Programming Research Group

A TAXONOMY OF WEB SERVICES USING CSP

Lee Momtahan, Andrew Martin and A. W. Roscoe

PRG-RR-04-22

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Abstract

Terms such as *conversational* and *stateless* are widely used in the taxonomy of web services. We give formal definitions of these terms using the CSP process algebra. Within this framework we also define the notion of *Service-Oriented Architecture*. These definitions are then used to prove important scalability properties of stateless services. The use of formalism should allow recent debates, concerning how and whether web services provide standardized access to state, to progress more rigorously.

1 Introduction

There is currently a debate within the web services and Grid communities over whether, and how, web services should allow for access to state. One view is "web services...have no notion of state" [Vog03] while others have argued that the critical role that state plays in distributed systems requires that it be addressed within the web services architecture [FKNT02].

The debate is hindered by a lack of formality and clarity in its discourse. This paper contributes by defining some of the key terms used in the debate, within the Communicating Sequential Processes (CSP) [Hoa85] formalism. We hope more principled comparisons between different proposals to standardize access to state can be made in the light of these definitions.

1.1 Overview

In section 2 we quote the existing natural language definitions used in the taxonomy of web services. The following section gives a series of definitions in CSP culminating in a formal version of the same taxonomy. Section 4 discusses some of the implications of this formalized taxonomy. Section 5 concludes with a summary of the main findings. The first appendix presents our definitions in an alternative form that can be used with the CSP model checker, FDR. The second appendix presents proofs of theorems used in the paper.

2 A taxonomy of state and services

In [FFGT03] the following taxonomy of web services is given:

- A *stateless service* implements message exchanges with no access or use of information not contained in the input message. A simple example is a service that compresses and decompresses documents, where the documents are provided in the message exchanges with the service.
- A *conversational service* implements a series of operations such that the result of one operation depends on a prior operation and/or prepares for a subsequent operation. The service uses each message in a logical stream of messages to determine the processing behaviour of the service. The behaviour of a given operation is based on processing preceding messages in the logical sequence. Many interactive Web sites implement this pattern through use of HTTP sessions and cookies.

- A *service that acts upon stateful resources* provides access to, or manipulates a set of logical stateful resources (documents) based on messages it sends and receives.

[FFGT03] continues:

> When we talk in the third model about a *service that acts upon stateful resources* we mean a service whose *implementation* executes against dynamic state, i.e., state for which the service is responsible between message exchanges with its requesters. A service that acts upon stateful resources may be described stateless if it delegates responsibility for the management of the state to another component such as a database or file system.

Substantial modifications of the wording used in the definitions occurred between v1.0 and v1.1 of [FFGT03], perhaps indicating the difficulty of defining these concepts in natural language.

A related definition is that of a service in the context of Service-Oriented Architecture (SOA). [PWWR03] gives the following:

> A service is a well-defined set of actions, it is self-contained, stateless, and does not depend on the state of other services...
> Here, stateless means that each time a consumer interacts with a Web Service, an action is performed. After the results of the service invocation have been returned, the action is finished. There is no assumption that subsequent invocations are associated with prior ones.

[W3C] adds:

> The description of a service in a SOA is essentially a description of the messages that are exchanged. This architecture adds the constraint of stateless connections, that is where all the data for a given request must be in the request.

3 Web services in CSP

In this section a series of definitions is given which builds our model of web services and their taxonomy.

Definition 3.1.

$$Stateless'(P) \Leftrightarrow \forall\, s : traces(P) \bullet P = P/s$$

Our first attempt to define the notion of statelessness of a process P says that after communicating any events, the process returns to its initial state.

This definition is satisfactory only so long as a typical request-response operation is modelled as a single event. But we want to consider the interaction of the server with back-end stateful resources, which usually occurs between the request and response messages and therefore have to model the request and response as separate events.

3.1 Threads

We first define a CSP process W, which is willing to accept any events on the channels *response* and *request*, provided the events alternate between request and response and begin with request.

Definition 3.2.

$$\begin{aligned} W &= \mathit{request}?x \rightarrow W' \\ W' &= \mathit{response}?y \rightarrow W \end{aligned}$$

We also define for convenience αW as all request and response events.

