Introducing mobility into CSP||B

Steve Schneider, Helen Treharne, and Beeta Vajar

Department of Computing
University of Surrey
Guildford, Surrey, UK

Abstract

CSP||B is a combination of CSP and B which supports the design and verification of systems where control and state are both important. Recent work on combining the pi-calculus with B introduces mobility into the control of B machines and enables them to be passed around. This additional functionality is appropriate for modeling systems such as peer-to-peer networks. While the pi-calculus is appropriate for the description of mobility, the semantic foundation for pi |B is cumbersome for reasoning about systems, and a CSP based approach may be preferable. This paper considers how CSP can be extended to support mobile channels while retaining a trace-based semantics, within the context of CSP||B. While this is difficult for CSP in general, the restricted context provided by CSP||B does enable some progress to be made in this area. This is work in progress.

Keywords: CSP, B, mobility, combining formal methods

1 Introduction

CSP||B [6,5] is a combination of CSP and B which supports the design and verification of systems where control and state are important. CSP [4,7] is a process algebra particularly suited to describing and reasoning about concurrent systems in terms of interactions between components. The B-Method [1] is a state-based formal method based around abstract machines: self-contained components which maintain local state and provide operations on that state. It is particularly suited to dealing with information-rich systems.

A CSP||B system contains a number of controlled components. A controlled component consists of (1) a controlled B machine and (2) a CSP controller: a sequential CSP process which calls operations of the B machine and also interacts with other CSP controllers. This architecture enables verification of consistency between a controller and its controlled machine via a 'control loop invariant' technique, which enables proof that operation calls are always within their preconditions. Controlled components can then be combined to form larger and more complex systems. This approach results in static architectures in which a B machine M_i is tied to a particular CSP component P_i, as illustrated on the left in Figure 1.

More recently, the use of the pi-calculus as the control language has been introduced [2] to create a combination pi|B, in order to enable more dynamic architec-

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

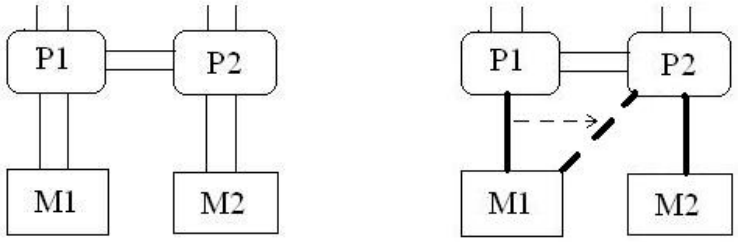

Fig. 1. CSP‖B architecture and pi|B architecture

tures in which communication channels for machines can be passed, as illustrated on the right in Figure 1. These ideas also appear in occam-pi [8]. Such architectures are suitable for modeling agent systems or peer-to-peer networks where consideration of mobility is important. The pi-calculus is natural for expressing mobile structures, but its operational semantics is cumbersome for reasoning about these structures. We are therefore interested in introducing the aspects of pi|B which support mobility into the CSP‖B framework in order to support reasoning at a more abstract level.

This paper introduces a limited form of mobility into CSP, specifically driven by the requirements of CSP‖B. We follow part of the approach taken in pi|B, which uses a special set MR of *machine references* as the links to interact with the B machines, and a special set CP of channels called *control points* on which machine references are passed around. We will also use a set C of regular CSP channels.

2 Mobile CSP‖B

A controlled component consists of a sequential CSP controller P in parallel with a B machine M. Operations op with inputs s and outputs t are declared in machines M as $t \longleftarrow op(s)$. In the combination they are treated as channels $op.s.t$. Morgan's failures-divergences semantics for action systems [3] gives a CSP semantics to M, providing a semantic foundation for reasoning about $P \parallel M$. Previous work [6,5] has developed techniques for proving that P always calls operations of M within their preconditions. This is what is meant by *consistency* of $P \parallel M$.

In order to pass machines between controllers, we introduce a unique machine channel z for each machine instance in the system. Operation calls correspond to the communication $z.op.s.t$, and the machine reference z can itself be passed between controllers. We require that only one sequential controller is in possession of z at any one time, so that when z is passed from P_1 to P_2 then P_1 is no longer able to use z to call the operations of the machine. The language for constructing controllers enforces this. This will be the cornerstone for reasoning about the action of controllers on a mobile machine: that a controller has an exclusive lock on a machine it is using, and other controllers cannot interfere with its use of the machine.

Each channel c in the set of regular channels C has a *type* denoted $type(c)$. The type of channels in CP (control points, which pass machine references) is MR: in other words, they pass values from the set MR. Each machine reference in MR is associated with a particular B machine. The type of a machine reference z is the set of operations (with inputs and outputs) of the unique machine M that is associated

with z.

The *channels* $\chi(P)$ of a process P are given by its communication channels and control points. Any particular control point in the alphabet of P will be either incoming or outgoing with respect to P, and is not permitted to be both. We identify the incoming control points within $\chi(P)$ as $\chi_i(P)$. The outgoing control points within $\chi(P)$ are denoted $\chi_o(P)$. The alphabet associated with communication channels is denoted $\chi_c(P)$. For any process these three sets are pairwise disjoint, and their union is $\chi(P)$.

