton, since

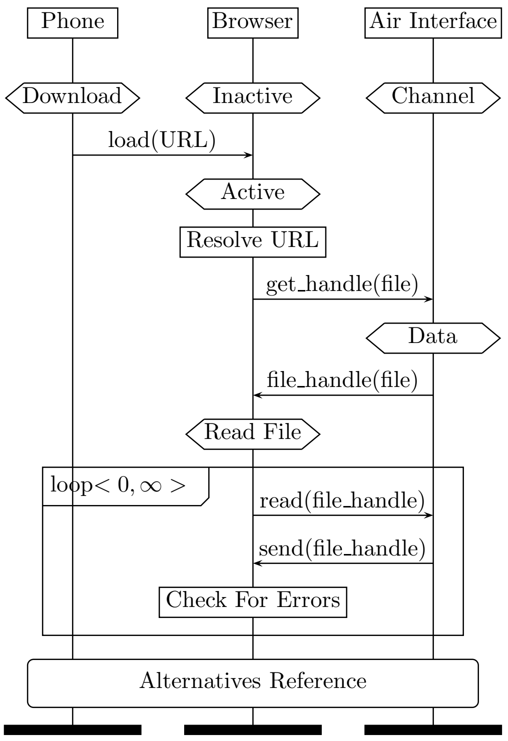

Figure 2. Iterative Browser Download

our aim is to provide a semantics that is a weakening of the state semantics, which will be suitable for more imprecise specifications. Each triple in a phase trace is referred to as an *annotated event*.

5. Dynamic Constraints

Phase names found in industrial examples are not consistently chosen, nor is their meaning consistent across large sets of MSCs. It is therefore necessary to further complicate phase semantics by introducing a mechanism to relate phase names within a given context.

In order to relate phase names within scenarios we will treat them as if they are temporal boolean propositions, the moments where they are true define when the relevant phases are active. We will use a Hennessy Milner style of temporal logic to permit phases to act as a type of temporal guard [8]. A temporal model T consists of a directed graph G, with vertex labelling $\nu : G_V \longrightarrow 2^{\mathsf{Ph}}$, edge labelling $\varepsilon : G_E \longrightarrow E$, and some vertex i that represents the initial moment. Temporal formulae are defined as usual:

- $T, v \vDash \langle e \rangle \phi$ iff there is an edge $(v, w) \in G_E$ such that $\varepsilon(v, w) = e$, and $T, w \vDash \phi$
- $T, v \vDash [e]\phi$ iff for every edge $(v, w) \in G_E$ where $\varepsilon(v, w) = e$, $T, w \vDash \phi$
- $T, v \vDash \Box\phi$ iff $T, v \vDash \phi$ and $T, w \vDash \Box\phi$ for every edge $(v, w) \in G_E$
- $T, v \vDash \Diamond\phi$ iff there is some vertex w reachable from v such that $T, w \vDash \phi$

The satisfiability of ordinary boolean formulae is defined as usual. Formula ϕ is satisfied in T when $T, i \vDash \phi$. ϕ is valid when it is satisfied in every model, when we write $\vdash \phi$.

Definition 5.1 *For a set $S \subseteq \mathsf{Ph}$, define $\bigwedge S = \bigwedge_{x \in S} x$. For a phase trace $t = (S, e, S') \cdot t'$, define its temporal semantics as*

$$\|t\| = \bigwedge S \wedge \langle e \rangle (\bigwedge S' \wedge \|t'\|)$$

A context $\mathcal{C}$ is any temporal formulae over $\mathcal{P}$ and E.

A temporal context controls how phases are related across the requirements scenarios.

Consider figure 2, which illustrates a scenario where a file is downloaded iteratively from the air interface by the WAP browser process. At each iteration the file is tested for errors. This example is based on a real requirements scenario used in one of the Motorola case studies. The rest of the scenario then continues in another MSC 'Alternatives Reference'. Hence the file download will terminate either when an error is detected, or the 'EOF' signal is received.

