Investigating a file transmission protocol using CSP and B (Extended Abstract)

Neil Evans, Steve Schneider and Helen Treharne

Department of Computer Science, Royal Holloway,
University of London, Egham, Surrey, TW20 0EX, UK.

E-mail: {neile,steve,helen}@cs.rhul.ac.uk

Abstract. In this paper a file transmission protocol specification is developed using our CSP || B approach. This demonstrates that it is possible to integrate two approaches whilst maintaining their individual advantages. We discuss how we compositionally verify the specification and ensure that it preserves some abstract properties. We also discuss how the structure of the specification follows a particular style which may be generally applicable when modelling other protocols using CSP || B.

Keywords: B-Method, CSP, composing specifications, combining formalisms, concurrency.

1 Introduction

Our approach to the development of distributed system centres around two formal approaches CSP [6] and B [1]. In this paper we present an overview of a file transmission protocol to illustrate how we develop a distributed system using the CSP || B combination [7–10] . We believe the detailed aspects of the protocol fall naturally into either state or event based descriptions. Our previous work has shown that CSP || B allows the two approaches to be integrated in a straightforward way, by relating CSP events with B operations, and that the integration preserves the original semantics of both languages. The overhead of the integration is minimal and involves consistency checks to ensure that the state and event based descriptions are compatible. Without this compatibility we cannot say that the file transmission protocol is deadlock- and divergence-free. These consistency checks are akin to the establishment of design contracts between the individual components of a specification. One of the main benefits of our approach is that these verification checks are carried out compositionally. Other verification opportunities enable us to capture abstract properties of the protocol using only the event based approach and demonstrate that the CSP || B specification preserves these properties.

In this paper we show that for this particular kind of application, the modelling of the Bounded Retransmission Protocol [5], we can adopt a specific style of specification so that the details within a specification which are needed to enable consistency checking can be identified easily or added systematically.

The protocol described in the paper has also been developed using other approaches including Event B [2], PVS [4], and I/O Automata [5].

The rest of the paper is structured as follows: Section 2 introduces the CSP controller language; Section 3 details the protocol itself and addresses several interesting points relating to its verification; and Section 4 highlights similarities and differences in the characterisation of the protocol as compared with two of the above mentioned approaches.

2 Controller Language

The AMN notation for the B machine descriptions is standard and further details can be found in [1]. We use a subset of CSP to describe the controllers for B machines and it is identical to the subset described in [10]. This is essentially the sequential part of CSP with I/O communications over channels. Initially, when specifying a system, we allow the CSP description to contain $c?x$ and $d!v$ where c and d can be both *communication* and *synchronisation channels*, x represents all data variables on a channel, and v represents all data values being passed along a channel. Furthermore, we allow $e!v?x$ where e is a *machine channel* and v and x are as above. Subsequently, during consistency checking, we may annotate these variables and values with assumptions and guards to strengthen the CSP description. For example, the event $e!v?x\{E(x)\}$ (corresponding to a B operation) can accept any x as input, but will diverge if $E(x)$ is not satisfied. Similarly, the guard $E(x)$ in the annotated event $e!v?x\langle E(x)\rangle$ can accept any x as input, but will deadlock if $E(x)$ is not satisfied. These annotations are only needed to make the design contracts explicit as we shall see in Section 3.1.

3 Outline of the Protocol

There are two main components to the protocol: the sender and the receiver. The sender is willing to accept a file from the environment, and controls the transfer of a sequence of packets that make up the file, over an unreliable medium to the receiver who accepts the file packet by packet. Reporting flags are used to indicate whether a file transfer has been successful (i.e. the system has reached a point in which all packets have been received and acknowledged — *conf.ok*), a possible success (i.e. the last packet has been sent without a subsequent acknowledgement — *conf.dont_know*), or aborted (i.e. after a bounded number of attempts to send an intermediate packet, the file transfer has been abandoned — *conf.not_ok*). The overall CSP || B architecture is defined in Figure 1. The full CSP and B descriptions can be downloaded[1].

The *SenderCtrl* **process** accepts a file, as a sequence of packets, from the external environment. It then passes the whole file to the *Sender* machine so that it can hold the data for subsequent transmission on the medium, whenever such

