TOVEsim: A Distributed Simulator for TOVE

John F. Chionglo and Chris Beck

Enterprise Integration Laboratory
Department of Industrial Engineering
University of Toronto
4 Taddle Creek Road
Toronto, Ontario Canada M5S 1A4
tel: (416) 978-6823
email: chionglo@ie.utoronto.ca

Introduction

TOVEsim is a distributed simulator. It provides users with an environment for performing distributed simulation. Distributed simulation is the execution of discrete event simulation programs on more than one computer. This document describes the C/C++ version of TOVEsim.

The Distributed Nature of Enterprises

We view the enterprise as a set of independent and asynchronous entities that interact with each other through message sending. We model each entity as separate software process or logical process called an agent. Interactions between entities are modeled by time stamped messages exchanged between agents. The time stamp associates the point in simulated time when the event occurs at the receiving agent.

To illustrate the distributed nature of an enterprise, consider the supply chain function of a manufacturing company. The supply chain functions span the operations of an enterprise from order entry, through production planning and control, and to distribution].

The supply chain consists of several agents such as an order entry (OE) agent, a resource manager (RM), a scheduler (ODO), and a dispatcher (DISP). When an order is placed by a customer through the order entry agent, several interactions take place between the agents before the order is accepted or denied. Such a scenario may look like this:

When an order is ‘generated’ (1), the OE agent asks the RM agent if the order can be met (2). The RM agent decides whether or not the order is satisfied from inventory, production or both. If production is needed, the RM agent asks ODO if it is possible to schedule the activities needed to satisfy the outstanding order (3). If ODO decides that it can schedule these activities, it notifies the RM agent that we can fit the activities in our current schedule (4). The RM agent then tells the OE agent that we can meet the order (5). The OE agent confirms the customer order (6), the RM agent confirms the activities (7), and ODO confirms the schedule (8).

FIGURE 1. ISCM Agents

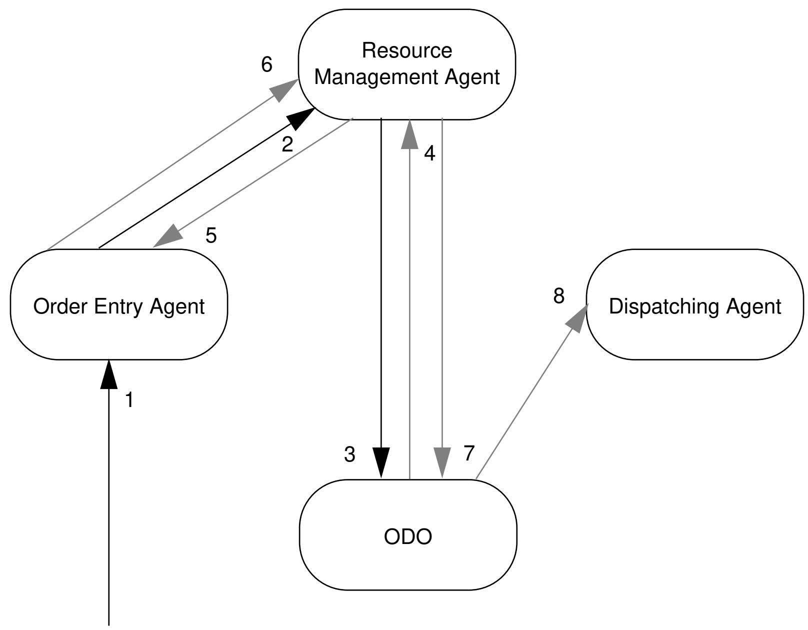


TABLE 1. Interaction Summary Between OE, RM, ODO, and DISP agents

message #	message content
1	incoming order
2	can we meet the order (assume RM agent responsible for resource allocation)?
3	can we schedule and meet order requirements?
4	yes, we can.
5	yes, we can.
6	order accepted.
7	confirm request.
8	new schedule.

TOVE is a simulation model of such an enterprise. The model was created using the ontologies created for the representation of enterprise wide knowledge [@refer(TOVE manual)]. TOVEsim is the simulation executive for TOVE.

