Specification Problem

Manfred Broy
Leslie Lamport

Sat 6 Aug 1994

1 The Procedure Interface

The problem calls for the specification and verification of a series of *components*. Components interact with one another using a procedure-calling interface. One component issues a *call* to another, and the second component responds by issuing a *return*. A call is an indivisible (atomic) action that communicates a procedure name and a list of *arguments* to the called component. A return is an atomic action issued in response to a call. There are two kinds of returns, *normal* and *exceptional*. A normal call returns a *value* (which could be a list). An exceptional return also returns a value, usually indicating some error condition. An exceptional return of a value e is called *raising exception* e. A return is issued only in response to a call. There may be "syntactic" restrictions on the types of arguments and return values.

A component may contain multiple *processes* that can concurrently issue procedure calls. More precisely, after one process issues a call, other processes can issue calls to the same component before the component issues a return from the first call. A return action communicates to the calling component the identity of the process that issued the corresponding call.

2 A Memory Component

The component to be specified is a memory that maintains the contents of a set $\mathsf{MemLocs}$ of locations. The contents of a location is an element of a set $\mathsf{MemVals}$. This component has two procedures, described informally below. Note that being an element of $\mathsf{MemLocs}$ or $\mathsf{MemVals}$ is a "semantic" restriction, and cannot be imposed solely by syntactic restrictions on the types of arguments.

Name	Read
Arguments	loc : an element of MemLocs
Return Value	an element of MemVals
Exceptions	BadArg : argument loc is not an element of MemLocs.
	MemFailure : the memory cannot be read.
Description	Returns the value stored in address loc.

Name	Write
Arguments	loc : an element of MemLocs
	val : an element of MemVals
Return Value	some fixed value
Exceptions	BadArg : argument loc is not an element of MemLocs, or argument val is not an element of MemVals.
	MemFailure : the write *might* not have succeeded.
Description	Stores the value val in address loc.

The memory must eventually issue a return for every Read and Write call.

Define an *operation* to consist of a procedure call and the corresponding return. The operation is said to be *successful* iff it has a normal (nonexceptional) return. The memory behaves as if it maintains an array of atomically read and written locations that initially all contain the value InitVal, such that:

- An operation that raises a BadArg exception has no effect on the memory.
- Each successful Read(l) operation performs a single atomic read to location l at some time between the call and return.
- Each successful Write(l, v) operation performs a sequence of one or more atomic writes of value v to location l at some time between the call and return.
- Each unsuccessful Write(l, v) operation performs a sequence of zero or more atomic writes of value v to location l at some time between the call and return.

A variant of the Memory Component is the Reliable Memory Component, in which no MemFailure exceptions can be raised.

Problem 1 (a) Write a formal specification of the Memory component and of the Reliable Memory component.

(b) Either prove that a Reliable Memory component is a correct implementation of a Memory component, or explain why it should not be.

(c) If your specification of the Memory component allows an implementation that does nothing but raise MemFailure exceptions, explain why this is reasonable.

3 Implementing the Memory

3.1 The RPC Component

The RPC component interfaces with two environment components, a *sender* and a *receiver*. It relays procedure calls from the sender to the receiver, and relays the return values back to the sender. Parameters of the component are a set Procs of procedure names and a mapping ArgNum, where ArgNum(p) is the number of arguments of each procedure p. The RPC component contains a single procedure:

Name	RemoteCall
Arguments	proc : name of a procedure
	args : list of arguments
Return Value	any value that can be returned by a call to proc
Exceptions	RPCFailure : the call failed
	BadCall : proc is not a valid name or args is not a syntactically correct list of arguments for proc.
	Raises any exception raised by a call to proc
Description	Calls procedure proc with arguments args

A call of RemoteCall(proc, args) causes the RPC component to do one of the following:

- Raise a BadCall exception if args is not a list of ArgNum(proc) arguments.
- Issue one call to procedure proc with arguments args, wait for the corresponding return (which the RPC component assumes will occur) and either (a) return the value (normal or exceptional) returned by that call, or (b) raise the RPCFailure exception.
- Issue no procedure call, and raise the RPCFailure exception.

The component accepts concurrent calls of RemoteCall from the sender, and can have multiple outstanding calls to the receiver.

Problem 2 Write a formal specification of the RPC component.

3.2 The Implementation

A Memory component is implemented by combining an RPC component with a Reliable Memory component as follows. A Read or Write call is forwarded to the Reliable Memory by issuing the appropriate call to the RPC component. If this call returns without raising an RPCFailure exception, the value returned is returned to the caller. (An exceptional return causes an exception to be raised.) If the call raises an RPCFailure exception, then the implementation may either reissue the call to the RPC component or raise a MemFailure exception. The RPC call can be retried arbitrarily many times because of RPCFailure exceptions, but a return from the Read or Write call must eventually be issued.

Problem 3 Write a formal specification of the implementation, and prove that it correctly implements the specification of the Memory component of Problem 1.

4 Implementing the RPC Component

4.1 A Lossy RPC

The Lossy RPC component is the same as the RPC component except for the following differences, where δ is a parameter.

- The RPCFailure exception is never raised. Instead of raising this exception, the RemoteCall procedure never returns.
- If a call to RemoteCall raises a BadCall exception, then that exception will be raised within δ seconds of the call.
- If a RemoteCall(p, a) call results in a call of procedure p, then that call of p will occur within δ seconds of the call of RemoteCall.
- If a RemoteCall(p, a) call returns other than by raising a BadCall exception, then that return will occur within δ seconds of the return from the call to procedure p.

Problem 4 Write a formal specification of the Lossy RPC component.

4.2 The RPC Implementation

The RPC component is implemented with a Lossy RPC component by passing the RemoteCall call through to the Lossy RPC, passing the return back to the caller, and raising an exception if the corresponding return has not been issued after $2\delta + \epsilon$ seconds.

Problem 5 (a) Write a formal specification of this implementation.

(b) Prove that, if every call to a procedure in Procs returns within ϵ seconds, then the implementation satisfies the specification of the RPC component in Problem 2.