Machine-Verifiable Responsiveness

J.N. Reed[1], A.W. Roscoe[2], and J.E. Sinclair[3]

[1] Armstrong Atlantic State University, Savannah Ga, US
Email: Joy.Reed@cs.armstrong.edu
[2] University of Oxford, UK
Email: Bill.Roscoe@comlab.oxford.ac.uk
[3] University of Warwick, Coventry, UK
Email: jane@dcs.warwick.ac.uk [†]

Abstract. In a system of inter-operating components, individual components may use services provided externally and will require assurance both of appropriate functionality and of responsiveness. We have developed properties which capture the notion of non-blocking responsive behaviour, together with machine-based checks implemented in the CSP model-checker, FDR. In this paper we illustrate the use of these properties in their application to a specific example, and provide a detailed analysis of the representation of these and similar properties. To this end we develop a new model of CSP with respect to which they are fully abstract.

Keywords: responsiveness, CSP, model-checking.

1 Introduction

The historic focus of formal verification of component-based systems has been to reason about the behaviour of a system based on the collective behaviour of its components. Typical inference rules allow the derivation of properties of an entire system from individual properties of its components. However in certain sorts of systems developed today, such as distributed services, it is more natural to reason from the point of view of an individual component. In particular, we want to reason about the effect on the behaviour of one component as a result of its interactions with another, perhaps independently developed, component.

Our work addresses the specific issues of specifying and verifying responsiveness, specifically, the requirement that one component will not cause another one to deadlock by not responding to it when expected. This is not equivalent to demanding that the parallel combination of the two components be deadlock-free. Rather, we require that a specific process P is itself not blocked by interacting process Q when P could have otherwise progressed. Ensuring responsive

[†] The work of authors Reed and Roscoe was partially supported by the US Office of Naval Research.

behaviour is of particular importance in systems of a critical nature or where guaranteed service is required.

In [RSR04], we defined a binary relationship between processes which we called *RespondsTo*, which characterises a desirable and mechanically-verifiable property of responsiveness for distributed components: Q is responsive to P if and only if Q does not cause P to deadlock, and furthermore, no refinement of Q causes any refinement of P to deadlock. The notation used is CSP, with model checking provided by the FDR tool. We showed that refinement-closure of the property is crucial. Once it is verified that Q responds to P, no further verifications of this property need be made for any valid refinements of Q or P. Both P's and Q's developers know that any valid modifications of Q will desirably respond to any valid modifications made to P.

The earlier paper [RSR04] concentrated on the theory of these properties in isolation. Here we provide an example illustrating how the theory can be used in practice, and provide a detailed comparison with some related work [FHRR04], shedding significant new light on the latter and yielding a new model for CSP.

Section 2 provides a brief overview of CSP. Section 3 gives failures-based definitions for our concept of responsiveness. In Section 4 we give an example of an on-line shopping network. Section 5 provides the theoretical comparison. Section 6 presents conclusions and relations to other work.

2 An introduction to CSP

CSP [Hoa85,Ros98] models a system as a *process* which interacts with its environment by means of atomic *events*. Communication is synchronous: an event takes place precisely when both process and environment agree on its occurrence. This, rather than assignments to shared variables, is the fundamental means of interaction between agents. An overview of the syntax of CSP is given in Appendix A.

A related series of semantic models capture different aspects of observable behaviours of processes. The simplest semantic model is the *traces* model which characterises a process as a set of all its finite traces, $traces(P)$, representing observable sequences of events it can perform. These events are drawn from a set Σ, containing all possible communications for processes in the universe of consideration. The traces model is sufficient for reasoning about safety properties, but not liveness properties. In this paper, we use the stable failures model in which a process P is modelled as sets of *traces* and *failures*. A *failure* consists of a pair (s, X) with s a finite trace of events drawn from Σ possibly followed by the termination signal $\checkmark$ and X a subset of events of $\Sigma^{\checkmark}$. Here, $\Sigma^{\checkmark}$ denotes the set of all communication events together with $\checkmark$. (In general, if $A \subseteq \Sigma$, then $A^{\checkmark}$ will be used as an abbreviation for $A \cup \{\checkmark\}$.) The pair (s, X) is a failure if

P may engage in the sequence s and then refuse all of the events in X. The set X is called a *refusal.*

A process P is a refinement of process S ($S \sqsubseteq P$) if any possible behaviour of P is also a possible behaviour of S:

$$\mathit{failures}(P) \subseteq \mathit{failures}(S) \wedge \mathit{traces}(P) \subseteq \mathit{traces}(S)$$

Intuitively, suppose S (for "specification") is a process for which all behaviours it permits are in some sense acceptable. If P refines S, then any behaviour of P is as acceptable as any behaviour of S. S can represent an idealised model of a system's behaviour, or an abstract property corresponding to a correctness constraint, such as deadlock freedom. A wide range of correctness conditions can be encoded as refinement checks between processes. Mechanical refinement checking is provided by the model-checker, FDR [For].

All our examples are *divergence-free*, meaning that the $\mathcal{F}$ (stable failures model) representation of a process is essentially equal to that in the more usual *failures-divergences* model [Ros98]. Checks for divergence-freedon can also be madein FDR.