Problems with Distributed Simulation

On a computer, discrete event simulation is performed as follows [@acknowledge(reed.fujimoto.87)]:

- There is a single queue of events and a single clock where time is maintained.
- The events are processed in increasing order of time.
- An event can cause a change in the state variables and/or generate new events that will occur in the future.
- The processing of an event causes an update on the clock to the time of the event being processed
- New events are inserted to the event queue and the simulation proceeds until the event queue is empty.

In a distributed model like ours, there is the problem of ensuring the execution of messages in proper time sequence. Since each agent can receive timestamped messages from other agents, we must ensure that these messages are processed in proper order. If we can ensure that messages are processed in non-decreasing timestamp order within an agent and if all interactions between agents are through timestamped messages, then these are sufficient to ensure global causality.

Distributed Simulation Strategies

There are three main strategies that address our problem [@acknowledge(reed.fujimoto.87)]. In ensuring that messages within an agent are processed in proper time sequence, deadlock situations occur for two of the strategies. A deadlock situation occurs when an agent blocks itself from other agents while waiting to receive new messages from neighboring agents.

- Strategy 1 avoids deadlock by generating “NULL” messages to distribute information concerning the simulated time of neighboring agents.
- Strategy 2 allows deadlock to occur but provides a mechanism to detect and recover from deadlock situations.
- Strategy 3 avoids deadlock by allowing agents to process events on any non-empty queue, regardless of the number of other input queues that are empty. This can lead to violation of local causality, so an additional rollback mechanism is provided. The rollback mechanism retracts the erroneous computations by returning to some point before the causality constraint was violated, giving the agent another chance to perform the computation correctly.

There are several assumptions made for each of the strategies enumerated above. The following table summarizes these assumptions.

TABLE 2. Strategies and Their Assumptions

Assumptions	Strategies 1 and 2	Strategy 3 (Time Warp mechanism)
messages sent from one process to another arrive in the same order.	yes.	no.
messages are transmitted through the network without error, and no messages are lost.	yes.	yes.
sufficient memory is available on each processor to hold the simulation program, message queues, local variables, etc.	yes.	yes.
the timestamps on messages transmitted from one process to another form a non-decreasing sequence. This is emphasized in the deadlock avoidance approach to require the sequence be strictly increasing.	yes.	yes, however, processes cannot generate messages with timestamps less than its local clock.
all computation occurs as the result of processing a received message. No computation can "spontaneously" begin.	yes.	yes.
the timestamp of each message must be at least as large as the timestamp of the message that initiated the computation generating the message.	yes	no, the timestamp of each message must be at least as large as the Global Virtual Time (GVT).
the communication paradigm is assumed to be synchronous where the process sending a message must block until the receiver executes the corresponding receive primitive.	yes	no, the communication paradigm is assumed to be asynchronous.
logical processes are not created dynamically once the simulation begins.	yes, dynamic creation of processes is difficult to implement because each process must know which other processes can send it a message that could violate local causality constraints.	no, dynamic creation of processes is allowed. However, the time warp mechanism does not address this issue.

