Part 2: CNCL

CNCL: Contents

- CNCL – C++ library for supporting event driven simulations
  - Overview
  - Main classes needed in simulations
- Learning CNCL by examples
- CNCL project work instructions

CNCL introduction

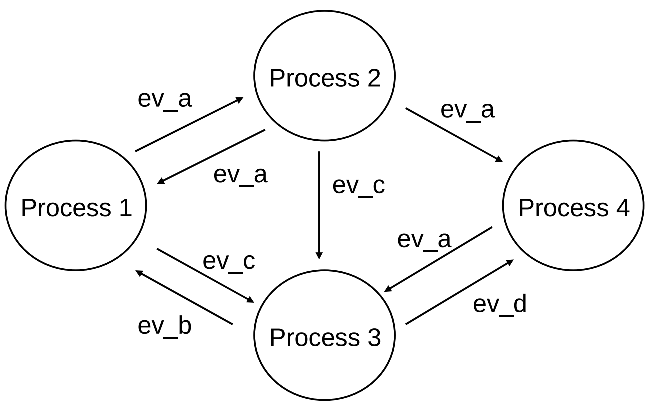
- Implemented by Communication Networks, Aachen University of Technology
  - freeware
  - package can be downloaded from

    http://www.comnets.rwth-aachen.de/

  - compiles on most Unix-type platforms
  - easy installation (package comes with an "install"-script)
- C++ class library
  - collection of classes for supporting event driven simulation
  - "light weight" simulation software
  - provides functionality for example for event handling/scheduling, random number generation, statistics collection, basic statistical analysis of results
- Usage
  - user writes his own code (in C/C++)
  - compiles (make) the code and links together own code and the class library

Modeling with CNCL (1)

- Basic philosophy
  - simulation model consists of processes and events
  - processes send events to each other



Modeling with CNCL (2)

- Process
  - implements a state machine (e.g., a server in a queue can be in state idle/serving)
  - receives events and depending on event's type executes an appropriate method (function)
  - while executing the method associated with an event, it typically changes the process's state and schedules new events
  - in practise, a process is a C++ class that has been derived (through inheritance) from the CNEventHandler – class
- Event
  - causes the state of a process to change
  - events drive the simulation's execution and (usually) imply the advancement of simulation time
  - for example, packets arriving at the queue, packet finishes service at the server, …
- Event vs. direct method call:
  - for reasons of modularity of the program design, a process can be implemented by using several classes
  - not all classes need to be able to handle events
  - often the internal overhead of event scheduling can be avoided by using just a direct method call

CNCL and C++ (1)

- CNCL programs implemented in C++

- CNCL library provides basic functionality for
  - pseudo random number generation, generation from given distributions, statistical analysis, basic queue elements, event management etc.
  - user can directly use these classes

- CNCL event handling
  - user must implement event handling logic himself
  - each process in the model is an event handler
  - event handler in practise
    - an event handler is a C++ class, that has been derived from an abstract event handler base class (CNEventHandler)
    - event handling logic implemented in predefined functions of the derived class (the function "void event_handler()" is declared virtual in base class)

CNCL and C++ (2)

- Memory management must be handled by the user
- Memory space for an object can be created either statically from the stack or dynamically from the heap
- Stack allocation (static)

```
{
  MyObject x;
  .
  .
  .
}
```

  - an object that is declared directly is created on the stack and objects are destroyed in reverse order of creation (done automatically on exit of the block they were created in)
  - user has no control over the creation timing and destroying
  - e.g. random number generators and event schedulers can be such static objects created in the main program

CNCL and C++ (3)

- Heap allocation (dynamic)

```
{
  MyObject* xPtr;
  xPtr = new MyObject;
  .
  .
  .
  delete xPtr;
}
```

  - objects created with the new operator are placed on the heap and will persist until explicitly destroyed, or the program terminates
  - every object created with new must be explicitly detroyed with a corresponding delete
  - e.g. in simulation of a network (or just single queue) packets going through the system should be dynamic objects

