Adam Reference Manual

1.0.0

Generated by Doxygen 1.4.1

Tue Jan 29 12:28:34 2008

Contents

Chapter 1

Adam Directory Hierarchy

1.1 Adam Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 2

Adam Namespace Index

2.1 Adam Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Adam Hierarchical Index

3.1 Adam Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Adam Class Index

4.1 Adam Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

Adam File Index

5.1 Adam File List

Here is a list of all documented files with brief descriptions:

Chapter 6

Adam Page Index

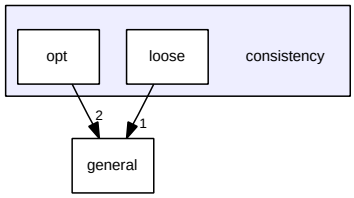
6.1 Adam Related Pages

Here is a list of all related documentation pages:

Chapter 7

Adam Directory Documentation

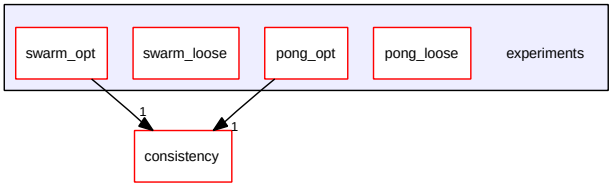
7.1 consistency/ Directory Reference



Directories

- directory **loose**
- directory **opt**

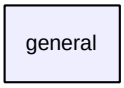
7.2 experiments/ Directory Reference



Directories

- directory **pong_loose**
- directory **pong_opt**
- directory **swarm_loose**
- directory **swarm_opt**

7.3 general/ Directory Reference



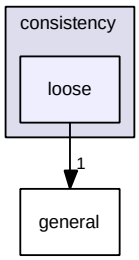
Files

- file **attribute.cc**
- file **attribute.hh**
- file **attribute_factory.cc**
- file **attribute_factory.hh**
- file **consistency.cc**
- file **consistency.hh**
- file **datatypes.h**
- file **disco_meter.cc**
- file **disco_meter.hh**
- file **eye.cc**
- file **eye.hh**
- file **gen.cc**
- file **gen.hh**

  Common includes and definitions for Adam.

- file **host.cc**
- file **host.hh**
- file **instance.cc**
- file **instance.hh**
- file **instance_factory.cc**
- file **instance_factory.hh**
- file **load.cc**
- file **load.hh**
- file **msg.cc**
- file **msg.hh**
- file **object_factory.cc**
- file **object_factory.hh**
- file **polar.cc**
- file **polar.hh**
- file **router.cc**
- file **router.hh**
- file **scenario.cc**
- file **scenario.hh**
- file **sliding_window.cc**
- file **sliding_window.hh**
- file **variance_meter.cc**
- file **variance_meter.hh**
- file **vector.cc**
- file **vector.hh**
- file **world.cc**
- file **world.hh**

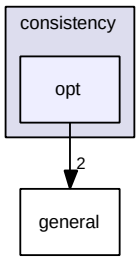
7.4 consistency/loose/ Directory Reference



Files

- file **consistency_loose.cc**
- file **consistency_loose.hh**

7.5 consistency/opt/ Directory Reference



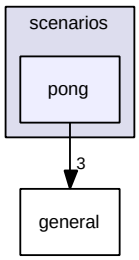
Files

- file **consistency_opt.cc**
- file **consistency_opt.hh**

  This file contains the description of cOptConsistency.

- file **message_hist.cc**
- file **message_hist.hh**
- file **world_opt.cc**
- file **world_opt.hh**

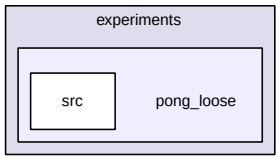
7.6 scenarios/pong/ Directory Reference



Files

- file **ball.cc**
- file **ball.hh**
- file **pong_scenario.cc**
- file **pong_scenario.hh**
- file **racket.cc**
- file **racket.hh**
- file **pong/score.cc**
- file **pong/score.hh**

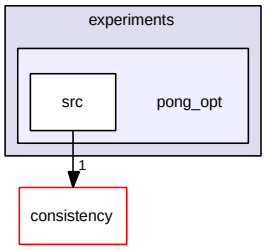
7.7 experiments/pong_loose/ Directory Reference



Directories

- directory **src**

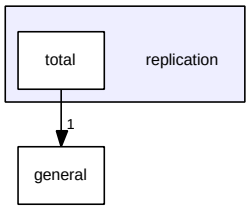
7.8 experiments/pong_opt/ Directory Reference



Directories

- directory **src**

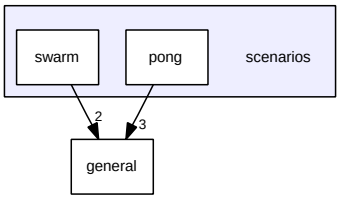
7.9 replication/ Directory Reference



Directories

- directory **total**

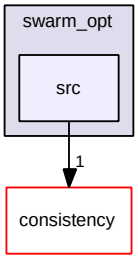
7.10 scenarios/ Directory Reference



Directories

- directory **pong**
- directory **swarm**

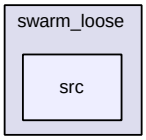
7.11 experiments/swarm_opt/src/ Directory Reference



Files

- file **swarm_opt_host.cc**
- file **swarm_opt_host.hh**

7.12 experiments/swarm_loose/src/ Directory Reference



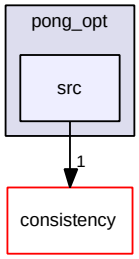
Files

- file **swarm_loose_host.cc**

  The implementation of the queen ownership passing scheme for loose consistency and Swarm.

- file **swarm_loose_host.hh**
- file **swarm_loose_scenario.cc**
- file **swarm_loose_scenario.hh**

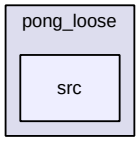
7.13 experiments/pong_opt/src/ Directory Reference



Files

- file **pong_opt_host.cc**
- file **pong_opt_host.hh**

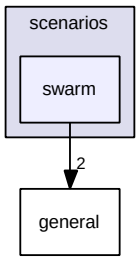
7.14 experiments/pong_loose/src/ Directory Reference



Files

- file **pong_loose_host.cc**
- file **pong_loose_host.hh**
- file **pong_loose_scenario.cc**
- file **pong_loose_scenario.hh**

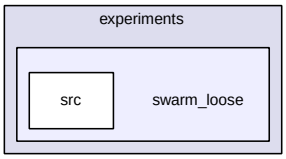
7.15 scenarios/swarm/ Directory Reference



Files

- file **bee.cc**
- file **bee.hh**
- file **queen.cc**
- file **queen.hh**
- file **swarm/score.cc**
- file **swarm/score.hh**
- file **swarm_scenario.cc**
- file **swarm_scenario.hh**

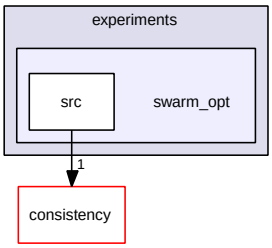
7.16 experiments/swarm_loose/ Directory Reference



Directories

- directory **src**

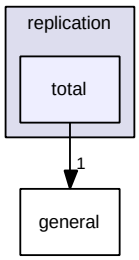
7.17 experiments/swarm_opt/ Directory Reference



Directories

- directory **src**

7.18 replication/total/ Directory Reference



Files

- file **message_router.cc**
- file **message_router.hh**

Chapter 8

Adam Namespace Documentation

8.1 Condel Namespace Reference

The namespace for our project.

Classes

- class **cAttribute**
  A general named variable class.
- class **cAttributeFactory**
  Attribute ID manager.
- class **cConsistency**
  The basic consistency algorithm interface.
- class **cDiscoMeter**
  Computes the discontinuity measure G.
- class **cEye**
  The eye-of-god for global measurements.
- class **cHost**
  The base class for simulation hosts.
- class **cHostComponent**
  A base class for all host-components.
- class **cInstance**
  Base class for all instances.
- class **cInstanceFactory**
  Creates instances.
- class **cLoadMeter**

A CPU/memory load measuring tool.

- class **cMsg**
  The base class for all messages.
- class **cMsgFactory**
  A message factory.
- class **cObjectFactory**
  Object ID manager.
- class **cPolar**
  A two-dimensional vector with polar coordinates.
- class **cRouter**
  The basic netlayer interface.
- class **cScenario**
  The basic scenario interface.
- class **cSlidingWindow**
  Gather statistical data in a sliding window.
- class **cVarianceMeter**
  Computes the divergence measure D.
- class **cVector**
  An n-dimensional numeric vector.
- class **cWorld**
  A set of instances.
- class **cLooseConsistency**
  Loose Consistency.
- class **cOptConsistency**
  Optimistic Consistency.
- class **cMessageHistory**
  An event history.
- class **cOptWorld**
  A world in an optimistic setting.
- class **cBall**
- struct **cRenderVec**
  Helper object for visualization.
- class **cPongScenario**
  The Pong Scenario.

- class **cRacket**

  Pong's racket object.

- class **cScore**

  Pong's score object.

- class **cBee**

  Swarm's bee object.

- class **cQueen**

  Swarm's queen object.

- class **cSwarmScenario**

  The Swarm Scenario.

- class **cSendout**
- class **cBMonitor**
- class **cTimeout**
- class **cMsgRoute**
- class **cSRMonitor**
- class **cMessageRouter**

  The network layer for a flat network model.

- class **cPongLooseHost**

  The host for Pong with loose consistency.

- class **cPongLooseScenario**

  The Pong scenario with loose consistency.

- class **cPongOptHost**

  The host for Pong with optimistic consistency.

- class **cSwarmLooseHost**

  The host for Swarm with loose consistency.

- class **cSwarmLooseScenario**

  The Swarm scenario with loose consistency.

- class **cSwarmOptHost**

  The host for Swarm with optimistic consistency.

Typedefs

- typedef unsigned **ATTRIBUTE_ID**

  Type for attribute IDs.

- typedef unsigned **OBJECT_ID**

Type for object IDs.

- typedef **cVector**< 2, double > **R_2**

  A typical vector.

- typedef **cVector**< 3, double > **R_3**
- typedef **cVector**< 2, INT32 > **Z_2**
- typedef **cVector**< 3, INT32 > **Z_3**

Functions

- void **condel_error** (const char *pWhat)

  Print error and exit.

- void **doPacking** (cCommBuffer *b, **cMsg** &obj)
- void **doUnpacking** (cCommBuffer *b, **cMsg** &obj)

Variables

- class typedef **cInstance** *(* **instanceCreator**)(const **OBJECT_ID** &, const string &)

  An instance creation callback.

8.1.1 Detailed Description

The namespace for our project.

Everything related to Adam is in the Condel namespace, except for a few minor things that cannot because their namespace is set by OMNeT. Also, local objects (visible only within some file) are usually not namespaced.

8.1.2 Typedef Documentation

8.1.2.1 typedef unsigned Condel::ATTRIBUTE_ID

Type for attribute IDs.

This typedef may change in the future, so use this defined type.

8.1.2.2 typedef unsigned Condel::OBJECT_ID

Type for object IDs.

This typedef may change in the future, so use this defined type.

8.1.2.3 typedef cVector<2, double> Condel::R_2

A typical vector.

Most vectors in Adam use this type.

8.1.3 Function Documentation

8.1.3.1 void Condel::condel_error (const char * *pWhat*)

Print error and exit.

Parameters:

pWhat The message shown to the user just before aborting.

Chapter 9

Adam Class Documentation

9.1 Condel::cAttribute Class Reference

A general named variable class.

`#include <attribute.hh>`

Public Member Functions

- **cAttribute** (const **ATTRIBUTE_ID** &id)

  A constructor.

- **cAttribute** (const string &name)

  A constructor.

- **cAttribute** (const **ATTRIBUTE_ID** &id, double value)

  A constructor.

- **cAttribute** (const string &name, double value)

  A constructor.

- **cAttribute** (const **cAttribute** &)
- const **ATTRIBUTE_ID** & **getID** (void) const
- double **getValue** (void) const
- void **setValue** (double value)

9.1.1 Detailed Description

A general named variable class.

An attribute is a tuple $(n, v), n \in N, v \in V_n$ of a name and a value, where N is the set of possible names and V_n the set of possible values; for our purposes, V_n always equals R (in other words, it's type (double)).

9.1.2 Constructor & Destructor Documentation

9.1.2.1 cAttribute::cAttribute (const ATTRIBUTE_ID & *id*)

A constructor.

The value of this attribute will be set to 0.

Parameters:

id The created attribute will have this ID. This must have been previously registered by **cAttributeFactory::registerAttribute(const string &)**(p. 40).

9.1.2.2 cAttribute::cAttribute (const string & *name*)

A constructor.

The value of this attribute will be set to 0.

Parameters:

name The created attribute will have the ID corresponding to name. This must have been previously registered by **cAttributeFactory::registerAttribute(const string &)**(p. 40).

9.1.2.3 cAttribute::cAttribute (const ATTRIBUTE_ID & *id*, double *value*)

A constructor.

Parameters:

id The created attribute will have this ID. This must have been previously registered by **cAttributeFactory::registerAttribute(const string &)**(p. 40).

value The value to initalize the attribute with.

9.1.2.4 cAttribute::cAttribute (const string & *name*, double *value*)

A constructor.

The value of this attribute will be set to 0.

Parameters:

name The created attribute will have the ID corresponding to name. This must have been previously registered by **cAttributeFactory::registerAttribute(const string &)**(p. 40).

value The value to initalize the attribute with.

The documentation for this class was generated from the following files:

- attribute.hh
- attribute.cc

9.2 Condel::cAttributeFactory Class Reference

Attribute ID manager.

`#include <attribute_factory.hh>`

Static Public Member Functions

- static **ATTRIBUTE_ID registerAttribute** (const string &name)
  Create a unique ID for a new attribute.
- static **ATTRIBUTE_ID nameToID** (const string &name)
  Returns the ID of a formerly registered attribute.
- static bool **nameExists** (const string &name)
  Check if an attribute was already registered.

9.2.1 Detailed Description

Attribute ID manager.

This is a purely static class used to globablly assign the IDs for attributes. Also handles the translation of attribute name->attribute ID.

Note that this scheme is a cludge, it would not be possible to do this on a truly distributed system. There are schemes to assign such numbers in a distributed fashion though, and we decided to ignore this sub- problem for greater clarity of the rest of the code.

See also:
cObjectFactory(p. 93)

9.2.2 Member Function Documentation

9.2.2.1 bool cAttributeFactory::nameExists (const string & *name*) [static]

Check if an attribute was already registered.

Parameters:
name The name of the attribute in question.

Returns:
True if this name exists already in the DB.

9.2.2.2 ATTRIBUTE_ID cAttributeFactory::nameToID (const string & *name*) [static]

Returns the ID of a formerly registered attribute.

If the attribute was NOT registered before, the application will exit because of a builtin assert here.

Parameters:
name The name of the existing attribute.

Returns:
The corresponding attribute ID.

See also:
nameExists(p. 39)

9.2.2.3 ATTRIBUTE_ID cAttributeFactory::registerAttribute (const string & *name*) [static]

Create a unique ID for a new attribute.

Parameters:
name The name of the new attribute.

Returns:
The newly assigned attribute ID.

The documentation for this class was generated from the following files:

- attribute_factory.hh
- attribute_factory.cc

9.3 cAttributeSet Class Reference

A set of attributes.

`#include <attribute.hh>`

Public Types

- typedef map< const ATTRIBUTE_ID, **cAttribute** >::**const_iterator const_iterator**
  An iterator.
- typedef map< const ATTRIBUTE_ID, **cAttribute** >::**iterator iterator**
  An iterator.
- typedef map< const ATTRIBUTE_ID, **cAttribute** >::**value_type value_type**
  An iterator.

Public Member Functions

- bool **hasAttribute** (const ATTRIBUTE_ID &id) const
  Check whether a particular attribute exists.
- const **cAttribute** & **get** (const ATTRIBUTE_ID &id) const
  Get an attribute.
- **cAttribute** & **get** (const ATTRIBUTE_ID &id)
  Get an attribute.
- void **add** (const **cAttribute** &attr)
  Add a new attribute.
- unsigned long **getCodedLength** (void) const
  Find out the size of the information in this set.
- **const_iterator begin** (void) const
  Start an iteration.
- **iterator begin** (void)
  Start an iteration.
- **const_iterator end** (void) const
  End of an iteration.
- **iterator end** (void)
  End of an iteration.
- unsigned long **size** (void) const
  Find the size of this set.

9.3.1 Detailed Description

A set of attributes.

This collects several attributes, usually all attributes of a given object.

9.3.2 Member Function Documentation

9.3.2.1 void cAttributeSet::add (const cAttribute & *attr*)

Add a new attribute.

This attribute will be copied. Fails if an attribute with the same name already exists.

Parameters:
attr An attribute. Ownership does not change.

9.3.2.2 cAttribute & cAttributeSet::get (const ATTRIBUTE_ID & *id*)

Get an attribute.

Parameters:
id The idea of the attribute in question.

Returns:
A reference to the attribute.

9.3.2.3 const cAttribute & cAttributeSet::get (const ATTRIBUTE_ID & *id*) const

Get an attribute.

Parameters:
id The idea of the attribute in question.

Returns:
A const reference to the attribute.

9.3.2.4 unsigned long cAttributeSet::getCodedLength (void) const [inline]

Find out the size of the information in this set.

Returns:
The # of bytes the whole information in these attributes will take when coded.

9.3.2.5 bool cAttributeSet::hasAttribute (const ATTRIBUTE_ID & *id*) const

Check whether a particular attribute exists.

Parameters:
id The idea of the attribute in question.

Returns:
True if the attribute exists within the set.

9.3.2.6 unsigned long cAttributeSet::size (void) const [inline]

Find the size of this set.

This function is slow, so cache the results if possible.

Returns:
The amount of attributes in this set.

The documentation for this class was generated from the following files:

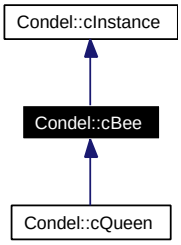
- attribute.hh
- attribute.cc

9.4 Condel::cBee Class Reference

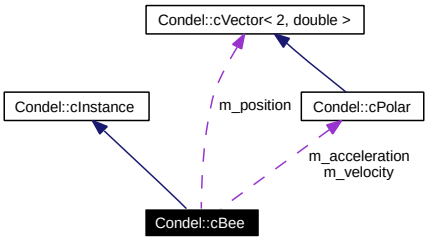
Swarm's bee object.

`#include <bee.hh>`

Inheritance diagram for Condel::cBee:



Collaboration diagram for Condel::cBee:



Public Member Functions

- **cBee** (const **OBJECT_ID** &id, const string &name, unsigned long size=4)
- void **setPosition** (const **R_2** &pos)
- void **setVelocity** (const **cPolar** &v)
- void **setAcceleration** (const **cPolar** &acc)
- const **R_2** & **getPosition** (void) const
- const **cPolar** & **getVelocity** (void) const
- const **cPolar** & **getAcceleration** (void) const
- bool **isAlive** (void) const
- double **getDead** (void) const
- void **setDead** (double time)
- void **addDead** (double time)
- virtual **cAttributeSet** **getAttributes** (void) const

  Get all attributes.

- virtual void **setAttributes** (const **cAttributeSet** &na)

  Set all attributes.

- virtual **cBee** * **clone** (void) const

  Create a deep copy of this instance.

- virtual double **getDistance** (const **cInstance** &other) const

Compares two instances.

- virtual double **getDirection** (const **cInstance** &other) const
  Get the direction to the other bee.

- virtual void **simulate** (double newtime, **cLoadMeter** &load)
  Simulate this instance.

Static Public Member Functions

- static **cInstance** * **create** (const **OBJECT_ID** &id, const string &name)

Static Public Attributes

- static const double **RADIUS** = 10
  Radius of a bee. In pels.

- static const double **ACCELERATION** = 10
  Typical acceleration in pels/s /s.

- static const double **MAX_V** = 200
  Maximum velocity.

- static **ATTRIBUTE_ID** **s_positionXID** = cAttributeFactory::registerAttribute("cBee::positionX")
  X-position of bee.

- static **ATTRIBUTE_ID** **s_positionYID** = cAttributeFactory::registerAttribute("cBee::positionY")
  Y-position of bee.

- static **ATTRIBUTE_ID** **s_velocityAID** = cAttributeFactory::registerAttribute("cBee::velocityAbs")
  X-velocity of bee.

- static **ATTRIBUTE_ID** **s_velocityPID** = cAttributeFactory::registerAttribute("cBee::velocityPhi")
  Y-velocity of bee.

- static **ATTRIBUTE_ID** **s_accelerationAID** = cAttributeFactory::registerAttribute("cBee::accelerationAbs")
  X-acceleration of bee.

- static **ATTRIBUTE_ID** **s_accelerationPID** = cAttributeFactory::registerAttribute("cBee::accelerationPhi")
  Y-acceleration of bee.

- static **ATTRIBUTE_ID** **s_deadID** = cAttributeFactory::registerAttribute("cBee::dead")
  Dead state of bee.