[1] http://www.cs.rhul.ac.uk/home/helen/papers/st.eve/sources.tar.gz

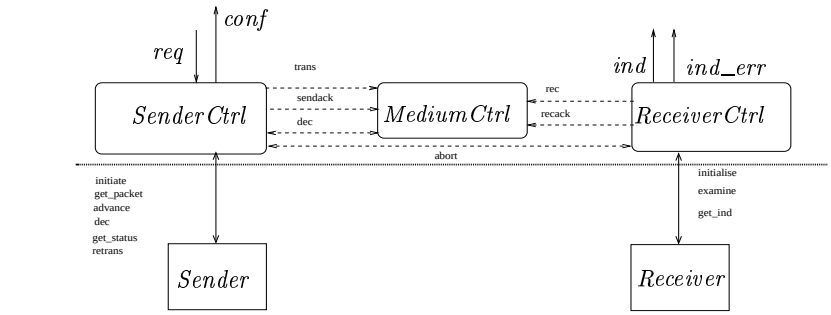

Fig. 1. Communication Architecture of the *Bounded Retransmission Protocol*

an attempt is made by the controller process. This means that the acceptance of a file triggers the machine to set its state to specific values so that the transfer of information can begin appropriately. Consequently, the value of the state variables in the *initialisation* clause of the machine are overridden. Therefore, as a pattern, if there is an initiating event in a controller process which causes the corresponding machine to configure all its state then we need not make the initialisation of the machine deterministic. The implication of this is simplified consistency checking as discussed in Section 3.1.

The *ReceiverCtrl* process is similar in structure to the *SenderCtrl* process in that is contains an initiating branch in its process description. However, the receiver process handles individual packets rather than entire files. Each new packet that is received is labelled and passed on to the external environment. (The labels indicate whether the packet is the beginning of a new file transfer, an intermediate packet, or the last packet in the current file transfer.) It is the duty of the *Receiver* machine to determine whether the packet is indeed new or just a retransmission of the previous packet. This is determined by an alternating bit (called the *toggle*) that is included in the packet.

The *MediumCtrl* process captures the notion of an unreliable medium by defining so-called 'lossy' channels. The medium can accept a packet from the sender on the *trans* channel and non-deterministically choose to pass it on to the receiver (via the *rec* channel) or lose it. Similarly, the medium can accept acknowledgements from the receiver process (a *recack* event) and non-deterministically choose to pass on the acknowledgement to the sender (a *sendack* event) or lose it. Note that the medium is unable to corrupt packets in any other way (such as by changing the contents of the packet). We are able to do this naturally because we are using CSP definitions to capture the control flow of the transfer protocol. It also means that we can experiment easily with different environments.

In the protocol description, time is used to determine when a retransmission

of a packet should occur, and when the protocol should abort. Here, as in all the other formal analyses of this protocol, we abstract away the notion of time by forcing a retransmission only when a packet has been lost. This requires an additional synchronisation between the sender and the medium (along the *dec* machine channel). Similarly, we must force the receiver to abort only when the sender has aborted. Thus, an additional synchronisation channel called *abort* is set up directly between the sender and the receiver.

From the above we observe that we have relaxed a restriction on our original CSP || B architecture. Previously, the alphabets of more than one process could not overlap with the alphabet of a B machine. Now we do allow such multi-way synchronisation between processes and machines. This extra synchronisation (along the *dec* channel) does not introduce different traces into the model and more importantly no new divergences are introduced. Therefore, consistency is preserved.

3.1 Consistency Checking

Consistency checking involves making sure that the whole system is deadlock-free (by checking $SenderCtrl \parallel MediumCtrl \parallel ReceiverCtrl$ is deadlock-free using FDR [3]) and also checking that each component ($SenderCtrl \parallel Sender$) and ($ReceiverCtrl \parallel Receiver$) is compatible. The latter check means that whenever an operation of a machine is called, as a consequence of a CSP event, its precondition must hold. For example, whenever a request is made to retrieve the next packet, as in the following CSP fragment of the *SenderCtrl* process:

$$SENDPACKET = retrans?num\_left \rightarrow if\ num\_left > 0\ then$$
$$get\_packet?p \rightarrow trans.p \rightarrow \ldots$$

then such a packet should be available in the *Sender* machine. This would always be case if the *get_packet* operation, defined in Figure 2, is called within its precondition. Verifying this formally involves demonstrating that the *SENDPACKET* process represents a state in which "there is definitely something to send" so that a relationship between this state and the *file* being non-empty holds true.

In previous examples, these relationships were reasonably straightforward to identify because they were between B variables and CSP process parameters. CSP parameters are often needed to assist with control flow in the controller processes. In the mutually recursive *SenderCtrl* process, parameters are implicit by virtue of the fact that there are two mutually recursive process definitions. We could identify appropriate parameters explicitly by creating a single vector of process definitions for *SenderCtrl* and then extract a set of parameters from the overall indexing set. This gives us a systematic way of explicity describing the relationship between the CSP controller and B variables of its corresponding machine. Thus, in the above example the relationship would be $index = sendpacket \Rightarrow size(file) > 0$. Defining unifying indexing sets is described

```
ff , ll , tt , mm ⟵ get_packet ≙
  PRE size ( file ) > 0 THEN
    ff := send_first || ll := send_last || tt := toggle || mm := first ( file ) ||
    IF send_last = TRUE THEN status := dont_know ELSE status := not_ok
    END
  END
```