Definition 3.3.

$$\alpha W = \{\!|\mathit{request}, \mathit{repsonse}|\!\}$$

We now define a *thread.*

Definition 3.4.

$$\begin{aligned} \mathit{Thread}(P) \Leftrightarrow{} & P = P \mathbin{[\![\alpha W]\!]} W \wedge \\ & \mathit{initials}(P) \subseteq \{\!|\mathit{request}|\!\} \end{aligned}$$

Thus if a process P is a thread, it alternates between request and response events, and can do nothing until its first request is received. Other than that, events may occur at any time.

3.2 Stateless Threads

Our definition of a *stateless thread* is as follows, where where $\mathit{last}(s)$ returns the last event in the trace s.

Definition 3.5.

$$\begin{aligned} \mathit{Stateless}(P) \Leftrightarrow{} & \mathit{Thread}(P) \wedge \\ & \forall\, s : \mathit{traces}(P) \mid \mathit{last}(s) \in \{\!|\mathit{response}|\!\} \bullet P/s = P \end{aligned}$$

Thus a thread P is stateless if it always returns to its initial state after communicating a response event.

3.3 Scalable Threads

We define a further property threads may exhibit we refer to as *scalability.*

Definition 3.6.

$$\begin{aligned} \mathit{Scalable}(P) \Leftrightarrow{} & \mathit{Thread}(P) \wedge \\ & P = (P \mathbin{|||} P) \mathbin{[\![\alpha W]\!]} W \end{aligned}$$

This says in the presence of W, which has the effect of limiting the number of outstanding requests to one, two copies of P behave like a single one.

In B.8 we show that it follows from this definition that $P = (P \mathbin{|||} P \mathbin{|||} P) \mathbin{[\![\alpha W]\!]} W$. Indeed when an arbitrary number of P's are interleaved in the presence of W the resulting combination is identical to P

We also show in B.6 that $\mathit{Stateless}(P) \Rightarrow \mathit{Scalable}(P)$. Interestingly the converse does not hold. Consider:

$$\begin{aligned} P(n) = {} & \mathit{request.up} \to \mathit{response.ok} \to P(n+1) \\ & \Box \\ & (n > 0)\&\mathit{request.down} \to \mathit{response.ok} \to P(n-1) \end{aligned}$$

$P(0)$ is scalable but not stateless. We note that although $P(0) = (P(0) \mathbin{|||} P(0)) \llbracket \alpha W \rrbracket P(0)$, a request to the interleaved combination must be forwarded to the right thread i.e. the one which can accept the event, and this feature of the interleaving operator seems hard to realize in practice. Of course if P is stateless, requests can be forwarded to either thread, since both always accept the same events.

3.4 Examples

The following defines a process P for which $\mathit{Stateless}(P)$ holds:

$$\begin{aligned} P = \mathit{request?name} \to {} & \text{if } \mathit{Cleared}(\mathit{name}) \text{ then} \\ & \quad (\mathit{store.name} \to \mathit{response.ok} \to P) \\ & \quad \Box \\ & \quad (\mathit{full} \to \mathit{reponse.failed} \to P) \\ & \text{else} \\ & \quad \mathit{response.failed} \to P \end{aligned}$$

This process models a thread used in a very simple airline booking system. A booking request is made with the passenger's name, then a security check is made with the function *Cleared*. If the passenger clears security, an attempt is made to add them to the passenger list (an auxiliary process), via the event *store.name*, otherwise they are rejected. The passenger can still be rejected if the flight is full.

The following defines a process $P(0)$ for which $\mathit{Thread}(P(0))$ holds, but $\mathit{Stateless}(P(0))$ does not:

$$\begin{aligned} P(x) = \mathit{request?n} \to {} & \text{if } \mathit{Cleared}(\mathit{name}) \text{ then} \\ & \quad ((x < \mathit{MaxPassengers})\&\mathit{response.ok} \to P(x+1)) \\ & \quad \Box \\ & \quad ((x = \mathit{MaxPassengers})\&\mathit{reponse.failed} \to P(x)) \\ & \text{else} \\ & \quad \mathit{response.failed} \to P(x) \end{aligned}$$

This process models the same booking system, but with no need for an auxiliary process to keep track of bookings. This process is not scalable: a single copy of $P(0)$ permits only *MaxPassengers* successful bookings whereas two copies of $P(0)$ might permit more.