Protected Attributes

- **R_2 m_position**
- **cPolar m_velocity**
- **cPolar m_acceleration**
- double **m_dead**

9.4.1 Detailed Description

Swarm's bee object.

This represents the players.

Note there are *many* bee objects, one for each player.

9.4.2 Member Function Documentation

9.4.2.1 cBee * cBee::clone (void) const [virtual]

Create a deep copy of this instance.

Returns:
An exact copy of this instance, including name. This must not be added to the same **cWorld**(p. 152) that this instance resides in.

Implements **Condel::cInstance** (p. 65).

Reimplemented in **Condel::cQueen** (p. 115).

9.4.2.2 cAttributeSet cBee::getAttributes (void) const [virtual]

Get all attributes.

Packs the current state of this instance into a **cAttributeSet**(p. 41).

Returns:
A copy of the current state.

Implements **Condel::cInstance** (p. 66).

Reimplemented in **Condel::cQueen** (p. 115).

9.4.2.3 virtual double Condel::cBee::getDirection (const cInstance & *other*) const [inline, virtual]

Get the direction to the other bee.

Returns:
The angle I'd have to choose to shoot at the other bee.

See also:
cPolar(p. 102)

9.4.2.4 virtual double Condel::cBee::getDistance (const cInstance & *other*) const [inline, virtual]

Compares two instances.

Normally, this is based on the cartesian distance. For the ideal case, velocity/acceleration is also used.

Todo
Need to fix ideal case here.

Parameters:
other Another instance of the same object.

Returns:
A metric of the similarity of the two instances.

Implements **Condel::cInstance** (p. 66).

9.4.2.5 void cBee::setAttributes (const cAttributeSet & *na*) [virtual]

Set all attributes.

Overwrite the internal state.

Parameters:
na A new attribute set. Every attribute found in na is overwritten locally. It is legal to pass a sparse set here.

Implements **Condel::cInstance** (p. 67).

Reimplemented in **Condel::cQueen** (p. 115).

9.4.2.6 void cBee::simulate (double *newtime*, cLoadMeter & *load*) [virtual]

Simulate this instance.

Accelerate the bee according to its acceleration parameter. Move the bee according to its new speed. Bounce off any walls. Move the b according to its speed. Stop at upper and lower boundaries.

Parameters:
newtime A simulation time in the future of **simTime()**(p. 67). Only the difference between the two times is advanced.
load A tool to measure CPU and memory usage during the simulation.

Implements **Condel::cInstance** (p. 67).

Reimplemented in **Condel::cQueen** (p. 116).

9.4.3 Member Data Documentation

9.4.3.1 const double cBee::MAX_V = 200 [static]

Maximum velocity.

$||v|| \leq$ MAX_V

9.4.3.2 ATTRIBUTE_ID cBee::s_deadID = cAttributeFactory::registerAttribute("cBee::dead") [static]

Dead state of bee.

If not 0, indicates how long dead yet.

The documentation for this class was generated from the following files:

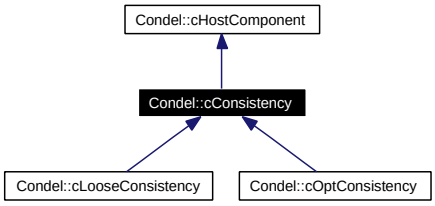
- bee.hh
- bee.cc

9.5 Condel::cConsistency Class Reference

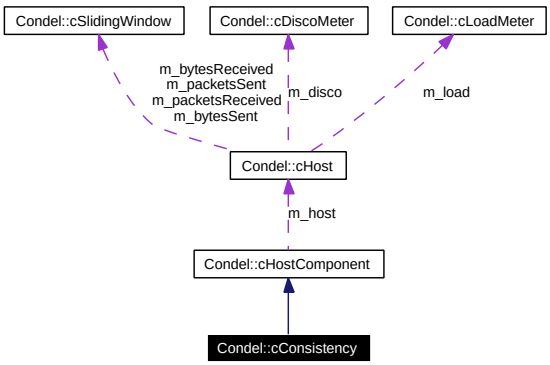
The basic consistency algorithm interface.

`#include <consistency.hh>`

Inheritance diagram for Condel::cConsistency:



Collaboration diagram for Condel::cConsistency:



Public Member Functions

- **cConsistency** (**cHost** &host)
- virtual void **initialize** (void)

  Reinitialize the object.

- virtual void **finish** (void)

  Object is done.

- bool **isReplicated** (const **OBJECT_ID** &objectID) const

  Check for local replication.

- **cInstance** & **getInstance** (const **OBJECT_ID** &objectID)

  Gets a locally replicated instance.

- const **cInstance** & **getInstance** (const **OBJECT_ID** &objectID) const

  Gets a locally replicated instance.

- virtual void **addInstance** (const **cInstance** &instance)=0

  Inserts a clone of this instance into the local world.

- virtual void **changeAttribute** (**OBJECT_ID** obj, **ATTRIBUTE_ID** attr, double val, bool preliminary=false)=0

  Change an (existing) instance's attribute.

- virtual void **updateInstance** (const **OBJECT_ID** &obj)=0

  Synchronizes an existing instance with other hosts.

- virtual void **handleTick** (void)=0

  Called every tick.

- virtual void **handleMessage** (**cMsg** *pMsg)=0

  Receive a message.

Protected Member Functions

- **cConsistency** (const **cConsistency** &)

  Prevent accidental usage of copy constructor.

Protected Attributes

- double **m_lastTickTime**

  The host-time of the last tick handled.

9.5.1 Detailed Description

The basic consistency algorithm interface.

This is the common base class of all actual consistency schemes.

Consistency schemes decide on the strategy to update/receive updates on local events/changes to local state. This can be as simple as loose (simply apply all local changes, send updates to the network, solve conflicts by simple overwriting), or it could be a complicated optimistic approach (apply local changes, send only nondeterministic actions to other hosts, reassemble the same global event history on all hosts, backtrack to synchronize the local state with the global).

9.5.2 Constructor & Destructor Documentation

9.5.2.1 Condel::cConsistency::cConsistency (const cConsistency &) [protected]

Prevent accidental usage of copy constructor.

This is declared, but not defined.

9.5.3 Member Function Documentation

9.5.3.1 virtual void Condel::cConsistency::addInstance (const cInstance & *instance*) [pure virtual]

Inserts a clone of this instance into the local world.

Handles insertion into the whole world (meaning other hosts) if necessary. Plus insertion into local world-copies (like ghosts etc.).

Parameters:

instance An instance. This is copied.

Implemented in **Condel::cLooseConsistency** (p. 76), and **Condel::cOptConsistency** (p. 97).

9.5.3.2 virtual void Condel::cConsistency::changeAttribute (OBJECT_ID *obj*, ATTRIBUTE_ID *attr*, double *val*, bool *preliminary* = false) [pure virtual]

Change an (existing) instance's attribute.

This may or may not happen immediately, depending on the consistency used. This may or may not update other hosts, depending on the consistency used.

Parameters:

obj The ID of the object whose instances are to be changed.

attr The ID of the attribute to change in the instances.

val The new value to set.

preliminary If preliminary is true, the effect is predicted, but the actual information on what actually happened will arrive later from another host.

Implemented in **Condel::cLooseConsistency** (p. 76), and **Condel::cOptConsistency** (p. 97).

9.5.3.3 virtual void Condel::cConsistency::finish (void) [inline, virtual]

Object is done.

Perform all pending calls to OMNeT. Calculate and write final analysis. When this is called, all other objects (and OMNeT) are still existant. ~cConsistency() will be called soon, but by then, much of the environment will be gone.

9.5.3.4 virtual void Condel::cConsistency::handleMessage (cMsg * *pMsg*) [pure virtual]

Receive a message.

This is called from **cRouter::handleMessage(cMsg *)**(p. 123).

Parameters:

pMsg A pointer to the new message. The host retains ownership of the message, do NOT delete it. If you need your own copy, you'll have to cMsg::dup() it.

Implemented in **Condel::cLooseConsistency** (p. 76), and **Condel::cOptConsistency** (p. 97).

9.5.3.5 virtual void Condel::cConsistency::handleTick (void) [pure virtual]

Called every tick.

This allow the consistency-layer to make automatic updates. This in turn will call simulate on the world or instances.

See also:
TICK_TIME

Implemented in **Condel::cLooseConsistency** (p. 77), and **Condel::cOptConsistency** (p. 98).

9.5.3.6 void cConsistency::initialize (void) [virtual]

Reinitialize the object.

Afterwards, same state as if freshly created.

Reimplemented in **Condel::cLooseConsistency** (p. 77), and **Condel::cOptConsistency** (p. 98).

9.5.3.7 bool Condel::cConsistency::isReplicated (const OBJECT_ID & *objectID*) const [inline]

Check for local replication.

Parameters:
objectID The ID of the object to check for.

Returns:
True if an instance of that object replicated locally.

9.5.3.8 virtual void Condel::cConsistency::updateInstance (const OBJECT_ID & *obj*) [pure virtual]

Synchronizes an existing instance with other hosts.

The update may or may not happen immediately (depending on the consistency used).

Parameters:
obj The ID of the object whose instances are synchronized.

Implemented in **Condel::cLooseConsistency** (p. 78), and **Condel::cOptConsistency** (p. 98).

The documentation for this class was generated from the following files:

- consistency.hh
- consistency.cc

9.6 Condel::cDiscoMeter Class Reference

Computes the discontinuity measure G.

`#include <disco_meter.hh>`

Public Member Functions

- void **addAttribute** (const **ATTRIBUTE_ID** &attribute, double weight)

  Add a new attribute to the watchlist.

- void **initWeights** (void)

  Run this once after all attributes have been added.

- void **computeDisco** (const **cInstance** &old, const **cInstance** &updated)

  Tally a discontinuous change.

- void **init** (map< const **ATTRIBUTE_ID**, double > &attributeNorm)

  Reinitialize this tool.

- double **averageDisco** (void) const

  Compute the average so far.

- double **countDisco** (void) const

  Count the changes so far.

- double **averageDisco** (**ATTRIBUTE_ID** aid) const

  Compute the average change in ONE attribute so far.

- double **countDisco** (**ATTRIBUTE_ID** aid) const

  Count the changes in ONE attribute so far.

9.6.1 Detailed Description

Computes the discontinuity measure G.

This tool is activated whenever a message arrives that updates an instance in the visible world. It calculates the size of the change, squares it, counts number and average sizes of such changes internally, supplying them to **cHost**(p. 58) for final analysis.

Changes can affect position as well as speeds, even angles and accelerations. Since these are perceived differently by human observers, each attribute needs to be weighted.

See also:
cVarianceMeter(p. 147)

9.6.2 Member Function Documentation

9.6.2.1 void cDiscoMeter::addAttribute (const ATTRIBUTE_ID & *attribute*, double *weight*)

Add a new attribute to the watchlist.

Parameters:

attribute The ID of the new attribute.

weight The relative importance of the attribute to the overall perception of disontinuity. In definitionen.pdf, this would be w'_a.

9.6.2.2 double cDiscoMeter::averageDisco (ATTRIBUTE_ID *aid*) const

Compute the average change in ONE attribute so far.

Parameters:

aid The ID of the attribute that for which changes are returned.

Returns:

The current average discontinuity for just this one attribute. This does not include the weight, but norm is already factored.

9.6.2.3 double cDiscoMeter::averageDisco (void) const

Compute the average so far.

Returns:

The current average discontinuity. This includes all changes until now, in all attributes, all weighted and added.

9.6.2.4 void cDiscoMeter::computeDisco (const cInstance & *old*, const cInstance & *updated*)

Tally a discontinuous change.

When an instance is changed by an external message, this member is called to calculate the discontinuity that arises (the possibly visible jump or teleportation). All attributes are looked at, each weighted with the weights set in **addAttribute(const ATTRIBUTE_ID &, double)**(p. 54).

Parameters:

old The old version of the instance (before the change).

updated The new version of the instance (after the change).

9.6.2.5 double cDiscoMeter::countDisco (ATTRIBUTE_ID *aid*) const

Count the changes in ONE attribute so far.

Parameters:

aid The ID of the attribute that for which the count is returned.

Returns:

The # of discontinuous changes that have occurred in the specified attribute until now.

9.6.2.6 double Condel::cDiscoMeter::countDisco (void) const [inline]

Count the changes so far.

Returns:

The # of discontinuous changes that have occurred until now. This is more or less just the # of calls to computeDisco(const cInstance &, const cInstance), except that if there appears to be no discernable difference between the instances, the call is not counted.

9.6.2.7 void cDiscoMeter::init (map< const ATTRIBUTE_ID, double > & *attributeNorm*)

Reinitialize this tool.

After this call, the tool looks like right after its creation.

Parameters:

attributeNorm A list of attribute NORMs to use for the attributes. This list is NOT copied, the original version specified here is used. The norms are independant from the weights, they are set by the scenario so that by multiplying it with its attribute, the attribute will have a natural domain of +/- 1. This does not mean the product cannot exceed this range, btw. An example would be the x position, where the weight is 1/width. Another example is score, where weight is 1.

9.6.2.8 void cDiscoMeter::initWeights (void)

Run this once after all attributes have been added.

Only call this ONCE. This fixes the weights to have $\sum w'_a == 1$.

The documentation for this class was generated from the following files:

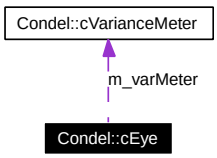
- disco_meter.hh
- disco_meter.cc

9.7 Condel::cEye Class Reference

The eye-of-god for global measurements.

`#include <eye.hh>`

Collaboration diagram for Condel::cEye:



Protected Member Functions

- virtual void **initialize** (void)

  Reset everything.

- virtual void **handleMessage** (cMessage *pMsg)

  Handles messages to eye.

- virtual void **finish** (void)

  System Done.

9.7.1 Detailed Description

The eye-of-god for global measurements.

This class is added to the system just like a host. This allows it to use OMNeT's methods to find all other hosts and survey their data. Thus, this object does something that is never possible in a real network: It looks at the states of all data on all hosts simultaneously. This ability is only used for measurements. The divergence measure, namely, is measured from here.

9.7.2 Member Function Documentation

9.7.2.1 void cEye::finish (void) [protected, virtual]

System Done.

This member is called by OMNeT when the system has ended.

9.7.2.2 void cEye::handleMessage (cMessage * *pMsg*) [protected, virtual]

Handles messages to eye.

This member is called by OMNeT when the internal tick-event-timer is up. Outside of that, you should never send a message to eye.

9.7.2.3 void cEye::initialize (void) [protected, virtual]

Reset everything.

This member is called by OMNeT when the system is about to start.

The documentation for this class was generated from the following files:

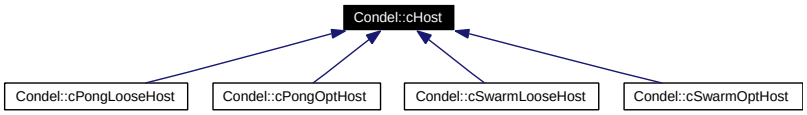
- eye.hh
- eye.cc

9.8 Condel::cHost Class Reference

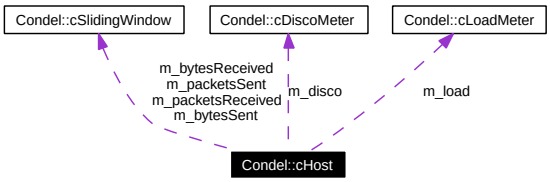
The base class for simulation hosts.

`#include <host.hh>`

Inheritance diagram for Condel::cHost:



Collaboration diagram for Condel::cHost:



Public Member Functions

- virtual const **cMsgFactory** & **getMsgFactory** (void) const
- virtual **cMsgFactory** & **getMsgFactory** (void)
- virtual const **cScenario** & **getScenario** (void) const
- virtual **cScenario** & **getScenario** (void)
- virtual const **cRouter** & **getRouter** (void) const
- virtual **cRouter** & **getRouter** (void)
- virtual const **cConsistency** & **getConsistency** (void) const
- virtual **cConsistency** & **getConsistency** (void)
- virtual const **cWorld** & **getWorld** (void) const
- virtual **cWorld** & **getWorld** (void)
- const **cDiscoMeter** & **getDisco** (void) const
- **cDiscoMeter** & **getDisco** (void)
- **cLoadMeter** & **getLoad** (void)
- virtual void **initialize** (void)

  Reset everything.

- virtual void **finish** (void)

  System Done.

- virtual void **handleMessage** (cMessage *pMsg)

  Accept a message.

- virtual void **handleTick** (void)

  A frame has passed.

- virtual void **handleCheck** (void)

A check interval has passed.

- virtual void **sendOut** (**cMsg** *pMsg, unsigned long id)
  Send out a message to another host.
- virtual void **split** (unsigned long id)
  Perform a network split.
- double **now** (void) const
  The current time.

Static Public Member Functions

- static double **roundTime** (double time)
  Round a time.

Public Attributes

- map< const **ATTRIBUTE_ID**, double > **m_attributeNorm**
  The norms for each attr.

Static Public Attributes

- static const double **TICK_TIME** = 0.05
  Time per frame.
- static const double **CHECK_TIME** = 0.001
  Fastest time-grain.

Protected Attributes

- **cDiscoMeter m_disco**
- list< string > **m_attNames**
- **cLoadMeter m_load**
- map< unsigned long, cGate * > **m_gate**
- cMessage * **m_pTickEvent**
- cMessage * **m_pCheckEvent**
- **cSlidingWindow m_packetsSent**
- **cSlidingWindow m_bytesSent**
- **cSlidingWindow m_packetsReceived**
- **cSlidingWindow m_bytesReceived**
- double **m_lossRate**
- double **m_drops**
- double **m_total**
- scoped_ptr< **cMsgFactory** > **m_factory**

Creates messages.

- scoped_ptr< **cRouter** > **m_router**
  Sending/routing/reception of messages.
- scoped_ptr< **cConsistency** > **m_consistency**
  Consistency algorithm.
- scoped_ptr< **cWorld** > **m_world**
  Currently visible world.
- scoped_ptr< **cScenario** > **m_scenario**
  scenario code

9.8.1 Detailed Description

The base class for simulation hosts.

cHost(p. 58) represents a physical computer, as such it also contains (manages) all the other parts (called components in this context) that run on the individual computers. This class also provides the entire interface to OMNeT.

9.8.2 Member Function Documentation

9.8.2.1 void cHost::finish (void) [virtual]

System Done.

This member is called by OMNeT when the simulation has ended.

Reimplemented in **Condel::cSwarmLooseHost** (p. 137).

9.8.2.2 void cHost::handleCheck (void) [virtual]

A check interval has passed.

This is called at the end of every check_interval, which are CHECK_TIME seconds apart. Called by **handleMessage(cMessage *)**(p. 60).

9.8.2.3 void cHost::handleMessage (cMessage * *pMsg*) [virtual]

Accept a message.

This can either be a network message sent by one of the other simulated hosts or it could be some tick event. This is called by OMNeT (and ONLY by OMNeT, please).

Parameters:

pMsg A pointer to the message. OMNeT (i.e. the caller of this function) retains ownership of the message, do NOT delete it. If you need your own copy, you'll have to dup() it.

Reimplemented in **Condel::cSwarmLooseHost** (p. 137).

9.8.2.4 void cHost::handleTick (void) [virtual]

A frame has passed.

This is called at the end of every frame, which are TICK_TIME seconds apart. Called by **handleMessage(cMessage *)**(p. 60).

Reimplemented in **Condel::cSwarmLooseHost** (p. 137).

9.8.2.5 void cHost::initialize (void) [virtual]

Reset everything.

Sets up the tick events (both tick and check) and reads the parameters from the OMNeT ini files. This member is called by OMNeT when the simulation is about to start.

Reimplemented in **Condel::cSwarmLooseHost** (p. 138), and **Condel::cSwarmOptHost** (p. 142).

9.8.2.6 double cHost::now (void) const

The current time.

Returns:
The current simtime, __ROUNDED__!

9.8.2.7 double cHost::roundTime (double *time*) [static]

Round a time.

This rounds a time to the builtin precision.

9.8.2.8 void cHost::sendOut (cMsg * *pMsg*, unsigned long *id*) [virtual]

Send out a message to another host.

A unicast message is sent to a single other host.

Parameters:
pMsg A pointer to the message to be sent away. CAUTION: The message now belongs to this function (or OMNeT, rather). Do not free/change it, just forget the pointer to it.

id The targeted host.

9.8.2.9 void cHost::split (unsigned long *id*) [virtual]

Perform a network split.

Split the specified host out of the network.

9.8.3 Member Data Documentation

9.8.3.1 const double cHost::CHECK_TIME = 0.001 [static]

Fastest time-grain.

This is used to check for message timeouts and such.

9.8.3.2 map<const ATTRIBUTE_ID, double> Condel::cHost::m_attributeNorm

The norms for each attr.

See also:
cDiscoMeter::init()(p. 55)

9.8.3.3 const double cHost::TICK_TIME = 0.05 [static]

Time per frame.

Once per frame, the world simulation is advanced and the AI is run.