3 Responsiveness

In the following descriptions, P and Q represent processes, with P regarded as the requesting (client) process which requires Q to respond in a non-blocking manner. We use J to denote the shared alphabet of P and Q, that is, the set of events in which they must both participate. We also assume that there is no other member of Σ which both P and Q are capable of communicating. Thus $P \,|[\, J \,]|\, Q$ will, in this paper, always be the same as Hoare's *alphabetised* parallel. Where convenient we will refer to their alphabets as αP and αQ, so that $J = \alpha P \cap \alpha Q$.

For Q to be suitably responsive to P, whenever P requires co-operation from Q in an event $j \in J$, Q must be willing to participate (after possibly completing a sequence of its own events). Q must not cause deadlock, although we allow the possibility of P behaving as it chooses. If P is happy to engage in any one of a set of joint events, Q must be willing to engage in at least one of these.

3.1 Failures-based formulation of responsiveness properties

Our formal definition of responsiveness is given in CSP over $\mathcal{F}$. It requires that, at any point in the joint execution of P and Q, if P demands participation in a set of joint events, Q complies for some non-empty subset of the events. In this definition:

- $\checkmark$ is the special termination event on which all CSP parallel operators effectively synchronise (distributed termination);
- $J^{\checkmark}$ is the joint alphabet with the $\checkmark$ added;
- $\mathit{initials}(P)$ is the set of all initial events in which P may engage;
- P/s is the process which behaves as P would after execution of trace s;
- $s \upharpoonright A$ is the subsequence of s formed by restricting s to elements of set A.

Definition 1. *For processes P and Q with joint alphabet J, Q RespondsTo P iff for all $s \in (\alpha P \cup \alpha Q)^*$, X:*

$$(s \upharpoonright \alpha P, X) \in \mathit{failures}(P) \;\wedge\; (J^{\checkmark} \cap \mathit{initials}(P/s)) - X \;\neq\; \{\} \;\Rightarrow$$
$$(s \upharpoonright \alpha Q, (J^{\checkmark} \cap \mathit{initials}(P/s)) - X) \notin \mathit{failures}(Q)$$

□

Given any failure (t, X) of P, the set $(J^{\checkmark} \cap \mathit{initials}(P/t)) - X$ describes a joint event set in which P may demand participation. Thus Q may not refuse the whole of this set after any trace u such that $u \upharpoonright J^{\checkmark} = t \upharpoonright J^{\checkmark}$.

In [RSR04] we showed that the above property is the weakest refinement-closed strengthening of a simpler property *RespondsToLive* (first investigated in [RS01]) which says that P and Q operating in parallel deadlock on their shared communications only if P could deadlock on them on its own. A binary property, H, is refinement-closed if whenever $H(P, Q)$ holds, then $H(P', Q')$ also holds for all refinements $P \sqsubseteq P'$, $Q \sqsubseteq Q'$.

Definition 2. *Q RespondsToLive P means that for every trace s*

$$(s, J^{\checkmark}) \in \mathit{failures}(P \parallel_J Q) \Rightarrow (s \upharpoonright \alpha P, J^{\checkmark}) \in \mathit{failures}(P)$$

□

An important aspect of our responsiveness properties is that they are mechanically verifiable. We have shown [RSR04] that both properties *RespondsTo* and *RespondsToLive* may be formulated as machine-checkable assertions suitable for verification as refinements. These results have been generalised by Roscoe [Ros03] who describes techniques for translating more-or-less arbitrary predicates on a process into refinement checks. If a property is closed and refinement-closed it can be formulated as a machine-checkable assertion of the form $\mathit{SPEC} \sqsubseteq G(P)$. A distributive predicate can be captured as an assertion of the form $F(P) \sqsubseteq G(P)$, again machine-checkable. Here, F and G are CSP contexts, that is, process expressions which may involve process variable P.

Suppose process P makes a *request* to a server, after which P is happy to deal with either of two different responses.

$$P = \mathit{request} \rightarrow (\mathit{response}1 \rightarrow P \;\Box\; \mathit{response}2 \rightarrow P)$$

The $\Box$ operator represents an external (deterministic) choice. Process Q offers only the service indicated by $\mathit{response}1$.

$$Q = \mathit{request} \rightarrow \mathit{response}1 \rightarrow Q$$

P is prepared to accept different responses and will regard Q as a suitable service since Q can supply one of the possible acceptable patterns. $P \parallel Q$ runs successfully (that is makes progress) without deadlocking on events they have in common: $\Sigma = \{\mathit{request}, \mathit{response}1, \mathit{response}2\}$. Thus we regard Q as responsive to P. In this particular case, P is also responsive to Q.

If P makes an internal choice between the replies:

$$P = \mathit{request} \rightarrow (\mathit{response}1 \rightarrow P \sqcap \mathit{response}2 \rightarrow P)$$

then Q should no longer be regarded as responsive to P, since if P chooses only $\mathit{response}2$, it would be forever blocked. From Q's perspective, it is also the case that P is not responsive to it.

In general, responsiveness is not symmetric, nor does it imply that processes can always progress. Below P is defined using internal choice. It may choose either to engage in x or to SKIP (indicating clean termination). Q is always prepared to offer event x.

$$P = (x \rightarrow P) \sqcap \mathit{SKIP}$$
$$Q = (x \rightarrow Q)$$

Q is regarded as being responsive to P, because Q is always willing to engage in x with P. However, at any time P can choose not to engage in x when expected, thereby blocking Q. Hence P is not responsive to Q.

