ODEMx3 Crashkurs

Magnus Müller (mamuelle@. . .)

Institut für Informatik
Humboldt Universität zu Berlin

2. Juni 2010

Einführung

Neues in ODEMx 3

- *setup.h*: Makros zur Konfiguration
- Referenzsemantik
- Smartpointer
- Logging-Konzept (Logchannels)
- Strukturierung der Bibliothek (Namespaces)
- Klassen umbenannt (*NegativeExponential*, RandomInt, . . .)
- Koroutinenkonzept (Fiber-API)
- . . .

Kompilierung von ODEMx 3

- Herunterladen von der Webseite
- 1. *cd odemx/GCC* – die Quellen müssen in einem Verzeichnis *odemx/* sein!
  2. *make* – gibt die verfügbaren Targets aus
  3. *make debug* – baut den Debugrelease
- Makefiles siehe Beispiele auf der Webseite

Ein erstes Projekt

```
#include <iostream>

#include <odemx/odemx.h>

int main () {
  odemx::base::Simulation& sim = odemx::getDefaultSimulation ();

  sim.setDescription ("Hello, World!");

  std::cout << sim.getDescription () << std::endl;

  return 0;
}
```

Uhr - einfacher Prozess I

- Idee: Ein Prozess, der alle 1 $[ZE]$ einen Punkt schreibt.
- Neues gegenüber *ODEMX 2*: Referenzsemantik

Uhr - einfacher Prozess I

Listing 1: Clock.h

```
#ifndef _CLOCK_H
#define _CLOCK_H

#include <odemx/odemx.h>
using namespace odemx;

class Clock: public base::Process {
  public:
    Clock (base::Simulation& sim);
    virtual int main ();
};

#endif
```

Listing 2: Clock.cpp

```
#include <odemx/odemx.h>
#include <iostream>
#include "Clock.h"
using namespace odemx;

Clock::Clock (base::Simulation& sim):
  base::Process (sim, "clock") {}

int Clock::main () {
  for (;;) {
    holdFor (1.0);
    std::cout << ".";
  }
  return 0;
}
```

Listing 3: main.cpp

```
#include <iostream>
using namespace std;

#include <odemx/odemx.h>
```

Uhr - einfacher Prozess II

```
using namespace odemx;

#include "Clock.h"

int main () {
  base::Simulation& simulation = getDefaultSimulation ();
  Clock myClock (simulation);
  myClock.activate ();

  cout << "Basic Simulation Example" << endl;
  cout << "========================" << endl;

  for (int i = 1; i < 5; i++) {
    simulation.step ();
    cout << endl << i << ".step time=" << simulation.getTime () << endl;
  }

  cout << endl;
  cout << "Continue until SimTime 13.0 is reached" << endl;
  simulation.runUntil (13.0);

  cout << endl << "time=" << simulation.getTime () << endl;

  cout << "========================" << endl;

  return 0;
}
```

Idee: Prozesssynchronisation mit Ports I

- Idee: zwei Prozesse kommunizieren über einen Port (unidirektional).
- Der lesende Prozess gibt die Nachricht aus.
- Neues gegenüber *ODEMX 2*: Smartpointer

Prozesssynchronisation mit Ports I

Listing 4: simpleport.h

```
#ifndef _SIMPLEPORT_H
#include <odemx/odemx.h>
using namespace odemx;

#include <string>

class Recipient: public base::Process {
  sync::PortHeadT<string>::Ptr readEnd;
  public:
    Recipient (base::Simulation& sim,
               sync::PortHeadT<string>::Ptr readEnd);
    virtual int main ();
};

class Sender: public base::Process {
  sync::PortTailT<string>::Ptr writeEnd;
  public:
    Sender (base::Simulation& sim,
            sync::PortTailT<string>::Ptr writeEnd);
    virtual int main ();
};
#endif      // ------ not _SIMPLEPORT_H ------
```

Prozesssynchronisation mit Ports II

Listing 5: simpleport.cpp

```
#include "simpleport.h"

#include <odemx/odemx.h>
using namespace std;

Recipient::Recipient (base::Simulation& sim, sync::PortHeadT<string>::Ptr readEnd):
  base::Process (sim, "Recipient"),
  readEnd (readEnd)
{}

int Recipient::main () {
  while (true) {
    cout << this << " received message: " << *(readEnd->get ()) << endl;
  }
  return 0;
}

Sender::Sender (base::Simulation& sim, sync::PortTailT<string>::Ptr writeEnd):
  base::Process (sim, "Sender"),
  writeEnd (writeEnd)
{}

int Sender::main () {
  while (true) {
    writeEnd->put ("Eine Nachricht");
    holdFor (10);
  }
  return 0;
```

