Instant CTL

Dipl. Inf. Cand. Boris Bügling

Institute for Scientific Computing, Technical University Braunschweig

3. Mai 2007

Outline

1. Motivation
2. The Component Template Library
3. Hands-On Tutorial (Java)
4. Conclusion

Outline

Why are we using software components?

- Code re-use
- Exchangeable software units
- Language interoperability
- Location transparency
- Support for distributed parallel run time systems

Why are we using the CTL?

- High performance (exception: CTL4j)
- Language support: C, C++, Fortran, Java, Python, Matlab
- Uniform behaviour across different transport protocols and local linkage
- Easy to understand protocol
- Almost no learning curve for implementing components and applications

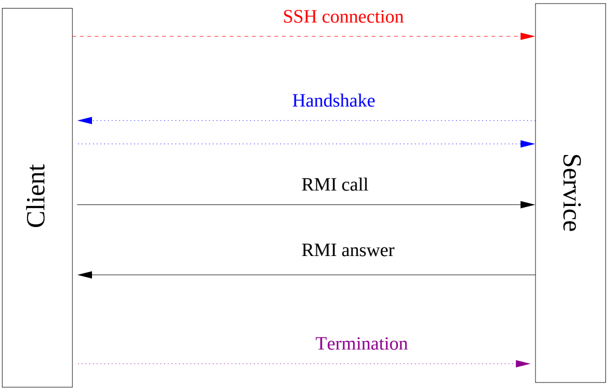
Outline

1. Motivation
2. The Component Template Library
3. Hands-On Tutorial (Java)
4. Conclusion

Overview

- Two implementations are available: CTL/C++ and CTL4j
- CTL: first partially realised as part of the parallel FE-code ParaFep at the Institute of Structural and Numerical Mechanics in Hanover, 1995
- Implemented as a C++ template library by Rainer Niekamp
- CTL4j: Java implementation of the protocol and concepts, developed since Fall 2005 at the Institute of Scientific Computing in Braunschweig
- Implemented as a Java library and toolset using Generics, Annotations and Reflection by Boris Buegling
- The support for Python and Matlab is based on the C APIs of their runtime environments

Basic structure of communication

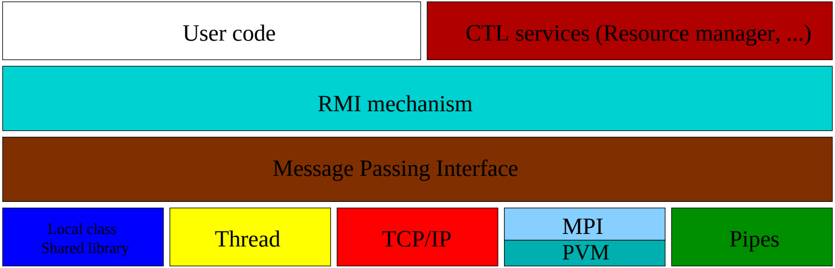


Breaking down complex types

- Fundamentals: integer types, floating point types, void
- Composites
  - Arrays (serialized as: size, e0, e1, ...)
  - String (serialized as: e0, e1, ..., 0)
  - Tuple (fixed size; serialized as: e0, e1, ...)
  - Reference (serialized as: typeID, true, data or typeID, false)
- All other types can be serialized as an aggregate of these types
- User-defined components do not have to deal with the binary data directly
- Protocol implementations just need to understand the binary stream → language independence

Separation of communication path and application

Layers of the CTL protocol



Separation of interface and implementation I

- Component interfaces are the **Interface Definition Language** (IDL)
- Example:

```
#define CTL_Class AddCI
#include CTL_ClassBegin
  #define CTL_Method1 int4, add (const int4,
    const int4), 2
#include CTL_ClassEnd
```

- → The implementation just needs to export this interface; multiple (different) solutions possible
- IDL is also important for language independence

Separation of interface and implementation II

- The code generator of the CTL4j works with Java classes
- CIs can be generated from those
- → Java example:

```
public class AddCI
{
  @export public int add (@const_ int arg0,
    @const_ int arg1)
  {
    [...]
  }
}
```

