APPENDIX G

Notes on Version 2.2

This book describes SR Version 2.0, which was released in August, 1992. Version 2.1 was released in March, 1993; it added tracing facilities, new utilities, an X-Windows interface, additional platforms, and performance improvements. Version 2.2—released in August, 1993—introduces dynamic operations and allows the use of operation capabilities in `in`, `receive`, and `P` statements.

This appendix describes the new language features as of August, 1993, and summarizes the additional tools. Further information on Versions 2.1 and 2.2 is included with the SR distribution.

Input from Capabilities

In earlier versions of SR, an operation capability could be used as a destination for an invocation but not as a source. This asymmetry is removed, to a reasonable extent, in Version 2.2. The remaining limitations ensure that an input statement requires at most a single exchange of messages with a different virtual machine.

An operation capability can now replace an operation name in an input statement, without regard to how the capability was created, subject to meeting *either* of the following restrictions:

- the input statement has just one arm and no `by` or `st` (but it may have an `else`), or
- the capability references an operation in the current resource.

Capabilities are also allowed in `P` and `receive` statements; these are shorthand forms of input statements meeting the first restriction.

As an example, a bag of tasks can be encapsulated in a resource by exporting an operation name that represents the bag. Processes in other resources can both contribute and withdraw tasks by using a capability for the exported operation.

The question mark operator (`?`) can be used to query the number of pending invocations available through a capability.

A `noop` capability is treated as an operation that is never invoked. Hence, input from a `noop` capability never succeeds. Querying a `noop` capability with the `?` operator always returns 0. Attempting to query or input from a `null` capability—or a capability for an operation that no longer exists—produces a runtime error.

A capability used with `P`, `receive`, or `?` can be any arbitrary expression. For syntactic reasons, a capability used in an `in` statement is limited to the form of an identifier, possibly qualified and/or subscripted.

Dynamic Operations

It is now possible to create new operations on demand. Any of the following expressions creates one instance of a new, anonymous operation and returns a capability for that operation:

```
new(optype_id )
new(op operation_specification )
new(op operation_specification operation_restriction )
new(sem)
```

The operation can be invoked or serviced until the resource or operation is destroyed.

The operation referenced by capability *ocap* can be destroyed by executing

```
destroy ocap
```

Only dynamically created operations can be destroyed. Note that in this case `destroy` and not `free` is the opposite of `new`; this is consistent with the destruction of resources using capabilities. A subsequent invocation, query, input, or destroy of a destroyed operation produces a runtime error.

Arrays of operations can be created using any of the forms

```
new([subscripts ] optype_id )
new([subscripts ] op operation_specification )
new([subscripts ] op operation_specification operation_restriction )
new([subscripts ] sem)
```

The array of operations referenced by capability *ocap_array* can be destroyed by executing

```
destroy ocap_array
```

Elements of an array can also be destroyed individually; destruction of any element renders wholesale destruction of the array illegal, because that would try to destroy the individual element a second time.

Additional Tools

The SRWin library provides an interface to the X Window System and makes it easy to create simple graphical applications in SR. A wide range of drawing operations is provided. The concurrency inherent in SR makes it especially easy for the programmer to handle window events; refresh events are handled automatically, so they require no attention at all. A technical report describing SRWin [Zhao 1993] is included in the distribution.

SR now provides support for tracing and profiling program execution. Setting the environment variable SR_TRACE to a file name outputs to that file a record of all invocations and other concurrency-related events. This can be used to aid in debugging. The file can be also be processed by the new **srprof** utility to produce an execution profile.

The SR compiler **sr** and linker **srl** have been extended to support the use of libraries of resources and globals. The new command **srlatex** formats SR programs with LaTeX.