Prozesssynchronisation mit Ports III

```
}

int main () {
  base::Simulation& sim = getDefaultSimulation ();

  // Erstelle einen Port
  // Eingabe
  sync::PortHeadT<string>::Ptr readEnd = sync::PortHeadT<string>::create (sim, "PortHead");
  // Ausgabe
  sync::PortHeadT<string>::TailPtr writeEnd = readEnd->getTail ();

  // Objekte erstellen
  Recipient recipient (sim, readEnd);
  Sender sender (sim, writeEnd);

  recipient.activate ();
  sender.activate ();

  sim.runUntil (200);

}
```

Idee: Zufallszahlen & Report I

- Idee: Ein Zufallszahlengenerator wird zur Erstellung eines Histogramms genutzt. Anhand dieses Histogramms werden zwei Möglichkeiten zur Erstellung eines Reports eingeführt.
- Neues gegenüber *ODEMX 2*: XML, Standardausgabe

Zufallszahlen & Report I

Listing 6: normal.cpp

```
#include <odemx/odemx.h>
using namespace odemx;

#include <iostream>
using std::cout;
using std::endl;

int main()
{
  base::Simulation &sim = getDefaultSimulation();

  random::Normal n(sim, "X", 1.0, 0.5);
  stats::Histogram hist(sim, "Probe", -1, 5.0, 20);

  for (int i = 0; i < 10000; i++) {
    hist.update(1 + n.sample());
  }

  // output report to stdout and create xml report
  data::output::OStreamReport oStreamReport(std::cout);
  data::output::XmlReport xmlReport("normal.xml");

  // add report producer
  oStreamReport.addReportProducer(hist);
  xmlReport.addReportProducer(hist);
```

Zufallszahlen & Report II

```
    // report
    oStreamReport.generateReport ();
    xmlReport.generateReport ();

    return 0;
}
```

Report

ODEMx XML Report

Histogram Statistics Summary

Name	Reset at	Uses	Low	High	Cells	Min	Max	Mean	Std Deviati
Probe	0	10000	-1	5	22	0.121683	3.86726	1.99607	0.499261

Histogram: Probe [10000 samples]

Low	High	Count	Percentage	Bar Diagram
-1.3	-1	0	0	
-1	-0.7	0	0	
-0.7	-0.4	0	0	
-0.4	-0.1	0	0	
-0.1	0.2	1	0.01	
0.2	0.5	18	0.18	
0.5	0.8	71	0.71	
0.8	1.1	280	2.8	
1.1	1.4	811	8.11	
1.4	1.7	1575	15.75	
1.7	2	2201	22.01	
2	2.3	2347	23.47	
2.3	2.6	1555	15.55	
2.6	2.9	804	8.04	
2.9	3.2	256	2.56	
3.2	3.5	70	0.7	
3.5	3.8	9	0.09	

Logging channels I

- Idee: Einfaches Logging mit der neuen Logging-Bibliothek

Logging channels I

Listing 7: logging.cpp

```
#include <odemx/odemx.h>
using namespace odemx;

class X: public base::Process {
  random::ContinuousDist& random;
  sync::Res& res;

  public:
    X(base::Simulation& sim, random::ContinuousDist& random, sync::Res& res):
      base::Process(sim, "X"),
      random(random),
      res(res) {}

    virtual int main() {
      while (getSimulation().getTime() < 10) {
        res.acquire(1);
        // mache irgendetwas

        holdFor(2);
        // statusmeldung in den info channel
        info << log("Ich habe etwas getan.");
        info << update("Zufallszahl", random.sample());
        warning << log("Obacht!");
        res.release(1);
      }
```

Logging channels II

```
      return 0;
    }
};

int main () {
  base::Simulation& sim = getDefaultSimulation ();

  random::NegativeExponential random (sim, "RNG", 0.1);

  sync::Res res (sim, "Resource", 1, 1);

  // registriere Statistikbuffer
  using data::buffer::StatisticsBuffer;
  std::tr1::shared_ptr<StatisticsBuffer> statisticsBuffer =
    StatisticsBuffer::create (sim, "Statisticsbuffer", true);

  // TODO was hat das dann mit den producern zu tun?
  sim.addConsumer (data::channel_id::statistics, statisticsBuffer);
  sim.addConsumer (data::channel_id::info, data::output::OStreamWriter::create( std::cout ));

  X x (sim, random, res);
  x.activate ();

  sim.run ();

  // report
  data::output::XmlReport report ("report.xml");
  report.addReportProducer (*statisticsBuffer);

  report.generateReport ();
```

Logging channels III

```
}
```