3.5 Services

We now build a model of services. We will assume that the set *Session* contains a number of session identifiers. Compared to to a single thread, every request and response to a service contains the session identifier as an additional parameter. If P is a thread, we form a *service* made of threads with P 's behaviour by defining the function *Service*.

Definition 3.7.

$$Service(P) = \mathbin{|||}\, s : Session \bullet P[request \leftarrow req.s, response \leftarrow resp.s]$$

Our model is deliberately simplistic in that we do not show how clients acquire sessions. We also assume there is a thread available for each session, so that clients never block waiting for available threads which is possibly unrealistic. An extra layer, modelling how sessions are assigned and limits on the number of concurrently active sessions could easily be added but is beyond the scope of this paper.

If P is scalable, then an obvious consequence is $Service(P)$ is scalable in the sense that:

$$Service(P) = (Service(P) \mathbin{|||} Service(P)) \mathbin{[\![} \{\!|req, resp|\!\} \mathbin{]\!]} Service(W)$$

where $Service(W)$ models the constraint that there is never more than one request outstanding per session.

In our example airline booking system (Sec. 3.4) we can see that building a service with the stateless thread version would be advantageous if the function *Cleared* takes considerable resources; the workload can be spread across multiple servers.

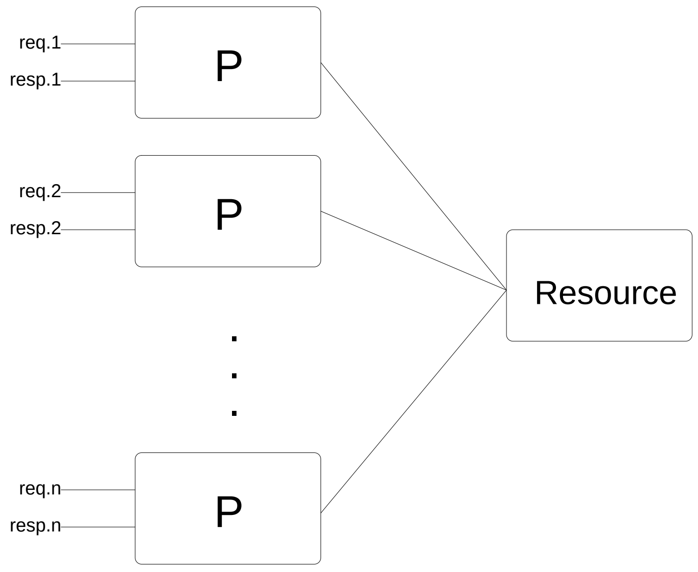
We model the *stateful resources* upon which a service may act simply as another process which is forbidden to communicate directly with clients. The following predicate determines if a process R is a resource.

Definition 3.8.

$$Resource(R) \Leftrightarrow Chaos(\Sigma - \{\!|req, resp|\!\}) \sqsubseteq R$$

(where Σ denotes the set of all events.)

The following diagram shows the communication channels and component processes in our general model of a a service that acts upon stateful resources



3.6 Formalized taxonomy

Using our preceding definitions of thread, stateless thread, service, and stateful resource, we redefine the [FFGT03] taxonomy of web services formally.

- A *stateless service* is a service made of stateless threads.
- A *conversational service* is a service made of threads (stateless or not).
- A *stateless service that acts upon stateful resources* is a stateless service in parallel with one or more stateful resources.
- A *conversational service that acts upon stateful resources* is a conversational service in parallel with one or more stateful resources.

This taxonomy can be made disjoint, by defining each category to exclude the ones which precede it in the above list. e.g. a conversational service is a service made of threads which are not stateless.

We can also formalize the definition of service in the context of Service-Oriented Architecture given in [PWWR03], [W3C]. Such services correspond to stateless services and stateless services that act upon stateful resources (the first and third types in the above taxonomy).

4 Discussion

Key to our distinction between stateless services that act upon stateful resources and conversational services that act upon stateful resources, is that the threads from which they are composed are not aware of which session they serve. That is although, each client makes requests of the form $req.s.x$, where s identifies the session, only the x component of the event is passed to the receiving thread. Suppose alternatively that s is also passed to the thread, so that the session can be identified. The threads from which a conversational service is composed, can be modified to load their state (indexed by each session s) from back-end stateful resources immediately after receiving a request, and to save their state to stateful resources immediately before each response. Thus (under our extra assuption), for every conversational service that acts upon stateful resources there exists a stateless service that acts upon stateful resources with the same behaviour.