CNCL properties

- Pros
  - as the user implements all functionality, the user also has full control of what functionality is needed and what is not
  - fast execution times (no unnecessary overhead)
  - (relatively) easy to learn (simple)
  - good supprt for random number generation and event driven simulation
- Cons
  - no ready made functional blocks for network simulations (e.g., different protocols, etc.)
  - implementation time may be substantial

S-38.148 Simulation of data networks / CNCL

CNCL: Contents

- CNCL – C++ library for supporting event driven simulations
  - Overview
  - Main classes needed in simulations
- Learning CNCL by examples
- CNCL project work instructions

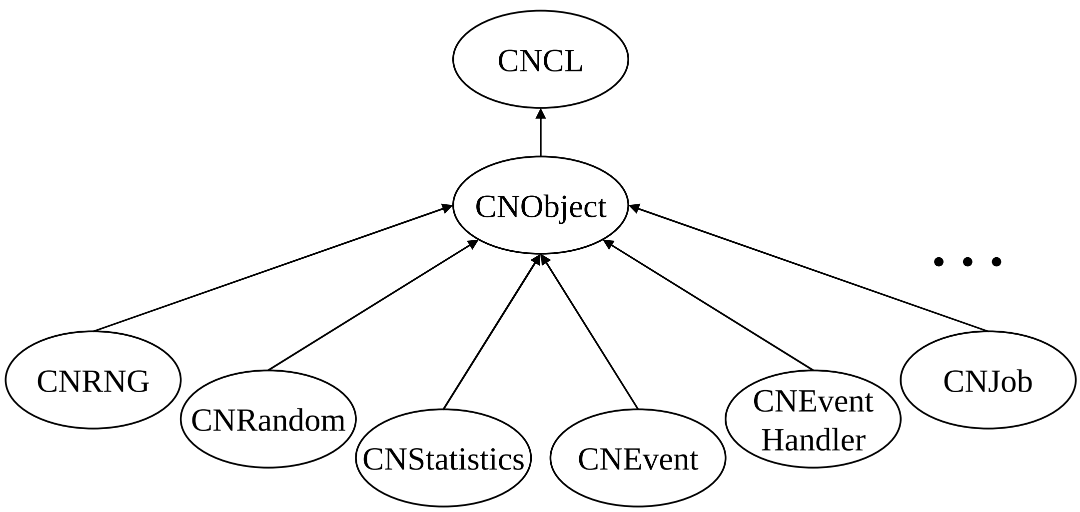
Essential functionality needed in every simulation program

- To some extent, all simulations need the same basic building blocks
  - pseudo random number generator
  - random number generators from given distributions
  - event scheduler (event exploders,…)
  - different queues (FIFO, priority queues, …)
  - event data structure
  - jobs (packets)

S-38.148 Simulation of data networks / CNCL

Class hierarchy

- Class CNCL provides functions for error handling
- CNObject is the root of the CNCL class hierarchy



Random number generators

- CNRNG is an abstract base class for all CNCL random number generators
  - the pseudo-random number generators to be used have been derived from CNRNG
- CNRNG supports following pseudo random number types
  - unsigned integer 0..2^{31-1}
  - float 0..1
  - double 0..1
- Actual RNGs differ in
  - quality of pseudo random number sequences (sequence lengths, overlapping sequences, correlation),
  - efficiency,
  - and memory consumption

Derived classes (1)

- CNLCG Linear Congruence RNG
  - simplest pseudo random number generator
  - may be used when performance is more important than perfect randomness
- CNMLCG Multiple Linear Congruence RNG
  - combines the results of two different CNLCGs
  - implementation taken from the GNU-library libg++
  - fairly long period, and has been shown to give good intersample-independence
- CNACG Additive RNG
  - high quality random number generator
  - requires a fair amount of memory for each instance of the generator
  - implementation taken from the GNU-library libg++

Derived classes (2)

- CNFiboG Fibonacci RNG
  - high quality generator with a huge period (in the CNCL implementation period = 2^{127})
  - relatively high memory usage
- CNFileG Data File RNG
  - data file random number generator class
  - reads random numbers from a disk file
  - “good” file must have a sufficient size
  - considerable memory usage and low speed can be expected when using this class
- CNTausG Tausworth RNG
  - main advantage of this generator is that it can easily be implemented as a fast hardware generator
  - statistical tests have shown some flaws in this generator so that its use is not recommended

