Scenario Synthesis from Imprecise Requirements

Bill Mitchell[1], Robert Thomson[2], Paul Bristow[2]

[1] w.mitchell@surrey.ac.uk,
Department of Computing, University of Surrey
Guilford, Surrey GU2 7XH, UK
[2] {brt007, paul.bristow}@motorola.com
Motorola UK Research Lab, RG22 4PD, UK
May 6, 2004

Abstract. Discovering faults in requirements specifications for distributed reactive systems is a challenging problem since many issues that need to be uncovered are a result of subtle component interactions that are implied by the requirements, but not explicitly described by them. A further difficulty is caused by the imprecise nature of industrial requirements specifications. This makes it difficult to construct valid models of the possible compositions between the requirements, which would be a valuable aid in uncovering such interactions.
The paper defines a formal semantics that characterizes a particular type of imprecise compositional semantics derived from industrial case studies, and a process algebra that describes the valid requirements compositions for that formal semantics.

1 Introduction

Telecommunications protocol requirements specifications often consist almost solely of normative MSC scenarios, together with english text. Requirement specification MSC scenarios tend not to provide a comprehensive set of examples, and contain implicit behavior that can easily be missed, or misinterpreted by software developers. Studies have shown that approximately a third of all serious defects are a result of poor requirements [11]. It is therefore important to derive a comprehensive set of scenarios describing implicit compositions between the requirements for use in uncovering potential defects in the specifications and as test purposes for the development process. However, in various case studies at Motorola it has been shown that although MSC scenarios are precise about message definitions and exchanges, industrial requirements specifications are often imprecise about their compositional semantics. That makes it difficult to construct a valid model of the requirements compositions. The MSC scenarios in the case studies were annotated with global state like information, which should make composition straightforward. However, these states were 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 a phase represents a set of global concurrent states with imprecise compositional semantics. Where the same phase occurs in two MSC scenarios it is not immediate that it represents the same global states in both scenarios. They can only safely be assumed to be the same states if the phase is reached in a consistent manner in both scenarios.

The paper defines a formal phase semantics for MSC scenarios, which was formulated from an industrial case study involving around three hundred MSC scenarios. This formalizes when two occurrences of a phase are consistently reached and define the same global concurrent states. This leads to a technique for synthesising *phase composition processes* from a collection of requirements scenarios. These characterise the 'valid' compositions of requirements specifications that have imprecise compositional semantics. The compositions are a subset of those given by regarding the phases as precisely defined concurrent global states. Note we use the term 'valid' within the context of the industrial case studies.

Related Work

Preliminary results were first reported in [7]. The current paper differs in that it allows phases to be simultaneously active, describes how to combine processes rather than traces and permits a temporal context that controls when features can be concurrent.

There appears to no work in the literature that attempts to define a formal semantics for composition of informal MSC scenarios. There are however several papers concerned with model synthesis of formal MSC scenarios. In [1] they describe how to generate some implied scenarios from basic MSCs. In [9], [8] they address the problem of synthesising statecharts from MSC scenarios. In [9] they compose synchronous MSC-s into statecharts by using global state names incorporated in the MSC scenarios. Phases are closely related to global states, but they are not the same. They have state like semantics when certain behavioral constraints hold. This state like semantics is dynamically determined by the concurrent behaviour described by the requirements. In [2], [3] they define scenario and program synthesis from live message sequence charts (LSCs). They do not use global state annotation, however the phase semantics here incorporates some of the ideas of mandatory behaviour from LSCs that permits the state like semantics to be determined dynamically.

2 MSC Phase Transition Scenarios

In this paper we assume MSCs are defined in accordance with the MSC 2000 standard [10]. An MSC scenario describes message exchanges between processes that achieve a transition between major operational phases of the system. Consider figure 4, which describes how a '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 shortened hexagonals are MSC condition symbols that describe which operational phase is active at any time. To emphasize this point we will refer to them as phase symbols from now on. Phase symbol labels will be identified with propositional boolean formulae in the paper. In figure 4 'Browser' starts the scenario in phase 'Inactive', that is the proposition 'Inactive' is given value true and all other phases are made false for that process. Then phases 'Active' and 'Load File' become active simultaneously, hence are given value true and 'Inactive' is given value false. The point at which a phase symbol is introduced into a scenario is defined as a phase transition. This interpretation of MSC condition symbols is an extension of the MSC 2000 standard where condition symbols have no formal semantics. Common engineering practise treats MSC condition symbols as global states, but unfortunately in an imprecise fashion that forces the phase semantics that are discussed in this paper.

