APPENDIX A

SLOOP SYNTAX QUICK REFERENCE

A.1 Notational conventions

The SLOOP syntax is given below in BNF. The significance of the typeface and symbols used in the BNF description is as follows:

Plain or boldface type: Terminal symbols
Italics: Non-terminal symbols
Braces: Grouping
Square brackets: Enclosed syntactical unit is optional
Asterisk: Zero or more occurrences of the syntactical unit
Plus: One or more occurrences of the syntactical unit
Vertical bar: Separates options
Single quotes: Enclose literals.

All names, such as program-name, class-name, instance-name and category-name, are strings that may contain letters, digits and the underscore character. They all start with a letter. If the plural form is used (for example as in instance-variable-names), then one or more name(s) may be present, each separated by a white-space character. Three consecutive colons separate a class-name from a package-name when it is necessary to qualify the class-name by a package-name.

A.2 The SLOOP program structure

A SLOOP-program has the following structure:

\[
\text{SLOOP-program} \rightarrow \text{program program-name activation-section } \{\text{package-description}\}^{+} \text{end-program}
\]

\[
\text{activation-section} \rightarrow \text{sequential statement-list end-sequential parallel statement-list end-parallel}
\]
A.3 Complete and partial class descriptions

Syntax of a class-description:

class-description →
  class class-name
  superclass superclass-name [from repository-name]
  [class variable names [class-variable-names]]
  [instance variable names [instance-variable-names]]
  [class macros [macros-section]]
  class properties [properties-section]
  methods-section

Syntax of a partial-class-description:

partial-class-description →
  class class-name from repository-name
  partial-class-methods-section

The class properties keywords are mandatory in order to indicate to the designer that all the relevant properties should always be listed. If this section had been optional, then it would have been unclear to the person reusing the class whether the original designer had merely chosen not to list the properties or whether there had been no relevant properties to list.

A.4 The macros-section

Syntax of a macros-section:

macros-section →  macro-list
  macro-list →  macro-definition
  macro-definition →  macro-variable = macro-expression
  macro-variable →  variable-name
  macro-expression →  simple-macro-expression | conditional-macro-expression
  simple-macro-expression →  message-expression | variable-name | literal
  conditional-macro-expression →  simple-macro-expression
    if boolean-expression
    {~ simple-macro-expression
    if boolean-expression}*

A message-expression is a Smalltalk-style message expression and a boolean-expression is a Smalltalk-style message expression that returns true or false. A Smalltalk-style message expression consists of a receiver, a selector and zero or more arguments. If there are no arguments, the message is called a unary message. For example, bufferedElements size is
a unary message expression. A binary message has a single argument following a selector consisting of one or two non-alphanumeric characters, the second of which may not be a minus sign. The message expression \( a + b \) is an example of a binary message expression. The third type of message is a keyword message. The selector consists of one or more keywords, each with its associated argument. A keyword consists of an identifier followed by a colon. For example, `bufferedElements addLast: newElement` is a keyword message expression.

A literal is a Smalltalk-style literal which may be a number, a symbol constant, a character constant, a string or an array constant.

Note that, as in Smalltalk-80, message expressions may be nested. The receiver of a message expression may itself be a message expression. Similarly, the argument(s) of a keyword message may also be message expression(s).

A message expression may also contain a block. The reader is referred to [GoRo89] for a detailed discussion of a Smalltalk-80 block. For the purposes of its application in SLOOP, the following description suffices.

A block represents a deferred sequence of actions. It consists of a sequence of expressions separated by periods and delimited by square brackets. The actions represented by a block are not necessarily executed when the block expression is encountered. For example, a block expression is used as the argument of the Smalltalk-80 `and:` keyword message. This message represents the logical "and" operation. If the first operand (i.e. the receiver) evaluates to true, the second operand (the argument of `and:`) is evaluated and its value is returned as result. However, if the first operand evaluates to false, the second operand is not evaluated. This is indicated syntactically via the fact that the argument of the `and:` message is a block expression.

A block is also used when the receiver of a message is a collection and the actions represented by the block need to be applied to each element of the receiver. In that case each element of the receiver is passed to the block as an argument. This is indicated syntactically by the presence of an identifier preceded by a colon at the beginning of the block. This identifier is separated from the rest of the contents of the block by a vertical bar.

For example, the Smalltalk-80 library `select:` and `detect:` messages are used frequently in the CallCentreSimulation example. The `select:` message evaluates the block received as argument of the message for each of the receiver's elements (the receiver is a collection object). It returns a collection that contains only those elements of the receiver for which the block evaluates to true. The message expression below returns the collection representing all employees earning a salary greater than $20,000:

```smalltalk
employees select: [:each | each salary > 20000]
```

The following message expression returns the object representing the first employee found earning a salary greater than $20,000 (if no such employee is found, then `nil` is returned):

```smalltalk
employees detect: [:each | each salary > 20000] ifNone: [nil]
```

For a formal description of the Smalltalk-80 syntax, the reader is referred to [GoRo89].

### A.5 The properties-section

Each class, as well as each method within a class may contain a `properties-section`. The syntax is as follows:
A property may be of the form:
\[ p \text{ unless } q, \]
stable \( p \),
invariant \( p \),
\( p \) ensures \( q \),
\( p \) leads-to \( q \),
\( p \) until \( q \),
\( p \) detects \( q \),
\( p \) precedes \( q \) or
\( p \) results-in \( q \),
where \( p \) and \( q \) are first-order predicates.

Since first order predicate logic is used, universal and existential quantification is allowed in the logical relations. The keywords for all and exists may be used as alternatives to the \( \forall \) (universal quantification) and \( \exists \) (existential quantification) symbols respectively. Instead of using a colon to denote the domain of a quantification, the reserved word where is used. This is to avoid confusion with the colon used in Smalltalk keyword expressions.

If a variable-list is used in a quantification, the variables are separated by a comma preceded by a backslash. This ensures that the comma cannot be mistaken for the Smalltalk concatenation symbol. If a '<' symbol is followed immediately by a quantification symbol, it denotes the start of a quantification construct. If a '>' symbol appears as the first non-white-space character on a line, it denotes the end of a quantification construct. The quantification constructs have the same parsing precedence as parentheses.

Smalltalk-style message expressions may be used in the predicates. However, these message expressions may not have any side-effects, i.e. they may not change the state of any object.

The \( \land \) (logical and), \( \lor \) (logical or) and \( \neg \) (negation) operators are defined in addition to the Smalltalk \& (logical and), | (logical or) and not (negation) operators. The additional logical operators serve a readability purpose only. One difference between the additional logical operators and the Smalltalk ones is in the parsing precedence. The Smalltalk unary, binary and keyword expressions are evaluated in that order. The additional logical operators are evaluated after the unary, binary and keyword expressions have been evaluated.

Further conventions about the priorities of logical relations are given next (those on the same line have equal priority and the lines represent the priorities from high to low):
\[ \neg \]
\[ \Rightarrow \]
\[ \equiv \]
\textit{unless, ensures, leads-to, stable, invariant, detects, until, precedes}

Properties are universally quantified over all the free variables occurring in them.
A.6 The methods-section

The syntax of the methods-section of a class is as follows:

\[
\begin{align*}
\text{methods-section} & \rightarrow [\text{class methods} \\
& \{\text{methods-implementation}\}+] \\
& [\text{instance methods} \\
& \{\text{methods-implementation}\}+] \\
\text{methods-implementation} & \rightarrow \text{category category-name} \\
& \{\text{method-description}\}+ \\
\text{method-description} & \rightarrow \text{sequential-method} | \text{parallel-method} \\
\text{sequential-method} & \rightarrow \text{message pattern Smalltalk-message-pattern} \\
& [\text{method macros macros-section}] \\
& \text{method properties properties-section} \\
& \text{sequential} \\
& \text{statement-list} \\
& \text{end-sequential} \\
\text{parallel-method} & \rightarrow \text{message pattern p_Smalltalk-message-pattern} \\
& [\text{method macros macros-section}] \\
& \text{method properties properties-section} \\
& \text{parallel} \\
& \text{statement-list} \\
& \text{end-parallel} \\
\text{partial-class-methods-section} & \rightarrow [\text{class methods} \\
& \{\text{selector}\}+] \\
& [\text{instance methods} \\
& \{\text{selector}\}+] \\
\text{selector} & \rightarrow \text{Smalltalk-selector} | \\
& \text{p_Smalltalk-selector} \\
\end{align*}
\]

A Smalltalk-message-pattern has the usual Smalltalk syntax, i.e. it comprises the message selector with the associated pseudo-variables to represent the arguments if there are any. A Smalltalk-selector has the usual syntax of a selector in a Smalltalk program.

Similarly to the properties of the class, all the relevant properties of the method should be listed if there are any. If this section had been optional, then it would have been unclear to the person reusing the class whether the original designer had merely chosen not to list the properties or whether there had been no relevant properties to list.

A.7 SLOOP statements

The SLOOP statement-list is defined as follows:

\[
\begin{align*}
\text{statement-list} & \rightarrow \text{statement}^1 \\
& \{[\text{statement}]\}^* \\
\text{statement} & \rightarrow \text{simple-statement} | \\
& \text{quantified-statement-list} \\
\text{quantified-statement-list} & \rightarrow <[\text{quantification statement-list}> \\
\end{align*}
\]

\[^1\text{This implies that, as in UNITY, a statement-list cannot be empty.}\]
quantification → variable-list where boolean-expr ::
variable-list → variable {\, variable}* 
simple-statement → statement-component
                { || statement-component }*
statement-component → enumerated-component
                | quantified-component
enumerated-component → component-part
                | conditional-component-part-list
quantified-component → \{\| quantification simple-statement\}
component-part → \{[^\}]variable := simple-expr\}^2 |
                ^message-expression
conditional-component-part-list → simple-component-part-list if boolean-expr
                {~ simple-component-part-list if boolean-expr }*
simple-component-part-list → component-part
                {+component-part}* 
simple-expr → message-expression | primary

A message-expression is a Smalltalk-style message expression and a boolean-expr is a Smalltalk-style message expression that returns true or false. A Smalltalk-style message expression consists of a receiver, a selector and zero or more arguments. If there are no arguments, the message is called a unary message. A binary message has a single argument following a selector consisting of one or two non-alphanumeric characters, the second of which may not be a minus sign. The third type of message is a keyword message. The selector consists of one or more keywords, each with its associated argument. A keyword consists of an identifier followed by a colon.

A primary is a Smalltalk-style primary which may be a variable name, a literal or a block. When a "<" symbol is immediately followed by the ""]" or "||" symbol, it denotes a quantification and the "<" symbol is not interpreted as a Smalltalk operator. If a ">" symbol appears as the first non-white-space character on a line, it denotes the end of a quantification construct.

No messages related to the Smalltalk-80 support for multiple processes may be used, since there is no concept of a process in a SLOOP program.

Cascaded message expressions are also not allowed. The motivation for this restriction was given in Chapter 4, Section 4.3.6.2.

A.8 Comments in a SLOOP program

Comments may be inserted anywhere in the program and are enclosed by double quotes.

2 The braces around the [\^]variable := simple-expr construct serve to identify the latter as a syntactic unit. This is needed because the '^' symbol has a higher precedence than the ':=' symbol.
APPENDIX B

A SLOOP PROGRAM FOR A CALL CENTRE

B.1 Scope of the first level of refinement of the design

The first level of refinement is only concerned with normal behaviour; no error conditions are specified. This implies that a service user is always served once it is connected to the system; the possibility of aborting the connection (hanging up before the service has been completed) or rejecting the service request (e.g. due to unavailability of the relevant service providers) is not specified at this stage.