The documentation for this class was generated from the following files:

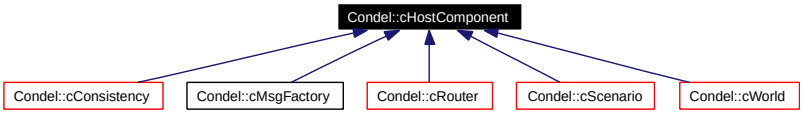
- host.hh
- host.cc

9.9 Condel::cHostComponent Class Reference

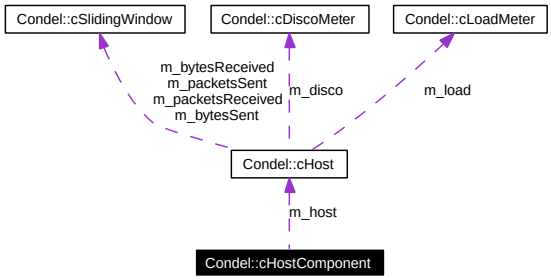
A base class for all host-components.

`#include <host.hh>`

Inheritance diagram for Condel::cHostComponent:



Collaboration diagram for Condel::cHostComponent:



Public Member Functions

- **cHostComponent** (**cHost** &host)
- const **cHost** & **host** (void) const
- **cHost** & **host** (void)

Protected Attributes

- **cHost** & **m_host**

9.9.1 Detailed Description

A base class for all host-components.

The documentation for this class was generated from the following file:

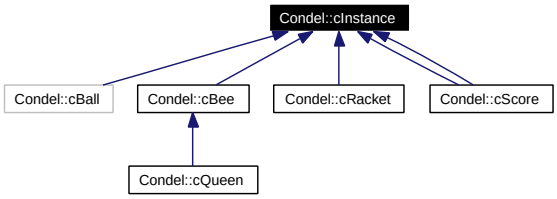
- host.hh

9.10 Condel::cInstance Class Reference

Base class for all instances.

`#include <instance.hh>`

Inheritance diagram for Condel::cInstance:



Public Member Functions

- **cInstance** (const **OBJECT_ID** &id, const string &name, const unsigned size)
  Constructor.
- **OBJECT_ID getObjectID** (void) const
  The ID of the corresponding object.
- const string & **getName** (void) const
  The name of the corresponding object.
- double **simTime** (void) const
  Query the instance's internal simulation time.
- void **setSimTime** (double newtime)
  Set the instance's internal simulation time.
- double **updateTime** (void) const
  Query the instance's last update time.
- void **setUpdateTime** (double newtime)
  Set the instance's last update time.
- virtual **cAttributeSet getAttributes** (void) const =0
  Get all attributes.
- virtual void **setAttributes** (const **cAttributeSet** &na)=0
  Set all attributes.
- virtual **cInstance** * **clone** (void) const =0
  Create a deep copy of this instance.
- virtual double **getDistance** (const **cInstance** &other) const =0
  Compares two instances.

- virtual void **simulate** (double newtime, **cLoadMeter** &load)=0

  Simulate this instance.

- unsigned **getCodedLength** (void)

  Estimate the entire in-memory size of this instance.

9.10.1 Detailed Description

Base class for all instances.

An instance is one version of an object (residing on some world on some host).

See also:
cInstanceFactory(p. 69), **cObjectFactory**(p. 93)

9.10.2 Constructor & Destructor Documentation

9.10.2.1 cInstance::cInstance (const OBJECT_ID & *id*, const string & *name*, const unsigned *size*)

Constructor.

Note that this is not usually called directly for two reasons.

- A pure **cInstance**(p. 64) class does not make sense. It needs to be inherited from. Besides, it's an abstract class.
- Even the inherited classes are not simply instantiated but rather created through the **cInstanceFactory**(p. 69).

Parameters:
id The id of the corresponding object.

name The name of the corresponding object.

size The amount of attributes this object has.

9.10.3 Member Function Documentation

9.10.3.1 virtual cInstance* Condel::cInstance::clone (void) const [pure virtual]

Create a deep copy of this instance.

Returns:
An exact copy of this instance, including name. This must not be added to the same **cWorld**(p. 152) that this instance resides in.

Implemented in **Condel::cRacket** (p. 118), **Condel::cScore** (p. 129), **Condel::cBee** (p. 46), **Condel::cQueen** (p. 115), and **Condel::cScore** (p. 129).

9.10.3.2 virtual cAttributeSet Condel::cInstance::getAttributes (void) const [pure virtual]

Get all attributes.

Packs the current state of this instance into a **cAttributeSet**(p. 41).

Returns:
A copy of the current state.

Implemented in **Condel::cRacket** (p. 118), **Condel::cScore** (p. 130), **Condel::cBee** (p. 46), **Condel::cQueen** (p. 115), and **Condel::cScore** (p. 130).

9.10.3.3 unsigned Condel::cInstance::getCodedLength (void) [inline]

Estimate the entire in-memory size of this instance.

Returns:
The # of bytes necessary to keep the attribute set of this instance.

See also:
cAttributeSet::getCodedLength()(p. 42)

9.10.3.4 virtual double Condel::cInstance::getDistance (const cInstance & *other*) const [pure virtual]

Compares two instances.

This measures the similarity, in an appropriate way fitting for this object. Please note that this means that this metric is not comparable between different instances of different objects, i.e. the distance between two balls cannot be compared to the distance between two scores.

Parameters:
other Another instance of the same object.

Returns:
A metric of the similarity of the two instances.

Implemented in **Condel::cRacket** (p. 119), **Condel::cScore** (p. 130), **Condel::cBee** (p. 47), and **Condel::cScore** (p. 130).

9.10.3.5 const string& Condel::cInstance::getName (void) const [inline]

The name of the corresponding object.

All instances of an object share the same name.

9.10.3.6 OBJECT_ID Condel::cInstance::getObjectID (void) const [inline]

The ID of the corresponding object.

All instances of an object share the same numerical ID.

9.10.3.7 virtual void Condel::cInstance::setAttributes (const cAttributeSet & *na*) [pure virtual]

Set all attributes.

Overwrite the internal state.

Parameters:

na A new attribute set. Every attribute found in na is overwritten locally. It is legal to pass a sparse set here.

Implemented in **Condel::cRacket** (p. 119), **Condel::cScore** (p. 131), **Condel::cBee** (p. 47), **Condel::cQueen** (p. 115), and **Condel::cScore** (p. 131).

9.10.3.8 void Condel::cInstance::setSimTime (double *newtime*) [inline]

Set the instance's internal simulation time.

Parameters:

newtime The instance's state has change, this is the corresponding time.

9.10.3.9 void Condel::cInstance::setUpdateTime (double *newtime*) [inline]

Set the instance's last update time.

Parameters:

newtime An update was sent at this time.

Todo

Currently, this mechanism only supports global updates (i.e. updates are sent to all other hosts).

9.10.3.10 double Condel::cInstance::simTime (void) const [inline]

Query the instance's internal simulation time.

Returns:

The instance's state is at this simulation time.

9.10.3.11 virtual void Condel::cInstance::simulate (double *newtime*, cLoadMeter & *load*) [pure virtual]

Simulate this instance.

Simulates from m_simtime until newtime, then sets simtime to newtime.

Parameters:

newtime A simulation time in the future of **simTime()**(p. 67). Only the difference between the two times is advanced.

load A tool to measure CPU and memory usage during the simulation.

Implemented in **Condel::cRacket** (p. 119), **Condel::cScore** (p. 131), **Condel::cBee** (p. 47), **Condel::cQueen** (p. 116), and **Condel::cScore** (p. 131).

9.10.3.12 double Condel::cInstance::updateTime (void) const [inline]

Query the instance's last update time.

Returns:
The last time an update was SENT for this instance.

The documentation for this class was generated from the following files:

- instance.hh
- instance.cc

9.11 Condel::cInstanceFactory Class Reference

Creates instances.

`#include <instance_factory.hh>`

Static Public Member Functions

- static void **registerInstance** (const string &obj, **instanceCreator** cbf)
  Register a creation callback.
- static **cInstance** * **createInstance** (const **OBJECT_ID** &oid, const string &inst)
  Create an instance of the given name.

9.11.1 Detailed Description

Creates instances.

An instance is one version of an object (residing on some world on some host).

It is then a frequent problem for a host to instantiate a new instance given just the object ID. This factory class here solves this problem.

This is a purely static class. It can create any instance in the virtual universe. In fact, all instances should be created through this class.

See also:
cInstance(p. 64), **cObjectFactory**(p. 93)

9.11.2 Member Function Documentation

9.11.2.1 cInstance * cInstanceFactory::createInstance (const OBJECT_ID & *oid*, const string & *inst*) [static]

Create an instance of the given name.

The instance creation callback must have been registered previously, or you'll get a **condel_error()**(p. 35) here (which in turn exits).

Parameters:
inst The name of the object for which to create an instance. The correct callback function is found by searching for the c++ class name of the instance, which is the first part of the instance name (up to any #, if there is one).

oid The object ID this corresponds to.

Returns:
A fresh instance of the correct type and with the given name.

Todo
The object ID is kind of redundant and could be supplied by **cObjectFactory::nameToID(const string &)**(p. 94).

9.11.2.2 void cInstanceFactory::registerInstance (const string & *obj*, instanceCreator *cbf*) [static]

Register a creation callback.

Each INSTANCE-class needs to call this function exactly ONCE (globally) to register itself. The createinstance function should create an instance of the appropriate type.

Parameters:

obj The C++ name of the corresponding instance class.

cbf The callback function to create an instance of class obj.

The documentation for this class was generated from the following files:

- instance_factory.hh
- instance_factory.cc

9.12 Condel::cLoadMeter Class Reference

A CPU/memory load measuring tool.

`#include <load.hh>`

Public Member Functions

- virtual void **initialize** (void)
  Reset everything.
- virtual void **nextFrame** (void)
  Finish a frame.
- void **addCPU** (unsigned long long ops)
  Record CPU usage.
- void **addMem** (unsigned long long bytes)
  Record memory usage.
- double **averageCPU** (void)
  Average CPU load so far.
- unsigned long long **maxCPU** (void)
  Maximum CPU load so far.
- double **averageMem** (void)
  Average memory use so far.
- unsigned long long **maxMem** (void)
  Maximum memory use so far.

Protected Attributes

- unsigned long long **m_CpuSum**
- unsigned long **m_CpuSumCount**
- unsigned long long **m_memSum**
- unsigned long **m_memSumCount**
- unsigned long long **m_CpuMax**
- unsigned long long **m_memMax**
- unsigned long long **m_CpuCurrent**

9.12.1 Detailed Description

A CPU/memory load measuring tool.

This class gathers data on cpu/memory load using instrumentation by hand.

Todo
This should inherit from OMNeT++'s cStatistic some day.

9.12.2 Member Function Documentation

9.12.2.1 void Condel::cLoadMeter::addCPU (unsigned long long *ops*) [inline]

Record CPU usage.

Call this whenever you wish to record some CPU load.

Parameters:

ops Some estimate of the pseudo ops that were just used in some calculation.

9.12.2.2 void Condel::cLoadMeter::addMem (unsigned long long *bytes*) [inline]

Record memory usage.

Call this once per frame to specify how many bytes of memory were used this tick.

Parameters:

bytes The total maximum amount of memory (in bytes) necessary to run all algorithms on this host (since there is only one load meter per host) for this frame. In our case, this is often an estimate for a production optimized version (i.e. NOT a hardware solution).

9.12.2.3 double Condel::cLoadMeter::averageCPU (void) [inline]

Average CPU load so far.

Returns:

The average CPU pseudo ops per frame up to now.

9.12.2.4 double Condel::cLoadMeter::averageMem (void) [inline]

Average memory use so far.

Returns:

The average # of bytes of memory per frame up to now.

9.12.2.5 unsigned long long Condel::cLoadMeter::maxCPU (void) [inline]

Maximum CPU load so far.

Returns:

The maximum CPU pseudo ops in any frame recorded up to now.

9.12.2.6 unsigned long long Condel::cLoadMeter::maxMem (void) [inline]

Maximum memory use so far.

Returns:

The maximum # of bytes of memory in any frame recorded up to now.

9.12.2.7 void cLoadMeter::nextFrame (void) [virtual]

Finish a frame.

This also starts a new frame of CPU measurements.

The documentation for this class was generated from the following files:

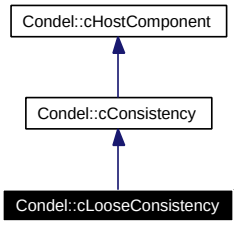
- load.hh
- load.cc

9.13 Condel::cLooseConsistency Class Reference

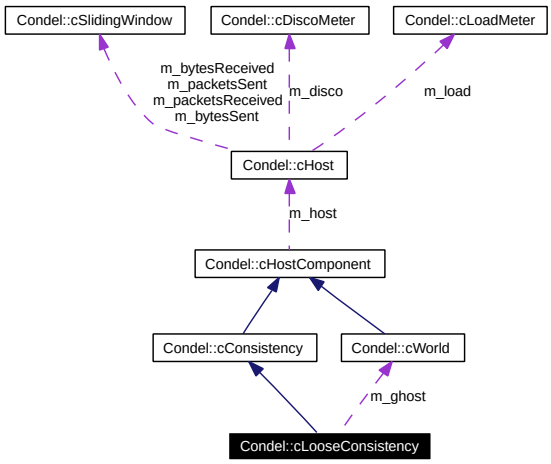
Loose Consistency.

`#include <consistency_loose.hh>`

Inheritance diagram for Condel::cLooseConsistency:



Collaboration diagram for Condel::cLooseConsistency:



Public Member Functions

- **cLooseConsistency** (**cHost** &host)
- virtual void **initialize** (void)

  Reinitialize the object.

- virtual void **addInstance** (const **cInstance** &inst)

  Do not *use this call.*

- virtual void **addInstance** (const **cInstance** &inst, double maxdist)

  Add an instance with its max-player-ghost distance.

- virtual void **handleTick** (void)

  Called every tick.

- virtual void **handleMessage** (**cMsg** *msg)

  Receive a message.

- virtual void **changeAttribute** (**OBJECT_ID** obj, **ATTRIBUTE_ID** attr, double val, bool preliminary=false)

  Change an (existing) instance's attribute.

- virtual void **updateInstance** (const **OBJECT_ID** &obj)

  Send an update for an object.

- virtual bool **isOwner** (const **OBJECT_ID** &obj)

  Am I the owner of this object?

- virtual void **setOwner** (const **OBJECT_ID** &obj, bool nval=true)

  Specify ownership of an object.

- virtual void **setMaxdist** (const **OBJECT_ID** &obj, double l)

  Reset maximum player/ghost distance.

Protected Attributes

- **cWorld m_ghost**
- double **m_regularUpdates**
- double **m_updateTimer**
- map< const **OBJECT_ID**, bool > **m_isOwner**
- map< const **OBJECT_ID**, double > **m_maxDistance**
- double **m_timeout**

9.13.1 Detailed Description

Loose Consistency.

This is the consistency algorithm for loose consistency.

Loose consistency works (roughly) as follows:

- Whenever a local action is to be performed, do it immediately.
- At regular intervals, inform all other hosts of the current state of all owned instances.
- Also, keep a ghost instance for every visible instance. Whenever the distance between the visible and the ghost becomes too large, send an update for that instance. Whenever an update is sent out for an instance, copy the visible to the ghost.
- When an update arrives for an instance, simply overwrite the local state.

There is a bit of science involved to determine which host *owns* which object.

This kind of consistency is really not consistent at all. In particular, there is no global truth that should be reached, the aggreed-upon truth hinges closely on the network delays. Nevertheless, this is a simple and robust approach that allows simple DVEs to be run.

Read more about the algorithms in TKN-07-005.

9.13.2 Member Function Documentation

9.13.2.1 void cLooseConsistency::addInstance (const cInstance & *inst*, double *maxdist*) [virtual]

Add an instance with its max-player-ghost distance.

Use this call to add an instance instead of the inherited **addInstance(const cInstance &)**(p. 76).

Parameters:

inst The new instance. This will be copied, you retain ownership of the specified one.

maxdist The maximum player/ghost distance before an update will be forced for this instance. Unit depends on the scenario. This value is compared to cInstance::getDistance(const cInstance &).

See also:

setMaxDist()

9.13.2.2 void cLooseConsistency::addInstance (const cInstance & *inst*) [virtual]

Do *not* use this call.

This is the inherited call from **cConsistency**(p. 49), but you need to use **addInstance(const cInstance &, double)**(p. 76) instead.

Implements **Condel::cConsistency** (p. 51).

9.13.2.3 void cLooseConsistency::changeAttribute (OBJECT_ID *obj*, ATTRIBUTE_ID *attr*, double *val*, bool *preliminary* = false) [virtual]

Change an (existing) instance's attribute.

In loose consistency, this simply performs the change, but only on the visible instance.

Parameters:

obj The ID of the object whose instance is to be changed.

attr The ID of the attribute to change in the instance.

val The new value to set.

preliminary Is ignored for loose consistency.

Implements **Condel::cConsistency** (p. 51).

9.13.2.4 void cLooseConsistency::handleMessage (cMsg * *msg*) [virtual]

Receive a message.

In loose consistency, if the message is for an object not yet on this host, it is replicated. Then, if this host is not the owner, read the data given in the message and overwrite the corresponding instances (both visible and ghost). During this change, discontinuity is also measured. This is called from **cRouter::handleMessage(cMsg *)**(p. 123).

Parameters:

msg A pointer to the new message. The host retains ownership of the message, do NOT delete it. If you need your own copy, you'll have to cMsg::dup() it.

Implements **Condel::cConsistency** (p. 51).

9.13.2.5 void cLooseConsistency::handleTick (void) [virtual]

Called every tick.

In loose consistency, this advances (calls **cWorld::simulate()**(p. 156)) both the real and the ghost world, then calls the AI. Finally checks for regular or indivual updates and sends them if necessary.

Implements **Condel::cConsistency** (p. 52).

9.13.2.6 void cLooseConsistency::initialize (void) [virtual]

Reinitialize the object.

Afterwards, same state as if freshly created.

Reimplemented from **Condel::cConsistency** (p. 52).

9.13.2.7 virtual bool Condel::cLooseConsistency::isOwner (const OBJECT_ID & *obj*) [inline, virtual]

Am I the owner of this object?

Parameters:

obj The object in question.

Returns:

True if this host is considered to be the owner of this object, i.e. the one who may send updates for it.

9.13.2.8 virtual void Condel::cLooseConsistency::setMaxdist (const OBJECT_ID & *obj*, double *l*) [inline, virtual]

Reset maximum player/ghost distance.

Parameters:

obj The object this information pertains to. The object must have been registered globally, but it need not yet have been instantiated or added here. Note, however, that if you use the recommended adding function **addInstance(const cInstance &, double)**(p. 76), the value given here will be overwritten.

l The maximum player/ghost distance before an update will be forced for this instance. Unit depends on the scenario. This value is compared to cInstance::getDistance(const cInstance &).

9.13.2.9 virtual void Condel::cLooseConsistency::setOwner (const OBJECT_ID & *obj*, bool *nval* = `true`) `[inline, virtual]`

Specify ownership of an object.

Parameters:

obj The object this information pertains to. The object must have been registered globally, but it need not yet have been instantiated anywhere.

nval True if this host is to be the owner of the object, false otherwise.

9.13.2.10 void cLooseConsistency::updateInstance (const OBJECT_ID & *obj*) `[virtual]`

Send an update for an object.

Finds the corresponding instance in the visible world, packs it into a message, sends that message off and copies the visible state onto the corresponding instance in the ghost world.

Parameters:

obj The object ID of the object to send an update about. If this host does not own that object, nothing happens.

Implements **Condel::cConsistency** (p. 52).

The documentation for this class was generated from the following files:

- consistency_loose.hh
- consistency_loose.cc

9.14 Condel::cMessageHistory Class Reference

An event history.

`#include <message_hist.hh>`

Public Types

- typedef list< **cMsg** >::iterator **iterator**

Public Member Functions

- virtual void **initialize** (void)
- iterator **add** (const **cMsg** &new_message)

  Add a new event into the queue.

- void **discard** (double new_horizon, list< **cMsg** >::iterator &before)

  Prune the past of this history.

- void **find** (list< **cMsg** >::iterator &it, const **cMsg** &m)

  Find the event in the queue.

- list< **cMsg** >::iterator **end** (void)

  The end of the list.

9.14.1 Detailed Description

An event history.

This is a simple list of events that may shape the development of a DVE.

9.14.2 Member Function Documentation

9.14.2.1 cMessageHistory::iterator cMessageHistory::add (const cMsg & *new_message*)

Add a new event into the queue.

Parameters:

new_message A message to be added. It will be added at the correct position. If that event is in the queue already, it will be ignored and **end()**(p. 80) is returned.

Returns:

An iterator pointing to the newly inserted message. If the event was in the queue already, **end()**(p. 80) is returned.

9.14.2.2 void cMessageHistory::discard (double *new_horizon*, list< cMsg >::iterator & *before*)

Prune the past of this history.

Starting from the oldest event, this will release and discard all event while *both* the conditions are true.

Parameters:

new_horizon All events older than new_horizon are discarded.

before All events before this iterator are discarded (the event pointed to by before is kept).

9.14.2.3 list<cMsg>::iterator Condel::cMessageHistory::end (void) [inline]

The end of the list.