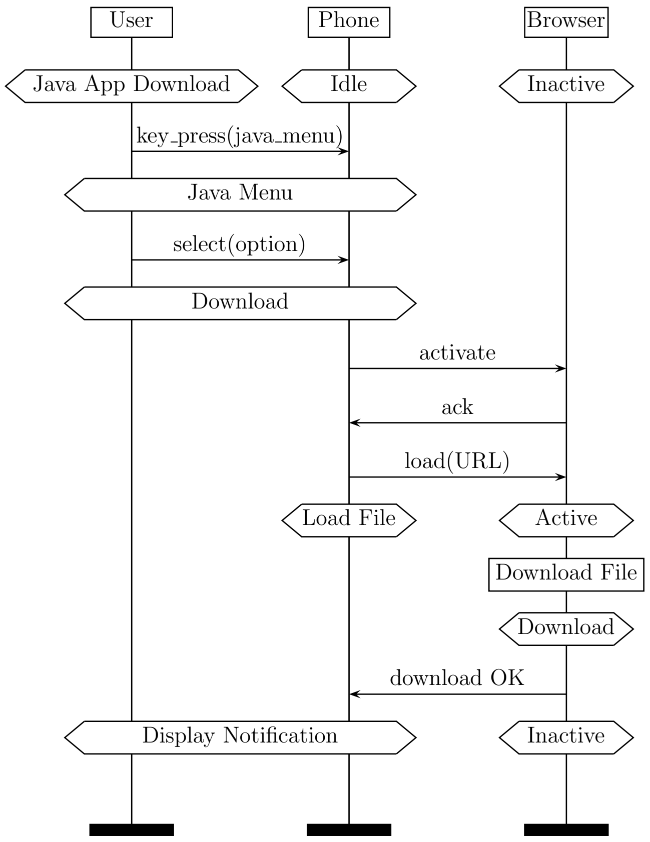

Fig. 1. Java Application Download

3 Phase Composition Semantics

Each process behavior described by an MSC scenario can be defined as a process algebra term that characterises this behavior up to strong bisimulation equivalence [5]. From now on we will identify an MSC with the set of process algebra terms it defines. For a process P we can extract from each MSC scenario M_i a process algebra term Q_i that defines the behavior of P in M_i. In section 4 we will define a process algebra that permits us to join together these different Q_i into a process that describes the implicit phase transitions of the requirements scenarios. In this section we will motivate the formal semantics with an informal definition.

For a set of requirements specification scenarios let $\mathcal{P}$ be the set of possible phases that can occur and E be the set of events that can occur. We regard $\mathcal{P}$ as a set of boolean propositions.

Definition 1. *A* phase trace *is a sequence of triples* (S_i, e_i, S_{i+1}), *for* $0 \leq i \leq n-1$, *where each* $e_i \in E$ *and* S_i, $S_{i+1} \subseteq \mathcal{P}$. S_i *denotes the set of phases that are active before event* e, *and* S_{i+1} *denotes the phases that are active immediately after* e.

In practise S_i and S_{i+1} are usually the same as they represent the major operational phases of the protocols defined by the requirements, which do not change after every single event. A triple (S_i, e_i, S_{i+1}) is referred to as an annotated event. When a phase trace ends in an annotated event (S, e, S') where $S \neq S'$ we say t is a phase transition trace and (S, e, S') is a phase transition.

Figure 1 describes a requirements scenario where the mobile handset has a dedicated key that causes a menu of java applications that are available for download to be presented to the user. Once the user selects one of these applications from the menu the 'Phone' process delegates the task of downloading the application to the (WAP) 'Browser' process. Within this scenario it is not specified how this downloading occurs, it is abstracted away by the action box 'Download File'.