In this example suppose that if a load(URL) message is received then the 'Load File' phase is valid whenever Active is valid. The context is then

$$\Box([?\mathsf{load(URL)}](\mathsf{Active} \Rightarrow \text{'Load File'}))$$

The temporal context permits phases defined by different development teams to be given a consistent meaning across scenarios without having to extensively rewrite the scenarios.

Definition 5.2 *For context $\mathcal{C}$ we define phase trace t to match phase trace t' when*

$$\vdash \mathcal{C} \Rightarrow (\|t\| \Rightarrow \Diamond\|t'\|)$$

The matching formula is true when some suffix of the sequence t contains exactly the same event trace as the whole of t', and the phase annotations of the corresponding events are logically consistent within the context defined by $\mathcal{C}$. Intuitively t matches t' if after some initial delay, t' becomes the same as t within the context defined by $\mathcal{C}$.

Definition 5.3 *Let $a = (S, e, S')$ be an annotated event. When $\nvdash \mathcal{C} \Rightarrow (\bigwedge S \Rightarrow \bigwedge S')$ define a to be a phase transition event.*

Define a phase transition trace to be a trace of annotated events terminating with a phase transition event.

Note in the case where $\mathcal{C}$ is a tautology this is equivalent to $S' \not\subseteq S$, so that there must be at least one new phase name in S' that is not in S. Hence at least one phase has changed.