Indicates an end-of-list. This iterator does *not* point to a legal event.

9.14.2.4 void cMessageHistory::find (list< cMsg >::iterator & *it*, const cMsg & *m*)

Find the event in the queue.

Parameters:

it This iterator is set to the event in the queue with the same id-data as the given one. If that event does not exist (the usual case), the iterator returns the next event after (so if you insert with the returned it, you'll be in order).

m A message only used for its ID data, namely **cMsg::getHostID()**(p. 87), cMsg::timestamp(), and **cMsg::getOrder()**(p. 88).

The documentation for this class was generated from the following files:

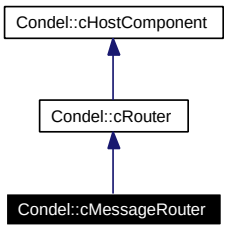
- message_hist.hh
- message_hist.cc

9.15 Condel::cMessageRouter Class Reference

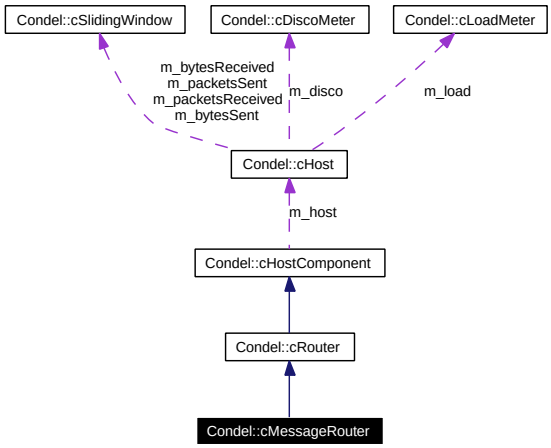
The network layer for a flat network model.

`#include <message_router.hh>`

Inheritance diagram for Condel::cMessageRouter:



Collaboration diagram for Condel::cMessageRouter:



Public Member Functions

- **cMessageRouter** (**cHost** &host)
- virtual void **initialize** (void)

  Reset everything.

- virtual void **send** (**cMsg** *pMsg)

  Send a message to the network.

- virtual void **handleMessage** (**cMsg** *pMsg)

  Receive a message from the network.

- virtual void **handleCheck** (void)

  Called each check-interval.

- virtual void **finish** (void)

  System Done.

9.15.1 Detailed Description

The network layer for a flat network model.

This works for total replication.

The class supports batching, which is a message aggregation based on time. If a message needs to be sent to a host, it may be delayed for a short time (the batching parameter). If other messages occur during that time that are destined for the same host, they are combined with the first message.

The class supports a selective repeat ARQ. If the timeout parameter is set in a message, that message is copied before sending it off. If such a message arrives at any host, it will be acknowledged (by sending an ACK message back to the original host). If the ACK message does not arrive within the timeout, that message will be resent.

On Batching & ARQ: Two possibilities exist:

1. Option I: Batch, THEN ARQ
   - + Only a single ACC for a batched package
   - - Must send batched package, but then filter out non-reliables (like ACKS)
   - - Can run into trouble if the head-package is a non-reliable
   - - Can run into trouble if the head-package is an ACK (weird id)
2. Option II: ARQ, THEN Batch
   - + Can resend just the packages with aggressive timeout
   - - timeout difficult to estimate (unless hooking into the batch-sendoff)
   - - More bandwith (many ACKS, all batched)

For now, we're installing option II, mostly because it's easier/more logical to do.

Note on the batching scheme:

- First message is the LAST message (timewise) to be batched.
- The encapsulated message is the one before (timewise).
- Last encapsulated message is the first one (timewise).

9.15.2 Member Function Documentation

9.15.2.1 void cMessageRouter::finish (void) [virtual]

System Done.

Prints out some statistics on of reliable messages, etc. This member is called by the host when the simulation has ended.

Reimplemented from **Condel::cRouter** (p. 123).

9.15.2.2 void cMessageRouter::handleCheck (void) [virtual]

Called each check-interval.

This handles timeouts both on messages waiting to be batched and on messages waiting to be ACKed in the ARQ. Called regularly (each **cHost::CHECK_TIME**(p. 62)) by the host.

Reimplemented from **Condel::cRouter** (p. 123).

9.15.2.3 void cMessageRouter::handleMessage (cMsg * *pMsg*) [virtual]

Receive a message from the network.

This deaggregates messages, handles ACK messages, and implements a reception delay (by rescheduling early messages). Finally hands off the message to the consistency. This is called by the host (i.e., a subclass of **cHost**(p. 58)).

Parameters:

pMsg A pointer to the message. The host (i.e. the caller of this function) retains ownership of the message, do NOT delete it. If you need your own copy, you'll have to dup() it.

Implements **Condel::cRouter** (p. 123).

9.15.2.4 void cMessageRouter::initialize (void) [virtual]

Reset everything.

This member is called by the host when the simulation is about to start.

Reimplemented from **Condel::cRouter** (p. 124).

9.15.2.5 void cMessageRouter::send (cMsg * *pMsg*) [virtual]

Send a message to the network.

The message is sent to each known host.

Parameters:

pMsg A pointer to the message. This message will be duped during this call, the original caller retains ownership of the supplied msg, so you may delete it after the call.

Implements **Condel::cRouter** (p. 124).

The documentation for this class was generated from the following files:

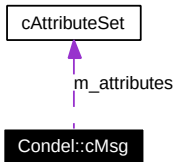
- message_router.hh
- message_router.cc

9.16 Condel::cMsg Class Reference

The base class for all messages.

`#include <msg.hh>`

Collaboration diagram for Condel::cMsg:



Public Types

- enum **MsgType** { **START** = -1, **DATA** = 0, **ACK** = 1 }

  Types of messages.

Public Member Functions

- **cMsg** (long host=0, unsigned long long order=0, double timeout=0.0, double batching=0.0, int type=cMsg::DATA, const char *name=NULL)

  The constructor.

- **cMsg** (const **cMsg** &other)
- **cMsg** & **operator**= (const **cMsg** &other)
- virtual cPolymorphic * **dup** (void) const
- virtual void **netPack** (cCommBuffer *b)
- virtual void **netUnpack** (cCommBuffer *b)
- virtual int **getType** (void) const
- virtual void **setType** (int type)
- virtual double **getTimeOut** (void) const

  Get the timeout.

- virtual void **setTimeOut** (double timeout)

  Set the timeout.

- virtual double **getBatching** (void) const

  Get the batching parameter.

- virtual void **setBatching** (bool batching)

  Set the batching parameter.

- virtual long **getHostID** (void) const

  Get the originating host ID.

- virtual void **setHostID** (long hostID)

  Set the originating host ID.

- virtual unsigned long long **getOrder** (void) const
  Get the order number.
- virtual void **setOrder** (unsigned long long order)
  Set the order number.
- virtual unsigned **getObjectID** (void) const
  Get the object ID.
- virtual void **setObjectID** (unsigned objectID)
  Set the object ID.
- virtual const char * **getObjectName** (void) const
  Get the object name.
- virtual void **setObjectName** (const char *objectName)
  Set the object name.
- virtual **cAttributeSet** & **getAttributes** (void)
  Get the encoded attributes.
- virtual const **cAttributeSet** & **getAttributes** (void) const
  Get the encoded attributes.
- virtual void **setAttributes** (const **cAttributeSet** &attributes)
  Encode attributes.
- bool **operator<** (const **cMsg** &other) const
  An order operator.
- bool **operator==** (const **cMsg** &other) const
  A comparison operator.

Protected Member Functions

- bool **operator==** (const **cMsg** &)
  Do not use this.

Protected Attributes

- int **m_type**
- double **m_timeout**
- double **m_batching**
- long **m_hostID**
- unsigned long long **m_order**
- unsigned **m_objectID**
- string **m_objectName**
- **cAttributeSet m_attributes**

9.16.1 Detailed Description

The base class for all messages.

All messages that are exchanged between hosts are based on this class.

Note that messages are not only interchanged and then interpreted immediately, they also often kept. This may be to have a copy if the transmission failed or to have a message history (useful for optimistic consistency).

Note that **getHostID()**(p. 87), timestamp(), and **getOrder()**(p. 88) together allow for a universally unique identification of this message. In other words, by using these three numbers as a serial number, each original message in the entire network has a different serial number. Note, however, that you may get the same message several times (because of the ARQ algorithm) and that the same message may be sent to many different hosts.

This file is partly inspired by the result of running opp_msgc on msg.opp. It thus interfaces with OMNeT very closely.

9.16.2 Member Enumeration Documentation

9.16.2.1 enum Condel::cMsg::MsgType

Types of messages.

More types could be added.

Enumeration values:

START The first message in an optimistic consistency setting.

DATA A normal message.

ACK An acknowledgement message.

9.16.3 Constructor & Destructor Documentation

9.16.3.1 cMsg::cMsg (long *host* = 0, unsigned long long *order* = 0, double *timeout* = 0.0, double *batching* = 0.0, int *type* = cMsg::DATA, const char * *name* = NULL)

The constructor.

Do NOT use this one. Use the **cMsgFactory**(p. 91), instead.

Parameters:

host The host this message was created on.

order A unique serial number for messages created on this host.

timeout If not 0.0, the message will be sent reliably and this is the timeout of the selective repeat algorithm.

batching If negative, no batching is performed. If 0.0, all messages created for the same target host during this CHECK_TIME will be combined. If > 0.0, The sending algorithm will delay this message for a maximum of batching seconds to combine it with other messages going to the same host.

type One of the types of messages, usually **cMsg::DATA**(p. 86).

name A name for this message, this is purely for debugging.

9.16.4 Member Function Documentation

9.16.4.1 virtual const cAttributeSet& Condel::cMsg::getAttributes (void) const [inline, virtual]

Get the encoded attributes.

Returns:
A **cAttributeSet**(p. 41) generated by some cInstance's **cInstance::getAttributes()**(p. 66). This is the payload of the message. The instance in question should have this message's **getObjectID()**(p. 87).

9.16.4.2 virtual cAttributeSet& Condel::cMsg::getAttributes (void) [inline, virtual]

Get the encoded attributes.

Returns:
A **cAttributeSet**(p. 41) generated by some cInstance's **cInstance::getAttributes()**(p. 66). This is the payload of the message. The instance in question should have this message's **getObjectID()**(p. 87).

9.16.4.3 virtual double Condel::cMsg::getBatching (void) const [inline, virtual]

Get the batching parameter.

Returns:
If negative, no batching is performed. If 0.0, all messages created for the same target host during this CHECK_TIME will be combined. If > 0.0, The sending algorithm will delay this message for a maximum of batching seconds to combine it with other messages going to the same host.

9.16.4.4 virtual long Condel::cMsg::getHostID (void) const [inline, virtual]

Get the originating host ID.

Returns:
The host this message was created on.

9.16.4.5 virtual unsigned Condel::cMsg::getObjectID (void) const [inline, virtual]

Get the object ID.

Most messages pertain to exactly one object. In those cases, this is the ID of that object. The attributes encoded in this message will overwrite the state of the corresponding instance on receiving hosts.

Returns:
ID of the object whose state is encoded in this message.

9.16.4.6 virtual const char* Condel::cMsg::getObjectName (void) const [inline, virtual]

Get the object name.

Most messages pertain to exactly one object. In those cases, this is the name of that object. The attributes encoded in this message will overwrite the state of the corresponding instance on receiving hosts.

Returns:
Name of the object whose state is encoded in this message.

9.16.4.7 virtual unsigned long long Condel::cMsg::getOrder (void) const [inline, virtual]

Get the order number.

Returns:
A unique serial number for messages created on this host.

9.16.4.8 virtual double Condel::cMsg::getTimeOut (void) const [inline, virtual]

Get the timeout.

Returns:
If not 0.0, the message is sent reliably and this is the timeout of the selective repeat algorithm.

9.16.4.9 bool Condel::cMsg::operator< (const cMsg & *other*) const [inline]

An order operator.

Using this operation, a total order of all events can be attained by simple sorting.

9.16.4.10 bool Condel::cMsg::operator== (const cMsg & *other*) const [inline]

A comparison operator.

if this operator evaluates to true, then the two messages carry the same extended ID and should therefore be equal (unless something went seriously wrong).

9.16.4.11 bool Condel::cMsg::operator== (const cMsg &) [protected]

Do not use this.

The comparison operator is protected, declared, but not implemented, to prevent accidental usage.

9.16.4.12 virtual void Condel::cMsg::setAttributes (const cAttributeSet & *attributes*) [inline, virtual]

Encode attributes.

Parameters:

attributes A **cAttributeSet**(p. 41) generated by some cInstance's **cInstance::getAttributes()**(p. 66). This is the payload of the message. The instance in question should have this message's **getObjectID()**(p. 87).

9.16.4.13 virtual void Condel::cMsg::setBatching (bool *batching*) [inline, virtual]

Set the batching parameter.

Make sure to set this before the message is actually sent out,

Parameters:

batching If negative, no batching is performed. If 0.0, all messages created for the same target host during this CHECK_TIME will be combined. If > 0.0, The sending algorithm will delay this message for a maximum of batching seconds to combine it with other messages going to the same host.

9.16.4.14 virtual void Condel::cMsg::setHostID (long *hostID*) [inline, virtual]

Set the originating host ID.

Usually, you do not want to change this.

Parameters:

hostID The host this message was created on.

9.16.4.15 virtual void Condel::cMsg::setObjectID (unsigned *objectID*) [inline, virtual]

Set the object ID.

Parameters:

objectID The object ID this message is about. The attributes encoded in this message will overwrite the states of the corresponding instance on receiving hosts.

9.16.4.16 virtual void Condel::cMsg::setObjectName (const char * *objectName*) [inline, virtual]

Set the object name.

Parameters:

objectName The object name this message is about. The attributes encoded in this message will overwrite the state of the corresponding instance on receiving hosts.

9.16.4.17 virtual void Condel::cMsg::setOrder (unsigned long long *order*) [inline, virtual]

Set the order number.

Usually, you do not want to change this.

Parameters:

order A unique serial number for messages created on this host.

9.16.4.18 virtual void Condel::cMsg::setTimeOut (double *timeout*) [inline, virtual]

Set the timeout.

Make sure to set this before the message is actually sent out, it makes little sense to change this on a received message.

Parameters:

timeout If not 0.0, the message will be sent reliably and this is the timeout of the selective repeat algorithm.

The documentation for this class was generated from the following files:

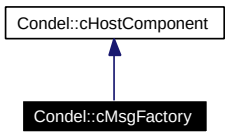
- msg.hh
- msg.cc

9.17 Condel::cMsgFactory Class Reference

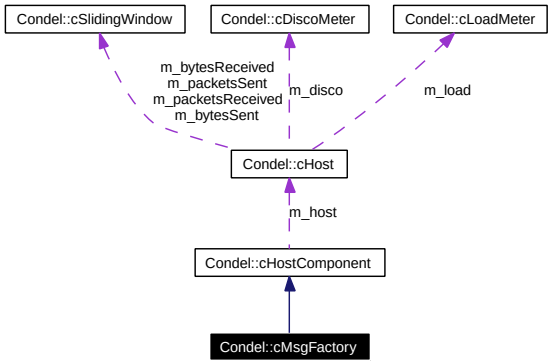
A message factory.

`#include <msg.hh>`

Inheritance diagram for Condel::cMsgFactory:



Collaboration diagram for Condel::cMsgFactory:



Public Member Functions

- **cMsgFactory** (**cHost** &host)
- auto_ptr< **cMsg** > **createMsg** (const char *name=NULL, double timeout=0.0, double batching=0.0)

  Create a **cMsg**(p. 84)*, with a running order number.*

- virtual void **initialize** (void)

  Reset all.

Static Public Member Functions

- static auto_ptr< **cMsg** > **createStartMsg** (void)

  Create a special start event **cMsg**(p. 84).

9.17.1 Detailed Description

A message factory.

Use this helper to construct cMsg'es for you. Basically, it's just a counter to keep order numbers sequential and unique.

9.17.2 Member Function Documentation

9.17.2.1 auto_ptr< cMsg > cMsgFactory::createMsg (const char * *name* = NULL, double *timeout* = 0.0, double *batching* = 0.0)

Create a **cMsg**(p. 84), with a running order number.

The host and order number will automatically be set by this function, timestamp() is the current cHost::simtime(). Type of the message is **cMsg::DATA**(p. 86).

Parameters:

name A name for this message, this is purely for debugging.

timeout If not 0.0, the message will be sent reliably and this is the timeout of the selective repeat algorithm.

batching If negative, no batching is performed. If 0.0, all messages created for the same target host during this CHECK_TIME will be combined. If > 0.0, The sending algorithm will delay this message for a maximum of batching seconds to combine it with other messages going to the same host.

Returns:

A pointer to a correctly instantiated message. Assign this to an auto_ptr<cMsg> to avoid any possibility of leakage.

See also:

cMsg::cMsg(long, unsigned long long, double, double, int,const char *)(p. 86)

9.17.2.2 auto_ptr< cMsg > cMsgFactory::createStartMsg (void) [static]

Create a special start event **cMsg**(p. 84).

You should only do this once per host.

The documentation for this class was generated from the following files:

- msg.hh
- msg.cc

9.18 Condel::cObjectFactory Class Reference

Object ID manager.

`#include <object_factory.hh>`

Static Public Member Functions

- static **OBJECT_ID registerObject** (const string &name)
  Create a unique ID for a new object.
- static **OBJECT_ID nameToID** (const string &name)
  Returns the ID of a formerly registered object.
- static const map< const string, **OBJECT_ID** > & **objectMap** (void)
  Returns a list of currently registered objects.
- static bool **nameExists** (const string &name)
  Check if an attribute was already registered.
- static void **clear** (void)
  Clears ALL previous entries.

9.18.1 Detailed Description

Object ID manager.

In our model, objects may have instances of themselves on some or all of the hosts. To be quite precise, each **cWorld**(p. 152) can house an instance of each object. Objects are differentiated both by their unique name and their corresponding unique object ID.

The name of each object is of the form <c++classname>[#<host_name>] , e.g. **cRacket**(p. 117)#2. Instances do not have names (apart from their object name) to further distinguish them.

There is no actual base class for objects, since objects are never created. Only their instances are created. All instances are based on **cInstance**(p. 64).

cObjectFactory(p. 93) is a purely static class used to globabally assign the IDs for objects. It also handles the translation of object name->object ID.

Note that this scheme is a cludge, it would not be possible to do this on a truly distributed system. There are schemes to assign such numbers in a distributed fashion though, and we decided to ignore this sub- problem for greater clarity of the rest of the code.

See also:
cInstance(p. 64), **cInstanceFactory**(p. 69), **cAttributeFactory**(p. 39)

9.18.2 Member Function Documentation

9.18.2.1 bool cObjectFactory::nameExists (const string & *name*) [static]

Check if an attribute was already registered.

Parameters:
name The name of the attribute in question.

Returns:
True if this name exists already in the DB.

9.18.2.2 OBJECT_ID cObjectFactory::nameToID (const string & *name*) [static]

Returns the ID of a formerly registered object.

If the object was NOT registered before, the application will exit because of a builtin assert here.

Parameters:
name The name of the registered object.

Returns:
The corresponding object ID.

See also:
nameExists(p. 93)

9.18.2.3 static const map<const string, OBJECT_ID>& Condel::cObjectFactory::objectMap (void) [inline, static]

Returns a list of currently registered objects.

Try not to use this if you don't really have to.

9.18.2.4 OBJECT_ID cObjectFactory::registerObject (const string & *name*) [static]

Create a unique ID for a new object.

Parameters:
name The name of the new object.

Returns:
The newly assigned object ID.

The documentation for this class was generated from the following files:

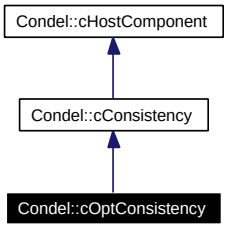
- object_factory.hh
- object_factory.cc

9.19 Condel::cOptConsistency Class Reference

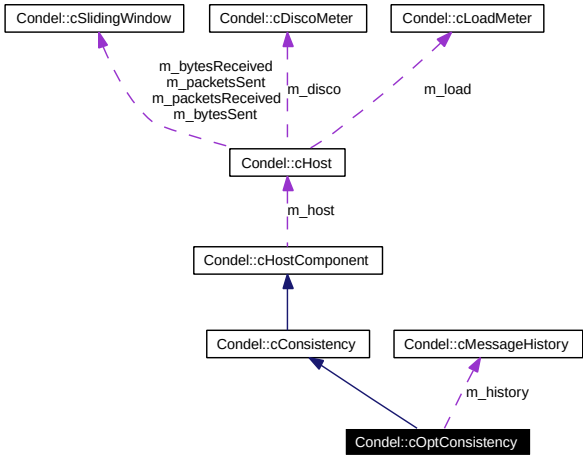
Optimistic Consistency.

`#include <consistency_opt.hh>`

Inheritance diagram for Condel::cOptConsistency:



Collaboration diagram for Condel::cOptConsistency:



Public Member Functions

- **cOptConsistency** (**cHost** &host)
- virtual void **initialize** (void)

  Reinitialize the object.