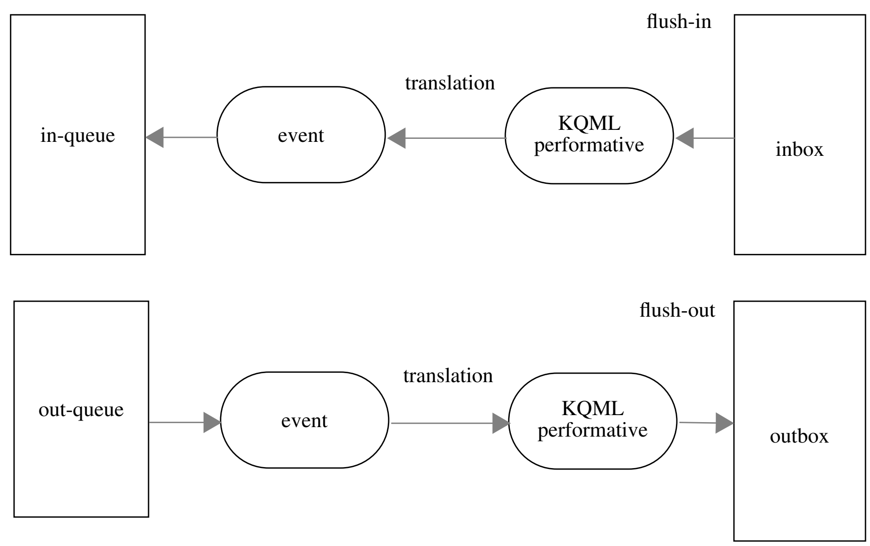
TOVEsim

Currently, TOVEsim is a simple event queue manager which communicates with a global event queue manager (called the facilitator). When using TOVEsim, we are basically performing multi-process simulation (as opposed to distributed simulation). There is a global event queue of events where all events are scheduled. This global event queue is maintained by the facilitator.

In the following sections, we will describe the entities in our simulation environment (i.e. event, agent, and facilitator), explain the simulation process (i.e. starting, executing, and terminating a simulation), and illustrate the creation of a simulation with an example.

Messages, Events and Performatives

FIGURE 2. Events and Performatives



We define an event as any timestamped message sent by an agent to another agent or to itself.

The protocol used for inter-agent communication is KQML. The messages that an agent sends to and receives from another agent is known as a performative. In this document, a performative will also be referred to as a remote event or just an event.

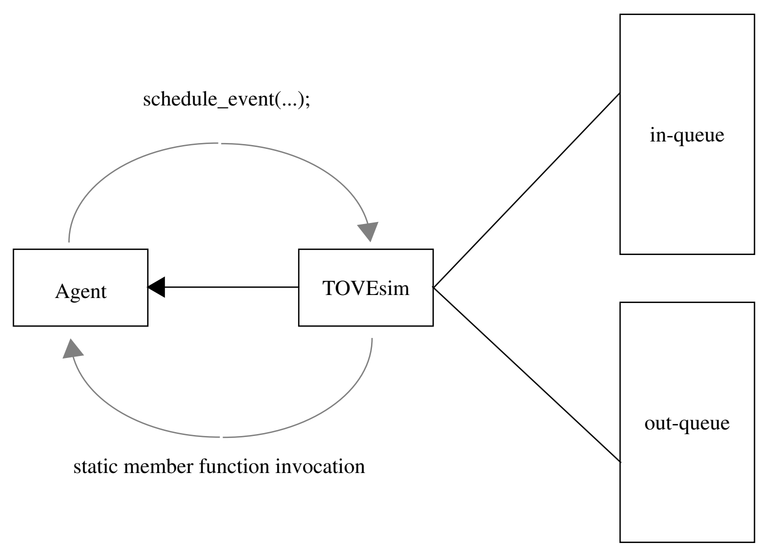
When a performative 'arrives', it is converted into an event by TOVEsim. Every piece of information in a performative is present in an event. Similarly, when a remote event is to be sent out, it is converted into a performative by TOVEsim before sending it to the receiving agent. Hence, a performative and an event differs only on how they are implemented.

The differences between a remote event and a local event are:

- The sender and receiver slots of a remote event are the string names of the respective agents. The sender slot of a local event is the string name of the agent while the "receiver" slot is a pointer to a C/C++ instance in the 'sending' agent's program.
- For each local event scheduled, a corresponding remote event is created by TOVEsim++ for scheduling to the facilitator. This special remote event's sender and receiver refer to the same agent. The 'content' slot of this special event is a pointer to the instance of the local event.
- The 'function/message' slot of a remote event is a KQML performative type. For a local event, this slot is a C/C++ pointer to a static member function of the 'receiver'.

Agents

FIGURE 3. TOVEsim and Agents



Each process in the simulation is identified as an agent. Each agent has a unique name associated with it. When an agent communicates with another agent, it addresses the receiving agent by this name and vice versa.

Agents do not manipulate performatives and they do not monitor their in-queue. They send and receive messages to and from TOVEsim. In particular, they schedule events (local or remote) through TOVEsim. TOVEsim takes care of the converting the events into performatives (when events are outgoing) or converting performatives into events (when events are incoming).