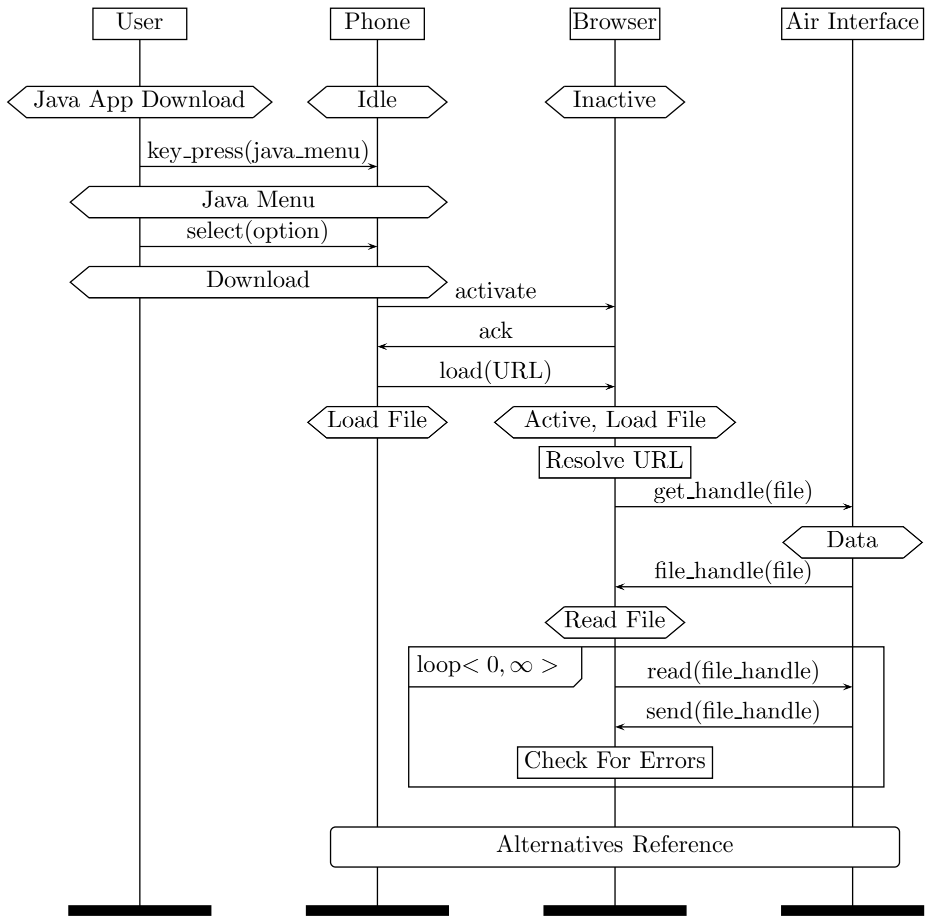

Figure 3. Overlap Scenario

Let t_1 be the phase transition trace of the phase trace t_0 consisting of

$$\begin{array}{rlll} t_1 = & (\{\mathsf{Inactive}\}, & ?\mathsf{activate}, & \{\mathsf{Inactive}\}) \\ & (\{\mathsf{Inactive}\}, & !\mathsf{ack}, & \{\mathsf{Inactive}\}) \\ & (\{\mathsf{Inactive}\}, & ?\mathsf{load(URL)}, & \{\mathsf{Active}\}) \end{array}$$

In figure 2 the initial annotated event of process 'Browser' is $t_2 = (\{\mathsf{Inactive}\}, ?\mathsf{load(URL)}, \{\mathsf{Load\ File}\})$. From this we can prove:

$$\begin{array}{ll} \vdash & \Box([\mathsf{load(URL)}](\mathsf{Active} \Rightarrow \\ & \text{`}\mathsf{LoadFile}\text{'})) \Rightarrow (\|t_1\| \Rightarrow \Diamond\|t_2\|) \end{array}$$

so that t_1 matches t_2.

6. Phase transition simulation

Given processes whose actions are annotated events we define first a simulation relation between them, and then a phase transition simulation relation. For annotated events $a = (S, e, S')$ and $b = (U, g, U')$ define $a \sqsupset_{\mathcal{C}} b$ when $e = g$, $\vdash \mathcal{C} \Rightarrow (\bigwedge U \Rightarrow \bigwedge S)$ and $\vdash \mathcal{C} \Rightarrow (\bigwedge U' \Rightarrow \bigwedge S')$.

Define P to simulate process Q within context $\mathcal{C}$, written as $P \sqsupset_{\mathcal{C}} Q$, if $\forall a$ such that $Q \xrightarrow{a} Q'$ there is some a' where $P \xrightarrow{a'} P'$ such that $a' \sqsupset_{\mathcal{C}} a$ and

$$P' \sqsupset_{\mathcal{C}} Q'$$

This simulation relation forces phases to be compatible as well as ensuring the events are simulated correctly.

For annotated events a_i and phase trace $t = a_0 \cdot a_1 \cdots a_{n-1}$, let $P \xrightarrow{t} P'$ denote that there are processes P_i, for $0 \le i \le n$, such that $P_i \xrightarrow{a_i} P_{i+1}$, $P_0 = P$ and $P_n = P'$.

Definition 6.1 *Define P to simulate the phase transitions of process Q within context $\mathcal{C}$, written as $P \trianglerighteq_{\mathcal{C}} Q$, when the following holds. For all phase transition traces t such that $Q \xrightarrow{t} Q'$, and for all phase traces τ that match t, whenever there is a process P' such that $P \xrightarrow{\tau} P'$ then $P' \sqsupset_{\mathcal{C}} Q'$.*

Definition 6.2 *Let $\{M_i \mid 0 \le i \le n\}$ be a set of scenarios, let Q_i be a process from M_i for each i. That is each Q_i defines exactly the observed behavior of one process in scenario M_i.*