This is a cyclic system, which should therefore not terminate. However, there may be conditions under which the system may need to be shut down. System shutdown is not shown at this level of refinement.

The service providers may only be in the 'BUSY' or 'IDLE' states. At this level of refinement there is no state instance variable. The 'BUSY' and 'IDLE' status of a service provider is determined by checking whether it can accept the next service request or not. The 'RESTRICTED-IDLE', 'RESTRICTED-BUSY' and 'UNAVAILABLE' states are introduced during subsequent refinements. The additional states enable a service provider to go out of service gracefully and it ensures that once a service request is allocated to a service queue that it will be serviced.

(When a service provider is operating in a restricted mode, it means that it will continue to remain active until all the service requests already present in the service queue at the time when it changes to the restricted mode have been serviced. This indicates to the scAllocator that no further service requests should be accepted for a specific service queue if all the service providers servicing that particular category are operating in a restricted fashion.)

At this level of refinement the service user is not informed of the progress of the service request.

An object diagram of the Call Centre system is shown in Figure B-1. Figure B-2 illustrates the contents of the various packages in the system.
Figure B-1. Call centre object diagram.
The property numbers are defined uniquely with respect to the class in which they appear. These numbers may appear in any order as long as they are unique. These numbers are independent of the numbers of the superclass properties. If a property overrides a property specified in an ancestor, the number of the ancestor property is used in the subclass followed by the name of the ancestor in brackets. When referring to a property that overrides a property in its superclass, the property identifier as described above must be followed by the words in class-name, where class-name is the name of the subclass.

In order to demonstrate that the concept of a package has no significance other than organizational, the classes in this Appendix are presented as individual classes and in any order.
B.2 The CC_Activation class

The CC_Activation class is an abstract superclass which leaves the activation of the interface classes up to the subclasses. It activates all the other classes in the system. The CC_SimulationActivation class overrides the methods related to the interface classes. The analysis level properties are not repeated here. They can be found in Chapter 5, Section 5.4.

class CC_Activation
"The CC_Activation instance instantiates all the classes of the system that need to present before the parallel methods start executing. It also activates all the parallel methods required by the system."

subclass SmalltalkLibPkg::Object from SmalltalkLibRepository

instance variable names
"The following variables represent the objects that are not instantiated as elements of a collection, i.e. they are not indexed instance variables."
config
"All configurable values (e.g. the maximum number of connections supported) are obtained via the config object."
commsAgent
"The commsAgent handles the interface with the communication provider."
userConnections
"This variable represents the collection of connections supported by the system."
inputQ
"The inputQ models the FIFO way in which service requests are accepted by the system."
scAllocator
"The scAllocator categorises the service requests and allocates them to the appropriate service queues."
scContainer
"This variable represents the collection of service categories supported by the system."
spAgentContainer
"The spAgentContainer contains all the service providers supported by the system."
timer
"The timer object provides timer services to the other objects in the system."
timerEventQ
"The timerEventQ is used to inform the requestors of the various timeouts that a requested timeout has occurred."

class macros
maxConn = config maximumConnections
"Number of simultaneous user connections supported"
[] maxCategories = config maximumServiceCategories
"Number of service categories supported"
[] maxSP = config maximumServiceProviders
"Number of service providers supported"

class properties
"These are the properties as identified during the analysis phase, as well as the following design level clean behaviour invariants:"
invariant
config notNil ^
commsAgent notNil ^
userConnections notNil ^
∀ i \text{ where } 1 \leq i \leq \text{maxConn} :: (\text{userConnections at: } i) \text{ notNil} > ^ \∧
\text{inputQ notNil} ^
\text{scAllocator notNil} ^
\text{scContainer notNil} ^
∀ j \text{ where } 1 \leq j \leq \text{maxCategories} :: (\text{scContainer at: } j) \text{ notNil} > ^ \∧
\text{spAgentContainer notNil} ^
∀ k \text{ where } 1 \leq k \leq \text{maxSP} :: (\text{spAgentContainer at: } k) \text{ notNil} > ^ \∧
\text{timer notNil} ^
\text{timerEventQ notNil} ^
\text{maxConn} > 0 ^
\text{maxCategories} > 0 ^
\text{maxSP} > 0 ^
"DS2-01"
"Clean behaviour"
"The CC Activation class is an abstract class and should not be instantiated"
∀ \text{anObject} :: \text{invariant} \text{anObject class} \sim CC\text{Activation} > ^"DS2-02"

instance methods

category private

message pattern initialize

method properties
"Total correctness"

true \text{ results-in}

methodReturnValue = self ^
\text{config notNil} ^
\text{self postconditions: (#initManagement) ^}
\text{commsAgent notNil} ^
\text{self postconditions: (#initCommsAgent) ^}
\text{userConnections notNil} ^
∀ i \text{ where } 1 \leq i \leq \text{maxConn} :: (\text{userConnections at: } i) \text{ notNil} > ^ \∧
\text{inputQ notNil} ^
\text{scAllocator notNil} ^
\text{self postconditions: (#initServiceCategoryAllocator) ^}
\text{scContainer notNil} ^
∀ j \text{ where } 1 \leq j \leq \text{maxCategories} :: (\text{scContainer at: } j) \text{ notNil} > ^ \∧
\text{spAgentContainer notNil} ^
∀ k \text{ where } 1 \leq k \leq \text{maxSP} :: (\text{spAgentContainer at: } k) \text{ notNil} > ^ \∧
\text{timer notNil} ^
\text{(timer class) postconditions: (#setup:) ^}
\text{withArguments: #(config)} ^
\text{timerEventQ notNil} ^
"DL1-02"

"Note that the receiver of the postconditions:withArguments:
message is the expression (timer class) instead of
SystemUtilitiesPkg::TimerServices. This is done to facilitate
subclassing without violating the correctness properties. If
the actual class name had been used here, then the property
would no longer have been valid if a subclass of TimerServices
had been instantiated at this point. Recall that correctness properties must be preserved during subclassing.

"At this stage (i.e. before the subclass has completed the execution of its instance creation method) the class invariants do not need to hold yet, so it should be stated explicitly that once the predicate self postconditions: (#initManagement) holds, it continues to hold. That is a requirement, since many of the subsequent statements in the method depend on it. Similarly for the other stable properties listed below."

**stable** config notNil ∧ self postconditions: (#initManagement)  

"Note that self postconditions: (#initManagement) implies that:
maxConn > 0 ∧
maxCategories > 0 ∧
maxSP > 0 "

**stable** userConnections notNil  
**stable** scContainer  
**stable** spAgentContainer notNil

**sequential**

config := self initManagement
commsAgent := self initCommsAgent
userConnections := SmalltalkLibPkg::Array new: maxConn
< [] i where 1≤i≤maxConn :: userConnections at: i
  put: (self initConnection: i)
>
inputQ := SmalltalkLibPkg::OrderedCollection new: maxConn
scAllocator := self initServiceCategoryAllocator
scContainer := SmalltalkLibPkg::Array new: maxCategories
< [] j where 1≤j≤maxCategories :: scContainer at: j
  put: (CC_CorePkg::ServiceCategory setup: config)
>
spAgentContainer := SmalltalkLibPkg::Array new: maxSP
< [] k where 1≤k≤maxSP :: spAgentContainer at: k
  put: (self initSPAgent)
>
timer := SystemUtilitiesPkg::TimerServices setup: config
timerEventQ := SmalltalkLibPkg::OrderedCollection new

**end-sequential**

**message pattern** initManagement

**method properties**

"Total correctness"

true **results-in**

(methodReturnValue notNil ∧ (methodReturnValue class) postconditions: (#setup))  

"Again the explicit reference to a class name (in this case CC_CorePkg::Configuration) is avoided in order to ensure that subclasses do not violate the correctness property."
message pattern initConnection: index
method properties
"Total correctness"
true results-in
  methodReturnValue notNil ∧
  (methodReturnValue (methodReturnVal class) postconditions: (#setup:))
withArguments: #\{index\}
"DL1-04"
"Again the explicit reference to a class name (in this case
CC_CorePkg:::Connection) is avoided."
sequential
  ^CC_CorePkg:::Connection setup: index
end-sequential

message pattern initCommsAgent
method properties
"Total correctness"
true results-in methodReturnValue notNil
sequential
  self subclassResponsibility
end-sequential

message pattern initSPAger
method properties
"Total correctness"
true results-in methodReturnValue notNil
sequential
  self subclassResponsibility
end-sequential

message pattern initServiceCategoryAllocator
method properties
"Total correctness"
true results-in methodReturnValue notNil
sequential
  ^CC_CorePkg:::ServiceCategoryAllocator setup
end-sequential

category cyclic
message pattern p_activate
method properties
"These are the properties as identified during the analysis phase."
"The p_activate method is only invoked once the CC_Activation
subclass has been instantiated. The class invariants of the
CC_Activation subclass that has been instantiated are therefore
guaranteed to hold before the p_activate method is executed.
Each statement executed by the p_activate method has to preserve
these invariants."

parallel
  self p_executeCPAgent
"The parallel methods of the commsAgent are not invoked
directly, but rather via the p_executeCPAgent method of the
CC_Activation class."

[] timer p_runTimer: timerEventQ
"Activate the parallel methods of the timer object. The timer
parallel statements have the following functionality: Whenever a
timeout occurs, the TimeoutElement instance representing the
timeout is added to the end of the timerEventQ, which indicates
to the requestor that the specified timer has expired."
The parallel methods of the scAllocator object are invoked via the p_categoriseAndAllocate method of the CC_Activation class. The scAllocator parallel statements have the following functionality: Once a service request has been categorised, it is removed from the inputQ and appended to the appropriate serviceQ.

n < () j where 1≤ j ≤ maxCategories :: (scContainer at: j)
   p_execute

"Activate the parallel methods of the ServiceCategory instances. Their parallel statements have the following functionality: For each service category the associated service queue and set of service provider agents are monitored. If the service queue is not empty and a service provider agent in the spSubset associated with the service category is available to process a new service request, the first element of the service queue is removed and assigned to a service provider agent."

n < () i where 1≤ i ≤ maxConn :: self p_executeConnection:
   (userConnections at: i)

"The p_executeConnection method of the CC_Activation class is executed for each Connection instance in order to invoke the parallel methods of the latter. The parallel statements of the Connection instances have the following functionality: When a connection has entered the 'TERMINATING' state, the communication provider agent is requested to terminate the connection. Once all the procedures have been completed to terminate the connection, the connection and its associated service request are reset to their initial states."

n < () k where 1≤ k ≤ maxSP :: self p_executeSPAgent:
   (spAgentContainer at: k)

"The parallel methods of the service provider agents are not invoked directly, but rather by executing the p_executeSPAgent method of the CC_Activation class for each of the service provider agents."

message pattern p_executeCPAgent
method properties
"These are the properties pertaining to the communication provider interface as identified during the analysis phase."
parallel
self subclassResponsibility
end-parallel

message pattern p_executeSPAgent: spAgent
method properties
"These are the properties pertaining to the service provider interface as identified during the analysis phase."
parallel
self subclassResponsibility
end-parallel

message pattern p_categoriseAndAllocate
method properties
"These are the properties pertaining to the service category allocator as identified during the analysis phase."
parallel
scAllocator p.categorise: inputQ using: scContainer
"The scAllocator monitors the inputQ. If it is not empty, it
enables the categorisation of the first element (a service
request)."
[] scAllocator p.allocate: scContainer from: inputQ
"Once the service request has been categorised, the scAllocator
removes it from the inputQ and appends it to the appropriate
serviceQ."
end-parallel

message pattern p_executeConnection: aConnection
method properties
"These are the properties pertaining to the Connection class as
identified during the analysis phase."
parallel
aConnection p_informCommsProvider: commsAgent
"When a connection has entered the 'TERMINATING' state, the
communication provider agent is requested to terminate the
connection."
[] aConnection p_doWrapUp
"Once all the procedures have been completed to terminate a
connection, the connection and its associated service request
are reset to their initial states."
end-parallel
B.3 The CC_SimulationActivation class

class CC_SimulationActivation
superclass CC_Activation
class properties


class methods
category instance creation
message pattern setup
method properties
"Total correctness: After the statements in the initialize method have been executed the clean behaviour invariant of the CC_Activation class will hold, i.e. all the objects that should be created upon start-up will exist. This implies that the invariants of the classes instantiated by the CC Activation class will also hold, as well as the correctness properties of their respective creation methods. All classes have to preserve their respective class invariants after initialisation."