All ‘incoming’ events are placed in the in-queue. TOVEsim ‘pops’ the in-queue in increasing timestamp order and sends the message to the agent directly (i.e. invokes the corresponding C/C++ function of the agent). Each message corresponds to a performative type in KQML (e.g. tell and untell) and every agent developer is responsible for providing a method for each message type. What this means is that each agent must have a C/C++ function associated with each KQML message type which TOVEsim can invoke. (explain local events)

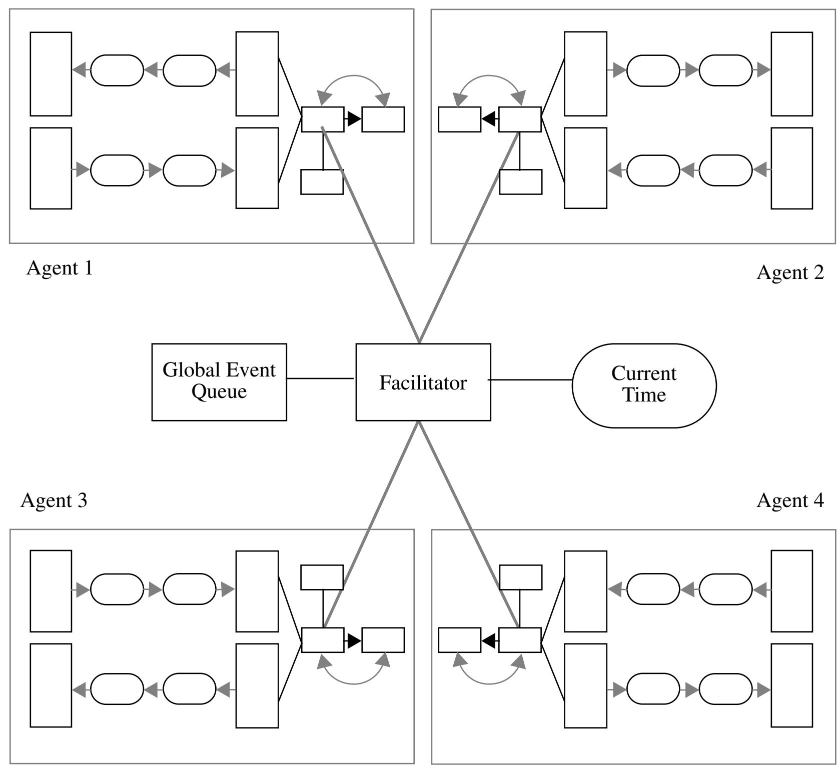
All ‘outgoing’ events are placed in the out-queue. TOVEsim ‘pops’ the out-queue in increasing timestamp order and sends the message to the facilitator. The last message sent out is always an acknowledge performative addressed to the facilitator.

The Facilitator

FIGURE 4. The Facilitator

The Facilitator is a separate process executing concurrently with the agents. Its main function is to sort events scheduled by agents in increasing order of receive (execution) time and send these messages to the respective agents. After sending a message, the facilitator waits for the receiving agent’s acknowledge performative before ‘popping’ the next event form the global event queue.

FIGURE 5.



A simulation is started as follows:

- Agents register to the facilitator.
- Once all agents have registered successfully, the facilitator sends out 'go' messages to each agent to start the simulation. While waiting for the 'go' message from the facilitator, agents may initialize their state/database and schedule 'seed' events (the 'seed' events are not actually sent out to the facilitator until the 'go' message has been received by the agent).

A simulation is executed as follows:

- When an agent receives an event from the facilitator, it is executed. After executing the event, the agent sends an acknowledge message addressed to the facilitator. During the execution of this event, other events may be scheduled by the agent to the facilitator. These events are sent out before sending the acknowledge.

TOVEsim C++ Implementation

There are one initialization function: *tovesim_initialize()*, a global pointer: *TOVEsim* and three objects implemented as C++ classes: *event*, *simuO*, and *agent* in the simulation package.

TOVEsim

Purpose

Global pointer of type *simuO*.