Summary of random number generators

	Randomness	Period length	Performance	Memory req.
LCG	+	+	+++++	+++++
MLCG	++	++	+	++++
ACG	+++	1)	++	+
FiboG	+++++	+++++	+++	++

1) depends on table size

Example

```
CNFiboG  rng1;
CNRNG   *rng2 = new CNFiboG();
unsigned  x1;
float     x2;
double    x3;

x1 = rng1.as_long();    // draw a random integer 0..2^31-1
x2 = rng2->as_float();  // draw a random float 0..1
x3 = rng2->as_double(); // draw a random double 0..1

delete rng2;
```


Random numbers with different distributions

- CNRandom is an abstract base class for different random number distributions
  - common interface to access all derived RNG classes
- CNRandom provides
  - a random number from the distribution
- CNRandom uses CNRNG
  - CNRandom initialized with a pointer to the used pseudo random number generator

Derived classes

- CNBeta
- CNBinomial
- CNDeterm
- CNDiracTab
- CNDiscUniform
- CNErlang
- CNGeometric
- CNHyperExp
- CNHyperGeom
- CNInterTab
- CNLogNormal
- CNMDeterm
- CNNegExp
- CNNormal
- CNPoisson
- CNRandomMix
- CNRayleigh
- CNRice
- CNTab
- CNUniform
- CNWeibull

Example

```
CNRNG   *rng = new CNFiboG();
double   mean = 2.0;
CNNegExp  rnd(mean, rng);
double    x;

x = rnd(); // draw a neg. exp. distributed random number

delete rng;
```

Statistical evaluation

- CNStatistics is an abstract base class for all statistics classes
  - defines a common interface
- CNStatistics allows
  - to put a value for statistical evaluation
  - to reset the evaluation
- CNStatistics provides e.g.
  - mean and variance of the input sequence
  - number of evaluated values
  - minimum and maximum of all evaluated values
- Derived classes
  - CNMoments
  - CNMomentsTime
  - CNConfidence
  - CNHistogram
  - (CNLREF, CNLREG, CNDLRE, CNBatchMeans)

Evaluation of moments for simulation data

- CNMoments provides e.g.
  - mean
  - variance and relative variance
  - 2nd and 3rd zero moment
  - 3rd central moment
  - relative deviation
  - skewness

- CNMomentsTime
  - moments of a time-weighted input sequence
  - useful for computing e.g. statistics of the queue length process

- CNConfidence
  - usual non-parametric statistics + functions for computing confidence intervals

- CNHistogram
  - support for computing histograms of sample statistics

Example(s)

```
CNMoments m;
double    x;
double    y;

m.put(2.0);
m.put(3.4);
m.put(5.1);

x = m.mean();
y = m.variance();

cout << m;
```

```
CNMomentsTime m;
double        x;
double        y;

m.put(2.0, 1.0);
m.put(3.4, 2.0);
m.put(5.1, 3.0);

x = m.mean();
y = m.variance();

cout << m;
```


Container classes

- Container classes work with pointers to CNObject
- Generic data structures
  - CNAVLTree AVL balanced tree structure
  - CNSLList Single Linked List of Objects
  - CNDLList Double Linked List of Objects
  - also iterators for lists
- Queue objects
  - CNQueueFIFO FIFO Queue
  - CNQueueLIFO LIFO Queue
  - CNQueueRandom Random queue
  - CNQueueSPT Shotest Processing Time queue (only for CNJobs)
  - CNPrioQueueFIFO FIFO priority queue
- Other classes
  - CNSink Queue that deletes all inserted jobs

FIFO queue

```
CNQueueFIFO queue;
CNJob* in_job = new CNJob;
CNJob* out_job;

queue.put(in_job);
.
.
out_job = queue.peek(); //job not removed from queue
.
.
out_job = queue.get(); //job is removed from queue
delete out_job;
```