\[
\forall k \text{ where } k \geq 0 \quad \text{self instanceCount} = k \quad \implies \text{self instanceCount} = k + 1 \land \text{methodReturnValue notNil}
\]

sequential
"DL1-01"

end-sequential

instance methods
category private
message pattern initCommsAgent
method properties
"Total correctness"
true results-in

\[
\forall j \text{ where } 1 \leq j \leq \text{maxCategories} \quad \text{(scContainer at: j) notNil} \quad \implies \text{methodReturnValue notNil} \land \text{CC_SimulationInterfacesPkg:::CommsProviderSimulator postconditions: (#startSimulation)}
\]

"DL1-05 (CC_Activation)"

end-sequential

message pattern initSPAgent
method properties
"Total correctness"
"The references to scContainer in the precondition of this method are required, because during initialization the ServiceProviderSimulator instance registers itself with each ServiceCategory instance that is serviced by a service provider category matching that of the current ServiceProviderSimulator instance."

\[
\forall j \text{ where } 1 \leq j \leq \text{maxCategories} \quad \text{(scContainer at: j) notNil} \land \text{config notNil} \land \text{self postconditions: (#initManagement) \land scContainer notNil} \land \text{methodReturnValue notNil} \land \text{CC_SimulationInterfacesPkg:::ServiceProviderSimulator postconditions: (#startSimulation:using:) withArguments: #(scContainer config)}
\]

"DL1-6 (CC_Activation)"

376
sequential
^CC_SimulationInterfacesPkg:::ServiceProviderSimulator
  startSimulation: scContainer using: config
end-sequential

category cyclic
message pattern p_executeCPAgent
method properties
"These are the properties pertaining to the communication
provider interface as identified during the analysis phase."
parallel
commsAgent p_simulate: timer timeoutEventsIn: timerEventQ
[] commsAgent p_generateEvent: userConnections target: inputQ
"The commsAgent simulates the establishment of new connections
at random intervals (within a configured range). A simulation
timer is started after initialization and restarted each time
after the establishment of a connection has been simulated. The
latter is done by placing the service request associated with
the new connection into the input queue. The commsAgent ensures
that the capacity of maxConn connections per call centre is not
exceeded, therefore a message is displayed indicating that all
connections are busy if the maximum number of connections are
currently assigned."
end-parallel

message pattern p_executeSPAgent: spAgent
method properties
"These are the properties pertaining to the service provider
interface as identified during the analysis phase."
parallel
spAgent p_simulate: timer timeoutEventsIn: timerEventQ
"When a service request has been assigned to a service provider
simulator, the latter simulates the time it takes to service the
service request by starting a random timer. When this timer
expires, it represents the completion of the service."
[] spAgent p_generateEvent
"When the service provider has completed the service, it
indicates that the connection should be terminated."
[] spAgent p_updateCategoryIndex: scContainer
"Update the index into the categoriesServed collection if the
serviceQ of the current category being served by this spAgent is
empty."
end-parallel
B.4 The Configuration class

class Configuration
"The purpose of this class is to ensure that the following parameters are configured:

Uses defaults to set the maximum number of connections, service categories and service providers. It also facilitates the configuration of the maximum allowable timeout value, the service request category names supported by the system, the service provider category names supported by the system and the mapping of service request to service provider category names. Subclasses may allow the operator to specify other values."

superclass Object

instance variable names
maximumConnections
"The maximum number of connections supported by the system."
maximumServiceCategories
"The number of service categories supported by the system."
maximumServiceProviders
"The number of service providers supported by the system."
maximumAllowableTimeout
"The maximum allowable timeout that may be requested by any object in the system."
srCategoryNames
"The collection of service request category names supported by the system."
spCategoryNames
"The collection of service provider category names supported by the system."
srToSpCategoryMap
"The mapping of service request categories to service provider categories."
categoriesAssigned
"This variable is used to keep track of the number of service request category names that have already been assigned. When a ServiceCategory instance is created and initialized, it obtains the name of the service category that it supports from the Configuration instance (via the assignSRCategory method). Each service request category name may only be assigned once (in order to ensure that each ServiceCategory instance will support a unique service request category)."

class properties
<∀ ( t, u, v, w) where
\( t > 0 \land u > 0 \land v > 0 \land w > 0 \) ::

invariant
  maximumConnections = t ∧
  maximumServiceCategories = u ∧
  maximumServiceProviders = v ∧
  maximumAllowableTimeout = w
> "DS2-01"
"The values of each of the maximumConnections, maximumServiceCategories, maximumServiceProviders and maximumAllowableTimeout instance variables are invariant and always greater than zero."
\[\forall u \text{ where } u > 0 ::
\]
\[
\text{invariant}
\]
\[\text{srCategoryNames notNil } \land \text{srCategoryNames size } = u
\]
"DS3-01"

"The number of service request category names that are configured is equal to maximumServiceCategories."

\[\forall \text{ (anSRCategoryNameX, anSRCategoryNameY)} \text{ where srCategoryNames includes: anSRCategoryNameX } \land \text{srCategoryNames includes: anSRCategoryNameY :: invariant}
\]
\[\text{anSRCategoryNameX } \not\sim \text{ anSRCategoryNameY}
\]
"DS3-02"

"Each configured service request category name is unique."

\[
\text{invariant}
\]
\[\text{spCategoryNames notNil } \land \text{~spCategoryNames isEmpty}
\]
"DS3-03"

"At least one service provider category name is configured."

\[\forall \text{ (anSPCategoryNameX, anSPCategoryNameY)} \text{ where spCategoryNames includes: anSPCategoryNameX } \land \text{spCategoryNames includes: anSPCategoryNameY :: invariant}
\]
\[\text{anSPCategoryNameX } \not\sim \text{ anSPCategoryNameY}
\]
"DS3-04"

class methods

category instance creation
message pattern setup
method properties
"A Configuration instance is created and initialized. The new instance is returned"

"Total correctness"
\[
\forall k \text{ where } k \geq 0 ::
\]
\[\text{self instanceCount } = k \text{ results-in}
\]
\[\text{self instanceCount } = k + 1 \land \text{methodReturnValue notNil}
\]
"DL1-01"

^super new configure
dend-sequential

instance methods

category private
message pattern configure
method properties
"Upon completion of the configure method, the maximumConnections, maximumServiceCategories, maximumService=Providers and maximumAllowableTimeout instance variables will each have a value greater than zero, the srCategoryNames and spCategoryNames collections will have been created, the number of elements in the srCategoryNames collection will be equal to maximumServiceCategories, there will be at least one element in the spCategoryNames collection, the srToSpCategoryMap will have been created, the number of mappings in this collection will be equal to maximumServiceCategories and the categoriesAssigned variable will have the value zero."
"Total correctness"

\[ \forall (t, u, v, w) \text{ where } t > 0 \land u > 0 \land v > 0 \land w > 0 :: \]
true results-in
maximumConnections = t \land
maximumServiceCategories = u \land
maximumServiceProviders = v \land
maximumAllowableTimeout = w \land
srCategoryNames notNil \land spCategoryNames notNil \land
srCategoryNames size = u \land
spCategoryNames isEmpty \land
srToSpCategoryMap notNil \land
srToSpCategoryMap size = u \land
categoriesAssigned = 0

> "DL1-02"

sequential
[] maximumConnections := 8
  "Maximum number of simultaneous user connections"
[] maximumServiceCategories := 1
  "Maximum number of service categories"
[] maximumServiceProviders := 3
  "Maximum number of service providers"
[] maximumAllowableTimeout := 5
  "Maximum allowable timeout"
[] srCategoryNames := SmalltalkLibPkg:::OrderedCollection
  new: maximumServiceCategories
[] srCategoryNames addLast: 'Default Service Request category'
[] spCategoryNames := SmalltalkLibPkg:::OrderedCollection
  new: maximumServiceProviders
"Multiple service providers may belong to the same service
provider category, but it is also possible that each service
provider could belong to a different service provider category.
The maximum size is therefore used when spCategoryNames is
created."
[] spCategoryNames addLast: 'Default Service Provider category'
[] srToSpCategoryMap = Dictionary new
[] srToSpCategoryMap at: 'Default Service Request category'
  put: spCategoryNames
[] categoriesAssigned := 0
end-sequential

category accessing

message pattern maximumConnections
method properties
"Total correctness"
true results-in methodReturnValue = maximumConnections "DL1-03"
sequential
"maximumConnections
end-sequential

message pattern maximumServiceCategories
method properties
"Total correctness"
true results-in methodReturnValue = maximumServiceCategories
sequential
"maximumServiceCategories
end-sequential
message pattern maximumServiceProviders
method properties
"Total correctness"
true results-in methodReturnValue = maximumServiceProviders
"DL1-05"

  sequential
  ^maximumServiceProviders
  end-sequential

message pattern maximumAllowableTimeout
method properties
"Total correctness"
true results-in methodReturnValue = maximumAllowableTimeout
"DL1-06"

  sequential
  ^maximumAllowableTimeout
  end-sequential

message pattern getSPCategories: srCategory
method properties
"The srCategory passed as a parameter is used as the index into the srToSpCategoryMap object in order to extract the collection of Service Provider Categories associated with the srCategory."
"Total correctness"
true results-in
methodReturnValue = srToSpCategoryMap at: srCategory
"DL1-07"

  sequential
  ^srToSpCategoryMap at: srCategory
  end-sequential

category modifying

message pattern assignSRCategory
method properties
"Returns a unique service request category (each service request category is only assigned once)"
"Total correctness"
\forall x \text{ where } 0 \leq x < \text{maximumServiceCategories} ::
categoriesAssigned = x results-in
  categoriesAssigned = x + 1 \land
  methodReturnValue = srCategoryNames at: (x + 1) \text{ at: categoriesAssigned}
"DL1-08"

  sequential
  categoriesAssigned := categoriesAssigned + 1
  [\text{srCategoryNames at: categoriesAssigned}]
  end-sequential

message pattern assignSPCategory
method properties
"Returns a service provider category (each service provider category may be assigned multiple times). Subclasses may use various algorithms to assign service provider categories"
"Total correctness"

$$\forall x \ where \ spCategoryNames \ includes: \ x ::
true \ results-in
   \ methodReturnValue = x
>$$

"DL1-09"

sequential

^spCategoryNames \ first

end-sequential
B.5 The EventSimulator class

The EventSimulator class is an abstract class. It is responsible for starting a timer if one is required. It also detects the expiry of the timer. The subclasses of EventSimulator are responsible for determining when a timer is required and also for generating the events resulting from the expiry of the timers.