4 A Simple On-line Shopping Network

This example of an on-line shopping network is similar to that used by Fournet et al. [FHRR04]. This choice is made specifically to allow direct comparison with the work on stuck-freeness. A Customer interacts with a trolley, which in turn acts as intermediary among the customer, warehouse, and billing service. We deal with behaviour of components only in terms of their communications events.

We provide stylised FDR scripts for a network consisting of four processes: *Customer*, *Trolley*, *Warehouse*, and *Invoicer*. *Customer* and *Trolley* communicate on shared channels belonging to *CTevents* defined below. *Trolley* and *Warehouse* communicate on shared channels belonging to *TWevents*, and *Trolley* and *Invoicer* communicate on shared channels belonging to *TIevents*. For data type T and channel c, $c.T$ is the set of all events associated with c.

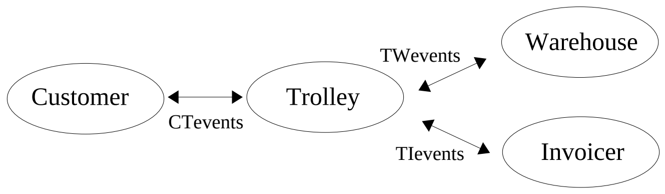

Fig. 1. On-line Shopping Network

$$
\begin{aligned}
&\mathit{CTevents} &&= \{\mathit{open}, \mathit{close}, \mathit{checkOut}, \mathit{cancel}, \mathit{invoice}\} \cup \mathit{addItem.Item} \cup \mathit{removeItem.Item}\\
&\mathit{TWevents} &&= \{\mathit{commitReserve}, \mathit{ack}, \mathit{cancelOrder}\} \cup \mathit{reserveItem.Item} \cup \mathit{cancelItem.Item}\\
&\mathit{TIevents} &&= \mathit{processOrder.Wkorder} \cup \mathit{processInvoice.Invoice}
\end{aligned}
$$

A customer non-deterministically chooses to add or remove items from the set *Item*, and at any time may terminate the session with an option of cancelling the purchase:

$$
\begin{aligned}
&\mathit{Customer} = \mathit{open} \to\\
&\quad (\sqcap_{item:Item}\ \mathit{addItem}!\mathit{item} \to \mathit{Customer})\\
&\quad \sqcap\ (\sqcap_{item:Item}\ \mathit{removeItem}!\mathit{item} \to \mathit{Customer})\\
&\quad \sqcap\ \mathit{checkOut} \to \mathit{invoice}?x \to \mathit{close} \to \mathit{SKIP}\\
&\quad \sqcap\ \mathit{cancel} \to \mathit{close} \to \mathit{SKIP}
\end{aligned}
$$

The trolley services the customer, by reserving or unreserving requested items from the warehouse. Upon checkout by the customer, the trolley commits to the warehouse and requests a workorder from the invoicer, which it passes on as an invoice to the customer. We abstract work orders and invoices with non-deterministic choice over data types *Wkorder* and *Invoice*.

$$
\begin{aligned}
&\mathit{Trolley} = \mathit{open} \to\\
&\quad \mathit{addItem}?\mathit{item} \to \mathit{reserveItem}!\mathit{item} \to \mathit{Trolley}\\
&\quad \Box\ \mathit{removeItem}?\mathit{item} \to \mathit{cancelItem}!\mathit{item} \to \mathit{Trolley}\\
&\quad \Box\ \mathit{checkOut} \to \mathit{commitReserve} \to \mathit{ack} \to\\
&\qquad (\sqcap_{y:Wkorder}\ \mathit{processOrder}!y \to \mathit{processInvoice}?x \to\\
&\qquad (\sqcap_{x:Invoice}\ \mathit{invoice}!x \to \mathit{close} \to \mathit{SKIP}))\\
&\quad \Box\ \mathit{cancel} \to \mathit{cancelOrder} \to \mathit{ack} \to \mathit{close} \to \mathit{SKIP})
\end{aligned}
$$

The warehouse reserves or releases items (as directed by the trolley). It terminates after receiving either a commit or cancel, which it acknowledges. A faulty warehouse behaves similarly, but does not acknowledge a commitReserve:

$$
\begin{aligned}
&\mathit{Warehouse} = \mathit{reserveItem}?\mathit{item} \to \mathit{Warehouse}\\
&\quad \Box\ \mathit{cancelItem}?\mathit{item} \to \mathit{Warehouse}\\
&\quad \Box\ \mathit{commitReserve} \to \mathit{ack} \to \mathit{SKIP}\\
&\quad \Box\ \mathit{cancelReserve} \to \mathit{ack} \to \mathit{SKIP}
\end{aligned}
$$

$$\begin{aligned}
&\mathit{FaultyWarehouse} = \mathit{reserveItem}?\mathit{item} \to \mathit{FaultyWarehouse}\\
&\quad \square\ \mathit{cancelItem}?\mathit{item} \to \mathit{FaultyWarehouse}\\
&\quad \square\ \mathit{commitReserve} \to \mathit{SKIP}\\
&\quad \square\ \mathit{cancelReserve} \to \mathit{ack} \to \mathit{SKIP}
\end{aligned}$$