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})

3.1 Informal Phase Composition Semantics

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

Consider two phase transition traces t_1 and t_2, where t_2 is a suffix of t_1 and terminates with a phase transition (S_0, e, S_1). I.e $t_1 = t_3 \cdot t_2$

for some t_3. In this case we say t_1 and t_2 match and that S_1 is reached consistently. Hence we can suppose each occurrence of S_1 defines the same set of concurrent global states. This leads us to the idea of phase transition simulation between processes based on the idea of one process simulating another once a common phase is shown to be consistently reached.

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 behavior of Q from then on. Hence, if there are traces t_1 and t_2 as above such that $P \xrightarrow{t_1} P_1$, and $Q \xrightarrow{t_2} Q_2$ then P_1 must be able to simulate Q_2 (in the conventional sense). 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 4. That canonical process captures the legitimate compositions of the scenarios within an imprecise setting.

Note the above semantics is true if we can assume a phase symbol is a global state name for some statechart, and is in fact a weakening of such state semantics. The phase semantics above allows a phase to act as a global state once there is a match between the behaviour of two different scenarios. By using such an overlap between scenarios to define when phase symbols can act in a state like way, we ensure that they can only be used to compose scenarios where they are consistently applied.

Consider the examples of figures 1 and 2. Figure 2 describes how the 'Browser' process downloads a file iteratively once it receives the ?load(URL) message in the 'Inactive' phase. Recall that figure 1 abstracted out the details of how the Java application is downloaded. We can suppose these two scenarios are defined by different feature teams, quite likely at different times. Perhaps figure 2 is a legacy requirement specification. Given the informal semantics we can see how the two 'Browser' processes can be joined together within a single process that represents some of the phase transitions implied by the two scenarios.

Suppose we observe the initial trace of annotated events for t_0 from figure 1 consisting of

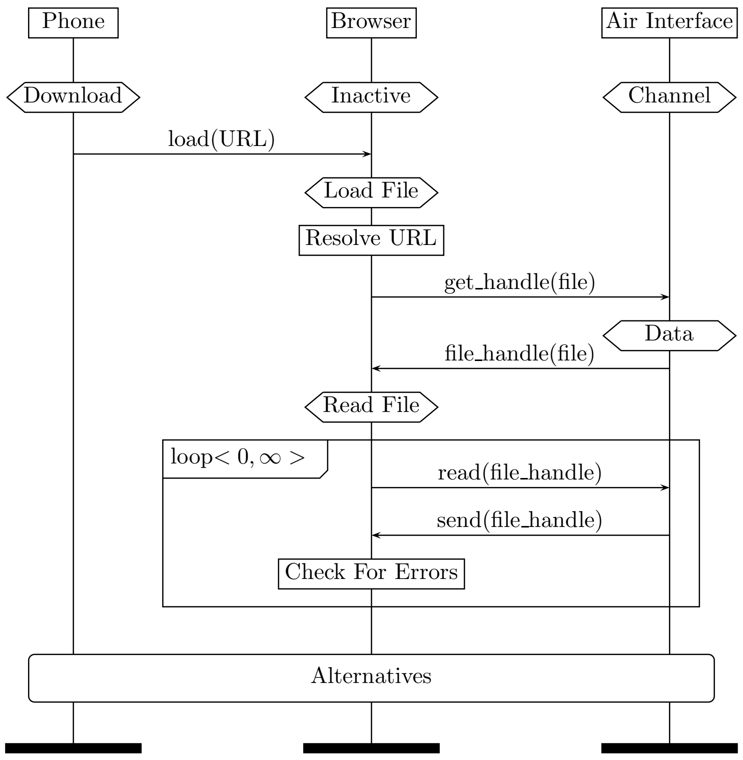

Fig. 2. Download File with Browser Process

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

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

This causes a phase transition from 'Inactive' to 'Load File'. Let us assume within the context of receiving load(URL) that whenever 'Active' is an active phase then so is 'Load File'. Hence the end of t_1 matches t_2. That means after the first two annotated events have occurred t_1 matches t_2 in that they contain the same events and are consistently annotated.