In fact this is a well-known technique for achieving what is in effect a conversational service in the context of Service-Oriented Architecture, known as *contextualization* [PWWR03]: every message passed between the service and its client contains a unique context identifier.

This being the case we may ask is if the distinction we draw matters? As a 'black box' there is little between a stateless service that acts upon stateful resources and a conversational one. However, a stateless service has important 'white box' properties that the conversational service does not: the ability to replicate its stateless front end to achieve scalability. The use of stateless services may also improve the modularity of a design.

We note also, that statelessness is not preserved by refinement e.g.

$$\begin{aligned} P = \mathit{request}?x \rightarrow (&\mathit{response}.1 \rightarrow P \\ &\sqcap \\ &\mathit{response}.2 \rightarrow P) \end{aligned}$$

is a stateless thread, whilst:

$$P' = \mathit{request}?x \rightarrow \mathit{response}.1 \rightarrow \mathit{request}?x \rightarrow \mathit{reponse}.2 \rightarrow P'$$

is not, even though though P' refines P. We argue that one should be concerned only with the statelessness of specifications and not implementations. For suppose *SPEC* is a stateless thread and *IMPL* is a refinement of it, which is not stateless. Although $(\mathit{IMPL} \mathbin{|||} \mathit{IMPL}) \mathbin{[\![\, \alpha W \,]\!]} W \neq \mathit{IMPL}$ in general, it still holds (by monotonicity) that $\mathit{SPEC} \sqsubseteq (\mathit{IMPL} \mathbin{|||} \mathit{IMPL}) \mathbin{[\![\, \alpha W \,]\!]} W$. *IMPL* still has the property that it can be replicated as required for scalability whilst satisfying its specification.

5 Conclusion

We have given formal definitions of stateless and conversational services and Service-Oriented Architecture, and explained their relationship. If contextualization is permitted, the distinction between stateless and conversational services, that act upon stateful resources cannot be determined by external behaviour; rather is it an internal property that can be used to achieve scalability. Finally we have explained how, in the presence of non-determinism, it is possible to have a stateful implementation of a stateless service specification, and thus it is only whether a service's specification is stateless that matters. We hope the ongoing debate into services and state is informed by these observations.

References

[FFGT03] I. Foster, J. Frey, S. Graham, and S. Tuecke. Modelling stateful resources with web services. Computer Associates International, Fujitsu Limited, IBM, The Hewlett-Packard Development Company, The University of Chicago, 2003.

[FKNT02] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.

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

[PWWR03] S. Parastatidis, J. Weber, P. Watson, and T. Rischbeck. A grid application framework based on web services specifications and practices. North East Regional e-Science Centre, School of Computing Science, University of Newcastle, UK, 2003.

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

[Vog03] Werner Vogels. Web services are not distributed objects. *IEEE Internet Computing*, 7(6):59–66, 2003.

[W3C] Web services architecture. W3C. http://www.w3.org/TR/2003/WD-ws-arch-20030808/.

A Checking thread properties with FDR

We have used in our definition of threads and stateless threads properties which cannot be readily checked with the FDR model checker for CSP. We here give alternative definitions which can.

To check $\mathit{initials}(P) \subseteq \{\!|\mathit{request}|\!\}$ we can ask FDR:

$$\mathit{request}?x \to \mathit{RUN}_\Sigma \sqsubseteq_T P$$

We now consider how to check $\forall\, s : \mathit{traces}(P) \mid \mathit{last}(s) \in \{\!|\mathit{response}|\!\} \bullet P/s = P$. Without loss of generality we assume *tock* is an event in the alphabet which is not used by the process P. (We can always enlarge the alphabet with a spare event if required.) We define:

$$\begin{aligned} S = {} & ?x : (\Sigma - \{\!|\mathit{response}|\!\}) \to S \\ & \Box \\ & ?x : \{\!|\mathit{response}|\!\} \to \mathit{tock} \to \mathit{STOP} \end{aligned}$$