The *Invoicer* takes in a request to process an order, and responds back with a work invoice:

$$\mathit{Invoicer} = \mathit{processOrder}?x \to \sqcap_{y:\mathit{WkInvoice}}\ \mathit{processInvoice}!y \to \mathit{Invoicer}$$

The shopping network is made up of the trolley, warehouse, and invoicer, with pairwise communication on their respective shared channels:

$$\mathit{ShopNet} = (\mathit{Trolley} \parallel_{\mathit{TWevents}} \mathit{WareHouse}) \parallel_{\mathit{TIevents}} \mathit{Invoicer}$$

Appendix B contains the definition for $G(P, Q)$ defined for processes P with alphabet H and process Q which synchronises with P on J. The appendix also contains a specification *SPEC* defined in terms of sets H and J. Q *RespondsTo* P is true exactly when the FDR validates the assertion: $\mathit{SPEC} \sqsubseteq G(P, Q)$

This check confirms that if P is taken as *Customer*, Q as *ShopNet*, H as *Customer*'s events (*CTevents*), and J as *CTevents* shared between *Customer* and *ShopNet*, then *ShopNet RespondsTo Customer*. Taking P as *Trolley* with H as the union of *CTevents*, *TWevents*, and *TIevents*, and Q as the *Warehouse* with J as *TWevents*, a check confirms that *Warehouse RespondsTo Trolley*. Analogously *Trolley RespondsTo Warehouse*, and *Invoicer RespondsTo Trolley*. If *FaultyWareHouse* replaces *Warehouse* then *ShopNet* does not respond to *Customer*, with FDR reporting failure. The source of the fault is revealed upon checking that *FaultyWareHouse RespondsTo Trolley*, which also fails.

Invoicer is modelled as a server, which is always ready to accept requests. *Trolley* does not respond to *Invoicer*, since *Trolley* terminates after the session with *Customer*, and indeed, may never even make a request (if *Customer* does not check out). Here the server is always prepared to respond to clients, but the client is allowed to terminate at will giving no notice to the server. We could have chosen to design and validate that the trolley and the invoicer each be responsive to the other. The trolley could inform the invoicer before terminating, so that the invoicer could then stop blocking on their shared channel.

Refinement-closure of *RespondsTo* allows developers of *Trolley*, *Warehouse*, and *Invoicer* to refine their implementation without worry that a modified component (satisfying standard refinement rules) would cause the overall system to be non-responsive to customers. Significantly, this offers component-side development which preserves responsiveness.

The shopping network might use distributed services, for example, the trolley might search dynamically for the best provider of individual items. If specifications of behaviour for warehouse services are published, the trolley could validate them on-the-fly in order to determine if their behaviours were responsive. Indeed, the warehouse and trolley could exchange behavioural specifications and negotiate before committing to interaction. Importantly it is *not* necessary to check the whole network to verify responsiveness, only the relevant pairwise interactions.

5 Comparison of models

Since doing our original work on the topic of responsiveness, some similar work by Fournet *et al* has appeared [FHRR04]. That work was based on a slightly different motivation, namely ensuring that a network of processes does not reach a state from which no further progress can be made while one of them is still requesting something from another. Comparing our work and theirs provides a fascinating insight into the relative qualities of CCS and CSP for specification, as well as illustrating their similarity.

The intention in [FHRR04] is that a combination does not *terminate* leaving one partner hanging. They call the absence of such behaviour *stuck-freeness*. It is noteworthy that this is not really an issue in CSP thanks to the termination signal $\checkmark$: the distributed termination condition of CSP means that the network can only seem to have terminated when they both actually have. This simply results from Hoare's decision to separate semantically between deadlock and successful termination: a stuck combination will appear as deadlock, whereas a pair that has terminated normally will have signalled $\checkmark$.

It follows that the absence of the type of behaviour identified as bad in [FHRR04] follows from a standard check for deadlock freedom (naturally, permitting processes to do nothing further after $\checkmark$).[1] This is at the expense of signalling termination via $\checkmark$, but that seems to us to be a distinction worth making.

The reason why simple termination-based reasoning will not work for *RespondsTo* is that we forbid some behaviours that are not final. We forbid one process from refusing another even when one, other, or even both processes have other things they can do. So we mind even if the refusing process has the potential, via other actions external to the binary parallel we are considering, to do more things and then reach a position where it can now satisfy its partner's request.

Nevertheless the way [FHRR04] chooses to address their issue is remarkably similar to the way we have addressed ours. They specify that the network N never reaches a state in which no further action can happen (i.e., it is deadlocked) but some $P \in N$ is still offering communications to another $Q \in N$.

[1] There would be one difference: the deadlock check would regard a state in which every single component process has individually deadlocked without terminating as incorrect even though there is no stuck-ness. This is impossible.

Over a pair of processes $P \mid Q$ this is conceptually equivalent to saying that any failure of our *RespondsTo* condition (in either direction) only occurs when either P or Q has some alternative action to the interactions in this parallel.