Sequential process terms are defined by the following BNF:

$$\begin{aligned} P ::= & \mathit{STOP} \mid c?x \rightarrow P(x) \mid c!v \rightarrow P \mid cp_1?w \rightarrow P(w) \\ & \mid z.op!s?t \rightarrow P(t) \mid cp_2!z \rightarrow P \ \ (z \notin \mathit{fv}(P)) \\ & \mid P_1 \,\Box\, P_2 \mid P_1 \sqcap P_2 \mid N(v_1, \ldots, v_n) \end{aligned}$$

where $c \in \chi_c(P)$, $cp_1 \in \chi_i(P)$, $cp_2 \in \chi_o(P)$, $v \in \mathit{type}(c)$, $z \in \mathit{MR}$, $t \longleftarrow \mathit{op}(s)$ is an operation of the B machine associated with z. $\mathit{fv}(P)$ is the set of free variables in P, including variables for machine references.

Sequential processes are then defined recursively as follows:

$$N(v_1, \ldots, v_n) \mathrel{\widehat{=}} P \qquad \text{where } \mathit{fv}(P) \subseteq \{v_1, \ldots, v_n\}$$

The machine references that N knows initially will appear in the list $v_1, \ldots, v_n$.

The trace semantics for this sequential controller language is straightforward and so will not be presented here for reasons of space.

3 Parallel combination

Sequential processes may be composed in parallel, provided they have no free variables in common. They must also differ on their incoming control points and their outgoing control points. Parallel combinations of controllers are defined by the following BNF:

$$Q ::= N(v_1, \ldots, v_n) \mid Q_1 \parallel Q_2$$

where $\mathit{fv}(Q_1) \cap \mathit{fv}(Q_2) = \emptyset$, $\chi_i(Q_1) \cap \chi_i(Q_2) = \emptyset$, $\chi_o(Q_1) \cap \chi_o(Q_2) = \emptyset$. N must be a recursively defined sequential process. The free variables of a parallel combination is given by $\mathit{fv}(Q_1 \parallel Q_2) = \mathit{fv}(Q_1) \cup \mathit{fv}(Q_2)$.

The alphabets for parallel combinations are given as follows:

$$\begin{aligned} \chi_i(Q_1 \parallel Q_2) &= \chi_i(Q_1) \cup \chi_i(Q_2) \\ \chi_o(Q_1 \parallel Q_2) &= \chi_o(Q_1) \cup \chi_o(Q_2) \\ \chi_c(Q_1 \parallel Q_2) &= \chi_c(Q_1) \cup \chi_c(Q_2) \end{aligned}$$

Note that incoming and outgoing control points need not be disjoint in parallel combinations: a control point that is both incoming and outgoing has both ends within the parallel combination and hence connects two of the parallel components. The rule for parallel ensures that no further parallel components will use that control point. We do not follow the standard approach of hiding such control points because we are aiming for a trace semantics and still wish communications over them to them to appear in the trace.

The language of process terms has been designed to ensure that at any point in an execution of a process, at most one parallel component has possession of any machine reference. Processes can be composed in parallel only if (1) they do not share any machine references to begin with, and (2) when a machine reference is passed along a control point to another process, it is not retained by the sending process.

In order to define the traces of parallel composition, it is necessary to keep track of the machine references as they are used and passed between the processes. We can define the projection of a trace onto a particular process P given the channels $\chi_i(P)$, $\chi_o(P)$, $\chi_c(P)$, provided we also know the set of machine references s owned by the process.

The projection of a trace tr onto $\chi(P)$ and a set of machine references s can be defined inductively as follows:

$$\langle\rangle \upharpoonright \chi(P), s = \langle\rangle$$

$$\langle cp.z\rangle \frown tr \upharpoonright \chi(P), s = \begin{cases} \langle cp.z\rangle \frown (tr \upharpoonright \chi(P), s \cup \{z\}) & \text{if } cp \in \chi_i(P) \wedge z \notin s \\ \langle cp.z\rangle \frown (tr \upharpoonright \chi(P), s - \{z\}) & \text{if } cp \in \chi_o(P) \wedge z \in s \\ tr \upharpoonright \chi(P), s & \text{if } cp \notin \chi_i(P) \cup \chi_o(P) \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\langle c\rangle \frown tr \upharpoonright \chi(P), s = \begin{cases} \langle c\rangle \frown (tr \upharpoonright \chi(P), s) & \text{if } c \in \chi_c(P) \\ tr \upharpoonright \chi(P), s & \text{if } c \notin \chi_c(P) \end{cases}$$

$$\langle z.op\rangle \frown (tr \upharpoonright \chi(P), s) = \begin{cases} \langle z.op\rangle \frown tr \upharpoonright \chi(P), s & \text{if } z \in s \\ tr \upharpoonright \chi(P), s & \text{if } z \notin s \end{cases}$$