class EventSimulator
superclass Object from SmalltalkLibRepository
instance variable names
rand
"This variable refers to an instance of the Random class from the Smalltalk library. The instance is created when the EventSimulator subclass is instantiated. The instance of the Random class maintains a seed from which the next random number is generated. The random number is used to start a timer with a random value."
newEventRequired
"When the value is equal to true it means that a new event is required. Once the variable has been set to true, a random timer will be started at some point afterwards. When the timer is started, newEventRequired is set to false. It is the responsibility of the subclass to set this variable to true when a new event is required, since each subclass will have its own conditions for requiring a new event. Once the timer expires, an event will be generated, as will be described in the comments section of the generatingEvent variable."
currentRandomTimeoutValue
"This variable contains the value of the random timeout currently being requested. The purpose of this variable is to provide a mechanism for referencing the current timeout value in the correctness arguments. Note that the SLOOP statements could therefore have been rewritten without this variable while still providing the same functionality. However, in that case it would not have been possible to formalise certain correctness properties (such as DL1-04)."
generatingEvent
"The value is equal to true if the timer has expired and an event has to be generated, otherwise it is equal to false. The subclass sets this variable to false at the time when the event is generated. The actual event that is generated is also the responsibility of the subclass, since each subclass will generate a different type of event."
timerOutstanding
"This variable is set to true when a timer is started and it is set to false when a timeoutElement is removed from the timerEventQ (i.e. when an expired timer has been processed). The purpose of this variable is to provide a mechanism for reasoning about the uniqueness of outstanding timers in the EventSimulator class. In this class only one timer requested by the EventSimulator may be outstanding at a time. The timerOutstanding variable is used in the preconditions of the startRandomTimer:withMaximum: method as well as in the postconditions of the resetTimerExpired: method. If subclasses need to support multiple simultaneous timers, then the preconditions of the startRandomTimer:withMaximum: method need to be weakened and the postconditions of the resetTimerExpired: method need to be strengthened. Since the purpose of the timerOutstanding variable is to facilitate correctness reasoning, the SLOOP statements could have been rewritten without this variable while still providing the same functionality."
timerId
"This variable contains the identifier of the timer currently
being requested.

class properties
"Liveness"
"When a simulation event is required, a simulation timer is eventually started.”
"AL2-01"

"Liveness"
"If a simulator timer expires, the simulator eventually has to generate an event.”
"AL2-02"

"Clean behaviour"
\(<V \text{anObject} \rightarrow \text{invariant anObject class \text{\textasciitilde} EventSimulator} >\)
"DS2-01"
"The EventSimulator class is an abstract class and should not be instantiated"

"Clean behaviour"
\(\text{invariant rand notNil \text{\&\& rand class = Random} \rightarrow DS2-02}\)
"Once rand has been initialized to refer to an instance of the Random class, it is never
set to nil while the instance of the EventSimulator subclass exists."

"It is therefore possible for the EventSimulator subclass instance to send messages
to rand at any stage after initialization.”

"Clean behaviour"
"The currentRandomTimeout value is always within the range specified by the
precondition of the start:id:for: method of the TimerServices class.”
"DS2-03"

"Global invariant"
"All outstanding timers requested by an EventSimulator subclass instance are
identified uniquely with respect to the requestor.”
"DS3-01"

"Thus, all the timers requested by this requestor that are
currently running or that are in the timerEventQ are uniquely
identified with respect to the requestor.”

instance methods
category private
message pattern initialize
"Creates an instance of class Random and sets newEventRequired,
generatingEvent and timerOutstanding to false. It also sets
currentRandomTimeoutValue to 1 so that the class invariant
referring to it will hold after instance creation and
initialization have been completed.”
method properties
"Total correctness"
true results-in methodReturnValue = self \&
rand notNil \& newEventRequired = false \&
currentRandomTimeoutValue = 1 \&
generatingEvent = false \&
timerOutstanding = false
"DL1-01"

sequential
rand := SmalltalkLibPkg:::Random new
\[ newEventRequired \rightarrow false \]
\[ currentRandomTimeoutValue \rightarrow 1 \]
generatingEvent := false
timerOutstanding := false
end-sequential

category accessing
message pattern nextRandomNumber: maximumValue
method properties
"Returns the next random number between 1 and maximumValue inclusive"
"Total correctness"
true results-in
methodReturnValue ≥ 1 ∧ methodReturnValue ≤ maximumValue
"DL1-02"
end-sequential

category testing
message pattern timerExpired: timerEventQ
method properties
"Returns true if the timerEventQ contains an element of which the requestor == self, otherwise it returns false. This method needs to be overridden when multiple simultaneous timers may be originated by the same requestor. In that case the identifier which uniquely identifies the timer with respect to the requestor has to match as well."
"Total correctness"
true results-in
methodReturnValue = (timerEventQ detect: [:each | each timeoutRequestor == self] ifNone: [nil]) notNil
"DL1-03"
end-sequential

category modifying
message pattern startRandomTimer: aTimerServices withMaximum: maximumValue
"Start a timer with a random value within the range between 1 and maximumValue. When the resulting start:id:for: message is sent to the TimerServices instance, a reference to the requestor (in this case the EventSimulator subclass instance) as well as an identifier are passed as parameters. The combination of the reference to the requestor and the identifier ensures that each timer request can be identified uniquely within the system. This facilitates the correlation of the subsequent timeout notifications with the timer requests.

In the EventSimulator class only one timer is outstanding at a time for a specific requestor, i.e. ~timerOutstanding is a precondition for starting a new timer for a specific instance of an EventSimulator subclass. Since the timers initiated by a specific EventSimulator subclass instance do not run concurrently, these timers can all have an identifier of 1. If a subclass requires multiple concurrent timers, unique values must be allocated to the corresponding identifiers. The startRandomTimer:withMaximum: method therefore needs to be overridden in order to achieve this. The total correctness property of the modified method also needs to be updated, viz. a disjunction needs to be added to the precondition to state that the proposed identifier of any new timer requested by that
EventSimulator subclass instance should not match any identifier of any other outstanding timer requested by that EventSimulator subclass instance. Thus, the precondition has to be weakened. In that case the value of timerOutstanding will no longer be relevant.

**method properties**

"Total correctness"

- timerOutstanding \( \rightarrow \) methodReturnValue = self \&
  self postconditions: (#nextRandomNumber:)
  withArguments: #(maximumValue) \&
  aTimerServices postconditions: (#start:id:for:)
  withArguments: #(self timerId currentRandomTimeoutValue) \&
  timerOutstanding

**sequential**

currentRandomTimeoutValue :=
(self nextRandomNumber: maximumValue)

- timerId := 1
- aTimerServices start: self id: timerId for:
  currentRandomTimeoutValue
- timerOutstanding := true

end-sequential

**message pattern** resetTimerExpired: timerEventQ

**method properties**

"Removes the first timeoutElement in timerEventQ where the requestor matches the receiver."

"Total correctness"

<3 \( \langle \) expiredTimeout \( \rangle \) where \( \text{expiredTimeout} = \langle \) (timerEventQ detect: [[:each | each timeoutRequestor == self]]

ifNone: [nil]) ::
  expiredTimeout notNil \( \rightarrow \) methodReturnValue = self \&
  (timerEventQ includes: expiredTimeout) not \&
  timerOutstanding not

**sequential**

timerEventQ remove:
(timerEventQ detect: [[:each | each timeoutRequestor == self]])

- timerOutstanding := false

end-sequential

**category** cyclic

**message pattern** p_simulate: aTimerServices timeoutEventsIn:

"If a new event is required, start a random timer, the expiry of which will cause an event to be initiated."

**method properties**

"This method ensures that properties DS2-03, AL2-01 and AL2-02 are satisfied by the EventSimulator class."

"Clean behaviour"

**invariant**

- currentRandomTimeoutValue > 0 \&
  currentRandomTimeoutValue \( \leq \) aTimerServices maximumTimeout

"A timeout requested by the EventSimulator subclass instance is always within the range that ensures that the precondition of the start:id:for: method of the
"Precedence"
newEventRequired \ ensures
  self postconditions: (#startRandomTimer:withMaximum:)
  withArguments:
    #{aTimerServices (aTimerServices maximumTimeout)}
^ ~newEventRequired
"DP1-01"
"When newEventRequired is true, it ensures that a simulation timer is started and
newEventRequired becomes false."

"Precedence"
self timerExpired: timerEventQ \ ensures
  generatingEvent ^
  self postconditions: (#resetTimerExpired:)
  withArguments: #{timerEventQ}
"DP1-02"
"When a simulation timer expires, it ensures that generatingEvent becomes true."

parallel
self startRandomTimer: aTimerServices withMaximum:
  (aTimerServices maximumTimeout) +
newEventRequired := false
  if newEventRequired
| generatingEvent := true |
self resetTimerExpired: timerEventQ
  if self timerExpired: timerEventQ
end-parallel
B.6 The CommsProviderSimulator class

The CommsProviderSimulator class simulates the actions of the communication provider. This class is replaced by the CommunicationProviderAgent class when the call centre interacts with the actual communication provider instead of simulating its actions.

The CommsProviderSimulator class is a subclass of EventSimulator. The newEventRequired instance variable is set to true as part of the initialization procedures. That results in the starting of a new timer. Once the timer has expired, an event is generated. A new connection is established if one is available and the associated service request is added to the end of the input queue. The timeout is ignored if all the connections are busy. This would correspond to a busy signal being received by the service user in an actual implementation.

class CommsProviderSimulator
  superclass EventSimulator from ApplicationsRepository

class properties
"The communication provider agent constructs the serviceRequest associated with the connection based on information received from the communication provider. For example, in the one case the service user identification information and the type of service required will be received. In another case no information will be received (perhaps because it is not relevant, e.g. when a directory enquiry is made). The simulation puts no information into the service request."

class methods

category instance creation
message pattern startSimulation
method properties

"The initialize method of the superclass sets newEventRequired to false. This method sends the initialize message to its superclass and immediately sets newEventRequired to true. The values of the other instance variables mentioned in the total correctness property of the initialize method of the superclass remain unchanged."
"Total correctness"
∀ k where k ≥ 0 ::
  self instanceCount = k results-in
  self instanceCount = k + 1 ∧
  methodReturnValue notNil

"DL1-01"

^(super new initialize) moreInit
end-sequential

instance methods

category private
message pattern moreInit
"Sets newEventRequired to true (which was set to false in the initialization routine of the superclass)."

method properties
"Total correctness"
true results-in methodReturnValue = self ∧
  newEventRequired = true

"DL1-02"
"This property ensures that property AS3-01, which was identified during the analysis phase, is achieved. Property AS3-01 specifies that instance creation results in a communication provider simulator event being required."
newEventRequired := true
end-sequential

category accessing
message pattern getIdleConnection: userConnections
"Returns the first connection that is idle"
method properties
"Total correctness"
true results-in methodReturnValue =
  userConnections detect: [:each | each isIdle] ifNone: [nil]
end-sequential
^ userConnections detect: [:each | each isIdle] ifNone: [nil]

category modifying
message pattern terminate: aConnection cause: reason
method properties
"Inform the communication provider that the connection has terminated."
"Total correctness"
true results-in methodReturnValue = self
increment
Transcript show: 'Connection'
Transcript show: (aConnection connectionIndex printString)
Transcript show: 'has terminated with cause'
Transcript show: reason
end-sequential

category cyclic
message pattern p_generateEvent: userConnections
target: inputQ
method macros
idleConnection = self getIdleConnection: userConnections
method properties
"Simulate an event from the communication provider."
"Safe liveness"
generatingEvent ^ idleConnection notNil ^
¬(inputQ includes: (idleConnection serviceRequest)) ensures
  ¬generatingEvent ^ newEventRequired ^ inputQ last = (idleConnection serviceRequest) ^
  idleConnection postconditions: (#assign) "AP1-01"
"If an event has to be generated and the maximum number of connections have not yet been established, the communication provider simulator ensures that a new connection is established, the associated service request is appended to the input queue and a new communication provider simulator event is again required."
"Safe liveness"
generatingEvent ^ idleConnection isNil ensures
  ¬generatingEvent ^ newEventRequired "AP1-02"
"If an event has to be generated and the maximum number of connections have already been established, the communication provider simulator ensures that the event is cancelled and a new communication provider simulator event is again required."
parallel
inputQ addLast: (idleConnection serviceRequest) \+
idleConnection assign
  if generatingEvent and: [idleConnection notNil] ~
Transcript show: 'All connections busy'
    if generatingEvent and: [idleConnection isNil]
    [newEventRequired := true \]
    generatingEvent := false
    if generatingEvent
end-parallel
B.7 The Connection class

class Connection
superclass Object from SmalltalkLibRepository
instance variable names
state
"The state of the connection"
terminatingReason
"The reason why the connection is being terminated."
serviceRequest
"The service request associated with the connection."
currentHandlerInformed
"This flag is used when a connection has to be terminated. It indicates whether the communication provider has been informed of the termination of the connection."
connectionIndex
"The index of this connection into the userConnections array."