- **cMessageHistory** & **getMessageHistory** (void)

  Gets you the message history.

- virtual void **addInstance** (const **cInstance** &inst)

  Add an instance.

- virtual void **handleTick** (void)

  Do scheduled work for each tick.

- virtual void **handleMessage** (**cMsg** *msg)

  Receive a new message from the network.

- virtual void **updateInstance** (const **OBJECT_ID** &obj)

  Update an instance.

- virtual void **changeAttribute** (**OBJECT_ID** obj, **ATTRIBUTE_ID** attr, double val, bool preliminary=false)

  Change an attribute on an instance.

Protected Member Functions

- void **update** (**cOptWorld** &rep, double newtime, bool &dirty, const **cOptWorld** *prev, bool measure_disco=false)

Protected Attributes

- **cMessageHistory m_history**
- scoped_array< double > **m_trailing**
- scoped_array< bool > **m_dirty**
- bool **m_dirtyVisible**
- scoped_ptr< **cOptWorld** > **m_pState** [TRAILING_STATES]
- double **m_localDelay**
- double **m_timeout**
- double **m_batching**

9.19.1 Detailed Description

Optimistic Consistency.

This is the consistency algorithm for optimistic consistency.

Optimistic consistency works (roughly) as follows:

- Whenever a local action is to be performed, do it immediately.
- Send the *action* to all other hosts.
- When an action is received from another host, backtrack to a time just before that, add it to the message history and recap everything to the present time.

For this to practicable, messages need to be transferred in a secure way (using a selective repeat in our case).

While this kind of consistency behaves just like loose consistency at first, the conflict resolution is much smarter and leads to an eventual consistency (i.e. past states are consistent) of all hosts with a global history. In addition, the global history would be logical even if this was a single-user system. Note that the price for these advantages is significantly higher CPU load and the tendency to break if any unforseen errors occur (like different rounding on different machines etc.).

Read more about the algorithms in TKN-07-005.

9.19.2 Member Function Documentation

9.19.2.1 void cOptConsistency::addInstance (const cInstance & *inst*) [virtual]

Add an instance.

The instance is not actually added here, rather an AddInstance message is generated which is then sent to all hosts and myself. When that message is evaluated, the instance is added.

Parameters:

inst A new instance. The values from this are copied, you retain ownership of it.

Implements **Condel::cConsistency** (p. 51).

9.19.2.2 void cOptConsistency::changeAttribute (OBJECT_ID *obj*, ATTRIBUTE_ID *attr*, double *val*, bool *preliminary* = false) [virtual]

Change an attribute on an instance.

This is assumed to be a nondeterministic event, so this entire effect is wrapped into a message, sent to all hosts and myself.

Parameters:

obj The ID of the object whose instances are to be changed.

attr The ID of the attribute to change in the instances.

val The new value to set.

preliminary If preliminary is true, this call returns immediately for this consistency.

Implements **Condel::cConsistency** (p. 51).

9.19.2.3 cMessageHistory& Condel::cOptConsistency::getMessageHistory (void) [inline]

Gets you the message history.

Returns:

A message/event list that defines the entire visible world. This is the only one this host is using. Note that all TSS'es are using that same queue.

9.19.2.4 void cOptConsistency::handleMessage (cMsg * *msg*) [virtual]

Receive a new message from the network.

This is called from **cRouter::handleMessage(cMsg *)**(p. 123). The message is simply added to the message history. Some of the trailing states may be marked as dirty, indicating they need to backtrack.

Parameters:

msg A pointer to the new message. The host retains ownership of the message, do NOT delete it. If you need your own copy, you'll have to cMsg::dup() it.

Implements **Condel::cConsistency** (p. 51).

9.19.2.5 void cOptConsistency::handleTick (void) [virtual]

Do scheduled work for each tick.

In optimistic consistency, this means backtracking/updating the TSSes, cropping the message history, and finally calling the scenario's AI.

Implements **Condel::cConsistency** (p. 52).

9.19.2.6 void cOptConsistency::initialize (void) [virtual]

Reinitialize the object.

Afterwards, same state as if freshly created.

Reimplemented from **Condel::cConsistency** (p. 52).

9.19.2.7 void cOptConsistency::updateInstance (const OBJECT_ID & *obj*) [virtual]

Update an instance.

Has no effect for this consistency.

Parameters:

obj The object whose instance's state is to be sent out to other hosts.

Implements **Condel::cConsistency** (p. 52).

The documentation for this class was generated from the following files:

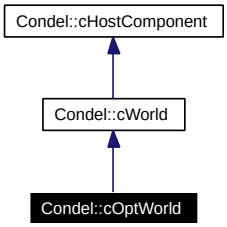
- **consistency_opt.hh**
- consistency_opt.cc

9.20 Condel::cOptWorld Class Reference

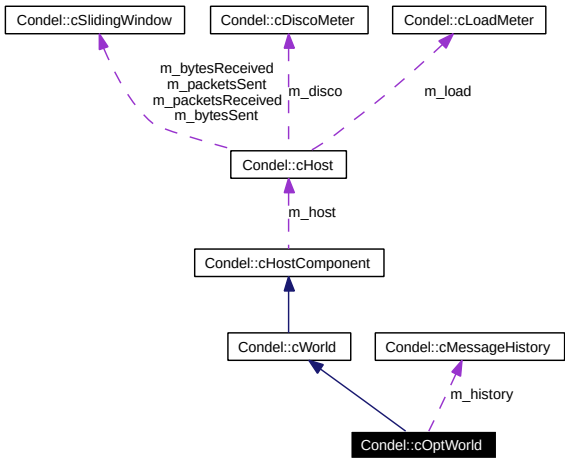
A world in an optimistic setting.

`#include <world_opt.hh>`

Inheritance diagram for Condel::cOptWorld:



Collaboration diagram for Condel::cOptWorld:



Public Member Functions

- **cOptWorld** (**cHost** &host, const string &name, **cMessageHistory** &msgHist)

  The constructor.

- const **cOptWorld** & **operator**= (const **cOptWorld** &o)
- auto_ptr< **cWorld** > **clone** (void)

  Create a deep clone of this world.

- void **initialize** (void)

  Reset everything.

- cMessageHistory::iterator & **current** (void)

  The last event.

- void **execute_event** (**cLoadMeter** &load)

  Execute a single event.

- virtual void **simulate** (double newtime, **cLoadMeter** &load)

  Perform deterministic actions.

Protected Attributes

- **cMessageHistory** & **m_history**
- cMessageHistory::iterator **m_current**

9.20.1 Detailed Description

A world in an optimistic setting.

9.20.2 Constructor & Destructor Documentation

9.20.2.1 cOptWorld::cOptWorld (cHost & *host*, const string & *name*, cMessageHistory & *msgHist*)

The constructor.

Parameters:

host The host this world is on.

name The name of this world.

msgHist A reference to the single message history being held in cConsistencyOpt. This reference is kept for use in this class.

9.20.3 Member Function Documentation

9.20.3.1 auto_ptr< cWorld > cOptWorld::clone (void) [virtual]

Create a deep clone of this world.

The world will have all the instances of the original, with the same state, etc. In fact, aside from the address, the new world will be indistinguishable from the old one.

Note this is a very slow process.

Returns:

A pointer to a newly created copy of this world. Assign this to an auto_ptr<cWorld> to avoid any possibility of leakage.

Reimplemented from **Condel::cWorld** (p. 154).

9.20.3.2 cMessageHistory::iterator& Condel::cOptWorld::current (void) [inline]

The last event.

This is the last event that was simulated on this world.

9.20.3.3 void cOptWorld::execute_event (cLoadMeter & *load*)

Execute a single event.

This finds the next event after **current()**(p. 100), executes it, then sets current to it.

9.20.3.4 void cOptWorld::initialize (void) [virtual]

Reset everything.

This member is called by the host when the simulation is about to start.

Reimplemented from **Condel::cWorld** (p. 155).

9.20.3.5 void cOptWorld::simulate (double *newtime*, cLoadMeter & *load*) [virtual]

Perform deterministic actions.

This function performs all deterministic actions in this world. This usually calls **cScenario::simulate()**(p. 127).

Parameters:

newtime Time to advance the system to. This should be in the future of **simTime()**(p. 156). Given in seconds.

load A tool to measure CPU and memory usage during the simulation.

Reimplemented from **Condel::cWorld** (p. 156).

The documentation for this class was generated from the following files:

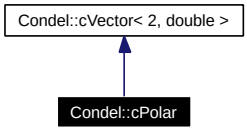
- world_opt.hh
- world_opt.cc

9.21 Condel::cPolar Class Reference

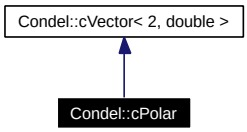
A two-dimensional vector with polar coordinates.

`#include <polar.hh>`

Inheritance diagram for Condel::cPolar:



Collaboration diagram for Condel::cPolar:



Public Member Functions

- **cPolar** (const **cVector**< 2, double > &src)
- void **update** (void)

  Update the local cache.

- const **cPolar** & **operator**= (const **cVector**< 2, double > &src)
- const double & **operator[ ]** (unsigned pos) const
- const double & **x** (void) const
- const double & **y** (void) const
- const double & **phi** (void) const
- const double & **abs** (void) const
- void **setPhiAbs** (double phi, double abs)

  Set a phi and an abs.

- void **setPhi** (double nphi)

  Change just phi.

- void **setAbs** (double nabs)

  change just abs.

- void **setX** (double x)
- void **setY** (double y)
- void **setXY** (double x, double y)
- const **cPolar** & **operator**+= (const **cVector**< 2, double > &src)
- const **cPolar** & **operator-**= (const **cVector**< 2, double > &src)
- const **cPolar** & **operator** *= (const double &value)
- **cPolar operator**+ (const **cVector**< 2, double > &src) const
- **cPolar operator-** (const **cVector**< 2, double > &src) const
- **cPolar operator** * (const double &value) const
- double **operator** * (const **cVector**< 2, double > &src) const

Static Public Member Functions

- static double **normalize_phi** (double phi)
  Normalize an angle.

Protected Attributes

- double **m_phi**
- double **m_abs**

9.21.1 Detailed Description

A two-dimensional vector with polar coordinates.

This is just a 2-d vector with some added support to read/write polar values (phi, abs). The True data is always the data in the vector.

Definition of the angle phi:

- phi=0 -> vector shows right (in direction of x-axis).
- phi increasing -> vector turns counterclockwise (mathematically positive).
- Given in radians (one full turn is 2Π).

So, phi((0,1)) := 0, phi((1,0)) := $\frac{\Pi}{2}$ Values of phi as returned by this class are always normalized (i.e. $\varphi \in [0, 2\Pi[$).

9.21.2 Member Function Documentation

9.21.2.1 double cPolar::normalize_phi (double *phi*) [static]

Normalize an angle.

The result will be between 0 and 2 M_PI.

9.21.2.2 void Condel::cPolar::setAbs (double *nabs*) [inline]

change just abs.

This keeps phi as it was, changes the vector to adjust abs.

9.21.2.3 void Condel::cPolar::setPhi (double *nphi*) [inline]

Change just phi.

This keeps abs as it was, changes the vector to adjust phi.

9.21.2.4 void cPolar::setPhiAbs (double *phi*, double *abs*)

Set a phi and an abs.

Sets the vector accordingly.

9.21.2.5 void cPolar::update (void)

Update the local cache.

This will update the cached phi and abs values to match the underlying vector data.

Call this whenever the underlying values in the vector may have changed without going through the members of this class.

The documentation for this class was generated from the following files:

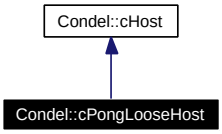
- polar.hh
- polar.cc

9.22 Condel::cPongLooseHost Class Reference

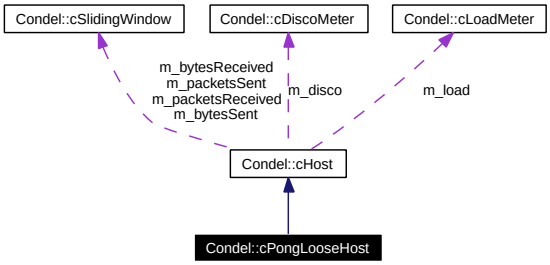
The host for Pong with loose consistency.

`#include <pong_loose_host.hh>`

Inheritance diagram for Condel::cPongLooseHost:



Collaboration diagram for Condel::cPongLooseHost:



Public Member Functions

- **cPongLooseHost** (void)

  The constructor.

9.22.1 Detailed Description

The host for Pong with loose consistency.

This is the class that creates the entire loose consistency/pong system.

9.22.2 Constructor & Destructor Documentation

9.22.2.1 cPongLooseHost::cPongLooseHost (void)

The constructor.

This sets cHost's component pointers to components instantiated here. The components chosen are **cPongLooseScenario**(p. 106), **cLooseConsistency**(p. 74), etc..

The documentation for this class was generated from the following files:

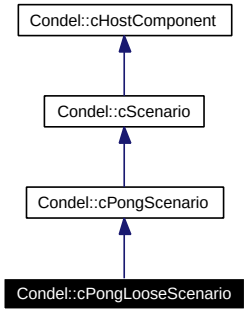
- pong_loose_host.hh
- pong_loose_host.cc

9.23 Condel::cPongLooseScenario Class Reference

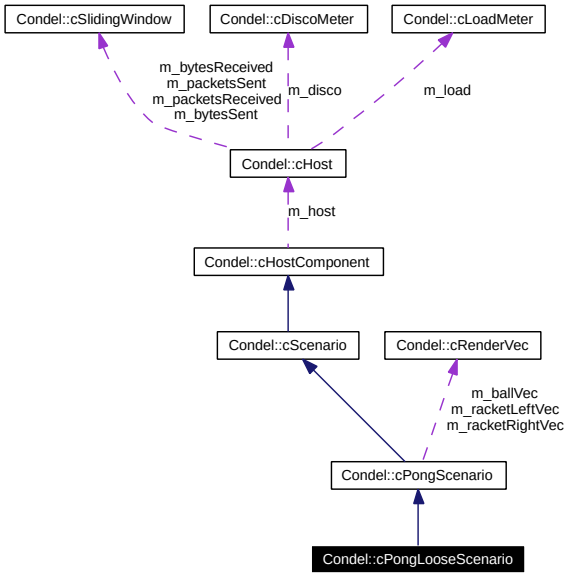
The Pong scenario with loose consistency.

`#include <pong_loose_scenario.hh>`

Inheritance diagram for Condel::cPongLooseScenario:



Collaboration diagram for Condel::cPongLooseScenario:



Public Member Functions

- **cPongLooseScenario** (**cHost** &host)
- virtual void **initialize** (void)

  Reset everything.

- virtual void **simulate** (**cWorld** &world, double newtime, **cLoadMeter** &load) const

  Perform deterministic actions.

9.23.1 Detailed Description

The Pong scenario with loose consistency.

This does things that are specific for loose consistency. That's really just the whole ownership and maximum player/ghost distance setting.

9.23.2 Member Function Documentation

9.23.2.1 void cPongLooseScenario::initialize (void) [virtual]

Reset everything.

Call **cPongScenario::initialize()**(p. 112), then set the player/ghost distances for racket and ball.

Reimplemented from **Condel::cPongScenario** (p. 112).

9.23.2.2 void cPongLooseScenario::simulate (cWorld & *world*, double *newtime*, cLoadMeter & *load*) const [virtual]

Perform deterministic actions.

Call **cPongScenario::simulate()**(p. 112), then take/yield ownership of the ball according to its new direction. The player that the ball moves away from owns it.

Parameters:

world The world that is advanced.

newtime Time to advance the system to. This should be in the future of the world's **cWorld::simTime()**(p. 156). Given in seconds.

load A tool to measure CPU and memory usage during the simulation.

Reimplemented from **Condel::cPongScenario** (p. 112).

The documentation for this class was generated from the following files:

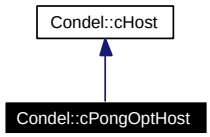
- pong_loose_scenario.hh
- pong_loose_scenario.cc

9.24 Condel::cPongOptHost Class Reference

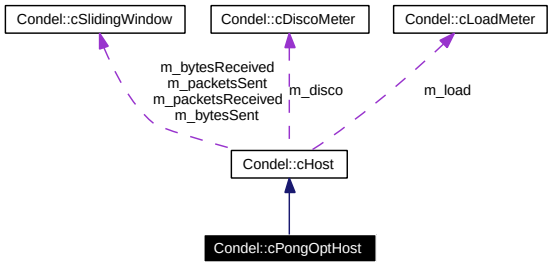
The host for Pong with optimistic consistency.

`#include <pong_opt_host.hh>`

Inheritance diagram for Condel::cPongOptHost:



Collaboration diagram for Condel::cPongOptHost:



Public Member Functions

- **cPongOptHost** (void)

  The constructor.

9.24.1 Detailed Description

The host for Pong with optimistic consistency.

This is the class that creates the entire optimistic consistency/pong system.

9.24.2 Constructor & Destructor Documentation

9.24.2.1 cPongOptHost::cPongOptHost (void)

The constructor.

This sets cHost's component pointers to components instantiated here. The components chosen are **cPongScenario**(p. 109), **cOptConsistency**(p. 95), etc..

The documentation for this class was generated from the following files:

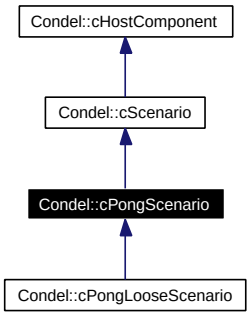
- pong_opt_host.hh
- pong_opt_host.cc

9.25 Condel::cPongScenario Class Reference

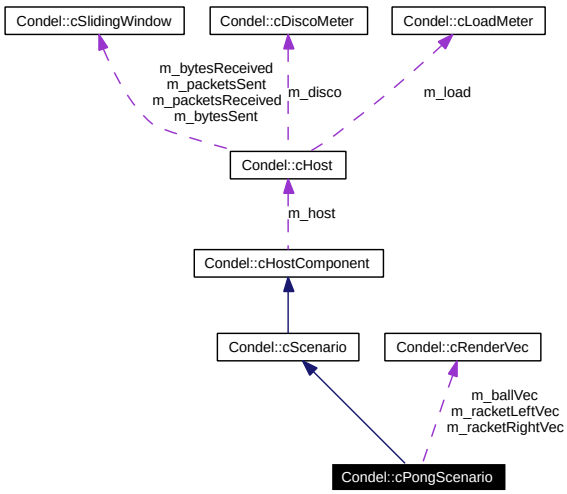
The Pong Scenario.

`#include <pong_scenario.hh>`

Inheritance diagram for Condel::cPongScenario:



Collaboration diagram for Condel::cPongScenario:



Public Member Functions

- **cPongScenario** (**cHost** &host)
- void **initialize** (void)

  Reset everything.

- virtual void **simulate** (**cWorld** &world, double newtime, **cLoadMeter** &load) const

  Perform deterministic actions.

- virtual void **ai** (double newtime)

  Perform the AI.

- virtual void **finish** (void)

System Done.

Static Public Attributes

- static const double **LEFT_BORDER** = 0
  X-coordinate of the left border.
- static const double **RIGHT_BORDER** = 1000
  X-coordinate of the right border.
- static const double **TOP_BORDER** = 0
  Y-coordinate of the upper border.
- static const double **BOTTOM_BORDER** = 500
  Y-coordinate of the lower border.
- static const double **INITIAL_X_SPEED** = 200
  Initial horizontal velocity of the ball, in pels/second.
- static const double **X_ACCELERATION** = 1.05
  Horizontal acceleration factor.
- static const double **RACKET_SIZE** = 50
  Height of the racket, in pels.
- static const double **RACKET_SPEED** = 500
  Vertical racket velocity.

Protected Member Functions

- void **checkCollision** (**cWorld** &world, cBall &ball, const **cRacket** *racket, const **OBJECT_ID** &scoreID, **cLoadMeter** &load, double newX) const
- void **initRendering** (void)
- void **render** (void)

Protected Attributes

- bool **m_doRender**
- **cRenderVec m_ballVec**
- **cRenderVec m_racketLeftVec**
- **cRenderVec m_racketRightVec**
- cOutVector **m_scoreLeftVec**
- cOutVector **m_scoreRightVec**
- unsigned long **m_lastScore**
- unsigned long **m_scoreCounter**
- bool **m_isPlayerLeft**

Static Protected Attributes

- static **OBJECT_ID s_racketLeftID** = cObjectFactory::registerObject("cRacket#left")
- static **OBJECT_ID s_racketRightID** = cObjectFactory::registerObject("cRacket#right")
- static **OBJECT_ID s_ballID** = cObjectFactory::registerObject("cBall#")
- static **OBJECT_ID s_scoreLeftID** = cObjectFactory::registerObject("cScore#left")
- static **OBJECT_ID s_scoreRightID** = cObjectFactory::registerObject("cScore#right")

9.25.1 Detailed Description

The Pong Scenario.

This is a simple game, fashioned after the classic video game Pong. Two players play a simplified tennis vs. each other. Each player controls a racket (can only move this up and down). A ball is thrown in and the player has to move the racket so that it reflects the ball and does not cross the player's rear line.