Thus $Q = P \mathbin{[\![\Sigma - \{\mathit{tock}\}]\!]} S$ behaves like P up to and including the first response event, and then becomes $\mathit{tock} \to \mathit{STOP}$. So the process $(Q \mathbin{[\![\{\mathit{tock}\}]\!]} \mathit{tock} \to P) \setminus \{\mathit{tock}\}$ similarly behaves like P up to and including the first response event, and then behaves like P. Thus we check whether P is stateless by checking P is a thread, and then asking FDR if:

$$(Q \mathbin{[\![\{\mathit{tock}\}]\!]} \mathit{tock} \to P) \setminus \{\mathit{tock}\} = P$$

B Proofs of theorems

Let P be a thread.

The following predicate holds exactly when process X can diverge immeadiately.

Definition B.1.

$$\mathit{diverges}(X) \Leftrightarrow X = X \sqcap \mathit{div}$$

Lemma B.2.

$$\forall\, s : \mathit{traces}(P) \bullet \#s \upharpoonright \{\!|\mathit{request}|\!\} \geq \#s \upharpoonright \{\!|\mathit{response}|\!\}$$

Proof. This is an easy consequence of $\mathit{Thread}(P)$. □

Definition B.3.

Write Q_s *for* P/s *if* $s : \mathit{traces}(P) \wedge \#s \upharpoonright \{\!|\mathit{request}|\!\} = \#s \upharpoonright \{\!|\mathit{response}|\!\}$
Write R_s *for* P/s *if* $s : \mathit{traces}(P) \wedge \#s \upharpoonright \{\!|\mathit{request}|\!\} > \#s \upharpoonright \{\!|\mathit{response}|\!\}$

Lemma B.4.

$$\begin{aligned} &\mathit{initials}\ Q_s \cap \{\!|\mathit{response}|\!\} = \emptyset \\ &\mathit{initials}\ R_s \cap \{\!|\mathit{request}|\!\} = \emptyset \end{aligned}$$

Proof. This is an easy consequence of $\mathit{Thread}(P)$. $\square$

Convention B.5. The interface of the parallel operator ($\|$) is αW unless stated otherwise. The interleaving operator ($|||$) binds more tightly than than the parallel operator. e.g. $P \mathbin{|||} P \parallel W$ stands for $(P \mathbin{|||} P) \mathbin{[\![\alpha W]\!]} W$

Theorem B.6.

$$\mathit{Stateless}(P) \Rightarrow \mathit{Scalable}(P)$$

Proof. Suppose $\mathit{Stateless}(P)$. We show that under our assuptions, for every trace t of $P \mathbin{|||} P \parallel W$ and every trace t of P:

$$\begin{array}{l}(P \mathbin{|||} P \parallel W)/t = (P/t) \mathbin{|||} P \parallel W = P/t \\ \quad \text{if } t = \langle\rangle \vee \mathit{last}(t) \in \{\!|\mathit{response}|\!\} \\ \\ (P \mathbin{|||} P \parallel W)/t = (P/t) \mathbin{|||} P \parallel W' = P/t \\ \quad \text{otherwise}\end{array}$$

and hence $(P \mathbin{|||} P) \parallel W = P$, i.e. $\mathit{Scalable}(P)$.

To prove the above equality, we show that the initials, refusals and initial divergences of the terms are equal, and that after each initial event the result states are also equal if we assume the above statements. This can be formally justified by reference to the theory of constructive recursions and unique fixeds points (UFPs) in CSP [Ros98].

We note that due to $\mathit{Stateless}(P)$ if $x \in \{\!|\mathit{response}|\!\}$ then $Q_{s \frown \langle x \rangle} = Q_{\langle\rangle}$.

