SpecSim
Programmer's Manual

— Version 1.0 —

A Simulation Engine for the SpecC Language

Jianwen Zhu
Electrical and Computer Engineering
University of Toronto

Contents

1

Component Object Model

com.h

Names

The Component Object Model (COM) is an architecture and infrastructure for building fast, robust, and extensible component-based software. SpecSim uses COM model to specify, document, and implement various APIs. The complete COM specification is available from Microsoft's web site (`http://www.microsoft.com/com/`).

At its lowest level, COM is merely a language-independent binary-level standard defining how software components within a single address space can rendezvous and interact with each other efficiently, while retaining a sufficient degree of separation between these components so that they can be developed and evolved independently. To achieve this goal, COM specifies a standard format for *dynamic dispatch tables* associated with objects. These dispatch tables are similar in function to the virtual function tables ("vtables") used in C++, but they are specified at the binary level rather than the language level, and they include additional functionality: in particular, a standardized run-time type determination ("narrowing") facility, and reference counting methods. This minimal basis allows a software component to dynamically determine the types of interfaces supported by another unknown component and negotiate a common "language" or set of interfaces through which further interaction can take place. ("Parlez vous Fran cais? Sprechen Sie Deutsch?") COM builds a whole range of services on top of this basic facility, such as cross-address-space RPC (MIDL), object linking and embedding (OLE), scripting (OLE Automation), etc. However, it is primarily this lowest-level facility that is relevant to us.

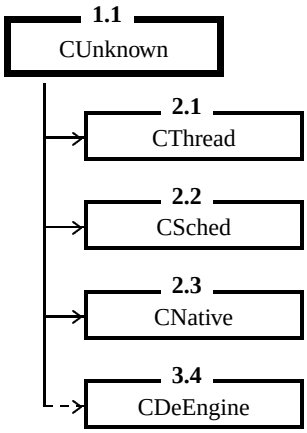
See Also: also www.microsoft.com/com
Author: Jianwen Zhu
Version: 1.0

1.1

class **CUnknown**

IUnknown interface wrapper

Inheritance



2 Native Thread API

native.h

Names

The purpose of the native thread API is to abstract away the implementation detail of different multi-threading packages we might use, for example, the POSIX, the QuickThread package among others. While sharing a similar functionality, these packages differ in both performance and portability. Since choices of the multi-threading package to use on different platforms may vary, it is best that we establish an multi-threading API that can wrap around different packages. Furthermore, since the functionality we need and only need from the thread layer is a simple abstraction of execution context as well as the mechanism to perform non-preemptive context switch, we can use this API to mask away the other functionalities the package may provide, or to create the abstraction using the package's native primitives.

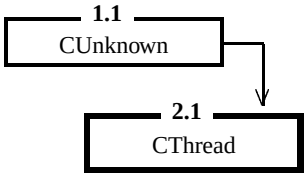
This section describes defines the native thread API.

Author: Jianwen Zhu
Version: 1.0

2.1 class **CThread** : public CUnknown

IThread interface wrapper

Inheritance



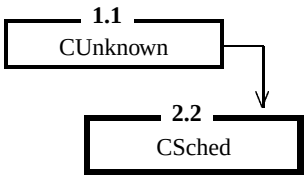
The `IThread` interface abstracts the behavior of the client thread object (user of the native thread API), which essentially defines a set of call back functions for `INative`.

2.2

class **CSched** : public CUnknown

ISched interface wrapper

Inheritance



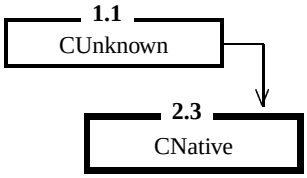
The `ISched` interface abstracts the behavior of the client scheduler object (user of the native thread API), which essentially defines a set of call back functions for `INative`.

2.3

class **CNative** : public CUnknown

INative interface wrapper

Inheritance



The `INative` interface abstracts the functionality of the underlying multithreading package that is needed by SpecC simulation.

3 Discrete Event API

de.h

Names

The Discrete Event (DE) API abstracts the behavior of a discrete event simulation engine, that serves as the basis for VHDL, Verilog and in our case, SpecC.

This section describes the public header file that defines DE API.

Author: Jianwen Zhu
Version: 1.0

3.1 typedef void* DeThread

an opaque data type

The opague data type Thread represents threads that can be executed concurrently. They are usually implemented as the client thread of native thread API by implementing the `IThread` interface.

3.2

typedef void* **DeClosure**

an opaque data type

DeClosure represents light-weight threads. They are essentially functions that can be executed atomically (there is no context switch before the completion of its execution).

3.3

typedef void* **DeEvent**

an opaque data type

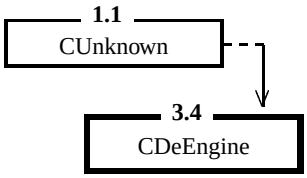
DeEvent represents events. They serves as the synchronization points between thread executions. A thread can *wait* on an event, whose execution will be blocked until the event is *notified*.