In comparison to the original Pong, there are two complications:

- The rackets have three zones (each one third of the racket). The middle zone reflects normally, but the lower zone deflects downwards, while the upper zone reflects upwards.
- The ball is accellerated (by 5%) in the X-direction whenever it is reflected.

The AI is very simple, it will only move the racket towards the estimated collision point. When the ball is reflected off the side-lines, this estimation changes, causing the AI to issue a new move command.

If the render parameter is true for this host, positions of ball, rackets and scores will be logged to the omnetpp.vec file each frame. This can be used to create a visualization later.

9.25.2 Member Function Documentation

9.25.2.1 void cPongScenario::ai (double *newtime*) [virtual]

Perform the AI.

Simply estimate the point of intersection between the ball and the baseline and move towards that.

Parameters:
newtime The current time.

Implements **Condel::cScenario** (p. 126).

9.25.2.2 void cPongScenario::finish (void) [virtual]

System Done.

Write the score to the log file (it's the application dependant yield measure).

Implements **Condel::cScenario** (p. 126).

9.25.2.3 void cPongScenario::initialize (void) [virtual]

Reset everything.

Create player's racket, opponents score and (if left player) the ball.

Reimplemented from **Condel::cScenario** (p. 126).

Reimplemented in **Condel::cPongLooseScenario** (p. 107).

9.25.2.4 void cPongScenario::simulate (cWorld & *world*, double *newtime*, cLoadMeter & *load*) const [virtual]

Perform deterministic actions.

Let each instance in the world simulate to newtime (moves ball/rackets). Check if ball moved over a baseline. If so, increment score and rethrow the ball. If reflected off racket, adjust ballspeed according to racket-zone.

Parameters:

world The world that is advanced.

newtime Time to advance the system to. This should be in the future of the world's **cWorld::simTime()**(p. 156). Given in seconds.

load A tool to measure CPU and memory usage during the simulation.

Implements **Condel::cScenario** (p. 127).

Reimplemented in **Condel::cPongLooseScenario** (p. 107).

The documentation for this class was generated from the following files:

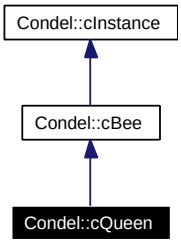
- pong_scenario.hh
- pong_scenario.cc

9.26 Condel::cQueen Class Reference

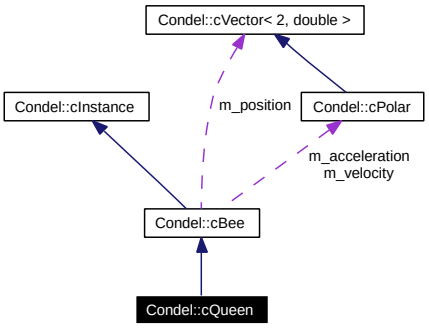
Swarm's queen object.

`#include <queen.hh>`

Inheritance diagram for Condel::cQueen:



Collaboration diagram for Condel::cQueen:



Public Member Functions

- virtual **cAttributeSet getAttributes** (void) const

  Get all attributes.

- virtual void **setAttributes** (const **cAttributeSet** &na)

  Set all attributes.

- virtual **cQueen** ∗ **clone** (void) const

  Create a deep copy of this instance.

- virtual void **simulate** (double newtime, **cLoadMeter** &load)

  Simulate this instance.

Static Public Member Functions

- static **cInstance** ∗ **create** (const **OBJECT_ID** &id, const string &name)

Protected Member Functions

- **cQueen** (const **OBJECT_ID** &id, const string &name)
- double **check_critical** (**cPolar** &warrow)

  Find out if current move will collide with a wall.

- void **avoid_wall** (**cPolar** &warrow)

  Turn away from the wall.

Static Protected Attributes

- static **ATTRIBUTE_ID s_positionXID** = cAttributeFactory::registerAttribute("cQueen::positionX")

  X-position of queen.

- static **ATTRIBUTE_ID s_positionYID** = cAttributeFactory::registerAttribute("cQueen::positionY")

  Y-position of queen.

- static **ATTRIBUTE_ID s_velocityAID** = cAttributeFactory::registerAttribute("cQueen::velocityAbs")

  X-velocity of queen.

- static **ATTRIBUTE_ID s_velocityPID** = cAttributeFactory::registerAttribute("cQueen::velocityPhi")

  Y-velocity of queen.

- static **ATTRIBUTE_ID s_accelerationAID** = cAttributeFactory::registerAttribute("cQueen::accelerationAbs")

  X-acceleration of queen.

- static **ATTRIBUTE_ID s_accelerationPID** = cAttributeFactory::registerAttribute("cQueen::accelerationPhi")

  Y-acceleration of queen.

- static **ATTRIBUTE_ID s_seedID** = cAttributeFactory::registerAttribute("cQueen::seed")

  Queen's random seed, used for random-walk.

9.26.1 Detailed Description

Swarm's queen object.

The queen is computer-controlled and does a random walk. This random walk avoids running into walls. The randomness of its walk is controlled by a builtin random seed.

Just like any other bee, the queen cannot simply set its position or its velocity, it can only control its acceleration.

9.26.2 Member Function Documentation

9.26.2.1 void cQueen::avoid_wall (cPolar & *warrow*) [protected]

Turn away from the wall.

Parameters:
warrow Points to a wall.

9.26.2.2 double cQueen::check_critical (cPolar & *warrow*) [protected]

Find out if current move will collide with a wall.

Parameters:
warrow Points to a wall.

Returns:
A critical number. If this is >0, then an action is required.

9.26.2.3 cQueen * cQueen::clone (void) const [virtual]

Create a deep copy of this instance.

Returns:
An exact copy of this instance, including name. This must not be added to the same **cWorld**(p. 152) that this instance resides in.

Reimplemented from **Condel::cBee** (p. 46).

9.26.2.4 cAttributeSet cQueen::getAttributes (void) const [virtual]

Get all attributes.

Packs the current state of this instance into a **cAttributeSet**(p. 41).

Returns:
A copy of the current state.

Reimplemented from **Condel::cBee** (p. 46).

9.26.2.5 void cQueen::setAttributes (const cAttributeSet & *na*) [virtual]

Set all attributes.

Overwrite the internal state.

Parameters:
na A new attribute set. Every attribute found in na is overwritten locally. It is legal to pass a sparse set here.

Reimplemented from **Condel::cBee** (p. 47).

9.26.2.6 void cQueen::simulate (double *newtime*, cLoadMeter & *load*) [virtual]

Simulate this instance.

Picks a random-walk direction. Checks if this would run into a wall and avoids that if necessary. Finally, call **cBee::simulate()**(p. 47) for everything else.

Parameters:

newtime A simulation time in the future of **simTime()**(p. 67). Only the difference between the two times is advanced.

load A tool to measure CPU and memory usage during the simulation.

Reimplemented from **Condel::cBee** (p. 47).

The documentation for this class was generated from the following files:

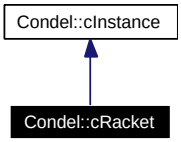
- queen.hh
- queen.cc

9.27 Condel::cRacket Class Reference

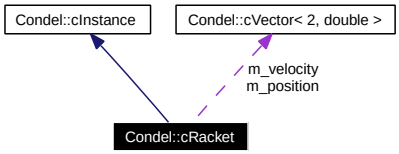
Pong's racket object.

`#include <racket.hh>`

Inheritance diagram for Condel::cRacket:



Collaboration diagram for Condel::cRacket:



Public Member Functions

- **cRacket** (const **OBJECT_ID** &id, const string &name)
- void **setPosition** (const **R_2** &pos)
- void **setVelocity** (const **R_2** &pos)
- const **R_2** & **getPosition** (void) const
- **R_2** & **getPosition** (void)
- const **R_2** & **getVelocity** (void) const
- **R_2** & **getVelocity** (void)
- double **upperEnd** (void) const
- double **lowerEnd** (void) const
- virtual **cAttributeSet** **getAttributes** (void) const

  Get all attributes.

- virtual void **setAttributes** (const **cAttributeSet** &na)

  Set all attributes.

- virtual **cRacket** * **clone** (void) const

  Create a deep copy of this instance.

- virtual double **getDistance** (const **cInstance** &other) const

  Compares two instances.

- virtual void **simulate** (double newtime, **cLoadMeter** &load)

  Simulate this instance.

Static Public Member Functions

- static **cInstance** * **create** (const **OBJECT_ID** &id, const string &name)

Static Public Attributes

- static **ATTRIBUTE_ID** **s_positionXID** = cAttributeFactory::registerAttribute("cRacket::positionX")
  racket's X position.
- static **ATTRIBUTE_ID** **s_positionYID** = cAttributeFactory::registerAttribute("cRacket::positionY")
  racket's Y position
- static **ATTRIBUTE_ID** **s_velocityXID** = cAttributeFactory::registerAttribute("cRacket::velocityX")
  racket's X velocity.
- static **ATTRIBUTE_ID** **s_velocityYID** = cAttributeFactory::registerAttribute("cRacket::velocityY")
  racket's Y velocity

9.27.1 Detailed Description

Pong's racket object.

This represents the players.

Note there are *two* racket objects, one for the left player and one for the right player.

9.27.2 Member Function Documentation

9.27.2.1 cRacket * cRacket::clone (void) const [virtual]

Create a deep copy of this instance.

Returns:
An exact copy of this instance, including name. This must not be added to the same **cWorld**(p. 152) that this instance resides in.

Implements **Condel::cInstance** (p. 65).

9.27.2.2 cAttributeSet cRacket::getAttributes (void) const [virtual]

Get all attributes.

Packs the current state of this instance into a **cAttributeSet**(p. 41).

Returns:
A copy of the current state.

Implements **Condel::cInstance** (p. 66).

9.27.2.3 double cRacket::getDistance (const cInstance & *other*) const [virtual]

Compares two instances.

Normally, this is based on the cartesian distance. For the ideal case, velocity is also used.

Parameters:

other Another instance of the same object.

Returns:

A metric of the similarity of the two instances.

Implements **Condel::cInstance** (p. 66).

9.27.2.4 void cRacket::setAttributes (const cAttributeSet & *na*) [virtual]

Set all attributes.

Overwrite the internal state.

Parameters:

na A new attribute set. Every attribute found in na is overwritten locally. It is legal to pass a sparse set here.

Implements **Condel::cInstance** (p. 67).

9.27.2.5 void cRacket::simulate (double *newtime*, cLoadMeter & *load*) [virtual]

Simulate this instance.

Move the racket according to its speed. Stop at upper and lower boundaries.

Parameters:

newtime A simulation time in the future of **simTime()**(p. 67). Only the difference between the two times is advanced.

load A tool to measure CPU and memory usage during the simulation.

Implements **Condel::cInstance** (p. 67).

9.27.3 Member Data Documentation

9.27.3.1 ATTRIBUTE_ID cRacket::s_positionYID = cAttributeFactory::registerAttribute("cRacket::positionY") [static]

racket's Y position

Note this is constant.

9.27.3.2 ATTRIBUTE_ID cRacket::s_velocityYID = cAttributeFactory::registerAttribute("cRacket::velocityY") [static]

racket's Y velocity

Note this is 0.

The documentation for this class was generated from the following files:

- racket.hh
- racket.cc

9.28 Condel::cRenderVec Struct Reference

Helper object for visualization.

`#include <pong_scenario.hh>`

Public Attributes

- cOutVector **posX**
- cOutVector **posY**
- cOutVector **velX**
- cOutVector **velY**

9.28.1 Detailed Description

Helper object for visualization.

This is used in dumping all data for later visualization. Only used within the scenario-file.

The documentation for this struct was generated from the following files:

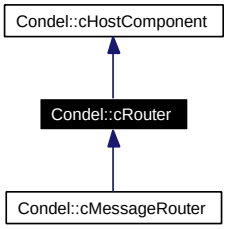
- pong_scenario.hh
- swarm_scenario.hh

9.29 Condel::cRouter Class Reference

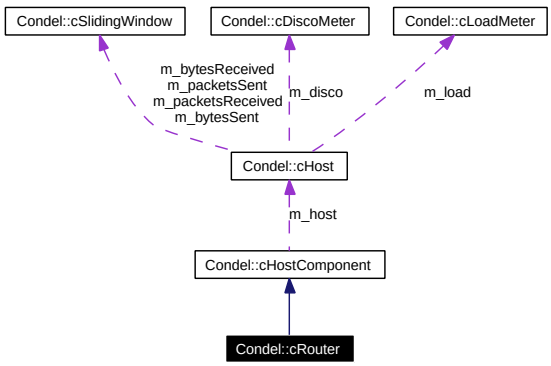
The basic netlayer interface.

`#include <router.hh>`

Inheritance diagram for Condel::cRouter:



Collaboration diagram for Condel::cRouter:



Public Member Functions

- **cRouter** (**cHost** &host)
- virtual void **send** (**cMsg** *pMsg)=0

  Send a message to the network.

- virtual void **handleMessage** (**cMsg** *pMsg)=0

  Receive a message from the network.

- virtual void **handleTick** (void)

  Called each tick.

- virtual void **handleCheck** (void)

  Called each check-interval.

- list< unsigned long >::const_iterator **beginHID** (void) const

  An iterator.

- list< unsigned long >::const_iterator **endHID** (void) const

  An iterator.

- virtual void **initialize** (void)

  Reset everything.

- virtual void **finish** (void)

  System Done.

Protected Attributes

- list< unsigned long > **m_hid**

9.29.1 Detailed Description

The basic netlayer interface.

This is the common base class of all actual routing/network libraries.

Routers basically send and receive information to/from other hosts on the networks. They contain the algorithms needed to find lists of other valid hosts, send packets to a subset of them, even participate in a peer2peer network with other hosts.

9.29.2 Member Function Documentation

9.29.2.1 virtual void Condel::cRouter::finish (void) [inline, virtual]

System Done.

This member is called by the host when the simulation has ended.

Reimplemented in **Condel::cMessageRouter** (p. 82).

9.29.2.2 virtual void Condel::cRouter::handleCheck (void) [inline, virtual]

Called each check-interval.

Called regularly (each **cHost::CHECK_TIME**(p. 62)) by the host.

Reimplemented in **Condel::cMessageRouter** (p. 82).

9.29.2.3 virtual void Condel::cRouter::handleMessage (cMsg * *pMsg*) [pure virtual]

Receive a message from the network.

This is called by the host (i.e., a subclass of **cHost**(p. 58)).

Parameters:

pMsg A pointer to the message. The host (i.e. the caller of this function) retains ownership of the message, do NOT delete it. If you need your own copy, you'll have to dup() it.

Implemented in **Condel::cMessageRouter** (p. 83).

9.29.2.4 virtual void Condel::cRouter::handleTick (void) [inline, virtual]

Called each tick.

Called regularly (each **cHost::TICK_TIME**(p. 62)) by the host.

9.29.2.5 void cRouter::initialize (void) [virtual]

Reset everything.

This member is called by the host when the simulation is about to start.

Reimplemented in **Condel::cMessageRouter** (p. 83).

9.29.2.6 virtual void Condel::cRouter::send (cMsg * *pMsg*) [pure virtual]

Send a message to the network.

Note the message is sent to ALL hosts.

Parameters:

pMsg A pointer to the message. This message will be duped during this call, the original caller retains ownership of the supplied msg, so you may delete it after the call.

Implemented in **Condel::cMessageRouter** (p. 83).

The documentation for this class was generated from the following files:

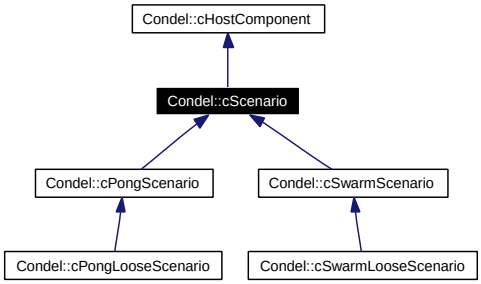
- router.hh
- router.cc

9.30 Condel::cScenario Class Reference

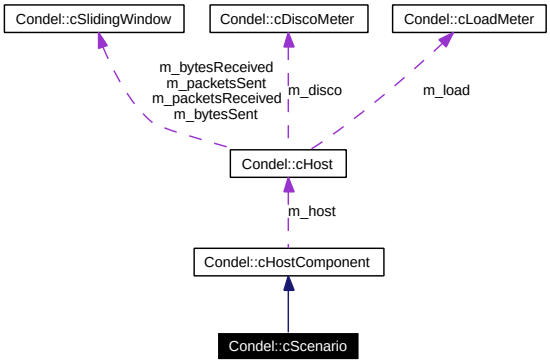
The basic scenario interface.

`#include <scenario.hh>`

Inheritance diagram for Condel::cScenario:



Collaboration diagram for Condel::cScenario:



Public Member Functions

- **cScenario** (**cHost** &host)
- void **setDelayHint** (double newhint)

  This is a cludge, the AI could *find this out on its own.*

- virtual void **initialize** (void)

  Reset everything.

- virtual void **simulate** (**cWorld** &world, double newtime, **cLoadMeter** &load) const =0

  Perform deterministic actions.

- virtual void **ai** (double newtime)=0

  Perform the AI.

- virtual void **finish** (void)=0

  System Done.

Protected Attributes

- double **m_delayHint**

9.30.1 Detailed Description

The basic scenario interface.

This is the common base class of all scenario codes.

The scenario contains all the code that represent the particular game etc. to be simulated. This includes in-game rules, initialization code, and the AI.

9.30.2 Member Function Documentation

9.30.2.1 virtual void Condel::cScenario::ai (double *newtime*) [pure virtual]

Perform the AI.

This function aims to supplant the user and therefore issues the nondeterministic actions a human user would cause.

This will look at the main world (the visible one) available through the host only.

Parameters:
newtime The current time.

Implemented in **Condel::cPongScenario** (p. 111), and **Condel::cSwarmScenario** (p. 145).

9.30.2.2 virtual void Condel::cScenario::finish (void) [pure virtual]

System Done.

This member is called by the host when the simulation has ended.

Implemented in **Condel::cPongScenario** (p. 111), and **Condel::cSwarmScenario** (p. 145).

9.30.2.3 virtual void Condel::cScenario::initialize (void) [inline, virtual]

Reset everything.

This member is called by the host when the simulation is about to start. This does NOT initialize **setDelayHint()**(p. 126), it's set before this call.

Reimplemented in **Condel::cPongScenario** (p. 112), **Condel::cSwarmScenario** (p. 145), and **Condel::cPongLooseScenario** (p. 107).

9.30.2.4 void Condel::cScenario::setDelayHint (double *newhint*) [inline]

This is a cludge, the AI *could* find this out on its own.

Human users, for example, adjust to lag within seconds. But the information is known, so if it is also constant, it is easiest to pass it through here.

9.30.2.5 virtual void Condel::cScenario::simulate (cWorld & *world*, double *newtime*, cLoadMeter & *load*) const [pure virtual]

Perform deterministic actions.

This function performs all deterministic actions in the given world. In effect, it calls **simulate()**(p. 127) on all instances in world, then does all the things that happen in the scenario that are NOT done in the single instance's simulate function, e.g. collision detection. This is const, so side-effects can only happen in the instances.

Parameters:

world The world that is advanced.

newtime Time to advance the system to. This should be in the future of the world's **cWorld::simTime()**(p. 156). Given in seconds.

load A tool to measure CPU and memory usage during the simulation.

Implemented in **Condel::cPongScenario** (p. 112), **Condel::cSwarmScenario** (p. 146), and **Condel::cPongLooseScenario** (p. 107).

The documentation for this class was generated from the following files:

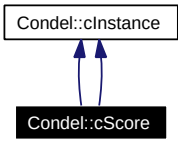
- scenario.hh
- scenario.cc

9.31 Condel::cScore Class Reference

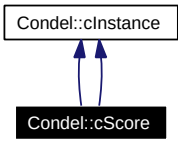
Pong's score object.

`#include <score.hh>`

Inheritance diagram for Condel::cScore:



Collaboration diagram for Condel::cScore:



Public Member Functions

- **cScore** (const **OBJECT_ID** &id, const string &name)
- void **setScore** (double score)
- double **getScore** (void) const
- double & **getScore** (void)
- virtual **cAttributeSet** **getAttributes** (void) const

  Get all attributes.

- virtual void **setAttributes** (const **cAttributeSet** &na)

  Set all attributes.

- virtual **cScore** * **clone** (void) const

  Create a deep copy of this instance.

- virtual double **getDistance** (const **cInstance** &other) const

  Compares two instances.

- virtual void **simulate** (double newtime, **cLoadMeter** &load)

  Simulate this instance.

- **cScore** (const **OBJECT_ID** &id, const string &name)
- void **setScore** (double score)
- double **getScore** (void) const
- double & **getScore** (void)
- virtual **cAttributeSet** **getAttributes** (void) const

  Get all attributes.

- virtual void **setAttributes** (const **cAttributeSet** &na)

  Set all attributes.

- virtual **cScore** ∗ **clone** (void) const

  Create a deep copy of this instance.

- virtual double **getDistance** (const **cInstance** &other) const

  Compares two instances.

- virtual void **simulate** (double newtime, **cLoadMeter** &load)

  Simulate this instance.