Case (i) $P1 = Q_{\langle\rangle} \wedge P2 = Q_{\langle\rangle} \mathbin{|||} Q_{\langle\rangle} \parallel W$.

$$\begin{array}{l}\mathit{initials}(P1) = \mathit{initials}(P2) \\ \mathit{refusals}(P1) = \mathit{refusals}(P2) \\ \mathit{diverges}(P1) \Leftrightarrow \mathit{diverges}(P2) \\ \\ \forall x : \mathit{initals}(P1) \bullet P1/\langle x \rangle = R_{\langle x \rangle} \\ \\ \forall x : \mathit{initials}(P2) \bullet P2/\langle x \rangle \\ \quad = (R_{\langle x \rangle} \mathbin{|||} Q_{\langle\rangle} \parallel W') \mathbin{\Box} (Q_{\langle\rangle} \mathbin{|||} R_{\langle x \rangle} \parallel W') \\ \quad = R_{\langle x \rangle} \mathbin{|||} Q_{\langle\rangle} \parallel W'\end{array}$$

Case (ii) $P1 = R_s \wedge P2 = R_s \mathbin{|||} Q_{\langle\rangle} \parallel W'$.

$$\begin{array}{l}\mathit{initials}(P1) = \mathit{initials}(P2) \\ \mathit{refusals}(P1) = \mathit{refusals}(P2) \\ \mathit{diverges}(P1) \Leftrightarrow \mathit{diverges}(P2) \\ \\ \forall x : \mathit{initals}(P1) \bullet P1/\langle x \rangle = \\ \quad \text{if } x \in \{\!|\mathit{response}|\!\} \text{ then } Q_{s \frown \langle x \rangle} \text{ else } R_{s \frown \langle x \rangle} = \\ \quad \text{if } x \in \{\!|\mathit{response}|\!\} \text{ then } Q_{\langle\rangle} \text{ else } R_{s \frown \langle x \rangle} \\ \\ \forall x : \mathit{initials}(P2) \bullet P2/\langle x \rangle = \\ \quad \text{if } x \in \{\!|\mathit{response}|\!\} \text{ then } Q_{s \frown \langle x \rangle} \mathbin{|||} Q_{\langle\rangle} \parallel W \\ \quad \text{else } R_{s \frown \langle x \rangle} \mathbin{|||} Q_{\langle\rangle} \parallel W' = \\ \quad \text{if } x \in \{\!|\mathit{response}|\!\} \text{ then } Q_{\langle\rangle} \mathbin{|||} Q_{\langle\rangle} \parallel W \\ \quad \text{else } R_{s \frown \langle x \rangle} \mathbin{|||} Q_{\langle\rangle} \parallel W'\end{array}$$

$\square$

Theorem B.7.

$$P \interleave P \interleave P \parallel W = P \interleave (P \interleave P \parallel W) \parallel W$$

Proof. The proof of this theorem is based on the a similar technique to the previous one. We cover all reachable states by showing the following equalities:

$$\begin{array}{c}
Q_s \interleave Q_t \interleave Q_v \parallel W = Q_s \interleave (Q_t \interleave Q_v \parallel W) \parallel W \\
R_s \interleave Q_t \interleave Q_v \parallel W' = R_s \interleave (Q_t \interleave Q_v \parallel W) \parallel W' \\
Q_s \interleave R_t \interleave Q_v \parallel W' = Q_s \interleave (R_t \interleave Q_v \parallel W') \parallel W' \\
Q_s \interleave Q_t \interleave R_v \parallel W' = Q_s \interleave (Q_t \interleave R_v \parallel W') \parallel W'
\end{array}$$

That is, we show the initials, refusals and initial divergences of the terms are equal and that the result states are also equal if we assume the above statements.

Case (i) $P1 = Q_s \interleave Q_t \interleave Q_v \parallel W \wedge P2 = Q_s \interleave (Q_t \interleave Q_v \parallel W) \parallel W$.