$$\begin{array}{rcl}
P \mid Q & = & P \lhd Q + P \rhd Q \\
a \cdot P \lhd b \cdot Q & = & a \cdot P \lhd\!| \; b \cdot Q \;\text{ if }\; a \sqsupset_{\mathcal{C}} b \\
a \cdot P \lhd b \cdot Q & = & a \cdot (P \lhd b \cdot Q) \;\text{ if }\; a \not\sqsupset_{\mathcal{C}} b \\
P \rhd Q & = & Q \lhd P \\
0 \lhd Q & = & 0 \\
a \cdot P \lhd\!| \; b \cdot Q & = & (a \cup b) \cdot (P \lhd\!| \; Q) \;\text{ if }\; a \sqsupset_{\mathcal{C}} b \;\text{ and }\; \neg\eta_{\mathcal{C}}(a) \\
a \cdot P \lhd\!| \; b \cdot Q & = & (a \cup b) \cdot (P \parallel Q) \;\text{ if }\; a \sqsupset_{\mathcal{C}} b \;\text{ and }\; \eta_{\mathcal{C}}(a) \\
a \cdot P \lhd\!| \; b \cdot Q & = & a \cdot P + b \cdot Q \;\text{ if }\; a \not\sqsupset_{\mathcal{C}} b \;\text{ and }\; \eta_{\mathcal{C}}(a) \\
a \cdot P \lhd\!| \; b \cdot Q & = & a \cdot P \;\text{ if }\; a \not\sqsupset_{\mathcal{C}} b \;\text{ and }\; \neg\eta_{\mathcal{C}}(a) \\
0 \lhd\!| \; Q & = & 0 \\
a \cdot P \parallel b \cdot Q & = & (a \cup b) \cdot (P \parallel Q) \;\text{ if }\; a \sqsupset_{\mathcal{C}} b \\
P \parallel Q & = & Q \parallel P \\
0 \parallel Q & = & Q \\
a \cdot P \parallel b \cdot Q & = & a \cdot P + b \cdot Q \;\text{ if }\; a \not\sqsupset_{\mathcal{C}} b \;\text{ and }\; b \not\sqsupset_{\mathcal{C}} a
\end{array}$$

Figure 4. Phase Transition Process Algebra

We define process P to be the phase transition representation of processes Q_i when $P \trianglerighteq_{\mathcal{C}} Q_i$ for each i. Define the overlaps *of P to be those phase transition traces of P that are not contained in any of the Q_i.*

7. Phase Transition Processes

In figure 4 we briefly describe a process algebra that defines how to synthesise a phase transition representation from a set of processes described by the requirements scenarios. The algebra essentially defines an algorithm for the construction of a minimal phase transition representation as explained in theorem 7.1.

Let $\mathcal{A}$ be the set of annotated events. Let $+$ be the usual choice operator over process terms. Let $\cdot$ be the usual composition operator of atomic actions and process terms. Let $\eta_{\mathcal{C}} : \mathcal{A} \longrightarrow \mathbb{B}$ be a boolean valued function that defines when an annotated event is a phase transition. That is $\eta_{\mathcal{C}}(S, e, S') = \mathsf{t}$ when $\not\models \mathcal{C} \Rightarrow (\bigwedge S \Rightarrow \bigwedge S')$. For annotated events $a = (S, e, S')$ and $b = (U, e, U')$ define $a \cup b = (S \cup U, e, S' \cup U')$.

Proposition 7.1 *Given a set Q of processes Q_i from requirements scenarios M_i for $0 \leq i \leq n$, then*

$$P = Q_0 \mid Q_1 \mid \cdots \mid Q_n$$

is a phase transition representation of Q. Where $\mid$ is defined by the axioms of figure 4.

If P' is another phase transition representation of Q, then $P' \sqsupset_{\mathcal{C}} P$. That is P is canonical up to simulation equivalence. Define P to be the phase transition process for Q.

Figure 3 describes one of the overlaps given by the phase transition process of the 'Browser' processes in figure 1 and figure 2.

Given that we have a context where Active and 'Load File' are valid at the same time, figure 3 can be viewed as an abstract form of 'cut' and 'paste' of the two scenarios in figure 1 and figure 2.

In order to simply depict a phase transition process we define a straightforward translation into a state machine description.

Definition 7.2 *A phase automaton consists of a set of events E, states $\mathcal{P}$ and transitions from $\mathcal{P} \times E \times \mathcal{P}$. A phase automaton also has a function $\psi : \mathcal{P} \longrightarrow 2^{\mathsf{Ph}}$.*

Given a process that has annotated events for actions, we can simply translate it into a phase automaton consisting of the following state transitions. Each action transition $P \xrightarrow{a} P'$, where $a = (S, e, S')$, defines a state transition $u \xrightarrow{e} u'$ for each $u \in \psi^{-1}(S)$, and $u' \in \psi^{-1}(S')$.