Therefore whenever process 'Browser' initially follows the scenario given by figure 1 up to the first phase transition, it must be able to simulate the subsequent scenario given by figure 2. That means we can combine the two scenarios into that described by fig-

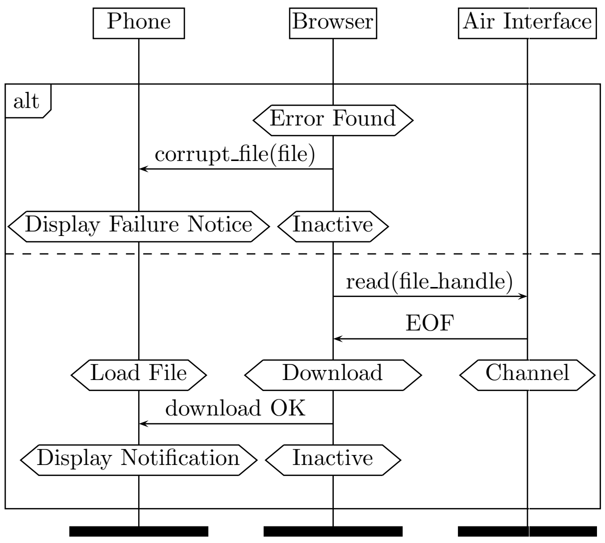

Fig. 3. Alternatives Reference for Figures 2 and 4

ure 4. Note that although figure 1 gave no account of what might happen if the file being downloaded was corrupted the new scenario describes this case. This would make a valuable test purpose.

3.2 Formal Semantics of Phase Compositions

We will use a Hennessy Milner style of temporal logic [6] to permit phases to act as a type of temporal guard. A temporal model M consists of a directed graph G, with vertex labelling $\nu : G_V \longrightarrow 2^{\mathcal{P}}$, edge labelling $\varepsilon : G_E \longrightarrow E$, and some vertex i that represents the initial moment. We can think of M as representing a model of the system global states and execution traces.

Temporal formulae are defined as usual:

- $M, v \vDash \langle e \rangle \phi$ iff there is an edge $(v, w) \in G_E$ such that $\varepsilon(v, w) = e$, and $M, w \vDash \phi$. I.e. there is some execution trace from v starting with e where ϕ holds.
- $M, v \vDash [e]\phi$ iff for every edge $(v, w) \in G_E$ where $\varepsilon(v, w) = e$, $M, w \vDash \phi$. I.e. for every execution trace from v starting with e, ϕ holds.
- $M, v \vDash \Box\phi$ iff $M, v \vDash \phi$ and $M, w \vDash \Box\phi$ for every edge $(v, w) \in G_E$. I.e. for all execution traces from v, ϕ holds.