class properties
"Note that there are no safety properties specifying the allowed values of the state instance variable. There are also no safety properties specifying the allowed state transitions. The reason for this is to avoid overspecification, i.e. it avoids restricting subclasses to certain specified values. Recall that preconditions may not be strengthened and postconditions may not be weakened during subclassing."

class methods
category instance creation
message pattern setup: indexOfConnection
method properties
"Total correctness"
<\ k where k \geq 0 ::
self instanceCount = k results-in
self instanceCount = k + 1 ^
methodReturnValue notNil
>
sequential
super new initialize: indexOfConnection
end-sequential

instance methods
category private
message pattern initialize: indexOfConnection
method properties
"Total correctness"
true results-in methodReturnValue = self ^ state = 'IDLE' ^
serviceRequest notNil ^ currentHandlerInformed = false ^
connectionIndex = indexOfConnection
"DLl-11"
sequential
state := 'IDLE'
[] serviceRequest := CC_CorePkg:::ServiceRequest setup: self
[] currentHandlerInformed := false
[] connectionIndex := indexOfConnection
end-sequential
category accessing
message pattern terminatingReason
method properties
"Total correctness"
true results-in methodReturnValue = terminatingReason "DL1-02"
sequential
^ terminatingReason
end-sequential

message pattern connectionIndex
method properties
"Total correctness"
true results-in methodReturnValue = connectionIndex "DL1-03"
sequential
^ connectionIndex
end-sequential

message pattern serviceRequest
method properties
"Total correctness"
true results-in methodReturnValue = serviceRequest "DL1-04"
sequential
^ serviceRequest
end-sequential

category testing
message pattern isIdle
method properties
"Total correctness"
true results-in methodReturnValue = (state = 'IDLE') "DL1-05"
sequential
^ state = 'IDLE'
end-sequential

message pattern isTerminating
method properties
"Total correctness"
true results-in methodReturnValue = (state = 'TERMINATING') "DL1-06"
sequential
^ state = 'TERMINATING'
end-sequential

category modifying
message pattern assign
method properties
"Total correctness"
state = 'IDLE' results-in methodReturnValue = self ∧ state = 'CONNECTED' "DL1-07"
sequential
state := 'CONNECTED'
if state = 'IDLE'
end-sequential
message pattern terminate: reason
method properties
"Total correctness"
state = 'CONNECTED' results-in
   methodReturnValue = self ∧
   state = 'TERMINATING' ∧ terminatingReason = reason
"DL1-08"
"Total correctness"
state = 'TERMINATING' results-in methodReturnValue = self
"DL1-09"
"This allows for terminate collision."

"Total correctness"
state = 'IDLE' results-in methodReturnValue = self
"DL1-10"
"This ensures that the transition from 'IDLE' to 'TERMINATING' is not possible"

  sequential
  terminatingReason := reason ∨
  state := 'TERMINATING'
  if state = 'CONNECTED'
end-sequential

category cyclic
message pattern p_informCommsProvider: commsAgent
method properties
"Safe liveness"
state = 'TERMINATING' ∧ terminatingReason = 'completed' ∧ ¬currentHandlerInformed ensures
   commsAgent postconditions: (#terminate:cause:)
   withArguments: #(self terminatingReason) ∧
   currentHandlerInformed
"DP1-01"
  parallel
  commsAgent terminate: self cause: terminatingReason ∨
  currentHandlerInformed := true
  if state = 'TERMINATING' and:
    [(terminatingReason = 'completed')
     and: [currentHandlerInformed not]]
end-parallel

message pattern p_doWrapUp
method properties
"Safe liveness"
currentHandlerInformed ensures
state = 'IDLE' ∧ serviceRequest postconditions: (#reset) ∧ ¬currentHandlerInformed
"DP1-02"
  parallel
  state := 'IDLE' ∨
  serviceRequest reset ∨
  currentHandlerInformed := false
  if currentHandlerInformed
end-parallel

393
class ServiceCategoryAllocator
superclass Object from SmalltalkLibRepository
instance variable categorising
"The categorising variable is used as a flag to indicate whether
the categorisation of the service request at the head of the
inputQ has been initiated or not."
class properties
"Safe liveness"
<∀ aServiceRequest where inputQ includes: aServiceRequest ::
  inputQ first = aServiceRequest ∧ ¬categorising ensures
  inputQ first = aServiceRequest ∧ categorising
> "DP1-01"
"Safe liveness"
<∀ aServiceRequest where inputQ includes: aServiceRequest ::
  inputQ first = aServiceRequest ∧ categorising ensures
  ∃ aServiceQueue where
  (scContainer detects: [:each | each serviceQ = aServiceQueue]
  ifNone: [nil] notNil ::
  aServiceQueue includes: aServiceRequest)
  ∧ ¬(inputQ includes: aServiceRequest) ∧ ¬(categorising)
> "DP1-02"
class methods
category instance creation
message pattern setup
method properties
"Total correctness"
<∀ k where k ≥ 0 ::
  self instanceCount = k results-in
  self instanceCount = k + 1 ∧
  methodReturnValue notNil
> "DL1-01"
sequential
super new initialize
end-sequential
instance methods
category private
message pattern initialize
method properties
"Total correctness"
true results-in methodReturnValue = self ∧ ¬categorising
"DL1-02"
sequential
categorising := false
end-sequential
category modifying
message pattern categoriseServiceRequest: serviceRequest
using: scContainer
"When the service request does not contain any categorisation
data, it is categorised as belonging to the first service
category in scContainer. This is the default behaviour which
facilitates usage of this class without further subclassing if
only one service category is supported by the system. This
method needs to be reimplemented in the subclasses if multiple
service categories are supported. In that case the default
category is only used if the categorisation data is not
provided, otherwise the service request is categorised according to the data provided by the service user (e.g. via the IVR)."

**method properties**

"Total correctness"

true **results-in**

- methodReturnValue = self ∧
- serviceRequest serviceRequestCategory notNil ∧
- (scContainer detects:
  - [:each | each serviceQCategory =
    - serviceRequest serviceRequestCategory] ifNone: [nil])
  notNil

**sequential**

- serviceRequest serviceRequestCategory:
  - (scContainer first) serviceQCategory

**end-sequential**

**message pattern** assignToSQ: serviceRequest using: scContainer

**method macros**

match = scContainer detect:
  - [:each | each serviceQCategory =
    - serviceRequest serviceRequestCategory]
  ifNone: [nil]

**method properties**

"Further refinements would override this method to include error conditions. For example, the service request would be rejected if the service providers in the service provider container were all operating in the restricted mode."

"Total correctness"

- serviceRequest serviceQ isNil ∧
- serviceRequest serviceRequestCategory notNil ∧
- match notNil

**results-in**

- methodReturnValue = self ∧
- serviceRequest serviceQ notNil ∧
- serviceRequest serviceRequestCategory notNil ∧
- (match serviceQ) includes: serviceRequest

**sequential**

- serviceRequest serviceQ: (match serviceQ) \+
  (match serviceQ) addLast: serviceRequest
  if match notNil

**end-sequential**

**category** cyclic

**message pattern** p_categorise: inputQ using: scContainer

**method properties**

"Safe liveness"

- (inputQ isEmpty) ∧ ¬categorising until
  - categorising ∧
  - self postconditions: (#categoriseServiceRequest:using:)
  - withArguments: {((inputQ first) scContainer)

**parallel**

- categorising := true \+
- self categoriseServiceRequest: (inputQ first)
- using: scContainer
  if inputQ isEmpty not and: [categorising not]

**end-parallel**

**message pattern** p_allocate: scContainer from: inputQ

**method macros**

- serviceRequest = inputQ first
  if inputQ isEmpty not ~
  nil
  if inputQ isEmpty
method properties
"Safe liveness"

∀ aServiceRequest where

¬(inputQ isEmpty) ∧ inputQ first == aServiceRequest ::

aServiceRequest serviceRequestCategory notNil ∧
aServiceRequest serviceQ isNil ∧

∃ aServiceCategory where

scContainer includes: aServiceCategory ::

aServiceCategory serviceQCategory =
aServiceRequest serviceRequestCategory

> ensures

self postconditions: (#assignToSQ:using:)

withArguments: #(aServiceRequest scContainer) ∧

¬categorising ∧

¬(inputQ includes: aServiceRequest)

> "DP1-04"

parallel

self assignToSQ: serviceRequest using: scContainer +
categorising := false +
inputQ removeFirst

if serviceRequest notNil and:

[serviceRequest serviceRequestCategory notNil and:

[serviceRequest serviceQ isNil]]

end-parallel
B.9 The ServiceRequest class

class ServiceRequest
superclass Object from SmalltalkLibRepository
instance variable names
serviceQ
"The service queue to which the service request has been assigned."
serviceRequestCategory
"The category of this service request."
connection
"The connection associated with this service request."
serviceProvider
"The service provider to which this service request has been assigned."
categorisationData
"The data which is used to categorise this service request."

class methods
category Instance creation
message pattern setup: aConnection
method properties
"Total correctness"
∀ k where k ≥ 0 ::
self instanceCount = k results-in
  self instanceCount = k + 1 ∧
  methodReturnValue notNil
>
sequential
^super new initialize: aConnection
end-sequential

instance methods
category private
message pattern initialize: aConnection
method properties
true results-in methodReturnValue = self ∧
  connection = aConnection ∧
  self postconditions: (#reset)
sequential
connection := aConnection
[] self reset
end-sequential

category accessing
message pattern serviceQ
method properties
"Total correctness"
true results-in methodReturnValue = serviceQ
sequential
^ serviceQ
end-sequential
message pattern serviceRequestCategory
method properties
"Total correctness"
true results-in methodReturnValue = serviceRequestCategory
   "DL1-04"
   ^ serviceRequestCategory
end-sequential

message pattern connection
method properties
"Total correctness"
true results-in methodReturnValue = connection
   "DL1-05"
   ^ connection
end-sequential

message pattern serviceProvider
method properties
"Total correctness"
true results-in methodReturnValue = serviceProvider
   "DL1-06"
   ^ serviceProvider
end-sequential

message pattern categorisationData
method properties
"Total correctness"
true results-in methodReturnValue = categorisationData
   "DL1-07"
   ^ categorisationData
end-sequential

category modifying
message pattern serviceQ: sq
method properties
"Total correctness"
true results-in methodReturnValue = self ∧ serviceQ = sq
   "DL1-08"
   ^ serviceQ := sq
end-sequential

message pattern serviceRequestCategory: srCategory
method properties
"Total correctness"
true results-in methodReturnValue = self ∧ serviceRequestCategory = srCategory
   "DL1-09"
   ^ serviceRequestCategory := srCategory
end-sequential

message pattern serviceProvider: sp
method properties
"Total correctness"
true results-in methodReturnValue = self ∧ serviceProvider = sp
   "DL1-10"
   ^ serviceProvider := sp
end-sequential
message pattern categorisationData: newData
method properties
"Total correctness"
true results-in methodReturnValue = self ∧
categorisationData = newData

sequential
categorisationData := newData
end-sequential

message pattern reset
method properties
"Total correctness"
true results-in methodReturnValue = self ∧ serviceQ isNil ∧
serviceRequestCategory isNil ∧ serviceProvider isNil ∧
categorisationData isNil

sequential
serviceQ := nil
serviceRequestCategory := nil
serviceProvider := nil
end-sequential
Subclasses may override the `canAssignSR` and `assignToSP` methods to implement algorithms to assign service requests to service providers as required by individual applications. The algorithm implemented for the `ServiceQueueCategory` class is to assign the service request to the first available service provider in the container.