Just as we, in [RSR04], observed that *RespondsToLive* is not refinement closed, they observe that their condition cannot be specified in a refinement-closed manner over the failures model. While our reaction was to strengthen the condition to the weakest refinement-closed one which implied the original, namely *RespondsTo*, theirs was to devise a special equivalence over processes to support it. They call this *conformance* and in it two processes are equivalent if they have identical behaviours of the form (s, X, Y), in which (s, X) is a failure where, in the same stable state which witnesses the failure, every event from Y is available. They restrict Y to be of size 0 or 1. Necessarily, of course, $X \cap Y = \emptyset$.

We make two observations about the conformance equivalence.

- Firstly, if the restriction to $\mid Y \mid \leq 1$ were removed, one gets a different congruence which is equivalent to the *Ready-Sets* model of Olderog and Hoare [OH86]. In that, processes are associated with sets of pairs (s, A) in which s is a trace and A is the set of events which are on offer in some stable state reachable on s. In the absence of the $\mid Y \mid \leq 1$ assumption, every triple (s, X, Y) extends to a maximal one in which $X \cup Y = \Sigma$, and it is clear that the two models will then be the same identifying Y with the ready set.
- Secondly, conformance can be developed into a model which is fully abstract with respect to properties like stuck-freeness and precise operational characterisations of *RespondsTo*.

The stable revivals model In order to turn the idea of conformance into a CSP model we separate the two cases of $Y = \emptyset$ (only necessary for deadlock traces) and $Y = \{a\}$. The latter can be represented as a triple (s, X, a) for $a \notin X$. Since the a represents *revival* from the stable failure represented by (s, X), that is what we shall call the triple.

On the basis (already adopted in [Ros98] relating to the stable failures model) that it is always a good idea to know a process's traces[2] for reasons of safety specification, our new model $\mathcal{R}$ equates a process with three components, respectively

- The finite traces T (a prefix-closed nonempty subset of Σ^*).
- The deadlock traces D (a subset of T).

[2] It is possible to get a compositional version of either this model or the stable failures model (see [Ros98]) without the trace component provided one omits the CSP interrupt operator $\triangle$. For this reason, the full abstraction result quoted below is only true for the language including this operator.

- The revivals R, namely triples of the form (s, X, a) where $s \in \Sigma^*$, $X \subseteq \Sigma$ and $a \in \Sigma - X$, such that R1: $s \frown \langle a \rangle \in T$, R2: $(s, X, a) \in R$ and $Y \subseteq X$ implies $(s, Y, a) \in R$, and R3: $(s, X, a) \in R$ and $b \in \Sigma$ implies that either $(s, X, b) \in R$ or $(s, X \cup \{b\}, a) \in R$.

This yields a model which is a congruence for CSP and which yields the natural fixed point under subset-least fixed points, like $\mathcal{F}$. It is straightforward to recover the $\mathcal{F}$ representation of any process from the $\mathcal{R}$ one: the new one is strictly less abstract.

The most interesting point in it being a congruence arises in hiding: the triple $(s, X, a) \in \mathit{revivals}(P)$ only gives rise to a revival of $P \setminus Y$ if $Y \subseteq X$ (because, analogously with the usual CSP hiding operator, $P \setminus Y$ is not stable unless P refuses Y). It follows that $a \notin Y$ and therefore is not hidden – something which would have caused a problem as we would have lost our next step.[3]

That this equivalence is weaker than ready sets is demonstrated by the following example. Let $\Sigma = \{a, b\}$ and let

$$P = (a \rightarrow \mathit{Stop}) \sqcap (b \rightarrow \mathit{Stop})$$

$$Q = P \sqcap (a \rightarrow \mathit{Stop} \,\Box\, b \rightarrow \mathit{Stop})$$

These two processes are equivalent under conformance/stable revivals semantics, since both can refuse any subset of $\{a\}$ and offer b, or vice-versa. They are not equivalent under ready sets since Q can refuse $\emptyset$ and offer both a and b at the same time.

Just as the concept of ✓ in CSP gives a convenient solution not available in CCS, the nature of the parallel and restriction operators in CCS makes stuck-freeness rather more natural to specify there. As stated in [FHRR04], it is that no unsynchronised label of a sort local to the network can be available when nothing else is in a stable state: it is thus definable as a property of the process representing the network (unrestricted) rather than of the individual network components.

The following definition captures this CCS style in language which is also appropriate to CSP.

Definition 3. *The process N is $\mathcal{R}$-stuck-free with respect to the set of actions A provided it has no revival of the form $(s, \Sigma - A, a)$ with $s \in (\Sigma - A)^*$ and $a \in A$.*

Most interestingly, the mechanisation of the *RespondsToLive* specification we presented in [RSR04] used a modified parallel composition of the pair, with

[3] This problem means that one cannot, for example, modify this congruence so that it records traces of length two or less after a refusal: the result would not be compositional under hiding.

much in common with the ordinary CCS one: parallel processes are enabled to perform an unsynchronised event as an alternative to parallel ones.

Exactly the same thing could be done in CSP to test stuck-freeness for networks: simply rename all synchronised events to both themselves and a special event *stuck* as an alternative, which is not synchronised. The network is then stuck-free if it has no revival $(s, \Sigma - \{stuck\}, stuck)$ for any trace s not containing *stuck*.