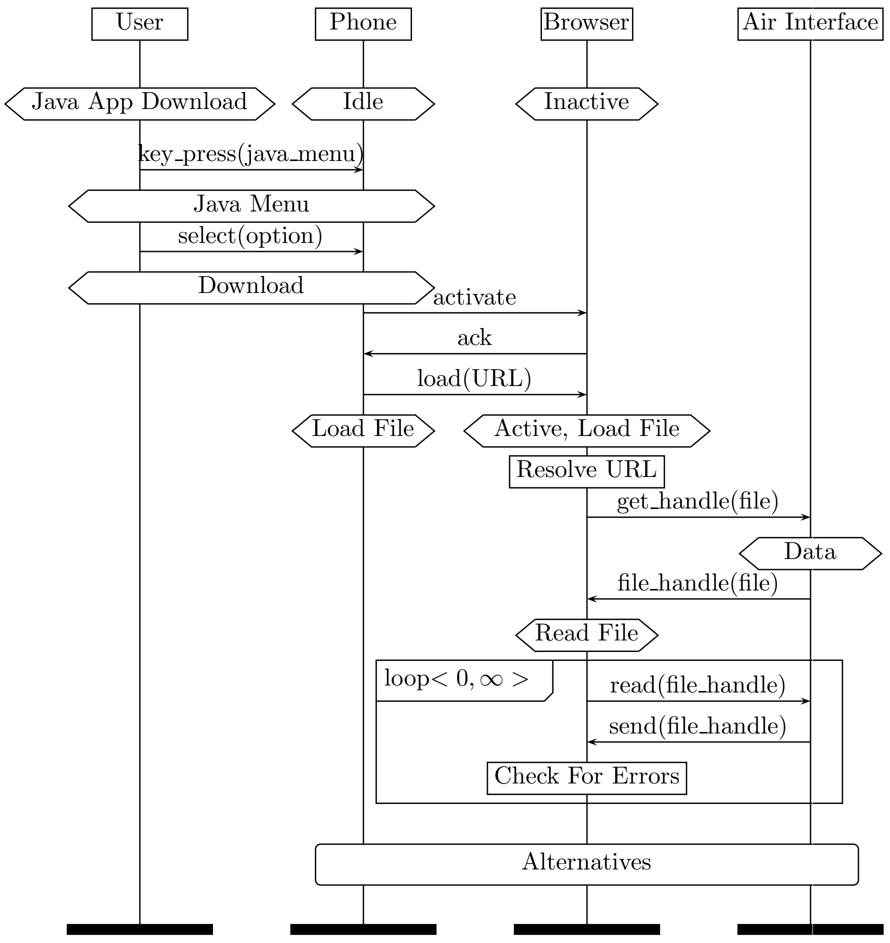

Fig. 4. Synthesized Scenario of Error Checking with Java App Download

- $M, v \vDash \Diamond\phi$ iff there is some vertex w reachable from v such that $M, w \vDash \phi$. I.e. for some execution trace from v, ϕ holds.
- $M, v \vDash \psi\,\mathcal{U}\,\phi$ iff there is some vertex w reachable from v such that $M, w \vDash \phi$, and for every vertex u on that path to w $M, u \vDash \psi$. I.e there is an execution trace from v where ψ holds until we reach w when ϕ becomes true.

The satisfiability of ordinary boolean formulae is defined as usual. Formula ϕ is satisfied in M when $M, i \vDash \phi$. ϕ is valid when it is satisfied in every model, when we write $\vdash \phi$. For formulae ψ and ϕ we write $\psi \vdash \phi$ to denote that $\vdash \psi \Rightarrow \phi$.

Definition 2. *For a set $S \subseteq \mathcal{P}$, define $\bigwedge S = \bigwedge_{x \in S} x$. For a phase trace $t = (S, e, S') \cdot t'$, define its temporal semantics as*

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

This formula represents that somewhere within the model M there should be at least one execution trace with states and events that match those of t. A context $\mathcal{X}$ is any temporal formulae over $\mathcal{P}$ and E. It controls how phases are related across the requirements scenarios. For the example above, where we assumed that whenever a load(URL) message is received then Active implies 'Load File' until Inactive, the context would be

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

The temporal context also permits phases defined by different development teams to be given a consistent meaning across all the scenarios.

Definition 3. *For context $\mathcal{X}$ we define phase trace t to match phase trace t' when*

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

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 $\mathcal{X}$. Note t_1 matches t_2 in the informal semantics example since

$$\Box([\mathsf{load(URL)}](\mathsf{Active} \Rightarrow (\text{'}\mathsf{Load\ File}\text{'}\ \mathcal{U}\ \mathsf{Inactive}))) \vdash (\|t_1\| \Rightarrow \Diamond \|t_2\|)$$

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

Definition 4. *Define P to simulate process Q within context $\mathcal{X}$, written as $P \sqsupset_{\mathcal{X}} Q$, if $\forall a$ such that $Q \xrightarrow{a} Q'$ there is some a' where $P \xrightarrow{a'} P'$ such that $a' \Leftarrow_{\mathcal{X}} a$ and*

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

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

For a 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 $0 \leq i \leq n$, such that $P_i \xrightarrow{a_i} P_{i+1}$, $P_0 = P$ and $P_n = P'$.