Synopsis

#include "/ie1/msfgrp/tove/current/include/tovesim++.h"

simuO **TOVEsim*;

tovesim_initialize

Purpose

Initializes "global" parameters for distributed simulation.

Synopsis

int tovesim_initialize(int argc, char **argv);

Description

This function currently checks for the use of a local or a global facilitator. If a '-l' flag is present in argv, then the local facilitator will be recognized by TOVsim; otherwise, it defaults to the global facilitator.

event

Purpose

event class

Synopsis

#include "/ie1/msfgrp/tove/current/tovesim++/eventQ.h"

Description

The *event* class contains all information for "executing" a time-stamped "message". They are seldom manipulated directly by the user except when retrieving or initializing member values. The creation of event instances are done through the scheduling interface provided in the *simuO* class.

isLocal

Purpose

Returns the value of the member variable *etype*.

Synopsis

event_type event::isLocal(void)

Return Value

Returns an enumerated type: REMOTE, LOCAL, INVALID

set_send_time

Purpose

Assigns a new value to the member variable *send_time*.

Synopsis

void event::set_send_time(double *st*)

Description

send_time is assigned the value *st*. *Warning. Using this member function by the user may result in an invalid simulation.*

set_receive_time

Purpose

Assigns a new value to the member variable *receive_time*.

Synopsis

void event::set_receive_time(double *rt*)

Description

receive_time is assigned the value *rt*. *Warning. Using this member function by the user may result in an invalid simulation.*

set_force

Purpose

Assigns a new value to the member variable *force*.

Synopsis

void event::set_force(int *f*)

Description

force is assigned the value *f*.

set_reply_with

Purpose

assigns a new value to the member variable *reply_with*.

Synopsis

void event::set_reply_with(char* *s*)

Description

Copies *s* to *reply_with*.

set_sender

Purpose

Assigns a new value to the member variable *sender*.

Synopsis

void event::set_sender(char **s*)

Description

Copies *s* to *sender*.

set_receiver

Purpose

Assigns a new value to the member variable *receiver.*

Synopsis

void event::set_receiver(char **s*)

Description

Copies *s* to *receiver.*

set_function

Purpose

Assigns a new value to the member variable *function*.

Synopsis

void event::set_function(EventProc *p*)

Description

function is assigned the value *p*.

set_in_reply_to

Purpose

Assigns a new value to the member variable *in_reply_to.*

Synopsis

void event::set_in_reply_to(char **s*)

Description

Copies *s* to *in_reply_to*.

set_comment

Purpose

Assigns a new value to the member variable *comment*.

Synopsis

void event::set_comment(char **s*)

Description

Copies *s* to *comment*.

set_aspect

Purpose

Assigns a new value to the member variable *aspect*.

Synopsis

void event::aspect(char **s*)

Description

Copies *s* to *aspect*.

set_name

Purpose

Assigns a new value to the member variable *name*.

Synopsis

void event::set_name(char **s*)

Description

Copies *s* to *name*.

set_to

Purpose

Assigns a new value to the member variable *to*.

Synopsis

void event::set_to(char **s*)

Description

Copies *s* to *to*.

set_from

Purpose

Assigns a new value to the member variable *from*.

Synopsis

void event::set_from(char **s*)

Description

Copies *s* to *from*.

set_ontology

Purpose

Assigns a new value to the member variable *ontology*.

Synopsis

void event::set_ontology(char **s*)

Description

Copies *s* to *ontology*.

set_language

Purpose

Assigns a new value to the member variable *language*.

Synopsis

void event::set_language(char **s*)

Description

Copies *s* to *language*.

set_content

Purpose

Assigns a new value to the member variable *content*.

Synopsis

void event::set_content(char **s*)

Description

Copies *s* to *content* if *language* is not NULL. The user must make sure to set the language first before setting the content. Otherwise, this member function has effect (i.e. if language is NULL, set_content does not set the content).

set_language_and_content

Purpose

Aconvenience function for setting the *language* and *content*.

Synopsis

int event::set_language_and_content(char **l*, char **c*)

Return values

0 - not set; 1 - set.