In order to formulate *RespondsTo* for $\mathcal{R}$ we need to extend the latter to include the termination signal $\checkmark$. The traces component T is extended to include members of the form $s \frown \langle\checkmark\rangle$, where $s \in \Sigma^*$ (recall that $\checkmark \notin \Sigma$). The deadlock component D is unchanged: still members of Σ^* (for a terminated process is not deadlocked). A revival is of the form (s, X, a), where $s \in \Sigma^*$, $X \subseteq \Sigma$ and $a \in \Sigma^{\checkmark}$. In other words, $\checkmark$ is not recorded in the refusal set, but can be the successor event a. This comes from the philosophy, described in detail in [Ros98] that termination is a "signal" event: not one the environment can refuse or which can meaningfully be offered as an alternative to another visible event. If $s \frown \langle\checkmark\rangle \in T$ then we specify $(s, \Sigma, \checkmark) \in R$: this states that a process which can terminate does not have to offer any other alternative (even τ implicitly).

The structure expressed here allows us to decide whether a process which can terminate after trace s can refuse to do so. For then $s \in D$ or $(s, X, a) \in R$ for some $a \neq \checkmark$: implicitly every revival with $a \neq \checkmark$ implies the refusal of $\checkmark$.

Note that if $P = (T, D, R)$ is a process represented in $\mathcal{R}$ we can easily calculate (bearing in mind the conventions set out in [Ros98] for $\mathcal{F}$):

$$\begin{aligned} failures(P) = {} & \{(s, X) \mid X \subseteq \Sigma^{\checkmark} \wedge s \in D\} \\ & \cup\{(s \frown \langle\checkmark\rangle, X) \mid X \subseteq \Sigma^{\checkmark} \wedge s \frown \langle\checkmark\rangle \in T\} \\ & \cup\{(s, X) \mid (s, X, a) \in R\} \\ & \cup\{(s, X \cup \{\checkmark\}) \mid (s, X, a) \in R \wedge a \neq \checkmark\} \end{aligned}$$

The representation in the Stable Failures Model $\mathcal{F}$ of P is then $(T, failures(P))$.

Now we have extended our model, it is capable of giving a completely precise definition of *RespondsTo*.

Definition 4. *We say that Q $\mathcal{R}$-RespondsTo P if for every trace s, there do not exist $(s \upharpoonright \alpha P, X, a) \in R_P$ and $(s \upharpoonright \alpha Q, Y) \in failures(Q)$ such that $a \in J^{\checkmark}$ and $J^{\checkmark} \subseteq X^+ \cup Y$. Here, $X^+ = X$ if $a = \checkmark$, and $X \cup \{\checkmark\}$ otherwise.*

This precisely captures the concept of P having a communication it wants to make with Q, but them being unable to agree on any.

This *implies RespondsToLive* over $\mathcal{F}$ since if $(s, J^{\checkmark}) \in failures(P \mathbin{[\![} J \mathbin{]\!]} Q)$ is created by the maximal failures $(s \upharpoonright \alpha P, X)$ of P and $(s \upharpoonright \alpha Q, Y)$ of Q, then if P, Q satisfy the condition above, $(s \upharpoonright \alpha P, X)$ either comes from a deadlock trace $s \upharpoonright \alpha P$ or a revival $(s \upharpoonright \alpha P, X, b)$ with $b \notin A$. In the second case, by the healthiness condition R2 above, and Q $\mathcal{R}$-*RespondsTo* P, we get that $J \subseteq X$. In either case $(s \upharpoonright \alpha P, J^{\checkmark}) \in failures(P)$.

Our new definition is very close to the original definition of *RespondsTo* over $\mathcal{F}$. The old definition says that if P can refuse X and do other things in $J^{\checkmark}$ besides X, then Q cannot refuse them. Our new definition, in fact, says precisely the same except that it is now able to couple the "do other things" more closely to X: they are necessarily from the same state. With this in mind it is straightforward to see that the definition over $\mathcal{F}$ implies the one over $\mathcal{R}$.

Both these implications are what we might have hoped for. Furthermore, if P is deterministic in the usual CSP sense (with each process fully characterised by its traces), all three conditions are equivalent. Note that in practical networks, parallel components are nearly always deterministic.

RespondsTo is both refinement-closed and distributive over $\mathcal{R}$.

In this section we have shown that the concept of responsiveness can be captured more precisely in a model we have created specially for this purpose. Indeed, this model is *fully abstract* with respect to both the natural operational characterisation of this or alternatively that of stuck-freeness.

The question then arises of which model we should generally choose to reason about *RespondsTo*. The obvious disadvantage of creating an *ad hoc* model to capture a condition is that it requires new theoretical work, new tool support, and places a substantial burden in ensuring that the rest of one's development is consistent with it. It also requires significant extra understanding on the part of anyone using it. Since we believe that in the vast majority of practical cases it will be possible to use the $\mathcal{F}$ version of *RespondsTo*, we think that pragmatically it is best to use that as the first line of attack, holding more sophisticated models for the rare cases where it is inadequate. As and when there is proper tool support for the refusal testing model of CSP [Muk93], based on Phillips's work [Phi87], it will make sense to reformulate our conditions in that ([FHRR04] observe that refusal testing can capture stuck-freeness[4]). For $\mathcal{R}$ is a weaker equivalence than refusal testing, so the latter can express our properties precisely.