It turns out that the phase automaton defined by the phase transition process of a collection of processes from MSC scenarios is always regular:

Proposition 7.3 *The phase automaton of a phase transition process is always finite.*

Figure 5 is the phase transition process of the two 'Browser' processes defined in figures 1 and 2. Those states that belong to the same phase are grouped together in a box labelled with the phase name, resulting in a state chart like diagram. The dotted arrows represent the part of the process behavior that is exclusive to figure 1. The solid arrows are the behavior that is defined by figure 2.

The grey box denotes where phase trace t_1 matches t_2. This match defines where the two 'Browser' processes from figures 1 and 2 are joined together. The process is depicted as a phase automaton. The temporal context here causes Active and Load File to be simultaneously valid, hence both phase names label the same phase in the automaton.

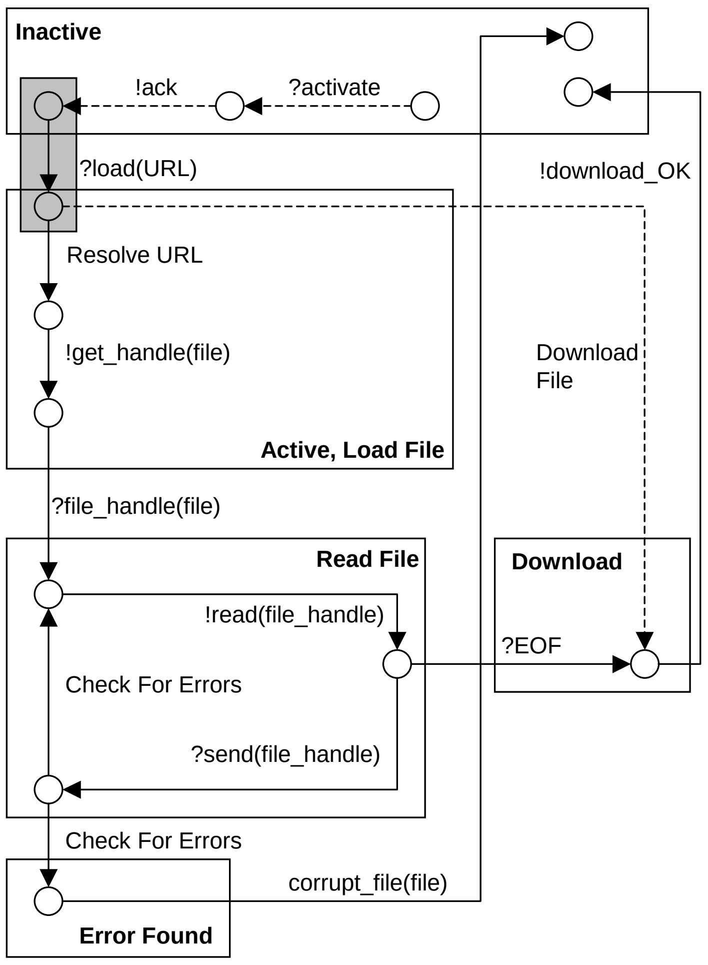

Figure 5. 'Browser' Phase Transition Process

The phase transition process P is built by joining together specification scenario processes wherever there is a match between phase transition traces. The process algebra of figure 4 captures this idea formally.

For an annotated event $a = (S, e, S')$ let $*a = e$. Let P be a process that has annotated events as actions. Let A be a state machine that accepts some subset of E^*. Define $A \sqsupset P$, if for all $P \xrightarrow{a} P'$ there is some $A \xrightarrow{*a} A'$ such that $A' \sqsupset P'$. That is when reduced to a process over plain events P can be simulated by A in the usual sense.