Definition 5. *Define P to simulate the phase transitions of process Q within context $\mathcal{X}$, written as $P \trianglerighteq_{\mathcal{X}} Q$, 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 $P' \sqsupset_{\mathcal{X}} Q'$.*

In other words, if after being active for some arbitrary time, P subsequently generates a trace of annotated events that match a phase transition trace of Q, then P must be able to simulate Q from that time onwards. This implies that a phase transition trace of Q acts as a kind of temporal guard. If ever the guard is triggered, in the sense that P can match the phase transition trace, then the rest of the behavior of Q is then simulated. Note this is a strict weakening of the global state semantics as in [9], [8] where the phase symbols of the MSC scenarios are identified with global state names in UML statecharts.

Let $\{M_i \mid 0 \leq i \leq n\}$ 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.

Definition 6. *We define process P to represent the phase transitions of processes Q_i when $P \trianglerighteq_{\mathcal{X}} Q_i$ for each i. The overlaps of P are those phase transition traces of P that are not contained in any of the Q_i.*

Figure 4 describes one of the overlaps given by the phase transition representation of the 'Browser' processes in figure 1 and figure 2.

4 Phase Composition Processes

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 $\rho(\mathcal{X}) : \mathcal{A} \longrightarrow \mathbb{B}$ be a boolean valued function that defines when an annotated event is a

$$\begin{aligned}
P \parallel Q \quad &= (P\langle\Leftarrow\rangle Q) + (P\langle\Rightarrow\rangle Q)\\
a\cdot P\langle\Leftarrow\rangle b\cdot Q &= a\cdot P\langle\Leftarrow\!\!|\rangle b\cdot Q \ \text{ when } \ a \Leftarrow_{\mathcal{X}} b\\
a\cdot P\langle\Leftarrow\rangle b\cdot Q &= a\cdot (P\langle\Leftarrow\rangle b\cdot Q) \ \text{ when } \ a \not\Leftarrow_{\mathcal{X}} b\\
a\cdot P\langle\Leftarrow\!\!|\rangle b\cdot Q &= (a+b)\cdot (P\langle\Leftarrow\!\!|\rangle Q) \ \text{ when } \ a \Leftarrow_{\mathcal{X}} b \ \text{ and } \ \neg\rho(\mathcal{X})(a)\\
a\cdot P\langle\Leftarrow\!\!|\rangle b\cdot Q &= (a+b)\cdot (P \mid Q) \ \text{ when } \ a \Leftarrow_{\mathcal{X}} b \ \text{ and } \ \rho(\mathcal{X})(a)\\
a\cdot P\langle\Leftarrow\!\!|\rangle b\cdot Q &= a\cdot P + b\cdot Q \ \text{ when } \ a \not\Leftarrow_{\mathcal{X}} b \ \text{ and } \ \rho(\mathcal{X})(a)\\
a\cdot P\langle\Leftarrow\!\!|\rangle b\cdot Q &= a\cdot P \ \text{ when } \ a \not\Leftarrow_{\mathcal{X}} b \ \text{ and } \ \neg\rho(\mathcal{X})(a)\\
a\cdot P \mid b\cdot Q \ &= (a+b)\cdot (P \mid Q) \ \text{ when } \ a \Leftarrow_{\mathcal{X}} b\\
a\cdot P \mid b\cdot Q \ &= a\cdot P + b\cdot Q \ \text{ when } \ a \not\Leftarrow_{\mathcal{X}} b \ \text{ and } \ b \not\Leftarrow_{\mathcal{X}} a
\end{aligned}$$

Fig. 5. Phase Composition Process Algebra

phase transition. That is $\rho(\mathcal{X})(S, e, S') = \mathsf{t}$ when $\mathcal{X} \not\vdash (\bigwedge S \Rightarrow \bigwedge S')$. Note when $\mathcal{X}$ is a tautology, then $\rho(\mathcal{X})(S, e, S')$ denotes that S and S' are disjoint. For annotated events $a = (S, e, S')$ and $b = (U, e, U')$ define $a + b = (S \cup U, e, S' \cup U')$. For a set of processes Q consisting of processes Q_i, for $1 \leq i \leq n$ let πQ denote $Q_0 \parallel Q_1 \parallel \cdots \parallel Q_n$.