Static Public Member Functions

- static **cInstance** ∗ **create** (const **OBJECT_ID** &id, const string &name)
- static **cInstance** ∗ **create** (const **OBJECT_ID** &id, const string &name)

9.31.1 Detailed Description

Pong's score object.

This is used for scorekeeping; it's actually just an integer.

Note there are *two* score objects, one for the left player and one for the right player.

9.31.2 Member Function Documentation

9.31.2.1 virtual cScore∗ Condel::cScore::clone (void) const [virtual]

Create a deep copy of this instance.

Returns:
An exact copy of this instance, including name. This must not be added to the same **cWorld**(p. 152) that this instance resides in.

Implements **Condel::cInstance** (p. 65).

9.31.2.2 cScore ∗ cScore::clone (void) const [virtual]

Create a deep copy of this instance.

Returns:
An exact copy of this instance, including name. This must not be added to the same **cWorld**(p. 152) that this instance resides in.

Implements **Condel::cInstance** (p. 65).

9.31.2.3 virtual cAttributeSet Condel::cScore::getAttributes (void) const [virtual]

Get all attributes.

Packs the current state of this instance into a **cAttributeSet**(p. 41).

Returns:
A copy of the current state.

Implements **Condel::cInstance** (p. 66).

9.31.2.4 cAttributeSet cScore::getAttributes (void) const [virtual]

Get all attributes.

Packs the current state of this instance into a **cAttributeSet**(p. 41).

Returns:
A copy of the current state.

Implements **Condel::cInstance** (p. 66).

9.31.2.5 virtual double Condel::cScore::getDistance (const cInstance & *other*) const [virtual]

Compares two instances.

This measures the similarity, in an appropriate way fitting for this object. Please note that this means that this metric is not comparable between different instances of different objects, i.e. the distance between two balls cannot be compared to the distance between two scores.

Parameters:
other Another instance of the same object.

Returns:
A metric of the similarity of the two instances.

Implements **Condel::cInstance** (p. 66).

9.31.2.6 double cScore::getDistance (const cInstance & *other*) const [virtual]

Compares two instances.

This measures the similarity, in an appropriate way fitting for this object. Please note that this means that this metric is not comparable between different instances of different objects, i.e. the distance between two balls cannot be compared to the distance between two scores.

Parameters:
other Another instance of the same object.

Returns:
A metric of the similarity of the two instances.

Implements **Condel::cInstance** (p. 66).

9.31.2.7 virtual void Condel::cScore::setAttributes (const cAttributeSet & *na*) [virtual]

Set all attributes.

Overwrite the internal state.

Parameters:

na A new attribute set. Every attribute found in na is overwritten locally. It is legal to pass a sparse set here.

Implements **Condel::cInstance** (p. 67).

9.31.2.8 void cScore::setAttributes (const cAttributeSet & *na*) [virtual]

Set all attributes.

Overwrite the internal state.

Parameters:

na A new attribute set. Every attribute found in na is overwritten locally. It is legal to pass a sparse set here.

Implements **Condel::cInstance** (p. 67).

9.31.2.9 virtual void Condel::cScore::simulate (double *newtime*, cLoadMeter & *load*) [virtual]

Simulate this instance.

Does nothing.

Parameters:

newtime A simulation time in the future of **simTime()**(p. 67). Only the difference between the two times is advanced.

load A tool to measure CPU and memory usage during the simulation.

Implements **Condel::cInstance** (p. 67).

9.31.2.10 void cScore::simulate (double *newtime*, cLoadMeter & *load*) [virtual]

Simulate this instance.

Does nothing.

Parameters:

newtime A simulation time in the future of **simTime()**(p. 67). Only the difference between the two times is advanced.

load A tool to measure CPU and memory usage during the simulation.

Implements **Condel::cInstance** (p. 67).

The documentation for this class was generated from the following files:

- pong/score.hh
- swarm/score.hh
- pong/score.cc
- swarm/score.cc

9.32 Condel::cSlidingWindow Class Reference

Gather statistical data in a sliding window.

`#include <sliding_window.hh>`

Public Member Functions

- **cSlidingWindow** (double windowSizeTime, double currentTime=0.0, bool exclusive=true)

  Construct a new sliding window.

- void **initialize** (double start)

  Reset everything.

- void **collect** (double time, double value)

  Gather data.

- double **getAverage** (double time=-1.0)

  Get results gathered so far.

- double **getMax** (double time=-1.0)

  Get the highest measured result so far.

- double **getTotalAverage** (double time=-1.0)

  Get the average over all time.

9.32.1 Detailed Description

Gather statistical data in a sliding window.

This window will give you moving averages over a specified length. Note the window-size will be smaller as long as the measurement-time is below windowSizeTime.

Todo

This should inherit from OMNeT++'s cStatistic some day

9.32.2 Constructor & Destructor Documentation

9.32.2.1 cSlidingWindow::cSlidingWindow (double *windowSizeTime*, double *currentTime* = `0.0`, bool *exclusive* = `true`)

Construct a new sliding window.

This window will supply you with moving averages over the specified length. The applicable data is the cost of any ongoing process, such as bytes sent out over a network or energy units used. Start is the time when the window is created.

Parameters:

windowSizeTime The length of the measuring window in seconds. Note the window-size will be smaller as long as the measurement-time is below windowSizeTime.

currentTime The start time of this measuring window.

exclusive If true, the data that will be given via **collect()**(p. 134) applies to AFTER the time given in **collect()**(p. 134). If false, the data applies to BEFORE the time given in **collect()**(p. 134). Use the true for data you're about to send off, false for data already received.

9.32.3 Member Function Documentation

9.32.3.1 void cSlidingWindow::collect (double *time*, double *value*)

Gather data.

Parameters:

time The time this data applies to. The time *must* be non-decreasing with previous calls (and also with calls to **getAverage()**(p. 134)). Also, time must be >= 0.0.

See also:

exclusive in **cSlidingWindow()**(p. 133)

Parameters:

value The data to be gathered. No interpretation here, you may use this object for anything you like.

9.32.3.2 double cSlidingWindow::getAverage (double *time* = `-1.0`)

Get results gathered so far.

Parameters:

time The time for which to calculate the moving average. The time *must* be non-decreasing with previous calls (and also with calls to **collect()**(p. 134)). Also, time must be >= 0. If time is not given, the time of the latest call to either collect or getMean() is used.

Returns:

A unit per second average of all data from time -windowSize to time.

9.32.3.3 double cSlidingWindow::getMax (double *time* = `-1.0`)

Get the highest measured result so far.

Returns:

The maximum calculated average. An average is calculated for every call to **collect()**(p. 134) and to **getAverage()**(p. 134). Note this is not the maximum of the collected data. Note that because the window-size is smaller in the beginning, the first measurements would be more erratic and therefore have a tendency to dominate this value here. For this reason, this value is -infinity until time>=windowSize.

9.32.3.4 double cSlidingWindow::getTotalAverage (double *time* = -1.0)

Get the average over all time.

Parameters:
time The current time.

Returns:
A unit per second measurement running over all data ever given.

9.32.3.5 void cSlidingWindow::initialize (double *start*)

Reset everything.

Parameters:
start The new starting time, in seconds.

The documentation for this class was generated from the following files:

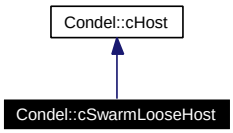
- sliding_window.hh
- sliding_window.cc

9.33 Condel::cSwarmLooseHost Class Reference

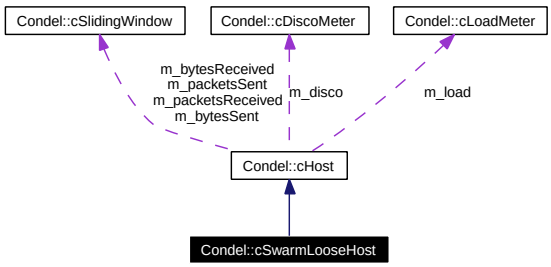
The host for Swarm with loose consistency.

`#include <swarm_loose_host.hh>`

Inheritance diagram for Condel::cSwarmLooseHost:



Collaboration diagram for Condel::cSwarmLooseHost:



Public Member Functions

- **cSwarmLooseHost** (void)
  The constructor.
- virtual void **initialize** (void)
  Reset everything.
- virtual void **finish** (void)
  System Done.
- virtual void **initialize_2nd** (void)
  A secondary initialization phase.
- virtual void **handleMessage** (cMessage *pMsg)
  Accept a message.
- virtual void **handleTick** (void)
  A frame has passed.

9.33.1 Detailed Description

The host for Swarm with loose consistency.

This is the class that creates the entire loose consistency/swarm system.

It also does quite a few extra things having to do with creating a complete host list. This host list ist used for:

- Move the ownership of the score from each host to the next once. Thus, no host owns its own score (fairness!).
- Move the ownership of the queen in a round-robin manner to the next host every few seconds (SWITCHOVER_TIME).

To be able to construct such a list, all hosts must have already registered, which they do during the **initialize()**(p. 138) call. To construct the list, a *second* initialize phase is added.

Note that the whole list is a cludge, but the addressing *could* be done in a clean way without adding noticable network traffic, we just chose to go the easy route.

9.33.2 Constructor & Destructor Documentation

9.33.2.1 cSwarmLooseHost::cSwarmLooseHost (void)

The constructor.

This sets cHost's component pointers to components instantiated here. The components chosen are **cSwarmLooseScenario**(p. 139), **cLooseConsistency**(p. 74), etc..

9.33.3 Member Function Documentation

9.33.3.1 void cSwarmLooseHost::finish (void) [virtual]

System Done.

This member is called by OMNeT when the simulation has ended.

Reimplemented from **Condel::cHost** (p. 60).

9.33.3.2 void cSwarmLooseHost::handleMessage (cMessage * *pMsg*) [virtual]

Accept a message.

This can either be a network message sent by one of the other simulated hosts or it could be some tick event. It is overloaded here so that the 2nd initialization event can be caught. Other than that, the parent's **cHost::handleMessage()**(p. 60) is called. This is called by OMNeT (and ONLY by OMNeT, please).

Parameters:

pMsg A pointer to the message. OMNeT (i.e. the caller of this function) retains ownership of the message, do NOT delete it. If you need your own copy, you'll have to dup() it.

Reimplemented from **Condel::cHost** (p. 60).

9.33.3.3 void cSwarmLooseHost::handleTick (void) [virtual]

A frame has passed.

Call **cHost::handleTick()**(p. 61), then take/yield ownership of the queen in a round-robin scheme. This only happens every SWITCHOVER_TIME seconds. This is called at the end of every frame, which are TICK_TIME seconds apart. Called by **handleMessage(cMessage *)**(p. 137).

Reimplemented from **Condel::cHost** (p. 61).

9.33.3.4 void cSwarmLooseHost::initialize (void) [virtual]

Reset everything.

This registers the current host in **cSwarmScenario::registerHost()**(p. 145), calls the parent's **cHost::initialize()**(p. 61), Then sets up a second initialization phase. This member is called by OMNeT when the simulation is about to start.

Reimplemented from **Condel::cHost** (p. 61).

9.33.3.5 void cSwarmLooseHost::initialize_2nd (void) [virtual]

A secondary initialization phase.

This is called right after all hosts have initialized. This sets the player/ghost distances, then finds out where this host is in the host list, finally takes control of the prior host's score and yields its own score.

The documentation for this class was generated from the following files:

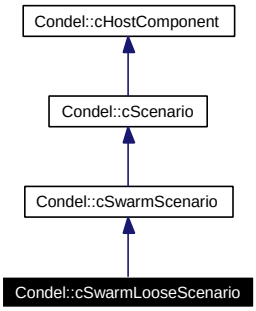
- swarm_loose_host.hh
- **swarm_loose_host.cc**

9.34 Condel::cSwarmLooseScenario Class Reference

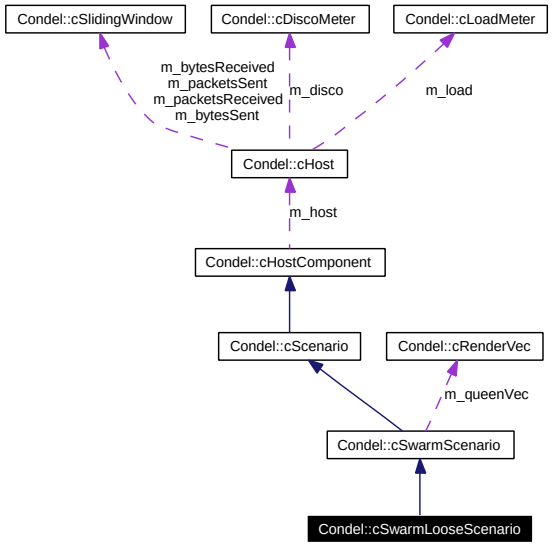
The Swarm scenario with loose consistency.

`#include <swarm_loose_scenario.hh>`

Inheritance diagram for Condel::cSwarmLooseScenario:



Collaboration diagram for Condel::cSwarmLooseScenario:



Public Member Functions

- **cSwarmLooseScenario** (**cHost** &host)

9.34.1 Detailed Description

The Swarm scenario with loose consistency.

This is currently just a wrapper for **cSwarmScenario**(p. 143), it does not do anything additional.

The documentation for this class was generated from the following files:

- swarm_loose_scenario.hh

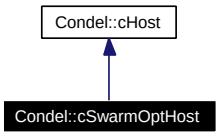
- swarm_loose_scenario.cc

9.35 Condel::cSwarmOptHost Class Reference

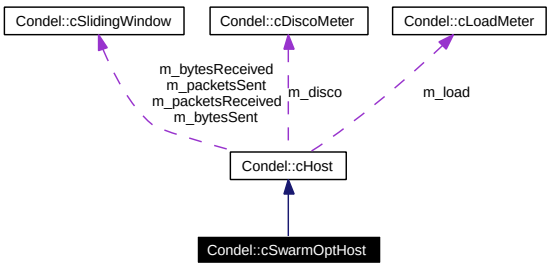
The host for Swarm with optimistic consistency.

`#include <swarm_opt_host.hh>`

Inheritance diagram for Condel::cSwarmOptHost:



Collaboration diagram for Condel::cSwarmOptHost:



Public Member Functions

- **cSwarmOptHost** (void)

  The constructor.

- virtual void **initialize** (void)

  Reset everything.

Protected Attributes

- unsigned long **m_id**

9.35.1 Detailed Description

The host for Swarm with optimistic consistency.

This is the class that creates the entire optimistic consistency/swarm system.

9.35.2 Constructor & Destructor Documentation

9.35.2.1 cSwarmOptHost::cSwarmOptHost (void)

The constructor.

This sets cHost's component pointers to components instantiated here. The components chosen are **cSwarmScenario**(p. 143), **cOptConsistency**(p. 95), etc..

9.35.3 Member Function Documentation

9.35.3.1 void cSwarmOptHost::initialize (void) [virtual]

Reset everything.

Sets up the tick events (both tick and check) and reads the parameters from the OMNeT ini files. This member is called by OMNeT when the simulation is about to start.

Reimplemented from **Condel::cHost** (p. 61).

The documentation for this class was generated from the following files:

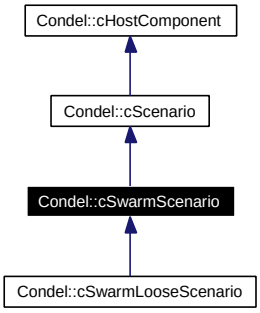
- swarm_opt_host.hh
- swarm_opt_host.cc

9.36 Condel::cSwarmScenario Class Reference

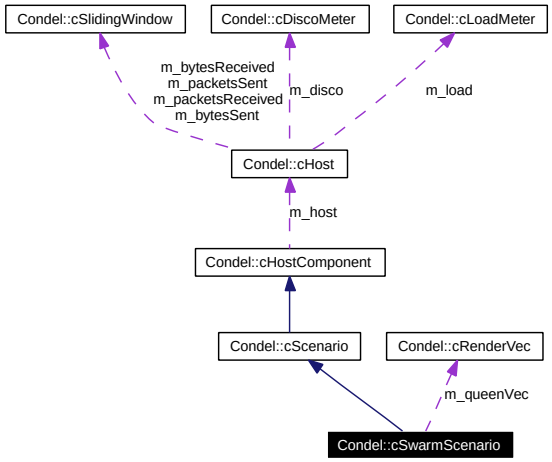
The Swarm Scenario.

`#include <swarm_scenario.hh>`

Inheritance diagram for Condel::cSwarmScenario:



Collaboration diagram for Condel::cSwarmScenario:



Public Member Functions

- **cSwarmScenario** (**cHost** &host)
- virtual void **registerHost** (unsigned long id)

  Register a new host/player/bee.

- void **initialize** (void)

  Reset everything.

- virtual void **simulate** (**cWorld** &world, double newtime, **cLoadMeter** &load) const

  Perform deterministic actions.

- virtual void **ai** (double newtime)

  Perform the AI.

- virtual void **finish** (void)

  System Done.

Static Public Attributes

- static const double **LEFT_BORDER** = 0

  X-coordinate of the left border.

- static const double **RIGHT_BORDER** = 1000

  X-coordinate of the right border.

- static const double **TOP_BORDER** = 0

  Y-coordinate of the upper border.

- static const double **BOTTOM_BORDER** = 500

  Y-coordinate of the lower border.

- static **OBJECT_ID s_queenID** = cObjectFactory::registerObject("cQueen#")
- static map< unsigned long, string > **s_beeName**
- static map< unsigned long, string > **s_scoreName**

Protected Member Functions

- void **initRendering** (void)
- void **render** (void)
- void **collision_analyze** (const **cBee** &me, const **cBee** &other, double &time, **cPolar** &v) const

Protected Attributes

- bool **m_doRender**
- **cRenderVec m_queenVec**
- map< unsigned long, shared_ptr< **cRenderVec** > > **m_beeVec**
- map< unsigned long, shared_ptr< cOutVector > > **m_deadVec**
- map< unsigned long, shared_ptr< cOutVector > > **m_scoreVec**
- double **m_lastScore**
- double **m_scoreCounter**
- long **m_actionCount**
- double **m_safetyQueen**
- double **m_safetyBee**
- double **m_lastDead**
- double **m_lastDecisionAbs**
- double **m_lastDecisionPhi**

9.36.1 Detailed Description

The Swarm Scenario.

This is a simple non-cooperative multi-player game. A computer-controlled queen performs a random-walk and each player attempts to stay as close to the queen as possible. Every tick, honey is awarded to each bee according to their distance from the queen (shorter distance -> more honey). If a bee collides with any other (or the queen), it is considered dead for a short time, not collecting honey and not being able to control its flight.

There is a physical model installed, each bee represents a fully elastic solid ball. If a bee collides with the queen, the queen is considered a wall (i.e. it has infinite weight).

If the render parameter is true for this host, positions of all bees and all scores will be logged to the omnetpp.vec file each frame. This can be used to create a visualization later.

9.36.2 Member Function Documentation

9.36.2.1 void cSwarmScenario::ai (double *newtime*) [virtual]

Perform the AI.

A form of potential field/gradient steering is performed to move towards the queen but stay away from all obstacles (including the queen itself).

A simple learning algorithm is installed to adjust to the level of collisions that occur. The algorithm adjusts the safety boarders in the steering algorithm.

Parameters:
newtime The current time.

Implements **Condel::cScenario** (p. 126).

9.36.2.2 void cSwarmScenario::finish (void) [virtual]

System Done.

Write the score (total honey) to the log file (it's the application dependant yield measure).

Implements **Condel::cScenario** (p. 126).

9.36.2.3 void cSwarmScenario::initialize (void) [virtual]

Reset everything.

Create your own bee and score. One player (controlled by start_queen parameter) creates the queen.

Reimplemented from **Condel::cScenario** (p. 126).

9.36.2.4 void cSwarmScenario::registerHost (unsigned long *id*) [virtual]

Register a new host/player/bee.

This is a cludge to allow SwarmLooseHost to find the relevant other hosts fast&easy. Should soon be replaced by a true join/leave function.

Parameters:

id The id of the newly joined host.

9.36.2.5 void cSwarmScenario::simulate (cWorld & *world*, double *newtime*, cLoadMeter & *load*) const [virtual]

Perform deterministic actions.

Let each instance in the world simulate to newtime (moves ball/rackets). Then, check for collisions (a-posteriori). Resolve them if necessary. Award honey to each bee.

Parameters:

world The world that is advanced.

newtime Time to advance the system to. This should be in the future of the world's **cWorld::simTime()**(p. 156). Given in seconds.

load A tool to measure CPU and memory usage during the simulation.

Implements **Condel::cScenario** (p. 127).

The documentation for this class was generated from the following files:

- swarm_scenario.hh
- swarm_scenario.cc

9.37 Condel::cVarianceMeter Class Reference

Computes the divergence measure D.

`#include <variance_meter.hh>`

Public Member Functions

- void **addHost** (const **cHost** *pHost)

  Add a new host to the watchlist.

- void **addAttribute** (const **ATTRIBUTE_ID** &attribute, double weight)

  Add a new attribute to the watchlist.

- void **initWeights** (void)

  Run this once after all attributes have been added.