Proposition 7.4 *Let P be the phase transition representation of a set of processes Q_i from MSC scenarios M_i, where the temporal context $\mathcal{C}$ is a tautology.*

Let A be the state chart of the Q_i processes defined according to the semantics of [10] where each set of phase names attached to a phase symbol from the M_i is mapped to a unique state name. Then $A \sqsupset P$.

This shows that the phase transition semantics here weakens the state based semantics of [10] so that less compositions are legitimate, which is intended to reflect the imprecise nature of industrial scenarios.

7.1. Conclusions

Around a third of significant defects can be traced to requirements specifications. Hence it is important to be able to construct a model of possible compositions of the requirements as an analytic tool to facilitate the detection of such defects. Such a model is also useful in ensuring sufficient coverage of test cases for feature interactions implied by the requirements, which are often caused by composition between requirements for different features.

Unfortunately MSC requirements scenarios usually have imprecise compositional semantics that makes it hard to synthesise an analytical model of their possible compositions. Here we have defined a process algebra that defines how such imprecise scenarios can be composed. The algebra allows phase symbols to have global state like semantics when there is a suitable overlap of concurrent behaviour between scenarios. This ensures composition occurs only where phase symbols have consistent state like definitions.

The research reported in this paper is a consequence of case studies of Motorola requirements scenarios. The work reported here has been incorporated into the *ptk* tool suite [6], has been validated against a suite of industrial requirements specifications, and is being applied in a pilot study for a new mobile handset for Motorola.

References

[1] R. Alur and M. Yannakakis, Model checking of message sequence charts, Proceedings of the Tenth International Conference on Concurrency Theory, Springer Verlag, 1999

[2] R. Alur, K. Etessami, M. Yannakakis, Inference of Message Sequence Charts, Proceedings 22nd International Conference on Software Engineering, pp 304-313, 2000.

[3] Yves Bontemps, Patrick Heymens, Turning high-level live sequence charts into automata, Proc of Scenarios and State Machines: Models Algorithms and tools, 24th International Conf. on Software Engineering, May 2002, ACM.

[4] European Telecommunications Standards Institute, `http://www.etsi.org`

[5] T. Gehrke, M. Hilhn, H. Wehrkeim, An Algebraic Semantics for Message Sequence Chart Documents, in Formal Description Techniques, Chapman Hall 1998.

[6] P. Baker, P. Bristow, C. Jervis, D. King, B. Mitchell, Automatic Generation of Conformance Tests From Message Sequence Charts, Proceedings of 3rd SAM Workshop 2002, Telecommunications and Beyond: The Broader Applicability of MSC and SDL, pp 170-198, LNCS 2599.

[7] Yves Bontemps, Pierre-Yves Schobbens, Synthesis of Open Reactive Systems from Scenario-Based Specifications, Third International Conference on Application of Concurrency to System Design (ACSD'03)

[8] M. Hennessy, R. Milner, Algebraic Laws for Nondeterminism and Concurrency, Journal of the ACM, 32: 137-161, 1985.

[9] Werner Dam, David Harel, LSCs: Breathing life into message sequence charts, Formal Methods in System Design, 19(1):45-80, 2001.

[10] Sebastian Uchitel, Jeff Kramer, Jeff Magee, Synthesis of Behavioral Models from Scenarios, IEEE Transactions on Software Engineering, vol. 29, no. 2, February 2003

[11] Annex C, Service Diagrams related to the model of Mobile user, Terrestrial Trunked Radio (TETRA); Voice plus Data (V+D); Designers' guide; Part 2: Radio channels, network protocols and service performance, European Telecommunications Standards Institute 1997.

[12] E. Wong, J. R. Horgan, W. Zage, D. Zage and M. Syring, Applying Design Metrics to a Large-Scale Software System, (Motorola), Proceedings of the 9th International Symposium on Software Engineering Reliability (ISSRE 98), Paderborn, Germany, November 4-7, 1998.

[13] Z.120 (11/99)ITU-T Recommendation - Message Sequence Chart (MSC)