In figure 5 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. For this algebra we further define $\mid$ to be commutative and $(P\langle\Rightarrow\rangle Q) = (Q\langle\Leftarrow\rangle P)$ and the process 0 to act as a multiplicative zero element for these two operators, so that $(0\langle\Leftarrow\rangle Q) = 0$, and $0 \mid Q = 0$. Notice that $a + b$ is equivalent to $b + a$, hence in the penultimate axiom

$$a \cdot P \mid b \cdot Q = (a + b) \cdot (P \mid Q)$$

when $a \Leftarrow_{\mathcal{X}} b$ or when $b \Leftarrow_{\mathcal{X}} a$. Finally we assume all the defined compositional operators in the algebra distribute over summation of processes. The algebra essentially defines an algorithm for the construction of a minimal phase transition representation as explained in theorem 1.

The process $P \parallel Q$ will consist of P and Q glued together along traces from each process that match. The process $P\langle\Leftarrow\rangle Q$ defines joins between the processes where a trace from P matches a trace from Q. Process $P\langle\Leftarrow\rangle Q$ acts like P until it reaches an action that Q is able to perform (if there is such a place). It then changes to the process $P\langle\Leftarrow\!\!|\rangle Q$. This process now allows P and Q to unfold in lock step. If this continues until there is a phase transition, then we have a match between a trace in P and Q. The process $P\langle\Leftarrow\!\!|\rangle Q$ will now become $P \mid Q$. If there is no such match then essentially Q is discarded. Process $P \mid Q$ allows P and Q to unfold in lock step until they diverge, at which point it splits into the summation of the two processes.

Hence if we have two traces $P \xrightarrow{t_1} P'$ and $Q \xrightarrow{t_2} Q'$ where t_1 matches t_2, then there is a trace $P \parallel Q \xrightarrow{t_1} P' \mid Q'$. If there are no matches between any traces of P and Q then $P \parallel Q$ degenerates into $P + Q$. Because it is possible for a process to have a non-degenerate match between two of it's own traces, it is *not* the case that $P \parallel P$ is necessarily equivalent to P.

Theorem 1. *Given a set Q of processes Q_i from requirements scenarios M_i for $0 \leq i \leq n$, then*

$$P = \pi Q$$

is a phase transition representation of Q.

Process P is canonical upto simulation equivalence. That is if P' is another phase transition representation of Q, then $P' \sqsupset_{\mathcal{X}} P$. Define P to be the phase composition process for Q.

Figure 6 is the phase composition process of the two 'Browser' processes defined in figures 1 and 2. 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 are joined together. The process is depicted as a finite state automaton, where the edges are labelled with annotated events. The temporal context here causes Active and Load File to be simultaneously valid, hence both phases are included in the relevant annotated events. Where a set of phases in an annotated event only contains a single phase we leave out the surrounding braces for that set and just write the phase on its own.

In general the phase composition process P is built by joining together specification scenario processes wherever there is a match between phase transition traces. Suppose there are two specification processes Q_0 and Q_1, where there is a phase transition trace t_0 within the body of Q_0 that matches some phase transition trace t_1 at the start of Q_1. Then P will contain a copy of Q_0 joined to Q_1 along

