1 OMNeT++

1.1 Q&A

1.1.1 Wie wird eine Topologie erstellt und dessen Funktionalität realisiert?

Eine Topologie, sprich eine Simulation wird in OMNeT++ mithilfe mehrerer Datentypen definiert. Es gibt:

- .NED - **NE**twork **D**escription File In diesen beschreibt man, wie der Name bereits verät, das Netzwerk mit all dessen Knoten, Empfängern wie Sendern. Neben den DataNodes beschreibt man auch den groben Funktionsumfang, etwa wann Daten gesendet werden sowie Delay. Mithilfe 2er Modi: Source und Design lässt sich das Netzwerk graphisch als auch programmiertechnisch beliebig anpassen.

- .CC - C++ Datensätze:
  In diesen beschreibt man die Subfunktionen e.g. das Verhalten aller Instanzen innerhalb des Netzwerkes. Für gewöhnlich bedient man sich vordefinierten Spezifikationen wie cMessage oder cSimpleModule, von welchen die eigenen Klassen erben.

- .INI - Initialisierungsdatei Diese beschreibt das zu simulierende Netzwerk, definiert im .NED-File. *Es gilt die Klasse anzugeben, nicht den Dateinamen.*

1.1.2 Wie kompiliert und startet eine Simulation?

Prinzipiell mithilfe des RUN-Buttons der IDE.

Für gewöhnlich gilt es mithilfe von cpp-makemake ein File zu erstellen, welches dann ausgeführt wird. Dieses orientiert sich an der jeweiligen omnetpp.ini-Datei, worin die auszuführenden Daten spezifiziert sind. Ähnlich einer Kettenreaktion wird dann der Rest ausgelesen, kompiliert und ausgeführt.

1.1.3 Wie fügt man folgende hinzu:

(a) Graphische Elemente:

Werden üblicherweise mithilfe der .ned Datei erstellt. Hierbei wird der TicToc mittels des network Blocks als Netzwerk gekennzeichnet.

(b) Ausgabe zur Fehlersuche:

Ausgaben werden können über Logger oder über folgenden Befehl ausgegeben werden:

```
EV << "Sending initial message";
```

(c) Zustandsvariablen:

Werden innerhalb einer Klasse definiert.

```
private:
int counter
```

(d) Zufällige Parameter:

Über Parameter eines Moduls können zufällige Werte generiert werden. Um diese letztendlich verwenden zu können, werden sie innerhalb der handleMessage() Methode eingelesen. Beispiel:

```
simtime_t delay = par("delayTime");
EV << "Message arrived, starting to wait";
tictocMsg = msg;
scheduleAt(simTime()+delay, event);
```

Die Parameter werden hierbei über die .ini Datei übergeben.

```
Tictoc.tic.delayTime = exponential(3s)
Tictoc.toc.delayTime = truncnormal(3s, 1s)
```

1.1.4 Wie setzt man Vererbung, Verzögerung, Zeitüberschreitung um oder hebt diese auf?

(a) Vererbung:
Umsetzung:
Aufhebung:

(b) Verzögerung:
Umsetzung:
Verzögerungen können mittels dem Aufruf der Methode scheduleAt() realisiert werden.
Beispiel: scheduleAt(simTime()+1.0, event);
Aufhebung:

(c) Zeitüberschreitung:
Umsetzung:
Aufhebung:

(d)

1.1.5 Wie funktionieren Netzwerktopologien mit mehr als 2 Knoten?

(a) zuerst muss in der .ned-Datei angegeben werden, dass mehrere Input/Output Gates enthalten sind, wobei [ ] die eigentliche Gates in Gate Vektoren ändert

```
gates:
input in[];
output out[];
```

(b) im nächsten Schritt werden die Module erstellt und miteinander verbunden

(c) wurden die Module erstellt und verbunden so kann das Programm gestartet werden

(d) Funktionsweise einer solche Topologie sieht wie folgt aus:

Ein Knoten erstellt hierbei die Nachricht, welche letztendlich durch einen anderen Knoten im Netzwerk in verschiedenen Richtungen solange weiterverbreitet wird bis ein vordefinierter Zielknoten erreicht wird.

1.1.6 Wie wird ein eigenes Nachrichtenformat definiert und wie werden sie verwendet?

Nachrichten können auf zwei Arten erstellt werden:

(a) indem eine eigene Unterklasse von cMessage erstellt, in der ein Ziel als Datenelement hinzugefügt wird.

(b) oder man erstellt eine eigene Message Klasse die wie folgt aussehen könnte:

```
message TicTocMsg13
{
int source;
int destination;
int hopCount = 0;
}
```

Verwendet wird die Nachricht dann wie folgt:

(a) zuerst muss die erzeugte .msg-Datei mittels des message compilers kompiliert werden, wodurch tictoc.h und tictoc.cc erzeugt werden (beinahlten die erstellte TicTocMsg Klasse, welche Unterklasse von cMessage ist und getter/setter enthalten)

(b) im nächsten Schritt muss die erzeugte Klasse importiert werden

```
#include "tictoc1
```

(c) um die Message verwenden zu können, wird sie innerhalb der C++ Datei wie folgt erstellt:

```
TicTicMsg13 *msg generateMessage(){
TicTocMsg13 *msg = new TicTocMsg13(msgname);
msg->setSource(src);
msg->setDestination(dest);
```

```
return msg;
}
```

(d) die handleMessage Methode muss nun wie folgt beginnen:

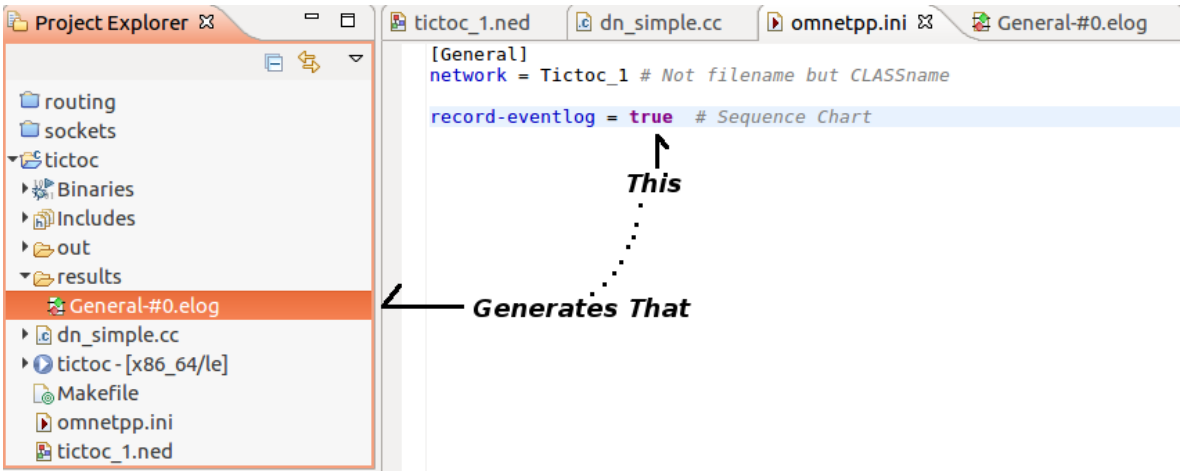
```
void Txc13::handleMessage(cMessage *msg){
TicTocMsg13 *ttmsg = check_and_cast<TicTocMsg13 *>(msg);
etc.
}
```

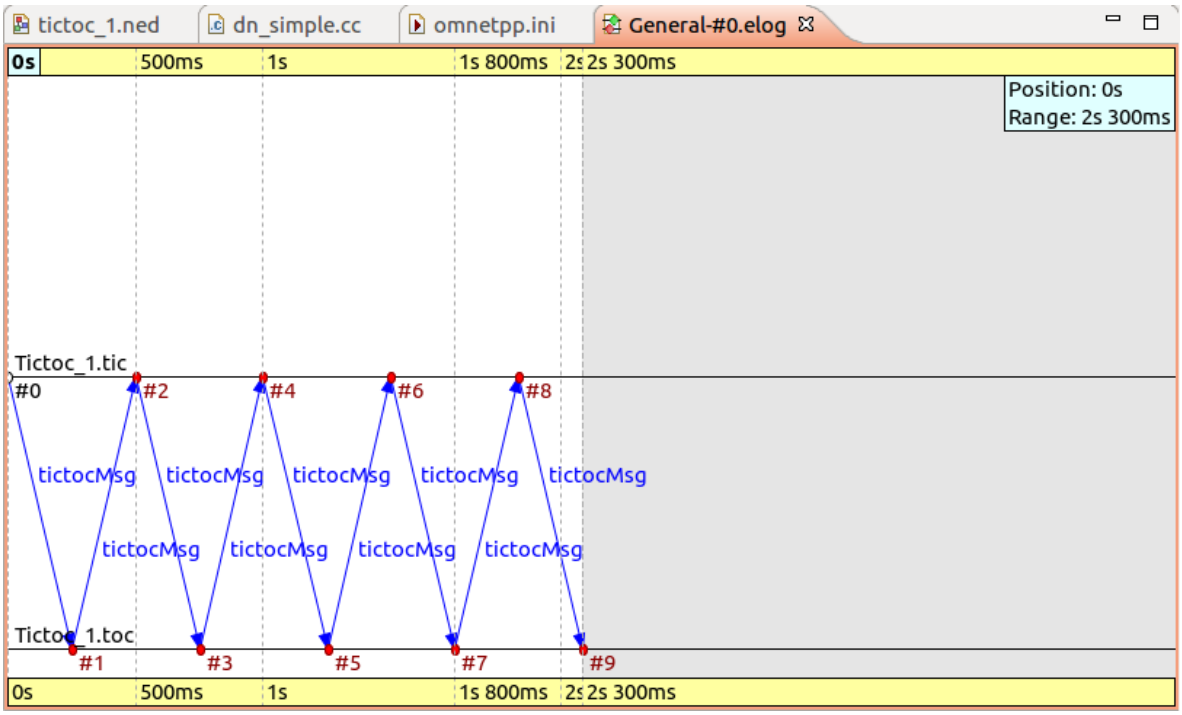
Das Argument, welches der handleMessage() übergeben wird, ist die Message, die wir als cMessage übergeben bekommen. Hierbei können wir nur dann auf dessen Felder zugreifen, wenn wir die Message zu TicTocMsg13* casten. Ein einfacher C-Style cast zu einem Crash führen würde, sollte die Message eine TicTocMsg13 sein. Daher ermöglicht C++ das verwenden von dynamischen casts, wie in unserem Fall:

```
check_and_cast<>()
```

Es versucht hier den Pointer mithilfe eines dynamischen casts umzuwandeln. Sollte dieser jedoch fehlschlagen, so wird eine ERROR MESSAGE aufgegeben.

1.1.7 Statistiken, Auswertung und Visualisierung - Wie setzt man sie um?




tictoc_1.ned
dn_simple.cc
omnetpp.ini
General-#0.elog
0s
500ms
1s
1s 800ms
2s 300ms
Position: 0s
Range: 2s 300ms
Tictoc_1.tic
#0
#2
#4
#6
#8
tictocMsg
tictocMsg
tictocMsg
tictocMsg
tictocMsg
tictocMsg
tictocMsg
tictocMsg
Tictoc_1.toc
#1
#3
#5
#7
#9
0s
500ms
1s
1s 800ms
2s 300ms