In anticipation of a refinement which might require that service providers be allocated in a round robin fashion, `OrderedCollection` rather than `Set` is used for the service provider container component of the `ServiceCategory` class. The main difference between `OrderedCollection` and `Set` is the fact that the elements of the former are ordered and those of the latter are not. By ordering the service providers within such a container, it makes it possible to implement an algorithm which will ensure that all service providers are utilised. For example, if service requests of a specific category arrive slowly enough that the second service request is only considered once the service provider has finished serving the first, then it could happen that the service requests are always allocated to the same service provider. An `OrderedCollection` implementation could assist in ensuring that the service requests are allocated to the service providers in a round robin fashion.

class ServiceCategory
superclass Object

instance variable names
serviceQCategory
"The category of the service requests enqueued in the serviceQ"
serviceQ
"The FIFO queue containing service requests matching serviceQCategory"
spCategories
"The collection of service provider categories that apply to the serviceQ"
spSubset
"The collection of service providers that may service the serviceQ"

class properties

class methods
category instance creation
message pattern setup: config
method properties
"Total correctness"
<∀ k where k ≥ 0 ::
self instanceCount = k ∧ config notNil results-in
  self instanceCount = k + 1 ∧
  methodReturnValue notNil
>
"DL1-01"

sequential
"super new initialize: config
end-sequential

instance methods
category private
message pattern initialize: config

method macros
maxConn = config maximumConnections
maxSP = config maximumServiceProviders

**method properties**

"Total correctness"

config notNil **results-in**

methodReturnValue = self ∧

serviceQ notNil ∧ serviceQCategory notNil ∧

spSubset notNil ∧ spCategories notNil ∧

self postconditions: (#registerSPCategories:)

withArguments: #(config)  "DL1-02"

**sequential**

serviceQ := SmalltalkLibPkg:::OrderedCollection new: maxConn

serviceQCategory := config assignSRCategory

spSubset := SmalltalkLibPkg:::OrderedCollection new: maxSP

spCategories := SmalltalkLibPkg:::OrderedCollection new

self registerSPCategories: config

end-**sequential**

**message pattern** registerSPCategories: config

**method properties**

"Total correctness"

config notNil ∧ serviceQCategory notNil **results-in**

methodReturnValue = self ∧ ¬spCategories isEmpty  "DL1-03"

**sequential**

spCategories addAll:

(config getSPCategories: serviceQCategory)

end-**sequential**

**category testing**

**message pattern** servicedBy: spCategory

**method properties**

"Total correctness"

spCategory notNil **results-in**

methodReturnValue = (spCategories includes: spCategory)  "DL1-04"

**sequential**

¬spCategories includes: spCategory

end-**sequential**

**message pattern** canAssignSR

**method properties**

"Total correctness"

true **results-in** methodReturnValue =

(spSubset detect:

[:each | each canAcceptNextSR: serviceQCategory]

ifNone: [nil] ) notNil  "DL1-05"

**sequential**

¬( (spSubset detect:

[:each | each canAcceptNextSR: serviceQCategory]

ifNone: [nil]) notNil)

end-**sequential**

**category accessing**

**message pattern** spSubset

**method properties**

"Total correctness"

true **results-in** methodReturnValue = spSubset  "DL1-06"
sequential
^spSubset
end-sequential

message pattern serviceQCategory
method properties
"Total correctness"
true results-in methodReturnValue = serviceQCategory
"DL1-07"
sequential
^serviceQCategory
end-sequential

message pattern serviceQ
method properties
"Total correctness"
true results-in methodReturnValue = serviceQ
"DL1-08"
sequential
^serviceQ
end-sequential

category modifying
message pattern addSP: anSP
method properties
"Total correctness"
anSP notNil results-in
methodReturnValue = self ^ spSubset includes: anSP
"DL1-09"
sequential
spSubset addLast: anSP
end-sequential

message pattern assignToSP: sr
method macros
availableServiceProvider =
spSubset detect: [:each | each canAcceptNextSR: serviceQCategory]
method properties
"Total correctness"
sr notNil ^ availableServiceProvider notNil results-in
methodReturnValue = self ^ sr postconditions: (#serviceProvider:)
withArguments: #(availableServiceProvider) ^
availableServiceProvider
postconditions: (#processServiceRequest:)
withArguments: #(sr)
"DL1-10"
sequential
sr serviceProvider: availableServiceProvider
[] availableServiceProvider processServiceRequest: sr
end-sequential

category cyclic
message pattern p_execute
method properties
"Safe liveness"
serviceQ isEmpty not ^ self canAssignSR ensures
self postconditions: (#assignToSP:) withArguments: #((serviceQ first)) ^
serviceQ postconditions: (#removeFirst)
"DP1-01"
parallel
self assignToSP: (serviceQ first) \+
  serviceQ removeFirst
if serviceQ isEmpty not and: [self canAssignSR]
end-parallel
B.11 The TimerServices class

The TimerServices class allows its clients to request that timers of specified durations be started. The timer resolution is in seconds and the maximum timeout value is denoted by maximumTimeout. The TimerServices class uses a circular array called timeoutCollection to implement the timers. Each position in the array represents one second. The object maintains an index into the array. This index is called currentTick and is advanced every second. Thus, the entry in the array which will be reached \( x \) seconds from the current moment can be calculated using the value of currentTick, the size of the array and the value of \( x \).

Each entry in the array is an ordered collection of TimeoutElement instances. When the TimerServices class receives a request to start a timer, it creates a TimeoutElement instance to represent that timer and enters it into the relevant collection of TimeoutElement instances, i.e. it enters it into the collection that will be reached after \( x \) seconds, where \( x \) is the timeout value specified for the timer. The TimeoutElement instance contains the timeout identifier. It also contains other attributes that may be used to obtain information about the timeout.

When a timer expires, its associated TimeoutElement instance is removed from the relevant collection in the circular array and it is added to timerEventQ. The latter is inspected by the clients of TimerServices in order to determine whether their requested timers have timed out yet.

The currentTick variable is used to calculate the read and write indices and is updated every second. When the TimerServices instance is created, currentTick is initialized to zero. Every second it is incremented modulo the size of the array. The size of the array is a function of the maximum timeout value.

Thus, when a TimerServices instance receives a request to start a timer, it has to determine which list of TimeoutElement instances will be processed after \((\text{duration} + 1)\) clock ticks. It is necessary to add the extra one in order to ensure that a timer does not expire prematurely. For example, if a timer for one second is started at the time when half a second of the currentTick period has already passed, then the new timer will expire after only half a second. By adding the extra one, a timer of one second could take up to two seconds to expire, but it will at least run for one second.

The write index is therefore calculated as follows:

\[
\text{writeIndex} := (((\text{currentTick} + \text{duration} + 1) \mod \text{timeoutCollection size}) + 1)
\]

The requested timeout value (represented by duration) plus one is added to the value of currentTick. The addition is performed modulo the size of the array, which is maximumTimeout + 2. The size of the array is explained as follows: Since one is always added to the duration in order to ensure that a timer does not expire prematurely, the maximum timeout value is actually maximumTimeout + 1. Another one has to be added to ensure that the read and write indices never have the same value. After the modulo addition has been performed, the writeIndex has to be adjusted by one, since SLOOP array indices start at one, not zero. The readIndex has the value currentTick plus one. It is also adjusted by one in order to compensate for the fact that the array indices start at one. Once the writeIndex has been calculated, the TimeoutElement instance is added to the end of the First In First Out list that is stored at that position in the array.

404
Figure B-3(a) illustrates where a new entry would be added for the following values of the various variables:

- maximumTimeout = 5
- duration = 5
- currentTick = 3

Figure B-3(b) shows the result if the currentTick value had been 0 at the time of the timer request.

Note that the currentTick value ranges from 0 to 6 inclusive in the above example, whereas readIndex and writeIndex may range from 1 to 7 inclusive.

![Diagram](image)

**Figure B-3(a).** Structures used by aTimerServices (currentTick = 3).
The value of the `currentTick` variable is updated whenever one second has passed since its last update and all the entries at the current `readIndex` have been processed. This is reflected by one of the parallel statements of the TimerServices class (it is executed infinitely often):

```plaintext
lastTime := currentTime +
currentTick := (currentTick + 1) \ (timeoutCollection size)
    if difference \geq 1 and: [currentTimeoutElement isNil]
```

This statement uses two macro-variables, viz. `difference` and `currentTimeoutElement`. They receive their values in the `macro-section` of the method containing the parallel statement, as shown below (the macro-variable `readIndex` is used in the definition of `currentTimeoutElement` and therefore has to be defined prior to its usage):

```plaintext
readIndex := currentTick + 1
[] difference = currentTime - lastTime
    if (currentTime - lastTime) \geq 0 -
currentTime + (86400 - lastTime)
    if (currentTime - lastTime) < 0
```

*Figure B-3(b). Structures used by aTimerServices (currentTick = 0).*
currentTimeoutElement :=
    (timeoutCollection at: readIndex) first
    if (timeoutCollection at: readIndex) isEmpty not -
    currentTimeoutElement := nil
    if (timeoutCollection at: readIndex) isEmpty

One of the other parallel statements of the TimerServices class ensures that the currentTime variable is updated on a regular basis:

currentTime := SmalltalkLibPkg::Time now asSeconds

Evaluation of the Time now asSeconds expression yields the number of seconds since midnight. The calculation of difference takes the rollover at midnight into account in the macros-section shown above.