Description

Copies *l* to *language* and *c* to *content*. if *l* is NULL, *c* is not copied to *content* and 0 is returned; otherwise, 1 is returned.

set_priority

Purpose

Assigns a new value to the member variable *priority.*

Synopsis

void event::set_priority(char **s*)

Description

Copies *s* to *priority.*

get_send_time

Purpose

Retrieves the value of the member variable *send_time*.

Synopsis

double event::get_send_time(void)

Returns

Returns the value of *send_time* [1 - DBL_MAX]

get_receive_time

Purpose

Retrieves the value of the member varible *receive_time*.

Synopsis

double event::get_receive_time(void)

Returns

Returns the value of *receive_time* [1 - DBL_MAX]

get_force

Purpose

Retrieves the value of the member varible *force*.

Synopsis

int event::get_force(void)

Returns

Returns the value of *force*.

get_reply_with

Purpose

Retrieves value of the member variable *reply_with*.

Synopsis

char* event::get_reply_with(void)

Returns

Returns the pointer to the string contained in *reply_with*.

get_sender

Purpose

Retrieves value of the member varible *sender*.

Synopsis

char* event::get_sender(void)

Returns

Returns the pointer to the string contained in *sender*.

get_receiver

Purpose

Retrieves value of the member varible *receiver*.

Synopsis

char* event::get_receiver(void)

Returns

Returns the pointer to the string contained in *receiver*.

get_message

Purpose

Retrieves value of the member varible *message*.

Synopsis

char* event::get_message(void)

Returns

Returns the pointer to the string contained in *message*.

get_in_reply_to

Purpose

Retrieves value of the member varible *in_reply_to*.

Synopsis

char* event::get_in_reply_to(void)

Returns

Returns the pointer to the string contained *in_reply_to*.

get_comment

Purpose

Retrieves value of the member varible *comment*.

Synopsis

char* event::get_comment(void)

Returns

Returns the pointer to the string contained in *comment*.

get_aspect

Purpose

Retrieves value of the member varible *aspect*.

Synopsis

char* event::get_aspect(void)

Returns

Returns the pointer to the string contained in *aspect*.

get_name

Purpose

Retrieves value of the member varible *name*.

Synopsis

char* event::get_name(void)

Returns

Returns the pointer to the string contained in *name*.

get_to

Purpose

Retrieves value of the member varible *to*.

Synopsis

char* event::get_to(void)

Returns

Returns the pointer to the string contained in *to*.

get_from

Purpose

Retrieves value of the member varible *from*.

Synopsis

char* event::get_from(void)

Returns

Returns the pointer to the string contained in *from*.

get_ontology

Purpose

Retrieves value of the member varible *ontology*.

Synopsis

char* event::get_ontology(void)

Returns

Returns the pointer to the string contained in *ontology*.

get_language

Purpose

Retrieves value of the member varible *language*.

Synopsis

char* event::get_language(void)

Returns

Returns the pointer to the string contained in *language*.

get_content

Purpose

Retrieves value of the member varible *content*.

Synopsis

char* event::get_content(void)

Returns

Returns the pointer to the string contained in *content*.

simuO

Purpose

simulation object class

Synopsis

#include “/ie1/msfgrp/tove/current/include/simuO.h”

Description

The simulation object class contains all the “housekeeping” functions required to perform discrete-event simulation. Users must instantiate a simulation object and initialize it before running a simulation.

simuO

Purpose

Constructor for instantiating a *simuO* (simulation object).

Synopsis

simuO::simuO(<your_agent> **name*);

simuO::simuO(<your_agent> **name*, double *start_time*);

simuO::simuO(<your_agent> **name*, double *start_time*, EventHook *f*);

Description

simuO is a collection of constructors for the class simuO. <your_agent> must be a subclass of the agent class.

add_method

Purpose

A member function which adds a method to a list of methods.

Synopsis

simuO::add_method(char **mname*, EventProc *method*);

Description

Add_method adds **mname* and *method* to a simulation object’s list of methods. When adding *method* and **mname* to the list, if an old *method* is associated with **mname* it will be replaced by the new *method*. Otherwise, **mname* and *method* will be added.

get_simulation_for

Purpose

Retrieves the agent for which the simulation was created.