3.4

class **CDeEngine** : CUnknown

IDeEngine Wrapper

Inheritance



Public Members

3.4.1

Jerror **start** (void)

initialization

Return Value: 0 if success, -1 if failed

3.4.2

Jerror **end** (void)

finalization

Return Value: 0 if success, -1 if failed

3.4.3

Jerror **newEvent** (EventKind kind, Jint nSrcs, IGeneric* pe)

create an event

Return Value: 0 if success, -1 if failed

3.4.4

Jerror **newValueEvent** (IGeneric src, IGeneric dst, Jint size, DeEvent* pe)

create a valued signal

Return Value: 0 if success, -1 if failed

3.4.5

Jerror **delEvent** (DeEvent e)

delete an event

Return Value: 0 if success, -1 if failed

3.4.6

Jerror **propEvent** (DeEvent esrc, Jint id, DeEvent edst)

propagate one event to another

Return Value: 0 if success, -1 if failed

3.4.7

Jerror **unpropEvent** (DeEvent esrc, DeEvent edst)

cancel the propagation of one event to another

Return Value: 0 if success, -1 if failed

3.4.8

Jerror **newThread** (RunProc run, void* arg, DeThread* pt)

create a thread

Return Value: 0 if success, -1 if failed

3.4.9

Jerror **delThread** (DeThread t)

delete a thread

Return Value: 0 if success, -1 if failed

3.4.10

Jerror **getThreadDoneEvent** (DeThread t, IGeneric* pe)

get the therad completion event

Return Value: 0 if success, -1 if failed

3.4.11

Jerror **goThread** (DeThread t)

execute the therad

Return Value: 0 if success, -1 if failed

3.4.12

Jerror **abortThread** (DeThread t)

abort the thread executition

Return Value: 0 if success, -1 if failed

3.4.13

Jerror **freezeThread** (DeThread t)

freeze the thread executition

Return Value: 0 if success, -1 if failed

3.4.14

Jerror **unfreezeThread** (DeThread t)

resume the executition of a frozen thread

Return Value: 0 if success, -1 if failed

3.4.15

Jerror **wait** (DeEvent e)

block the executition of a thread until an event is notified

Return Value: 0 if success, -1 if failed

3.4.16

Jerror **waitfor** (Time delay)

block the executition of a thread until sometime later

Return Value: 0 if success, -1 if failed

3.4.17

Jerror **notifyDelta** (Jint id, DeEvent e)

notify an event

Return Value: 0 if success, -1 if failed

3.4.18

Jerror **notifyOne** (Jint id, DeEvent e)

notify an event

Return Value: 0 if success, -1 if failed

3.4.19

Jerror **newClosure** (RunProc run, void* arg, IGeneric* pt)

create a closure

Return Value: 0 if success, -1 if failed

3.4.20

Jerror **delClosure** (DeClosure c)

delete a closure

Return Value: 0 if success, -1 if failed

3.4.21

Jerror **waitClosure** (DeClosure c, DeEvent e)

schedule a closure for execution on the notification of an event

Return Value: 0 if success, -1 if failed

3.4.22

Jerror **unwaitClosure** (DeClosure c, DeEvent e)

cancel a closure

Return Value: 0 if success, -1 if failed

3.4.23

Jerror **waitforClosure** (DeClosure c, Time delay)

schedule a closure for execution some time later

Return Value: 0 if success, -1 if failed

3.4.24

Jerror **getCurrentTime** (Time* pt)

get the current simulation time

Return Value: 0 if success, -1 if failed

3.4.25

Jerror **dump** (void)

dump the state of the engine for debugging purpose

Return Value: 0 if success, -1 if failed

4 SpecC Simulation API

specc.h

Names

SpecC defines an application-programming interface (SpecSim API) for simulation. The goal of the simulation API is to separate simulator implementation from compiler implementation. In this way, C++ code produced by any SpecC compiler can be linked with any SpecC API compliant simulation library to produce an executable.

This section describes the public header file that defines SpecSim API.

Author: Jianwen Zhu
Version: 1.0

4.1 class _specc

_specc name space

Names

4.1.1

static void **start** (void)

Finalization

This method initializes SpecSim's private data structure.

Return Value: none

4.1.2

static void **end** (void)

Initialization

This method finializes SpecSim's private data structure.

Return Value: none

4.1.3

static void **abort** (const char* formatString, ...)

Abortion

This method aborts the simulation process with a message

Return Value: none
Parameters: `formatString` — is a printf style format string
`...` — variable length arguments which give the content of the message. Must conform to what have specified in `formatString`.

4.1.4

static void **par** (fork* first, ...)

Parallel execution

This method executes the specified subbehaviors in parallel (fork). It terminates when all the subbehaviors terminates (join).

Return Value: none
Parameters: `first` — the first subbehavior encapsulated in a `fork` object
`...` — a null-terminated list of subbehaviors

4.1.5

static void **pipe** (fork* first, ...)