The TimerServices class is now presented in the SLOOP notation.
class TimerServices
superclass Object from SmalltalkLibRepository

instance variable names
maximumTimeout
"The maximum timeout value that may be requested"
timeoutCollection
"A circular array. Each element comprises a list of
TimeoutElement instances"
currentTick
"It points to a position in the timeoutCollection array. It is
used to calculate the read and write indices."
currentTime
"The most recent time (in number of seconds since midnight)
obtained from the system."
lastTime
"The time (in number of seconds since midnight) when the
currentTick was last updated."

class properties
"Global invariant: when a timeout is indicated to the
requestor, then a period greater than or equal to the value
specified by the requestor has expired."

invariant
\(\forall\, a\text{TimeoutElement}\, \text{where}
\quad \text{timerEventQ includes: aTimeoutElement:}
\quad \text{aTimeoutElement::}
\quad \text{aTimeoutElement.timerExpired}
\quad \text{leads-to}
\quad \text{aTimeoutElement.timerServicesCompleted}
\quad \text{"DS3-01"}

\(\forall\, a\text{TimeoutElement}\, \text{where}
\quad \exists\, i\, \text{where} \quad 1 \leq i \leq \text{timeoutCollection size}
\quad \text{(timeoutCollection at: i) includes: aTimeoutElement}
\quad \text{leads-to}
\quad \text{aTimeoutElement.timerServicesCompleted leads-to}
\quad \text{aTimeoutElement.timerServicesCompleted}
\quad \text{"DL2-01(TimerServices)"}

"Once a timer has been started, i.e. it is present in one of the lists associated with
timeoutCollection, the TimerServices instance will eventually complete its
responsibilities regarding the timer (i.e. the timer will either be stopped or the
TimerServices instance will indicate its expiry to the requestor of the timer)."
"This property refers to the timerServicesCompleted method
rather than the timerExpired method in order to make provision
for subclasses that may allow a timer to be aborted."

class methods
category Instance creation
message pattern setup: config
method properties
"Total correctness"
\(\forall\, k\, \text{where} \quad k \geq 0::
\quad \text{self instanceCount} = k \wedge \text{config notNil results-in}
\quad \text{self instanceCount} = k + 1 \wedge
\quad \text{methodReturnValue notNil}
\quad \text{"DL1-01"}
sequential
^super new initialize: config
end-sequential
instance methods
category private
message pattern initialize: config
method properties
"Total correctness"
true results-in timeoutCollection notNil ∧
maxTimeout = config maximumAllowableTimeout ∧
∀ i where 1 ≤ i ≤ timeoutCollection size ::
(timeoutCollection at: i) notNil
> ∧
currentTick = 0 ∧ currentTime notNil ∧ lastTime notNil
"DL1-02"

maximumTimeout := config maximumAllowableTimeout
timeoutCollection :=
SmalltalkLibPkg:::Array new: (maxTimeout + 2)
< i where 1 ≤ i ≤ timeoutCollection size ::
timeoutCollection at: i put: (OrderedCollection new)
> ∧
currentTick := 0
currentTime := SmalltalkLibPkg:::Time now asSeconds
lastTime := currentTime
end-sequential

category accessing
message pattern maximumTimeout
method properties
"Total correctness"
true results-in methodReturnValue = maximumTimeout
"DL1-03"

maximumTimeout
end-sequential

message pattern isTimerRunningFor: requestor with: identifier
method properties
"Total correctness"
true results-in methodReturnValue = (found notNil) ∧
found detects
∀ i where (1 ≤ i ≤ timeoutCollection size):
<timeoutElement ∧
(timeoutCollection at: i) includes: timeoutElement ::
timeoutElement timeoutRequestor = requestor ∧
timeoutElement timeoutIdentifier = identifier
> ∧
"DL1-04"

found := nil
< i where (1 ≤ i ≤ timeoutCollection size):
found := (timeoutCollection at: i) detect:
[:each | each timeoutRequestor = requestor and:
(each timeoutIdentifier = identifier)] ifNone: [nil]
if found isNil
> ∧ found notNil
end-sequential
category modifying
message pattern start: requestor id: identifier for: duration
method macros
   writeIndex =
   \((currentTick + duration + 1) \mod \text{timeoutCollection size}) + 1
   "This is because the array index starts at 1, not 0"

method properties
"Clean behaviour"
invariant 1 \leq writeIndex \land writeIndex \leq \text{timeoutCollection size}
       "DS3-02"
invariant writeIndex \equiv \text{readIndex}
       "DS3-03"

"Total correctness"
0 \leq \text{duration} \land \text{duration} \leq \maxTimeout results-in
   methodReturnValue = self \land
   \text{nextElement class = TimeoutElement} \land
   (\text{timeoutCollection at: writeIndex}) \includes: \text{nextElement} \land
   \text{TimeoutElement postconditions: \#setup:id:for:} \land
   \text{withArguments: \#(requestor identifier duration)}
   "DL1-05"

   sequential
   \text{nextElement := SystemUtilitiesPkg:::TimeoutElement setup: requestor}
   id: identifier for: duration
   \if 0 \leq \text{duration} \land \text{duration} \leq \maxTimeout
   (\text{timeoutCollection at: writeIndex}) addLast: \text{nextElement}
   \if 0 \leq \text{duration} \land \text{duration} \leq \maxTimeout
   end-sequential

category cyclic
message pattern p_runTimer: timerEventQ
method macros
   readIndex = currentTick + 1
   "This is because the array index starts at 1, not 0"
   difference = \text{currentTime} - \text{lastTime}
   \if \text{currentTime} - \text{lastTime} \geq 0
      \text{currentTimeoutElement = (timeoutCollection at: readIndex) first}
   \if \text{timeoutCollection at: readIndex} \isEmpty
      \Nil
   \if \text{timeoutCollection at: readIndex} \isEmpty

method properties
"Intermittent assertion"
\forall y \where 0 \leq y < \text{timeoutCollection size} : \:
   \text{currentTick} = y \land \text{difference} \geq 1 \land
   \text{currentTimeoutElement isNil leads-to}
   \text{currentTick} = (y + 1) \mod \text{timeoutCollection size}
   "DL2-02"

"Thus, all the entries at the current timeout position have to
be processed before the entries at the next position are
processed. The granularity of the timer is \geq 1\ second"
**Safe liveness**

```
<\aTimeoutElement where
  aTimeoutElement = currentTimeoutElement:::
  aTimeoutElement = notNil ensures
    ~((timeoutCollection at: readIndex) includes:
      aTimeoutElement) ∧
    timerEventQ includes: aTimeoutElement ∧
    aTimeoutElement timerServicesCompleted
>
```

```
parallel
  currentTime := SmalltalkLibPkg:::Time now asSeconds
  lastTime := currentTime +
  currentTick := (currentTick + 1) \ (timeoutCollection size)
    if difference ≥ 1 and: [currentTimeoutElement isNil]
      timerEventQ addLast: currentTimeoutElement +
      currentTimeoutElement updateEndTime +
      currentTimeoutElement timerServicesCompleted: true +
      (timeoutCollection at: readIndex) removeFirst
    if currentTimeoutElement notNil
end-parallel
```

Note that the macro-variable writeIndex cannot be defined in the class-macros section, because it refers to the pseudo-variable duration.

The clients of the TimerServices class are not restricted to removing only the first element of the timerEventQ. That ensures that each client will receive its timeout information even if other clients misbehave. Each client must remove the TimeoutElement instance from the timerEventQ as it processes it, otherwise it will process the same element multiple times.

The advantage of having a timerEventQ is that the interface between the TimerServices instance and its clients is very loosely coupled. Alternatively the TimerServices instance can invoke a client method when a timer expires, but in that case it is necessary to reserve the client together with the TimerServices instance when the timeout is processed.

The reason for defining timerEventQ as a peer class rather than as part of an aggregation (i.e. within TimerServices) is to allow for more parallelism. This way it is not necessary to reserve timerEventQ as well whenever a timer method is executed.
B.12 The TimeoutElement class

The TimeoutElement class is now presented using the SLOOK notation:

```
class TimeoutElement
superclass Object from SmalltalkLibRepository
instance variable names
startTime
  "The time when this timeout was started"
endTime
  "The time when this timeout expired."
timeoutRequestor
  "The requestor of this timeout."
timeoutIdentifier
  "The identifier of this timeout. It is unique with respect to
  the requestor."
requestedDuration
  "The requested duration of this timeout."
timerServicesCompleted
  "This flag indicates whether the TimerServices instance has
  completed its tasks regarding this timeout. This flag is used
  rather than checking the endTime and startTime in order to be
  able to cater for the case where the timeout is aborted."

class macros
currentTime = SmalltalkLibPkg:::Time total Seconds
  "currentTime contains the total number of seconds since January
  1, 1901."

class properties
invariant endTime ≥ startTime
self getCurrentDuration = 0 unless
  self getCurrentDuration > 0

class methods
category instance creation
message pattern setup: requestor id: identifier for: duration
method properties
"Total correctness"
∀ k where k ≥ 0 ::
  self instanceCount = k results-in
    self instanceCount = k + 1 ∧
    methodReturnValue notNil
> sequential
^super new initialize: requestor id: identifier for: duration
end-sequential

instance methods
category private
message pattern initialize: requestor id: identifier for: duration
method properties
"Total correctness"
true results-in methodReturnValue = self ∧
  timeoutRequestor = requestor ∧
  timeoutIdentifier = identifier ∧
  startTime notNil ∧ endTime notNil ∧
  requestedDuration = duration ∧ ¬timerServicesCompleted
```

412
sequential
timeoutRequestor := requestor
| timeoutIdentifier := identifier
| startTime := currentTime
| endTime := currentTime
| requestedDuration := duration
| timerServicesCompleted := false
end-sequential

category accessing
message pattern timeoutRequestor
method properties
"Total correctness"
true results-in methodReturnValue = timeoutRequestor "DL1-03"
sequential
^timeoutRequestor
end-sequential

message pattern timeoutIdentifier
method properties
"Total correctness"
true results-in methodReturnValue = timeoutIdentifier "DL1-04"
sequential
^timeoutIdentifier
end-sequential

message pattern startTime
method properties
"Total correctness"
true results-in methodReturnValue = startTime "DL1-05"
sequential
^startTime
end-sequential

message pattern requestedDuration
method properties
"Total correctness"
true results-in methodReturnValue = requestedDuration "DL1-06"
sequential
^requestedDuration
end-sequential

message pattern getCurrentDuration
method properties
"Total correctness"
true results-in methodReturnValue = (currentTime - startTime) "DL1-07"
sequential
^(currentTime - startTime)
end-sequential

message pattern getTimeoutDuration
method properties
"Total correctness"
true results-in methodReturnValue = (endTime - startTime) "DL1-08"
sequential
^(endTime - startTime)
end-sequential
category testing
message pattern timerExpired
method properties
"Total correctness"
true results-in methodReturnValue = (self getCurrentDuration - requestedDuration ≥ 0) "DL1-09"
sequential
"(self getCurrentDuration - requestedDuration ≥ 0)
end-sequential

message pattern timerServicesCompleted
method properties
"Total correctness"
true results-in methodReturnValue = timerServicesCompleted "DL1-10"
sequential
^timerServicesCompleted
end-sequential

category modifying
message pattern updateEndTime
method properties
"Total correctness"
true results-in methodReturnValue = self ∧ endTime = currentTime "DL1-11"
sequential
endTime := currentTime
end-sequential

message pattern timerServicesCompleted: newValue
method properties
"Total correctness"
true results-in methodReturnValue = self ∧ timerServicesCompleted = newValue "DL1-12"
sequential
timerServicesCompleted := newValue
end-sequential
B.13 The ServiceProviderSimulator class

**class** ServiceProviderSimulator  
**superclass** EventSimulator from ApplicationsRepository  
**instance variable names**  

serviceRequest  
"This variable refers to the service request currently being serviced by the service provider simulator. Note that the reference to the ServiceRequest instance is passed to the simulator as a parameter, i.e. the ServiceRequest instance is not created by the ServiceProviderSimulator instance and therefore does not form part of it."

serviceProviderCategory  
"This variable contains the name of the service provider category to which the service provider simulator belongs."

categoriesServed  
"This is an ordered collection containing the names of the service request categories serviced by this service provider. The purpose of this array is to facilitate a round robin servicing scheme of these categories. That prevents starvation of a specific service category."

nrOfCategoriesServed  
"This variable contains the number of service request categories serviced by this service provider. It is used in the calculation when the categoryIndex is updated."

categoryIndex  
"This variable is used as index into the categoriesServed collection. It is used to determine the next service request category to be serviced by this service provider. It is incremented modulo nrOfCategoriesServed. Its values range from 0 to nrOfCategoriesServed - 1"

**class properties**

\[
<∀ \text{categoryIndex} \text{ where categoryIndex} \geq 0 \land \text{categoryIndex} \leq \text{nrCategoriesServed} - 1 ::
\]

**invariant** serviceRequest notNil ⇒ ¬self canAcceptNextSR:

\( (\text{categoriesServed at: (categoryIndex + 1)}) \)  

"AS3-01"

"A service provider simulator services a single service request at a time."