$$\begin{array}{l}
\mathit{initials}\ P1 = \mathit{initials}\ Q_s \cup \mathit{initials}\ Q_t \cup \mathit{initials}\ Q_v = \mathit{initials}\ P2 \\
\mathit{refusals}\ P1 = \mathit{refusals}\ Q_s \cap \mathit{refusals}\ Q_t \cap \mathit{refusals}\ Q_v = \mathit{refusals}\ P2 \\
\mathit{diverges}\ P1 \Leftrightarrow \mathit{diverges}\ Q_s \vee \mathit{diverges}\ Q_s \vee \mathit{diverges}\ Q_v \Leftrightarrow \mathit{diverges}\ P2
\end{array}$$

```
∀ $x$ : $\mathit{initials}\ P1 \bullet P1/\langle x \rangle =$
    $\sqcap\, i : \{s, t, v\} \mid x \in \mathit{initials}(P/i) \bullet$
        if $x \notin \{\!|\mathit{request}|\!\}$ then
            $(i = s)\&(Q_s{}^\frown\langle x \rangle \interleave Q_t \interleave Q_v \parallel W)$
            $\square$
            $(i = t)\&(Q_s \interleave Q_t{}^\frown\langle x \rangle \interleave Q_v \parallel W)$
            $\square$
            $(i = v)\&(Q_s \interleave Q_t \interleave Q_v{}^\frown\langle x \rangle \parallel W)$
        else
            $(i = s)\&(R_s{}^\frown\langle x \rangle \interleave Q_t \interleave Q_v \parallel W')$
            $\square$
            $(i = t)\&(Q_s \interleave R_t{}^\frown\langle x \rangle \interleave Q_v \parallel W')$
            $\square$
            $(i = v)\&(Q_s \interleave Q_t \interleave R_v{}^\frown\langle x \rangle \parallel W')$
```

```
∀ $x$ : $\mathit{initials}\ P2 \bullet P2/\langle x \rangle =$
    $\sqcap\, i : \{s, t, v\} \mid x \in \mathit{initials}(P/i) \bullet$
        if $x \notin \{\!|\mathit{request}|\!\}$ then
            $(i = s)\&(Q_s{}^\frown\langle x \rangle \interleave (Q_t \interleave Q_v \parallel W) \parallel W)$
            $\square$
            $(i = t)\&(Q_s \interleave (Q_t{}^\frown\langle x \rangle \interleave Q_v \parallel W) \parallel W)$
            $\square$
            $(i = v)\&(Q_s \interleave (Q_t \interleave Q_v{}^\frown\langle x \rangle \parallel W) \parallel W)$
        else
            $(i = s)\&(R_s{}^\frown\langle x \rangle \interleave (Q_t \interleave Q_v \parallel W) \parallel W')$
            $\square$
            $(i = t)\&(Q_s \interleave (R_t{}^\frown\langle x \rangle \interleave Q_v \parallel W') \parallel W')$
            $\square$
            $(i = v)\&(Q_s \interleave (Q_t \interleave R_v{}^\frown\langle x \rangle \parallel W') \parallel W')$
```

Case (ii) $P1 = R_s \interleave Q_t \interleave Q_v \parallel W' \wedge P2 = R_s \interleave (Q_t \interleave Q_v \parallel W) \parallel W'$.

$$\begin{array}{l}
\textit{initials}\ P1 = \\
\quad (\textit{initials}\ R_s \cup \textit{initials}\ Q_t \cup \textit{initials}\ Q_v) - \{\!|\textit{request}|\!\} = \\
\quad \textit{initials}\ P2 \\
\textit{refusals}\ P1 = \\
\quad \{r : \textit{refusals}\ R_s \mid r - \{\!|\textit{request}|\!\} \in \textit{refusals}\ Q_t \cap \textit{refusals}\ Q_v\} = \\
\quad \textit{refusals}\ P2 \\
\textit{diverges}\ P1 \Leftrightarrow \\
\quad \textit{diverges}\ R_s \vee \textit{diverges} Q_t \vee \textit{diverges} Q_v \Leftrightarrow \\
\quad \textit{diverges}\ P2
\end{array}$$

```
$\forall x : \textit{initials}\ P1 \bullet P1/\langle x \rangle =$
    $\sqcap i : \{s, t, v\} \mid x \in \textit{initials}(P/i) \bullet$
        if $x \notin \{\!|\textit{response}|\!\}$ then
            $(i = s)\&(R_s{}^\frown\langle x \rangle \interleave Q_t \interleave Q_v \parallel W')$
            $\Box$
            $(i = t)\&(R_s \interleave Q_t{}^\frown\langle x \rangle \interleave Q_v \parallel W')$
            $\Box$
            $(i = v)\&(R_s \interleave Q_t \interleave Q_v{}^\frown\langle x \rangle \parallel W')$
        else
            $(Q_s{}^\frown\langle x \rangle \interleave Q_t \interleave Q_v \parallel W)$
```

```
$\forall x : \textit{initials}\ P2 \bullet P2/\langle x \rangle =$
    $\sqcap i : \{s, t, v\} \mid x \in \textit{initials}(P/i) \bullet$
        if $x \notin \{\!|\textit{response}|\!\}$ then
            $(i = s)\&(R_s{}^\frown\langle x \rangle \interleave (Q_t \interleave Q_v \parallel W) \parallel W')$
            $\Box$
            $(i = t)\&(R_s \interleave (Q_t{}^\frown\langle x \rangle \interleave Q_v \parallel W) \parallel W')$
            $\Box$
            $(i = v)\&(R_s \interleave (Q_t \interleave Q_v{}^\frown\langle x \rangle \parallel W) \parallel W')$
        else
            $(Q_s{}^\frown\langle x \rangle \interleave (Q_t \interleave Q_v \parallel W) \parallel W)$
```