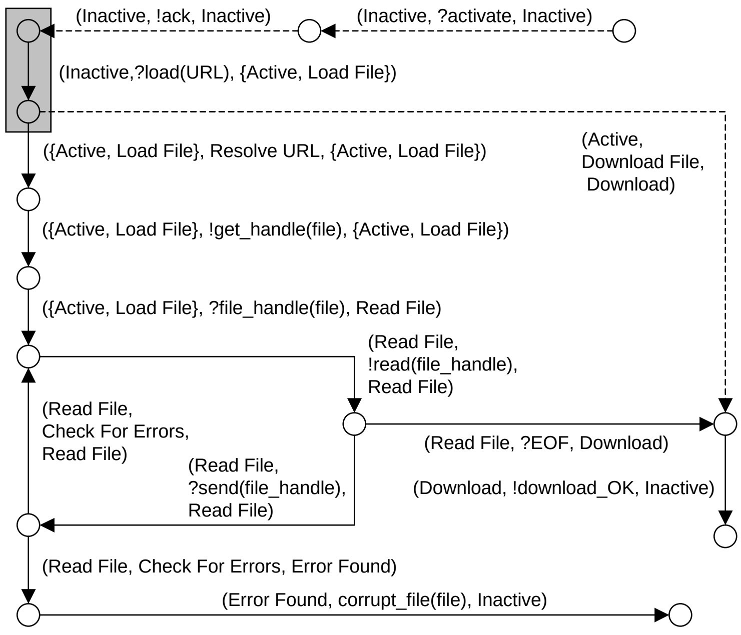
(Inactive, !ack, Inactive)
(Inactive, ?activate, Inactive)
(Inactive,?load(URL), {Active, Load File})
({Active, Load File}, Resolve URL, {Active, Load File})
(Active, Download File, Download)
({Active, Load File}, !get_handle(file), {Active, Load File})
({Active, Load File}, ?file_handle(file), Read File)
(Read File, !read(file_handle), Read File)
(Read File, Check For Errors, Read File)
(Read File, ?EOF, Download)
(Read File, ?send(file_handle), Read File)
(Download, !download_OK, Inactive)
(Read File, Check For Errors, Error Found)
(Error Found, corrupt_file(file), Inactive)

Fig. 6. 'Browser' Phase Composition Process

the end of t_1 that corresponds to t_0. By exhaustively joining all such matches together in a single process we construct P. The process algebra of figure 5 captures this idea formally.

Theorem 2. *The phase composition process of theorem 1 is regular. That is it can always be represented by a finite state automaton.*

If in fact phase symbols truly are global state names, then the phase composition process will always be simulated by the resultant statechart. Hence traces of the phase composition process are also traces of any future refinements of the scenarios that transform phase symbols to global states.

Motorola Pilot

The process algebra in figure 5 can be implemented in an efficient manner to provide an automated mechanism for generating phase transition representations of requirements scenarios. A prototype version of this has been implemented by Motorola UK Research Labs

[7] as an extension of their test generation tool set [4]. Their current prototype does not allow iterative processes, and has not implemented temporal contexts. The prototype has been used on various existing 3G requirements scenarios and is currently being used as part of a pilot study during the development of new products.

Phase Composition Test Purposes

It is possible to automatically generate test purposes from the phase composition process. The phase composition process can be used to generate new MSC scenarios that describe implicit phase transitions within the requirements. Figure 4 is derived in this way from figure 6. Such new MSC scenarios can be used to generate test suites via tools such as *ptk* [4]. *ptk* can derive TTCN2, TTCN3 or SDL test cases directly from MSC requirements using a number of algorithms to choose which MSC traces to generate the tests from.

5 Conclusion

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.

References

1. R. Alur, K. Etessami, M. Yannakakis, Inference of Message Sequence Charts, Proceedings 22nd International Conference on Software Engineering, pp 304-313, 2000.
2. 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)
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. 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.
5. T. Gehrke, M. Hilhn, H. Wehrkeim, An Algebraic Semantics for Message Sequence Chart Documents, in Formal Description Techniques, Chapman Hall 1998.
6. M. Hennessy, R. Milner, Algebraic Laws for Nondeterminism and Concurrency, Journal of the ACM, 32: 137-161, 1985.
7. Bill Mitchell, Robert Thomson, Clive Jervis, Phase Automaton for Requirements Scenarios, Feature Interactions in Telecommunications and Software Systems VII, 77-84, 2003, IOS Press.
8. J. Schumann, J. Whittle, Generating Statechart Designs From Scenarios, Proceedings 22nd international conference on on Software engineering, 2000.
9. S. Uchitel, J. Kramer, J. Magee, Synthesis of Behavioral Models from Scenarios, IEEE Transactions on Software Engineering, vol. 29, no. 2, February 2003
10. Z.120 (11/99)ITU-T Recommendation - Message Sequence Chart (MSC)
11. 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.