This section has described how the development of a new fully abstract model for CSP has arisen from our work on responsiveness and its comparison to stuck-freeness. Full details of the CSP semantics for this model are beyond the scope of this paper but can be found in [Ros05] together with justification of the claim of full abstraction.

[4] Note that $\mathcal{R}$ is the strongest congruence which is weaker than both ready sets and refusal testing.

6 Further remarks

In terms of the responsiveness property, the work most closely related is that of Fournet et al. [FHRR04] and a detailed comparison of the two approaches has been provided in the previous section. Other related work includes that of Treharne and Schneider [TS99,TS00] in which sufficient conditions are developed to ensure that a CSP controller successfully drives a state-based B specification. Although in a different setting, this requires the B to respond when called upon. Bolton and Lowe [BL03] investigate a class of non-standard refinement notions, one of which coincides with our formulation of *RespondsTo*. In an assume-guarantee setting, Amla et al. [AENT01] develop a rule both sound and complete (for safety and liveness properties) for reasoning about component decomposition. The idea of nondeterministic blocking is not at issue here. In contrast, we treat blocking as fundamental and undesirable.

We have developed a general property characterising responsiveness of interacting components formulated in CSP which can be verified using the techniques of FDR.. As shown in [RSR04], adding components which are responsive in our sense never introduces deadlock. These results have application both for component-side system development and for on-the-fly conformance checking and/or selection of distributed services.

Our comparison of the responsiveness work to the CCS-formulated stuck-freeness property has led to the development of the new CSP stable revivals model. It has also helped illuminate the relationship between CCS and CSP. Further details on the stable revivals model are given in [Ros05].

Future work is planned to include a larger case study which would also address aspects such as performance. Additional work is also required to investigate responsiveness in other settings, such as in the presence of divergence and for infinite traces.

Appendix A: Introduction to CSP

We use the syntax and semantics from [Ros98]. The CSP language describes interacting components of systems: *processes* whose external actions are the communication or refusal of instantaneous atomic *events*. All the participants in an event must agree on its performance. The following CSP algebraic operators are used for constructing processes.

STOP is the CSP process which never engages in any event, never terminates (deadlock).

SKIP similarly never performs any action, but instead terminates

$CHAOS(A)$ is the most non-deterministic, divergence-free process with alphabet A.

$a \rightarrow P$ performs event a and then behaves as process P. The same notation is used for outputs $(c!v \rightarrow P)$ and inputs $(c?x \rightarrow P(x)\ )$ of typed values on named channels, with $c.T = \{c.x \mid x \in T\}$.

$P \sqcap Q$ is *nondeterministic* or internal choice. It may behave as P or Q arbitrarily.

$P \,\square\, Q$ is external or *deterministic* choice. It first offers the initial events of both P and Q to its environment. Its subsequent behaviour is like P if the initial action chosen was possible only for P, and similarly for Q. If P and Q have common initial actions, its subsequent behaviour is nondeterministic (like $\sqcap$).

$\bigsqcap_{x:X} P(x)$ **and** $\square_{x:X} P(x)$ represent generalised forms of the choice operators allowing indexing over a finite set of indices where $P(x)$ is defined for each x in X. $c?x \rightarrow P$ is shorthand for $\square_{x:T}\, c.x \rightarrow P$.

$P \parallel_X Q$ is parallel (concurrent) composition. P and Q evolve separately, but events in X occur only when P and Q agree (i.e. *synchronise*) to perform them.

$P \parallel Q$ is parallel composition, with P and Q synchronising on all events, that is, on all of Σ.

$P \mid\mid\mid Q$ represents the interleaved parallel composition. P and Q evolve separately, and do not synchronise on their events.

$P \setminus A$ is the CSP abstraction or hiding operator. This process behaves as P except that events in set A are hidden from the environment and are solely determined by P; the environment can neither observe nor influence them.

$P[[x := y]]$ is the process formed by renaming x to y in P. Whenever P would offer x, this process instead offers y.

Failures/Divergences Model. A process P is modelled as a set of *failures* and *divergences.* The set Σ contains all possible communications events of processes. The set $\Sigma^{\checkmark} = \Sigma \cup \{\checkmark\}$, contains a special event $\checkmark$ that signals that a process has terminated cleanly. A *failure* is a pair (s, X) for s a finite trace of events of $\Sigma^{\checkmark}$, and X a subset of events of $\Sigma^{\checkmark}$. It is understood that whenever $\checkmark$ appears in a trace, it is the last event in the trace. The pair $(s, X) \in \mathit{failures}(P)$ means that P may engage in the sequence s and then refuse all of the events in X. The set X is called a *refusal.* The set $\mathit{divergences}(P)$ is the set of traces on which P can diverge, meaning perform an infinite unbroken sequence of internal events.

Appendix B: Mechanical verification of *RespondsTo*

We paraphrase results from [RSR04]. We work in the CSP failures model and assume that all processes are divergence-free (which can be mechanically checked).