Case (iii) $P1 = Q_s \interleave R_t \interleave Q_v \parallel W' \wedge P2 = Q_s \interleave (R_t \interleave Q_v \parallel W') \parallel W'$.

$$\begin{array}{l}
\textit{initials}\ P1 = \\
\quad (\textit{initials}\ Q_s \cup \textit{initials}\ R_t \cup \textit{initials}\ Q_v) - \{\!|\textit{request}|\!\} = \\
\quad \textit{initials}\ P2 \\
\textit{refusals}\ P1 = \\
\quad \{r : \textit{refusals}\ R_t \mid r - \{\!|\textit{request}|\!\} \in \textit{refusals}\ Q_s \cap \textit{refusals}\ Q_v\} = \\
\quad \textit{refusals}\ P2 \\
\textit{diverges}\ P1 \Leftrightarrow \\
\quad \textit{diverges}\ R_s \vee \textit{diverges} Q_t \vee \textit{diverges} Q_v \Leftrightarrow \\
\quad \textit{diverges}\ P2
\end{array}$$

```
$\forall x : \mathit{initials}\ P1 \bullet P1/\langle x \rangle =$
    $\sqcap i : \{s, t, v\} \mid x \in \mathit{initials}(P/i) \bullet$
        if $x \notin \{\!|\mathit{response}|\!\}$ then
            $(i = s)\&(Q_s{}^\frown\langle x\rangle \interleave R_t \interleave Q_v \parallel W')$
            $\Box$
            $(i = t)\&(Q_s \interleave R_t{}^\frown\langle x\rangle \interleave Q_v \parallel W')$
            $\Box$
            $(i = v)\&(Q_s \interleave R_t \interleave Q_v{}^\frown\langle x\rangle \parallel W')$
        else
            $(Q_s \interleave Q_t{}^\frown\langle x\rangle \interleave Q_v \parallel W)$
```

```
$\forall x : \mathit{initials}\ P2 \bullet P2/\langle x \rangle =$
    $\sqcap i : \{s, t, v\} \mid x \in \mathit{initials}(P/i) \bullet$
        if $x \notin \{\!|\mathit{response}|\!\}$ then
            $(i = s)\&(Q_s{}^\frown\langle x\rangle \interleave (R_t \interleave Q_v \parallel W') \parallel W')$
            $\Box$
            $(i = t)\&(Q_s \interleave (R_t{}^\frown\langle x\rangle \interleave Q_v \parallel W') \parallel W')$
            $\Box$
            $(i = v)\&(Q_s \interleave (R_t \interleave Q_v{}^\frown\langle x\rangle \parallel W') \parallel W')$
        else
            $(Q_s \interleave (Q_t{}^\frown\langle x\rangle \interleave Q_v \parallel W) \parallel W)$
```

Case (iv) $P1 = Q_s \interleave Q_t \interleave R_v \parallel W' \wedge P2 = Q_s \interleave (Q_t \interleave R_v \parallel W') \parallel W'$. Similarly.

$\Box$

Corollary B.8.

$$\mathit{Scalable}(P) \Rightarrow P = P \interleave P \interleave P \parallel W$$

Proof.

$$\begin{aligned}
&\mathit{Scalable}(P) \Rightarrow P = P \interleave P \parallel W = P \interleave (P \interleave P \parallel W) \parallel W\\
&P \interleave (P \interleave P \parallel W) \parallel W = P \interleave P \interleave P \parallel W\\
&\mathit{Scalable}(P) \Rightarrow P = P \interleave P \interleave P \parallel W
\end{aligned}$$

$\Box$