This enables a definition of the traces of a parallel combination to be given:

$$\begin{aligned} traces(Q_1 \parallel Q_2) = \{tr \mid\ & tr \upharpoonright \chi(Q_1), fv(Q_1) \in traces(Q_1) \\ & tr \upharpoonright \chi(Q_2), fv(Q_2) \in traces(Q_2)\} \end{aligned}$$

4 Example

For reasons of space, we can only illustrate the definitions above with the smallest of examples, given in Figure 2: a system with one machine, *Switch*, (which has a one-bit state) and two control processes which pass *Switch* back and forth. *Switch* has two operations, each with its own precondition. For the controllers there are two control points *cp* and *dp*, and each of them carries machine references to *Switch* machines. In *SYS*, P_1 begins with machine reference z: it calls the operation *on* on the corresponding machine, passes out the machine reference, and then accepts another one before beginning again. Considering P_1 in isolation, a machine received on *dp* could in principle be in any state. If such a machine is in the state $switch = 1$ then the operation call *on* will be outside its precondition and the system will be inconsistent. A similar situation holds for P_2, which relies on receiving a machine in state $switch = 1$.

```
MACHINE         Switch
VARIABLES       switch
INVARIANT       switch : 0..1
INITIALISATION  switch := 0
OPERATIONS
  on  = PRE switch = 0
        THEN switch := 1 END;
  off = PRE switch = 1
        THEN switch := 0 END
END
```

$$P_1(x) = x.on \rightarrow cp!x \rightarrow dp?w \rightarrow P_1(w)$$

$$P_2 = cp?y \rightarrow y.off \rightarrow dp!y \rightarrow P_2$$

$$CON = P_1(z) \parallel P_2$$

$$SYS = CON \parallel z : Switch$$

Fig. 2. A simple example: *Switch* and two controllers

We can see that for the system consisting of P_1 and P_2, this will not happen, since P_2 always sets *Switch* to the appropriate state before passing it back to P_1. More generally, we might hope to reason compositionally about the consistency of individual controllers by associating control points with assertions on the state of the machine being passed. *cp* would then have $switch = 1$ as its assertion, and *dp* would have $switch = 0$. The assertion needs to be guaranteed by the sending process, and can then be assumed by the receiving process. The general case is the subject of current research.

The trace definition for parallel establishes that $\langle z.on\ ,\ cp.z\ ,\ z.off\ ,\ dp.z\ ,\ z.on\rangle$ is a possible trace of CON. The projections to the two components are given by

$$\langle z.on\ ,\ cp.z\ ,\ z.off\ ,\ dp.z\ ,\ z.on\rangle \upharpoonright \chi(P_1), \{z\} = \langle z.on\ ,\ cp.z\ ,\ dp.z\ ,\ z.on\rangle$$

$$\langle z.on\ ,\ cp.z\ ,\ z.off\ ,\ dp.z\ ,\ z.on\rangle \upharpoonright \chi(P_2), \{\} = \langle cp.z\ ,\ z.off\ ,\ dp.z\rangle$$

P_1's involvement in the trace is: calling $z.on$, then passing z out along cp, then receiving z input on dp, and then calling $z.on$ again. P_1 is not involved in the call $z.off$. Conversely, P_2's involvement is: receiving z on cp, calling $z.off$, passing z on dp. P_2 is not involved in either of the $z.on$ calls.

5 Discussion

We have introduced a restricted notion of mobile channels into CSP in order to model the control of B machines which can be passed between processes. These definitions aim to give a formal foundation for mobility in CSP‖B.

The notion of mobility is currently very contained in order to obtain a clean trace semantics for parallel, since effectively only one channel end is ever passed around — the other end is always with the associated B machine. If we were to allow the more general case where arbitrary channel ends can be passed around, (such as in occam-pi [8]), then we enable more dynamic network structures, but the traces will be harder to define, and compositional reasoning will be more difficult.

Acknowledgements

We are grateful to the anonymous reviewers for their comments.

References

[1] J. R. Abrial, *The B Book: Assigning Programs to Meaning*, CUP 1996.

[2] D. Karkinsky, S. A. Schneider and H. E. Treharne, *Combining Mobility with State*, IFM 2007, LNCS 4591, Springer 2007.

[3] C. C. Morgan, *Of wp and CSP*. In *Beauty is our business: a birthday salute to Edsger W. Dijkstra*. Springer 1990.

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

[5] S. A. Schneider and H. E. Treharne, *CSP Theorems for Communicating B machines*, FACS 2004.

[6] S. A. Schneider and H. E. Treharne, *Communicating B machines*, ZB2002, LNCS 2272, Springer 2002.

[7] S. A. Schneider, *Concurrent and Real-Time Systems: the CSP Approach*, John Wiley 1999.

[8] P. H. Welch and F. R. M. Barnes, *Communicating mobile processes: introducing occam-pi*. 25 years of CSP, LNCS 3525, Springer 2005.