Assume that P is a process with alphabet H and Q is a process which synchronises on set J of events. We define functions $G(P, Q)$ and $SPEC$ such that Q *RespondsTo* P if and only if the FDR-checkable assertion succeeds:

assert $SPEC \sqsubseteq G(P, Q)$

Let H^* and $H^\diamond$ be distinct, disjoint copies of H. Define the lazy abstraction [Ros98] of Q to be the process which behaves like Q except that whenever Q can perform an abstracted event the new process has the choice of either not doing it or making it invisible:

$$LQ = (Q \parallel_{\Sigma - J} CHAOS(\Sigma - J)) \setminus (\Sigma - J)$$

P^* is a copy of P which can engage in $a^* \in H^*$ whenever P can engage in a:

$$P^* = P[[a := a^* \mid a \in H]]$$

$P^\dagger$ is a process which runs P and P^* in parallel, with a regulator process Reg^*. This runs P and $P*$ in a delayed lock-step manner, also ensuring that whenever P^* has demonstrates that there is something in $initialsInJ^{\checkmark}(P)$, say a, then $G(P, Q)$ only comes up with refusal sets X not containing a so that $initialsInJ^{\checkmark}(P) - X$ is nonempty: these being the one of interest for the condition.

$$\begin{aligned} Reg^* = \Box_{a:H}\, a^* \rightarrow & ((\Box_{b:H}\, b \rightarrow (a == b \& Reg^*)) \\ & \Box\ (a \in J) \& a^\diamond \rightarrow STOP \end{aligned}$$

where $a^\diamond \in H^\diamond$ is a further separate version of a.

$$P^\dagger = ((P \parallel\!\mid\ P^*) \parallel_{H \cup H^*} Reg^*)[[a^\diamond := a \mid a \in J]]$$

Q *RespondsTo* P if and only if $G(P, Q) = P^\dagger \parallel_J LQ$ has no deadlock after an odd-length trace whose last member is in J^*, or in other words if and only if it refines

$$\begin{aligned} Spec = (\Box_{a:J}\, a^* \rightarrow (& & (1) \\ & (\sqcap_{b:J}\, b \rightarrow Spec) & \\ & \Box & \\ & (STOP \sqcap (\Box_{b:H-J}\, b \rightarrow Spec))) & \\ \Box & & \\ (\Box_{a:H-J}\, a^* \rightarrow & (STOP \sqcap (\Box_{a:H}\, a \rightarrow Spec)))) & (2) \\ \sqcap\, STOP & & (3) \end{aligned}$$

The above specification provides three cases: (1) after odd length traces, if the last element is in J, then something in J (the a from $P^\dagger$) must be offered, and it does not care whether anything outside of J is offered or refused, (2) after odd length traces, if the last element is not in J, then the specification does not care what events are offered or refused, and (3) after even length traces, deadlock is acceptable since it means that P has reached a state for which its set of initial events is empty.

References

[AENT01] Nina Amla, E. Allen Emerson, Kedar Namjoshi, and Richard Trefler. Assume-guarantee based compositional reasoning for synchronous timing diagrams. *Lecture Notes in Computer Science*, 2031:465+, 2001.

[BL03] C. Bolton and G. Lowe. On the automatic verification of non-standard measures of consistency. In *6th International Workshop in Formal Methods*, Dublin, July 2003.

[FHRR04] C. Fournet, C.A.R. Hoare, S.K. Rajamani, and J. Rehof. Stuck-free conformance. In *Proceedings of the 16th International Conference on Computer Aided Verification (CAV'04)*, LNCS 3114, pages 242–254. Springer, July 2004.

[For] Formal Systems (Europe) Ltd. *Failures Divergence Refinement.* User Manual and Tutorial. http://www.formal.demon.co.uk/fdr2manual/index.html.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Muk93] A. Mukkaram. *A refusal testing model for CSP.* DPhil Thesis, Oxford University Computing Laboratory, 1993.

[OH86] E.R. Olderog and C.A.R. Hoare. Specification-oriented semantics for communicatiing processes. *Acta Informatica*, 23:9–66, 1986.

[Phi87] I. Phillips. Refusal testing. *Theoretical Computer Science*, 50:241–284, 1987.

[Ros98] A.W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1998.

[Ros03] A.W. Roscoe. On the expressive power of CSP refinement. In *Proceedings of 3rd International Workshop on Automated Verification of critical systems (AVoCS03)*, Southampton University, April 2003.

[Ros05] A.W. Roscoe. Revivals, stuckness and responsiveness. In preparation, 2005.

[RS01] J.N. Reed and J.E. Sinclair. Combining independent specifications. In *ETAPS-FASE 2001, European Joint Conferences on Theory and Practice of Software. Fundamental Approaches to Software Engineering*, pages 45–59, Genoa, Italy, 2001. Springer.

[RSR04] J.N. Reed, J.E. Sinclair, and A.W. Roscoe. Responsiveness of interacting components. *Formal Aspects of Computing*, 16(4):394–411, 2004.

[TS99] H. Treharne and Schneider S. Using a process algebra to control B operations. In K. Araki, A. Galloway, and K. Taguchi, editors, *Integated Formal Methods 1999*, pages 437–456, York, UK, June 1999. Springer Verlag.

[TS00] H. Treharne and S. Schneider. How to drive a B machine. In J.P. Bowen, S. Dunne, A. Galloway, and S. King, editors, *Proceedings of ZB2000*, LNCS 1878, pages 188–209, York,UK, September 2000. Springer Verlag.