- double **computeConsistency** (void)

  Tally the system divergence.

- void **init** (map< const **ATTRIBUTE_ID**, double > &attributeNorm)

  Reinitialize this tool.

- double **averageConsistency2** (void) const

  Get the arithmetic average so far.

- double **averageConsistency2** (**ATTRIBUTE_ID** aid) const

  Get the arithmetic average so far.

- double **averageConsistency1** (void) const

  Get the arithmetic average so far.

- double **averageConsistency1** (**ATTRIBUTE_ID** aid) const

  Get the arithmetic average so far.

9.37.1 Detailed Description

Computes the divergence measure D.

This tool is activated by **cEye**(p. 56), which ticks just after every normal tick. It traverses all hosts and measures the std deviation of the attributes of corresponding instances.

Differences can affect position as well as speeds, even angles and accelerations. Since these are perceived differently by human observers, each attribute needs to be weighted.

See also:
cDiscoMeter(p. 53)

9.37.2 Member Function Documentation

9.37.2.1 void cVarianceMeter::addAttribute (const ATTRIBUTE_ID & *attribute*, double *weight*)

Add a new attribute to the watchlist.

Parameters:

attribute The ID of the new attribute.

weight The relative importance of the attribute to the overall perception of disontinuity. In definitionen.pdf, this would be w_a.

9.37.2.2 void cVarianceMeter::addHost (const cHost * *pHost*)

Add a new host to the watchlist.

Parameters:

pHost The new host.

9.37.2.3 double cVarianceMeter::averageConsistency1 (ATTRIBUTE_ID *aid*) const

Get the arithmetic average so far.

Parameters:

aid An attribute. Note this attribute may exist in several different objects.

Returns:

The average of the standard deviation calculated for the specified attribute each frame.

9.37.2.4 double cVarianceMeter::averageConsistency1 (void) const

Get the arithmetic average so far.

Returns:

The average of the total standard deviation calculated each frame.

9.37.2.5 double cVarianceMeter::averageConsistency2 (ATTRIBUTE_ID *aid*) const

Get the arithmetic average so far.

Parameters:

aid An attribute. Note this attribute may exist in several different objects.

Returns:

The average of the variance calculated for the specified attribute each frame.

9.37.2.6 double cVarianceMeter::averageConsistency2 (void) const

Get the arithmetic average so far.

Returns:
The average of the total variances calculated each frame.

9.37.2.7 double cVarianceMeter::computeConsistency (void)

Tally the system divergence.

Compute the entire system's current consistency as defined by the divergence metric. $v := \sum(x_i - \bar{x})^2/N$, $\sigma := \sqrt{(}v)$

9.37.2.8 void cVarianceMeter::init (map< const ATTRIBUTE_ID, double > & *attributeNorm*)

Reinitialize this tool.

After this call, the tool looks like right after its creation.

Parameters:
attributeNorm A list of attribute NORMs to use for the attributes. This list is NOT copied, the original version specified here is used. The norms are independant from the weights, they are set by the scenario so that by multiplying it with its attribute, the attribute will have a natural domain of +/- 1. This does not mean the product cannot exceed this range, btw. An example would be the x position, where the weight is 1/width. Another example is score, where weight is 1.

9.37.2.9 void cVarianceMeter::initWeights (void)

Run this once after all attributes have been added.

Only call this ONCE. This fixes the weights to have $\sum w_a == 1$.

The documentation for this class was generated from the following files:

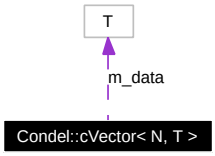
- variance_meter.hh
- variance_meter.cc

9.38 Condel::cVector< N, T > Class Template Reference

An n-dimensional numeric vector.

`#include <vector.hh>`

Collaboration diagram for Condel::cVector< N, T >:



Public Member Functions

- **cVector** (const **cVector**< N, T > &src)
- const **cVector**< N, T > & **assign** (const **cVector**< N, T > &src)
- const **cVector**< N, T > & **assign** (const T &value)

  Set all Elements to the same value.

- const **cVector**< N, T > & **operator**= (const **cVector**< N, T > &src)
- const **cVector**< N, T > & **operator**= (const T &value)
- const T & **at** (unsigned pos) const
- T & **at** (unsigned pos)
- const T & **operator[]** (unsigned pos) const
- T & **operator[]** (unsigned pos)
- const **cVector**< N, T > & **add** (const **cVector**< N, T > &src)
- const **cVector**< N, T > & **subtract** (const **cVector**< N, T > &src)
- const **cVector**< N, T > & **scale** (const T &value)
- const **cVector**< N, T > & **operator**+= (const **cVector**< N, T > &src)
- const **cVector**< N, T > & **operator-**= (const **cVector**< N, T > &src)
- const **cVector**< N, T > & **operator** *= (const T &value)
- **cVector**< N, T > **operator**+ (const **cVector**< N, T > &src) const
- **cVector**< N, T > **operator-** (const **cVector**< N, T > &src) const
- T **operator** * (const **cVector**< N, T > &src) const
- **cVector**< N, T > **operator** * (const T &value) const
- T **scalarProduct** (const **cVector**< N, T > &src) const

9.38.1 Detailed Description

template<unsigned N, class T> class Condel::cVector< N, T >

An n-dimensional numeric vector.

Type T can be anything numerical, but I really recommend (double) for the purposes of Adam.

9.38.2 Member Function Documentation

9.38.2.1 template<unsigned N, class T> const cVector< N, T > & cVector::assign (const T & *value*)

Set all Elements to the same value.

Parameters:

value A numerical value that all components of the vector are set to.

The documentation for this class was generated from the following files:

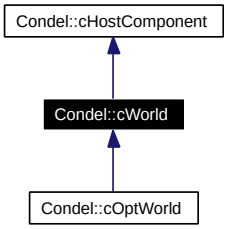
- vector.hh
- vector.cc

9.39 Condel::cWorld Class Reference

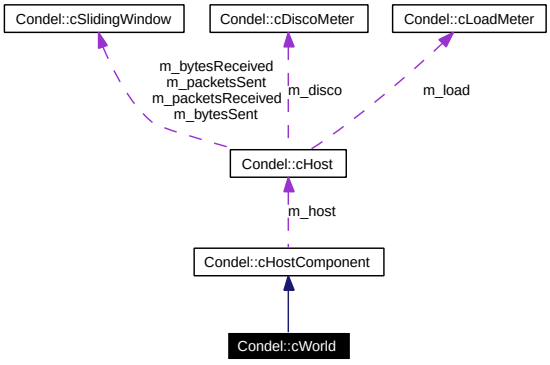
A set of instances.

`#include <world.hh>`

Inheritance diagram for Condel::cWorld:



Collaboration diagram for Condel::cWorld:



Public Types

- typedef map< const **OBJECT_ID**, shared_ptr< **cInstance** > >::**iterator iterator**

  An iterator.

- typedef map< const **OBJECT_ID**, shared_ptr< **cInstance** > >::**const_iterator const_iterator**

  An iterator.

Public Member Functions

- **cWorld** (**cHost** &host)
- **cWorld** (**cHost** &host, const string &name)
- virtual auto_ptr< **cWorld** > **clone** (void)

  Create a deep clone of this world.

- const **cWorld** & **operator**= (const **cWorld** &ow)

  A DEEP assignment.

- const string & **getName** (void) const

  Get the name of this world.

- double **simTime** (void) const

  Get the world's simulation time.

- void **setSimTime** (double newtime)

  Set the world's simulation time.

- bool **isReplicated** (const **OBJECT_ID** &objectID) const

  Find out if an object has an instance in this world.

- **cInstance** & **getInstance** (const **OBJECT_ID** &objectID)

  Get an instance.

- const **cInstance** & **getInstance** (const **OBJECT_ID** &objectID) const

  Get an instance.

- void **addInstance** (const **cInstance** *pInstance)

  Insert a new instance.

- virtual void **simulate** (double newtime, **cLoadMeter** &load)

  Perform deterministic actions.

- unsigned long long **getTotalLength** (void)

  Estimate the entire in-memory size of this world.

- virtual void **initialize** (void)

  Reset everything.

- virtual void **finish** (void)

  System Done.

Protected Attributes

- string **m_name**
- double **m_simtime**

9.39.1 Detailed Description

A set of instances.

A world holds a collection of instances, each with a unique object ID. Note this is A world, not THE world. IOW, it's one of the views of the world a single host can have. The collection of all worlds is called the universe. The universe is also entirely different from the single globally correct world (if there is such a thing, not true in some consistencies).

9.39.2 Member Function Documentation

9.39.2.1 void cWorld::addInstance (const cInstance * *pInstance*)

Insert a new instance.

Parameters:

pInstance A pointer to an instance to be added. An instance of the same object must not yet exist in this world. The instance is cloned during the call; the caller maintains ownership of the passed instance.

Todo

should use a reference, not a pointer.

9.39.2.2 auto_ptr< cWorld > cWorld::clone (void) [virtual]

Create a deep clone of this world.

The world will have all the instances of the original, with the same state, etc. In fact, aside from the address, the new world will be indistinguishable from the old one.

Note this is a very slow process.

Returns:

A pointer to a newly created copy of this world. Assign this to an auto_ptr<cWorld> to avoid any possibility of leakage.

Reimplemented in **Condel::cOptWorld** (p. 100).

9.39.2.3 virtual void Condel::cWorld::finish (void) [inline, virtual]

System Done.

This member is called by the host when the simulation has ended.

9.39.2.4 const cInstance & cWorld::getInstance (const OBJECT_ID & *objectID*) const

Get an instance.

Parameters:

objectID The object ID of an object that *does* have an instance in this world. To ensure this, use **isReplicated()**(p. 153) before this call.

Returns:

A reference to the actual instance, not a copy.

9.39.2.5 cInstance & cWorld::getInstance (const OBJECT_ID & *objectID*)

Get an instance.

Parameters:

objectID The object ID of an object that *does* have an instance in this world. To ensure this, use **isReplicated()**(p. 153) before this call.

Returns:

A reference to the actual instance, not a copy.

9.39.2.6 const string& Condel::cWorld::getName (void) const [inline]

Get the name of this world.

Returns:

An arbitrary name, used only for debugging.

9.39.2.7 unsigned long long cWorld::getTotalLength (void)

Estimate the entire in-memory size of this world.

Returns:

The # of bytes necessary to keep this entire world in memory.

See also:

cInstance::getCodedLength()(p. 66)

9.39.2.8 void cWorld::initialize (void) [virtual]

Reset everything.

This member is called by the host when the simulation is about to start.

Reimplemented in **Condel::cOptWorld** (p. 101).

9.39.2.9 const cWorld & cWorld::operator= (const cWorld & *ow*)

A DEEP assignment.

This copies everything over to this world except for the name and the host reference. Because of this limitation, do not use this for full initialization, only to copy state.

Parameters:

ow Another world on the same host.

9.39.2.10 void Condel::cWorld::setSimTime (double *newtime*) [inline]

Set the world's simulation time.

After all instances have been updated to a new time, set the world's time to the same value via this call

Parameters:

newtime The new simulation time, in seconds.

9.39.2.11 double Condel::cWorld::simTime (void) const [inline]

Get the world's simulation time.

Note the only time there is a difference between this and any instance in this world is during a **simulate()**(p. 156) run.

Returns:

The simulation time that all instances in the world are at, or were at if **simulate()**(p. 156) is currently running.

9.39.2.12 void cWorld::simulate (double *newtime*, cLoadMeter & *load*) [virtual]

Perform deterministic actions.

This function performs all deterministic actions in this world. This usually calls **cScenario::simulate()**(p. 127).

Parameters:

newtime Time to advance the system to. This should be in the future of **simTime()**(p. 156). Given in seconds.

load A tool to measure CPU and memory usage during the simulation.

Reimplemented in **Condel::cOptWorld** (p. 101).

The documentation for this class was generated from the following files:

- world.hh
- world.cc

Chapter 10

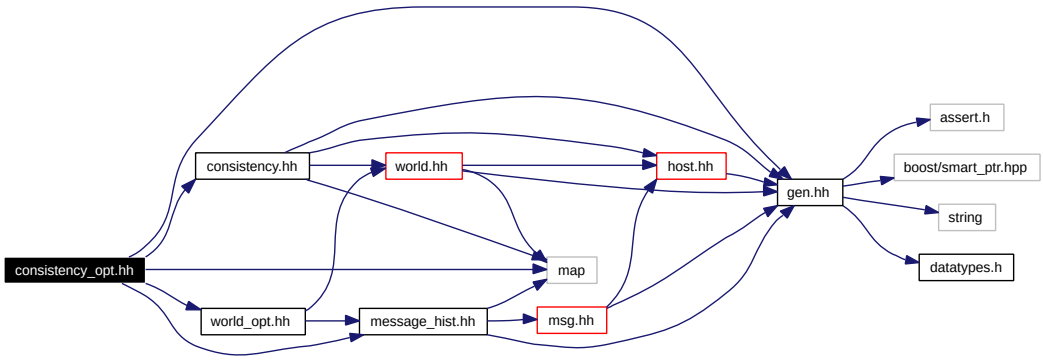
Adam File Documentation

10.1 consistency_opt.hh File Reference

This file contains the description of cOptConsistency.

```
#include <map>
#include "gen.hh"
#include "consistency.hh"
#include "message_hist.hh"
#include "world_opt.hh"
```

Include dependency graph for consistency_opt.hh:



Namespaces

- namespace **Condel**

Defines

- #define **TRAILING_STATES** 8

  Sets the # of trailing states to use.

10.1.1 Detailed Description

This file contains the description of cOptConsistency.

10.1.2 Define Documentation

10.1.2.1 #define TRAILING_STATES 8

Sets the # of trailing states to use.

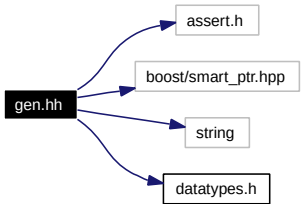
This could easily be increased or decreased. If everything works correctly, this should *not* change anything, except for the complexity. If the # of TSSes is too low, backtracks will waste too much calculation time. If the # of TSSes is too high, idle calculation of all those engines will waste too much calculation time.

10.2 gen.hh File Reference

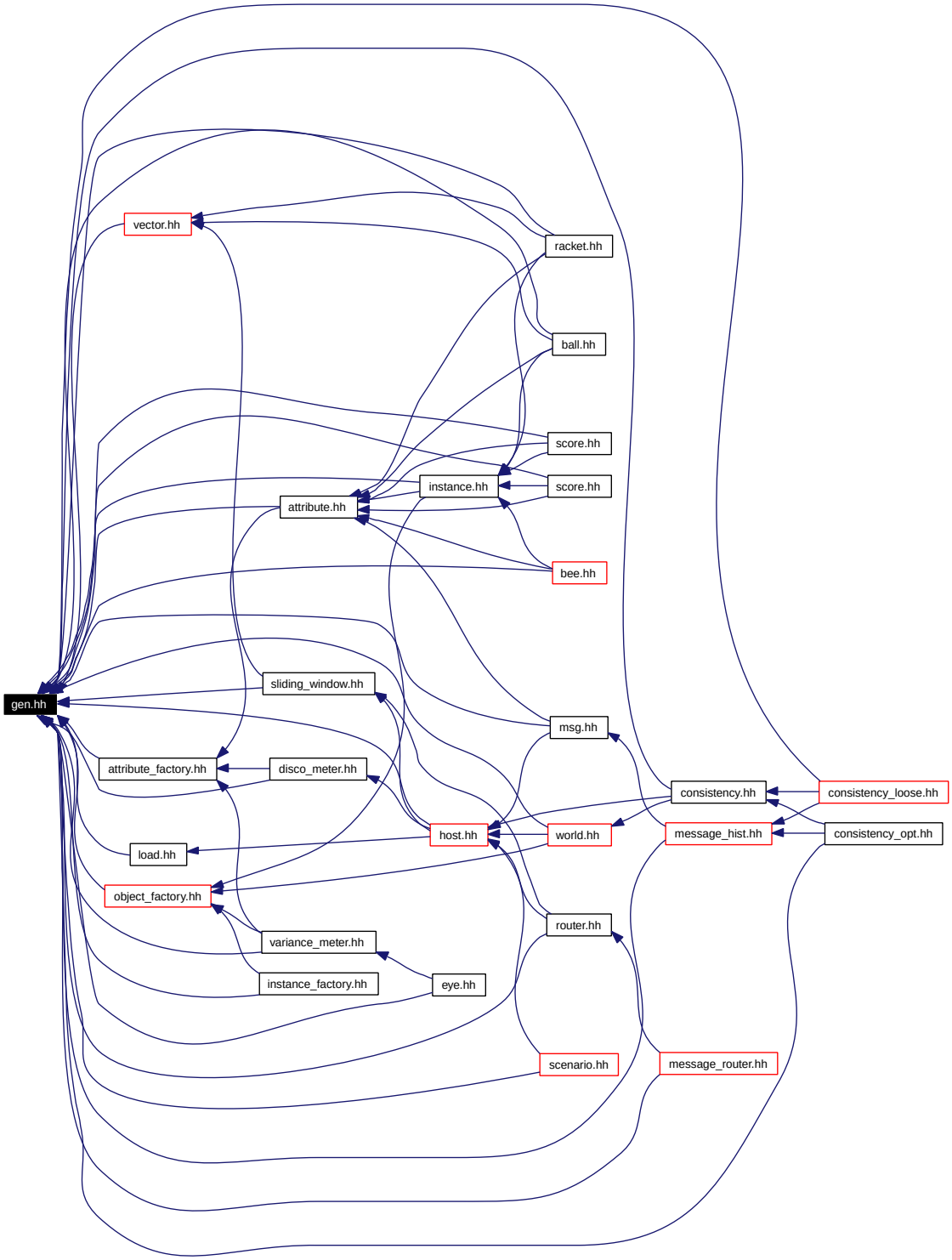
Common includes and definitions for Adam.

```
#include <assert.h>
#include <boost/smart_ptr.hpp>
#include <string>
#include <datatypes.h>
```

Include dependency graph for gen.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **Condel**

- namespace **boost**
- namespace **std**

Defines

- #define **DEBUG** 1
  Set the debug level.
- #define **NDEBUG**
  Set No debug mode.
- #define **VERBOSITY** 0
  Verbosity depends on level:.

10.2.1 Detailed Description

Common includes and definitions for Adam.

10.2.2 Define Documentation

10.2.2.1 #define DEBUG 1

Set the debug level.

Debugging depends on level:

- 0: none at all, this also means sanity checks are skipped
- 1: some (unexpected things occurring once in a while)
- 2: lots (i.e., checks and printouts per tick)
- 3: extreme (probably unnecessary checks)

Set DEBUG level either in **gen.hh**(p. 159) or in the Makefile via -DDEBUG=2.

Defaults to 1.

10.2.2.2 #define NDEBUG

Set No debug mode.

If defined, overrides the DEBUG setting (i.e. DEBUG=0).

10.2.2.3 #define VERBOSITY 0

Verbosity depends on level:.

- 0: Nothing is printed during a normal run
- 1: some progress info

- 2: lots (i.e., printouts per tick)
- 3: extreme (additionally shows massive amounts of internal information)

Defaults to 0.

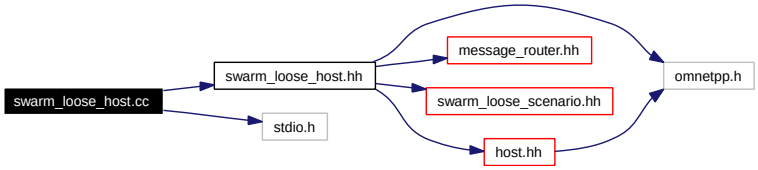
10.3 swarm_loose_host.cc File Reference

The implementation of the queen ownership passing scheme for loose consistency and Swarm.

`#include "swarm_loose_host.hh"`

`#include <stdio.h>`

Include dependency graph for swarm_loose_host.cc:



Defines

- #define **SWITCHOVER_TIME** 2.0
  Switch time.

10.3.1 Detailed Description

The implementation of the queen ownership passing scheme for loose consistency and Swarm.

10.3.2 Define Documentation

10.3.2.1 #define SWITCHOVER_TIME 2.0

Switch time.

Every SWITCHOVER_TIME seconds, the ownership of the queen transfers to the next host.

Chapter 11

Adam Page Documentation

11.1 Todo List

Member Condel::cBee::getDistance(p. 47)**(const cInstance &other) const** Need to fix ideal case here.

Member Condel::cInstance::setUpdateTime(p. 67)**(double newtime)** Currently, this mechanism only supports global updates (i.e. updates are sent to all other hosts).

Member Condel::cInstanceFactory::createInstance(p. 69)**(const OBJECT_ID &oid, const string &inst**
The object ID is kind of redundant and could be supplied by cObjectFactory::nameTo-ID(const string &).

Class Condel::cLoadMeter(p. 71) This should inherit from OMNeT++'s cStatistic some day.

Class Condel::cSlidingWindow(p. 133) This should inherit from OMNeT++'s cStatistic some day

Member Condel::cWorld::addInstance(p. 154)**(const cInstance *pInstance)** should use a reference, not a pointer.

Index