"If a service request is currently assigned to the simulator, no other service request from any of the categories being served by this simulator will be served by the latter."

\[
\text{serviceRequest isNil } \land \lnot \text{newEventRequired} \, \text{unless}
\]

\[
\text{serviceRequest notNil } \land \text{newEventRequired}
\]  

"AS4-01"

"When a new service request is assigned to the service provider simulator then a new service provider simulator event is required."

"Note: The parent class, viz. EventSimulator, contains a parallel method which monitors the value of newEventRequired. If it detects that newEventRequired is true, it starts a timer and sets newEventRequired to false."

\[
\text{serviceRequest notNil } \land \lnot \text{newEventRequired} \, \text{unless}
\]

\[
\text{serviceRequest isNil } \land \lnot \text{newEventRequired}
\]  

"AS4-02"
"If a service request has been assigned to the service provider simulator and newEventRequired is false, then newEventRequired remains false while the service request is still assigned to the service provider simulator."
"This has the effect that this simulator will not start another timer before the servicing of the current service request has been completed."

```
generatingEvent ∧ serviceRequest notNil ensures (serviceRequest connection)
postconditions: (#terminate:) withArguments: #('completed') ∧ serviceRequest isNil ∧ ¬generatingEvent
"AP1-01"
```

"If a service provider simulator has to generate an event, it ensures that the service provider simulator terminates the connection currently associated with it and becomes available to service a new service request."

"Note: The parent class, viz. EventSimulator, contains a parallel method which sets generatingEvent to true when a timer has expired."

```
✓∀ aServiceRequest where serviceRequest = aServiceRequest ::
serviceRequest = aServiceRequest ensures (serviceRequest connection)
postconditions: (#terminate:) withArguments: #('completed') ∧ serviceRequest isNil
> "AP1-02"
```

"A service request remains assigned to a service provider simulator until the latter completes the service and terminates the connection."

```
invariant categoryIndex ≥ 0 ∧
categoryIndex < nrOfCategoriesServed
"DS2-01"
```

"The categoryIndex is always greater than or equal to zero and less than nrOfCategoriesServed."

```
invariant categoriesServed notNil ∧
categoriesServed class = OrderedCollection
"DS2-02"
```

"Once categoriesServed has been initialized to refer to an instance of the OrderedCollection class, it is never set to nil while the ServiceProviderSimulator instance exists."

```
∀ categoryIndex where 0 ≤ categoryIndex ∧
categoryIndex < nrOfCategoriesServed ::
¬(self canAcceptNextSR:
(categoriesServed at: (categoryIndex + 1)))
leads-to
self canAcceptNextSR:
(categoriesServed at: (categoryIndex + 1))
> "DL2-01"
```

"For any service category serviced by the service provider simulator, the service provider simulator will eventually be able to service a request from that service category."
class methods

message pattern startSimulation: scContainer using: aConfiguration

method properties
"The initialize method of the superclasses sets newEventRequired
to false. The startSimulation:using: method invokes the
initialize message of its superclass and initialises the
ServiceProviderSimulator-specific instance variables. The
properties of the initialize method of the superclass hold.
"Total correctness"
true results-in methodReturnValue notNil ∧
super postconditions: (#initialize) ∧
serviceRequest isNil ∧
serviceProviderCategory notNil ∧
categoriesServed notNil ∧
.nrOfCategoriesServed ≥ 0 ∧
categoryIndex ≥ 0
"AL1-01"

"Instance creation results in the initialization of the instance variables of the
ServiceProviderSimulator class and its superclasses.
sequential
^((super new initialize) moreInit: scContainer using: aConfiguration)
end-sequential

instance methods

message pattern moreInit: scContainer using: aConfiguration
"Initializes the ServiceProviderSimulator instance"

method properties
"Total correctness"
true results-in methodReturnValue = self ∧
serviceRequest isNil ∧
aConfiguration postconditions: (#assignSPCategory) ∧
serviceProviderCategory notNil ∧
categoriesServed notNil ∧
selves postconditions: (#registerServiceProvider: using:) withArguments: #(scContainer aConfiguration)
"DL1-01"

"The initialization that is performed during instance creation results in the service
provider simulator being available to provide service and in the service provider
simulator being registered with each service category that has a matching service
provider category in its service provider categories component."
sequential
serviceRequest := nil
serviceProviderCategory := aConfiguration assignSPCategory
categoriesServed := OrderedCollection new
categoriesServed := OrderedCollection new
self registerServiceProvider: scContainer using: aConfiguration
end-sequential

message pattern serviceProviderCategory

method properties
"Total correctness"
true results-in methodReturnValue = serviceProviderCategory
"DL1-02"
sequential
"serviceProviderCategory"
end-sequential

message pattern serviceRequest
method properties
"Total correctness"
true results-in methodReturnValue = serviceRequest
"DL1-03"
sequential
^ serviceRequest
end-sequential

category testing
canAcceptNextSR: requestingServiceCategory
method properties
"Total correctness"
true results-in
methodReturnValue = ((requestingServiceCategory =
categoriesServed at: (categoryIndex + 1)) ^
(serviceRequest isNil))
"DL1-04"
sequential
^ (serviceRequest isNil)
  if requestingServiceCategory =
categoriesServed at: categoryIndex + 1 -
^false
  if requestingServiceCategory ==
categoriesServed at: categoryIndex + 1
end-sequential

category modifying
registerServiceProvider: scContainer
  using: aConfiguration
"Registers the ServiceProviderSimulator with the relevant service categories"
method macros
maxCategories = aConfiguration maximumServiceCategories
method properties
"Total correctness"
true results-in methodReturnValue = self ^
<VaServiceCategory where
  scContainer includes: aServiceCategory ^
  aServiceCategory servicedBy: serviceProviderCategory ::
  aServiceCategory postconditions: (#addSP:)
withArguments: #(self) ^
categoriesServed includes:
  (aServiceCategory serviceCategory)
> ^
nrOfCategoriesServed = categoriesServed size ^
categoryIndex ≥ 0
"DL1-05"
sequential
nrOfCategoriesServed := 0
[] < [] j where 1 ≤ j ≤ maxCategories ::
(scContainer at: j) addSP: self \+
nrOfCategoriesServed := nrOfCategoriesServed + 1 \+
categoriesServed addLast:((scContainer at: j) serviceCategory)
> 
[] categoryIndex := 0
end-sequential
message pattern processServiceRequest: aServiceRequest
method properties
"A new simulation is required each time when a new service request is processed."
"Total correctness"
∀ x where 0 ≤ x ∨ x < nrOfCategoriesServed ::
categoryIndex = x ∧
aServiceRequest notNil ∧
serviceRequestCategory := (aServiceRequest serviceRequestCategory)
results-in
  methodReturnValue = self ∧
  serviceRequest = aServiceRequest ∧
  newEventRequired ∧
  categoryIndex = (x + 1) \ nrOfCategoriesServed
"DL1-06"
sequential
  newEventRequired := true
 parallel
    serviceRequest := aServiceRequest \\
    categoryIndex := (categoryIndex + 1) \ nrOfCategoriesServed
end-parallel

message pattern p_generateEvent
method properties
"The newEventRequired attribute is not updated here. It is only set to true once a new service request has been received."
generatingEvent ∧ serviceRequest notNil ensures
  (serviceRequest connection)
  postconditions: (#terminate:)
  withArguments: #('completed') ∧
  serviceRequest isNil ∧ ~generatingEvent
"DP1-01"
parallel
  (serviceRequest connection) terminate: 'completed' \+
  serviceRequest := nil \+
  generatingEvent := false
  if generatingEvent
end-parallel

message pattern p_updateCategoryIndex: scContainer
method properties
∀ x where 0 ≤ x ∨ x < nrOfCategoriesServed ::
categoryIndex = x ∧
(scContainer detect:[:each |
  (each serviceQCategory =
    (categoriesServed at: categoryIndex))
  and: [each serviceQ isEmpty] ifNone: [nil] notNil]
ensures
categoryIndex = (x + 1) \ nrOfCategoriesServed
"DP1-02"
"If the serviceQ of the service category matching the categoriesServed entry at the current categoryIndex+1 is empty, the categoryIndex is incremented modulo nrOfCategoriesServed."
parallel

categoryIndex := (categoryIndex + 1) \ nrOfCategoriesServed
if [scContainer detect[:each |
  (each serviceQCategory =
    categoriesServed at: (categoryIndex+1))
  and: [each serviceQ isEmpty]] ifNone: [nil] notNil
end-parallel
BIBLIOGRAPHY


Bjor99  Bjorner, D. Where do software architectures come from? A systematic
development from domains and requirements; a re-assessment of software
eering. South African Computer Journal (SACJ), No. 22, pp. 3 - 13, March
1999.

software architecture: a system of patterns. John Wiley & Sons Ltd, England,
1996.

Butl99  Butler, M. csp2B: A practical approach to combining CSP and B. Formal
methods 1999, Vol. 1, LNCS 1708, pp. 490-508, Toulouse, France, September 20-
24, 1999. J. Wing, J. Woodcock, J. Davies (Eds). Springer-Verlag Berlin,
Heidelberg 1999.


ChCh99  Charpentier, M. and K. M. Chandy. Towards a compositional approach to the
design and verification of distributed systems. Formal methods 1999, Vol. 1,

ChMi88  Chandy, K.M. and J. Misra. Parallel program design: a foundation. Addison-

CHYLS-Web  Chung, P.E., Y. Huang, S. Yajnik, D. Liang, J.C. Shih, C.-Y. Wang and Y.-M.
Wang. DCOM and CORBA side by side, step by step and layer by layer. On-line

Dijk76  Dijkstra, E.W. A discipline of programming. Prentice-Hall International,

Dijk78  Dijkstra, E.W. Two starvation free solutions to a general exclusion problem.
EWD 625, Plataanstraat 5, 5671 Al Nuenen, The Netherlands.

EnKo98  Engelbrecht, R.L. and D. Kourage. Issues in translating Smalltalk to Java. In:
Compiler construction (Ed. Kai Koskimies). Proceedings of the 7th international

FGHVE96  Fang, W., S. Guyet, R. Haven, M. Vilmi and E. Eckmann. VisualAge for
Smalltalk Distributed – developing distributed object applications. Prentice-Hall,

Fran92  Francez, N. Program verification. Addison-Wesley, 1992.

GeRo89  Gehani, N. and W.D. Roome. The Concurrent C programming language. Prentice-
Hall, 1989.

GHJV95  Gamma, E., R. Helm, R. Johnson and J. Vlissides. Design patterns, elements of

GoRo89  Goldberg, A. and D. Robson. Smalltalk-80, the language. Addison-Wesley, 1989.

Grie96  Gries, D. The need for education in useful formal logic. An invitation to formal methods, Computer, pp. 29 - 30, April 1996.


Jone96 Jones, C. B. A rigorous approach to formal methods. An invitation to formal methods, Computer, pp. 20 - 21, April 1996.


Keen89 Keene, S.E. Object-oriented programming in Common Lisp - A programmer's guide to CLOS. Addison-Wesley, 1989.


Lott90  Lott, C. M. Correctness is congruent with quality. ACM SIGSOFT, Vol. 15, No 5, pp. 19-20, October 1990.


425


Seli93 Selic, B. An efficient object-oriented variation of the statecharts formalism for distributed real-time systems. Submitted to IFIP Conference on hardware description languages and their applications, April 26 - 28, 1993, Ottawa, Canada.


<table>
<thead>
<tr>
<th>Bib ID</th>
<th>Author(s)</th>
<th>Title</th>
<th>Source</th>
</tr>
</thead>
</table>