Synopsis

agent *simuO::get_simulation_for(void);

Returns

Returns a pointer to the agent of type agent.

get_current_time

Purpose

Retrieves the value of the member varible *current_time*.

Synopsis

double simuO::get_current_time(void)

Returns

Returns the value of *receive_time* [1 - DBL_MAX]

get_start_time

Purpose

Retrieves the value of the member varible *start_time*.

Synopsis

double simuO::get_start_time(void)

Returns

Returns the value of *start_time* [1 - DBL_MAX]

schedule_event

Purpose

schedules an event.

Synopsis

event *simuO::schedule_event(*t*, *relative*, *priority*, *obj*, *func*);
  double *t*;
  my_boolean *relative*;
  int *priority*;
  void **obj*;
  EventProc *func*;

event *schedule_event(t, *relative*, *priority*, *receiver*, *message*);
  double *t*;
  my_boolean *relative*;
  int *priority*;
  char **receiver*;
  char **message*;

event *simuO::schedule_event(*priority*, *obj*, *func*);
  int *priority*;
  void **obj*;
  EventProc *func*;

event *schedule_event(*priority*, *receiver*, *message*);
  int *priority*;
  char **receiver*;
  char **message*;

Description

schedule_event creates an event instance and adds the instance to the simulation object's event queue. The event instance will be executed at time *t* (if relative is MY_FALSE) or *t+current_time* (if relative is MY_TRUE).

The member functions which do not have time as an argument will set the send_time and receive_time as 0.0. These functions will often be used for scheduling events "now" or events created interactively through a GUI. The execution of the event results in the invocation of the method *func* (if it is a local event) or the invocation of the associated method of *message* for *receiver* at another process (if is a remote event).

func must be a C++ static member function. It should also take two arguments: (event *e*, void **obj*) and return a void. *e* is a copy of the actual event invoked and *obj* is a pointer to the C++ instance.

run_simulation

Purpose

Starts the simulation process.

Synopsis

int simuO::start_simulation(void)

Returns

Returns 0.

agent

Purpose

agent class

Synopsis

#include "/ie1/msfgrp/tove/current/include/agent++.h"

Description

Each process is identified as an agent. When agents "communicate", the sending agent addresses the receiving agent by the receiving agent's string name. The *agent* class contains one member variable *name*. Users may specialize the agent to suit their application.

agent

Purpose

Constructor for instantiating a *agent*.

Synopsis

agent::agent(char **s*);

Description

When instantiated, *s* is copied to the agent's member value *name*.

get_name

Purpose

Retrieves value of the member varible *name*.

Synopsis

char *agent::get_name(void)

Returns

Returns the pointer to the string contained in *name*.

agent_default_method

Purpose

Default member function used by *simuO* to "execute" a remote *event*.

Synopsis

static void agent_default_method(event *e*, void **obj*);

Description

Inter-agent communication is performed using a protocol (in our system, using KQML). Each message type (i.e. tell, untell, ...) is associated with a function. For example, if agent 1 tells agent 2 something, agent 2 executes tell by invoking a corresponding function associated with tell. By default, simuO will associate all KQML performative types to this member function. Users may override the default by using *simuO::add_method*.

When invoked, event *e* and a pointer to the agent instance (*obj*) is passed as arguments.

Release Information

The current release of the simulator is implemented using C/C++. A C/C++/ROCK version is available. This version is different from the C/C++ version in that the C++ events are automatically converted to a ROCK event frame.

To use the simulation package, you will need the following files:

path:	**/ie1/msfgrp/tove/current**
header files:	**include/tovesim++.h**
object files:	**lib/tovesim++.o** (C/C++ version) or
	lib/tovesim++_rk.o (C/C++/ROCK version)
ROCK files:	**fds/tovesim.rock** (C/C++/ROCK version only)

An example simulation of a three-airport scenario is in:

/ie1/msfgrp/tove/current/examples/airport_example

Please read the README file for more details about the sample program.

Note: Users must ensure that the ROCK initialization function: *kb_Initialize(int argc, char **argv);* is invoked before using any of the functions in C/C++/ROCK module. If not, a segmentation violation occurs.