Job

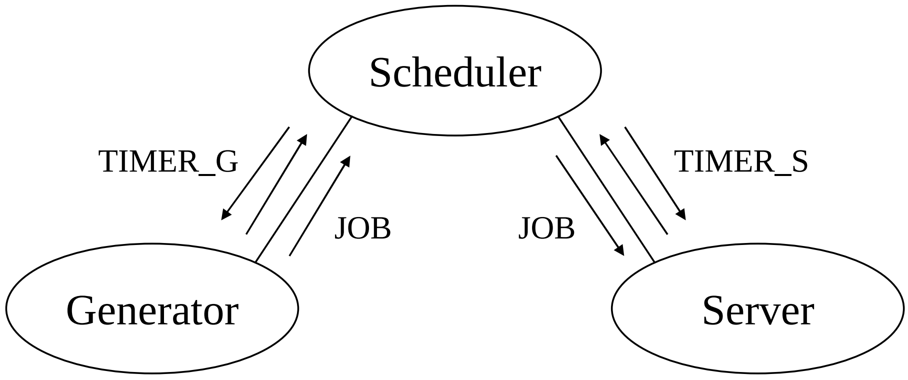
- CNJob (derived from CNObject) is a standard object for CNCL queues
- CNJob provides e.g. the following public member variables:
  - `CNSimTime in          // enter system`
  - `CNSimTime start       // service begins`
  - `CNSimTime out         // leave system`
  - `int priority          // priority of job`
- Useful for example when recording sojourn times

Example

```
CNMoments    m_queue;
CNMoments    m_total;
CNJob        *job = new CNJob;
.
.
job->in = now();           // job arrives at the queue
.
.
job->start = now();        // service begins
.
.
job->out = now();          // service ends
m_queue.put(job->start - job->in);
m_total.put(job->out - job->in);
delete job;                // job is no longer needed
```


Event driven simulation

- Event Handlers: Generator, Server
- Events: TIMER_G, TIMER_S, JOB
- Event Scheduler



Event handlers

- Derived from class CNEventHandler
  - state machine that receives and processes events
  - user implements event handling method: void event_handler(const CNEvent *ev)
  - executed routine depends on the state of the event handler and the type of the incoming event
  - may generate new events and change state
  - resembles a process

Example

```
Class Generator : public CNEventHandler {
private:
.
.
public:
.
.
};

void Generator::event_handler(const CNEvent *ev)
{
switch (ev->type())
    {
    .
    .
    }
};
```

Events

- Class CNEvent
  - data structure representing events in the simulation
- Includes
  - type
  - priority
  - sending time and scheduled time
  - sending and receiving event handlers
  - unique identifier
  - pointer to an arbitrary CNCL object
- Note! CNEvents are created with new operator, but they do not need to be explicitly deleted by user (scheduler takes care of that).

Example

```
CNEvent  *ev;
CNRandom *rnd;
Server   *server; // another event handler
...
ev = new CNEvent(EV_TIMER_G);
send_delay(ev, rnd( )); // send to myself as default
...
ev = new CNEvent(EV_JOB, server, new CNJob);
send_now(ev);           // send without delay
...
send_delay(new CNEvent(EV_TIMER_G), rnd( ));
```

Event scheduler

- Operation
  - controls the simulation run
  - receives events
  - orders events in the increasing order of time stamp and decreasing order of priority
  - passes events to the addressed event handlers
  - deletes all created CNEvent objects

- Two variants
  - CNEventScheduler
    - guarantees that events are processed in FIFO order even when time and priority compare equal
    - slow if nof managed simultaneous events grows large
  - CNEventHeapSched
    - more efficient than CNEventScheduler, but cannot guarantee FIFO processing if time and priority of events are equal

Examples

```
main( )
{
  ...
  CNEventScheduler scheduler;
  scheduler.start(new CNEvent(EV_TIMER_G,
                             &generator, 0.));
}

main( )
{
  ...
  CNEventScheduler scheduler;
  scheduler.send_event(new CNEvent(EV_TIMER_G,
                                  &generator, 0.));
  scheduler.start( );
}
```

Simulation time

- Note! Simulation time is accessible only in EventHandler – classes

- Within such classes, simulation time is accessed with ”now()” – method call
  - each event handler has inherited this method from the base class CNEventHandler