Pipelined execution

This method executes the specified subbehaviors in piplelined fashion. The method will never return unless it is aborted by an exception initiated by `_specc::tryTrapInterrpt`.

Return Value: none
Parameters: `first` — the first subbehavior encapsulated in a `fork` object
`...` — two null-terminated lists. The first list specifies the rest of subbehaviors. The second list specifies the set of piped variables.

4.1.6

static void **tryTrapInterrupt** (try_block *t, exception_block *f, ...)

Exception handling

This function executes a behavior with exception handlers and interrupt handles set up. During the execution of the behavior, if an exception (one of the events specified in the `_specc::exception_block` object) occurs, depending upon the nature of the exception, different action will be taken. If it is an trap, the behavior will be aborted and the exception handlers with be executed. If it is an interrupt, the behavior will be freezed until the corresponding interrupt handler completes its execution.

Return Value: none
Parameters: `t` — the `_specc::try_block` object which encapsultes the information on the behavior to be executed and monitored.
`e` — the `_specc::exception_block` object which encapsulates the information of the exception handler.
`...` — a null-terminated list of exception handlers.

4.1.7

static void **waitfor** (Time delay)

Delayed Execution

This method delays the execution of the calling behavior for the specified amount of simulation time.

Return Value: none
Parameters: `delay` — the amount of simulation time that the behavior should be delayed.

4.1.8

static void **wait** (event* first, ...)

Blocking

This method blocks the execution of the calling behavior until one of the specified events is notified.

Return Value: none
Parameters: `first` — the first events
`...` — a null-terminated list of events

4.1.9

static void **notify** (event *first, ...)

Event notification

This method notifies a set of events so that for each event, all the blocked threads can resume execution.

Return Value: none
Parameters: `first` — the first events
`...` — a null-terminated list of events

4.1.10

static void **notifyone** (event *first, ...)

Event notification

This method notifies a set of events so that for each event, one of the blocked threads can resume execution.

Return Value: none
Parameters: `first` — the first events
`...` — a null-terminated list of events

4.1.11

static Time **getCurrentTime** (void)

Simulation time

This method reports the current simulation time.

Return Value: none
Parameters: `first` — the first events
`...` — a null-terminated list of events

4.2

class _specc::event

SpecC event or signal data type

Public Members

4.2.1

event (void)

SpecC event constructor

This constructor initializes a SpecC event data type.

Return Value: none

4.2.2

event (void* src, void* dst, int size)

SpecC signal constructor

This constructor initializes a SpecC signal (with VHDL signal semantics).

Return Value: none
Parameters: `src` — the placeholder for the current value of the signal
`dst` — the placeholder for the next value of the signal
`size` — the size (number of bytes) of signal value

4.2.3

~**event** (void)

SpecC event or signal destructor

This destructor destroys a SpecC event or signal.

Return Value: none
Parameters: `none` —

4.3

class **_specc::behavior**

Root class of SpecC behavior classes

Public Members

4.3.1

behavior (void)

SpecC behavior constructor

This constructor initializes the SpecC behavaior object

Return Value: none

4.3.2

virtual ~**behavior** (void)

SpecC behavior constructor

This destructor finalizes the SpecC behavaior object.

Return Value: none

4.3.3

virtual void **main** (void)

Function Specification

This method contains the functionality of the SpecC behavior. Note that this method has to be made virtual.

Return Value: none

4.4

class **_specc::channel**

Root class of SpecC channel classes

Public Members

4.4.1

channel (void)

SpecC channel constructor

This constructor initializes the SpecC channel object

Return Value: none

4.4.2

~**channel** (void)

SpecC channel destructor

This constructor finalizes the SpecC channel object

Return Value: none

4.5

class _**specc::fork**

Encapsulator of Concurrent Behavior

Public Members

4.5.1

fork (behavior *b)

SpecC concurrent behavior constructor

This constructor initializes the `fork` object

Return Value: none
Parameters: `b` — the concurrent behavior to be forked

4.5.2

~**fork** (void)

SpecC concurrent behavior destructor

This destructor finalizes the `fork` object

Return Value: none

4.6

class _**specc::exception_block**

SpecC exception handler

Public Members

4.6.1

exception_block (bool isTrap, behavior* b, event* e1, ...)

SpecC exception handler constructor

This constructor initializes the exception handler object.

Return Value: none
Parameters: `isTrap` — indicates whether is a trap (TRUE) or interrupt (FALSE) handler
`b` — the handler behavior
`e1` — the first trigger event
`...` — a null-terminated list of trigger events

4.6.2

~**exception_block** (void)

SpecC exception handler destructor

This destructor finalizes the exception handler object.

Return Value: none

4.7

class **_specc::try_block**

SpecC exception monitor

Public Members

4.7.1

try_block (behavior* b)

SpecC exception monitor constructor

This constructor initializes the exception monitor object.

Return Value: none
Parameters: `b` — indicates the behavior to be monitored

4.7.2

~try_block (void)

SpecC exception monitor destructor

This destructor finalizes the exception monitor object.

Return Value: none

Class Graph