Fig. 2. *get_packet* operation from *Sender* Machine

fully in [8]. In this protocol, we do not need to make use of the indices corresponding to the initiating branches of the two main components because all the operations in such branches will always have *true* preconditions. However, we may still need to introduce design contracts. For example, verifying the receiver component involves introducing the design contract $\{l = true \Rightarrow i = last\_packet\}$ on the *initialise* channel. Without this we could have the situation where the sender believes it is sending packets from a new file whereas the receiver still thinks it is processing the previous file. In this example, an alternative to the introduction of the design contract would have been to include further branching in the *ReceiverCtrl* process and offer events whose values are given explicitly. (This would make the *Receiver* machine a redundant specification.) This alternative is only possible because the number of different permutations of possible values is very small. In general the introduction of extra branching should not be adopted as a specification style. The use of succinct design contracts in our approach avoids such cumbersome branching.

3.2 Verification of abstract properties

By describing the desirable properties of the protocol as an abstract CSP process, it is possible to show (using FDR) that the CSP || B protocol specification meets these properties by CSP refinement. This abstract process is defined in terms of the external events of the protocol specification (i.e. the events that communicate with the external environment).

In general, file transfer protocols should act like a buffer. However, this particular protocol has bounded behaviour and, as such, any desirable properties should reflect this. We therefore identify the following properties for verification: if a file is empty nothing is sent over the medium. Hence, $conf.ok$ should be the only outcome. If the file contains one packet then either the whole file is sent, or its packet is not sent at all (before an abort occurs). Thus, the only outcomes are $conf.ok$ or $conf.dont\_know$. If the file contains two packets then, in addition, one packet can be sent (i.e. all packets minus the final packet) before an abort occurs. Thus, there is the additional possibility of $conf.not\_ok$. If the file contains more than two events then, in addition, any other nonempty prefix of the file can be sent before an abort occurs.

4 Discussion

The development process for the file transfer protocol example is comprised of three stages: the first stage is the definition of the specification, the second stage concerns the definition of abstract properties, and the third state involves the verification of the specification against these properties. There is no reason why, in practice, the first two stages could not be carried out in either order. The method followed in [5] is similar to ours since it also comprises of separate stages: the definition of the individual I/O automata and their composition, and the identification of interesting invariants. Within the definition of an individual I/O automaton it is difficult to see the flow of control without analysing the details of all the guards carefully. In our approach we feel that we can naturally separate out the events under the environment's control and show that these trigger a set of internal actions that cause state updates. However, if the environment wanted to accept input which is complex in structure we would have to model a suitable abstraction of it, since the CSP description would not be rich enough to deal with the complex state.

In [2], the protocol is constructed following a series of refinements using Event-B. The advantage of this approach is that, in one atomic step, the overall property on the global state can be captured very abstractly using simple predicates. Such a simple abstraction is not possible with our approach because we have to include behavioural information from the outset. Nonetheless, we believe that during the refinement stages the Event-B approach suffers from the same problem as the I/O automata approach because it becomes increasingly difficult to clearly visualise the control flow within the protocol which we feel is important for this kind of application. We have shown in this paper that CSP is capable of capturing control flow and we can view this in isolation. Nonetheless, we can easily move between the two views of a system.

References

1. Abrial J.R.: *The B Book: Assigning Programs to Meaning*, CUP (1996).
2. Abrial J. R., Mussat L.: *Specification and Design of a Transmission Protocol by Successive Refinements using B.* Mathematical Methods in Program Development, eds. Broy M. and Schieder B., pp. 129–200, Springer (1997).
3. Formal Systems (Europe) Ltd.: `http://www.fsel.com`
4. Havelund K., Shankar N.: *Experiments in Theorem Proving and Model Checking*, FME'96, Oxford, LNCS 1051, Springer, March (1996).
5. Helmink L., Selling M.P.A., and Vaandrager F.W.:*Proof-checking a data link protocol.* "Types for Proofs and Programs", Nijmegen, LNCS 806 Springer (1994).
6. Hoare C. A. R.: *Communicating Sequential Processes*, Prentice Hall (1985).
7. Schneider S.,Treharne H.: *Communicating B Machines.* ZB2002, Grenoble, LNCS 2272, Springer, January (2002).
8. Schneider S.,Treharne H.: *CSP Theorems for Communicating B Machines.* Technical Report CSD-TR-02-12, Dept. of Computer Science, Royal Holloway (2002).
9. Treharne H.: *Controlling Software Specifications.* PhD Thesis, Royal Holloway, University of London (2000).
10. Treharne H., Schneider S., Bramble M.: *Composing Specifications using Communication.* ZB2003, Grenoble, LNCS 2651, Springer, June (2003).