Location independence

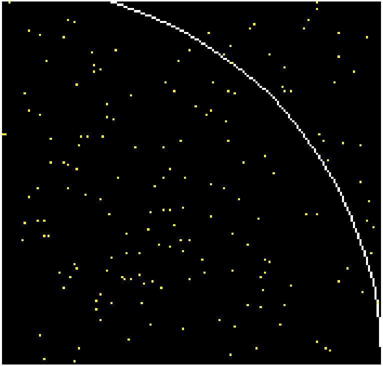
- *CTL.Types.Location* describes the address of a component
- Java example:

```
user:pass@moo.com:/tmp/Example.Server tcp
user:pass@moo.com:/tmp/Example.Server pipe
lib
./cpp/add/add.so lib
```

- By convention, there exists a file *locs.txt* in the CWD which contains locations.
- C++: Type is *ctl::location*; syntax for links is slightly different: '-l tcp' instead of 'tcp'
- There is a CI for a service which can be asked for known locations of components, called the resource manager. It keeps mappings from component names to locations.
- Component Query Language (CQL) for more specific queries

Outline

Problem definition I



Calculate π using **MonteCarlo**

Problem definition II

Algorithm

- Take the unit circle and look at the first quadrant only: $1 = x^2 + y^2$
- Choose some random $x, y \in [0, 1]$
- Those points can be either *inside* the circle ($1 \geq x^2 + y^2$) or *outside* ($1 < x^2 + y^2$)
- We know that the surface area of the circle is $A = \frac{\pi}{4} * r^2 = \frac{\pi}{4}$
- We also know that for an infinite number of points $A = \frac{points_{inside}}{points_{total}}$ holds
- Putting it all together, we can actually determine an approximation of π: $A = \frac{\pi}{4} \approx \frac{points_{inside}}{points_{total}} \rightarrow \pi \approx 4 * \frac{points_{inside}}{points_{total}}$

Designing an interface I

- ```
  class TestPoints
  {
    /* Test how many points in a given set
       are inside the unit circle */
    @export public static int testpoints
      (DoubleVector[] points);
  }
  ```
- Too much data is sent
- *@const_* was not used
- No encapsulation
- No interface for random number generation
- → Good interface design is **not** as easy as it seems
  ```

Designing an interface II

- ```
  class Pi
  {
    @export static double calculate (@const_ int i
  }

  class Random
  {
    @export double getDouble ();
  }
  ```
- However, the former interface uses a user-defined type
  ```

User-defined types I

```
package Impl;

import CTL.Serialize.*;
import CTL.Types.*;
import java.io.*;
import java.lang.reflect.*;

public class DoubleVector implements Writable {

private double x,y;

public DoubleVector(double x,double y){
        this.x = x;
        this.y = y;
}

public double getx(){
        return this.x;
}
```

User-defined types II

```
public double gety(){
        return this.y;
}

public void read (SerialIn in) throws IOException,
    ClassNotFoundException, IllegalAccessException,
    InvocationTargetException, InstantiationException
{
        x = in.readDouble();
        y = in.readDouble();
}

public void write (SerialOut out) throws IOException,
    IllegalAccessException, InvocationTargetException,
    CTLException
{
        out.writeDouble(x);
        out.writeDouble(y);
}
}
```

Component implementation I

```
package Impl;

import CTL.Annotate.*;

public class TestPoints {

static int pointsin; // Punkte im Kreis

        @export public static int testpoints(DoubleVector[]
            points){
                for(int i=0;i<=points.length-1;i++){
                        double x = points[i].getx();
                        double y = points[i].gety();
                        if ( (x*x + y*y) <= 1) pointsin++;
                }
                return pointsin;
        }
}
```

Service side I

```
import CTL.*;
import CTL.Types.*;

public class Server {

    public static void main (String[] args) {

            try {
                    boolean dmn = true;
                    int port = 0;

                    if (args.length > 0) {
                            port = RUtil.tryInt(args[0]);
                           dmn = (port != -1);
                    }

                    Group grp = null;
```

Service side II

```
                if (!dmn)
                        grp = new Group(args);
                else
                        grp = new Group("localhost",
                            port, 0, 2, Location.TCP,
                            null);
                grp.run();
            }
            catch (Exception e) {
                RUtil.except(e);
            }
        }
}
```

Client side I

```
import java.util.LinkedList;
import CTL.Types.*;
import Impl.*;
import javaSys.*;

public class Client {

static double x,y;

            public static void main (String args[])
            {
                  LinkedList<Location> locs = Location.
                        parseFile("locs.txt");
                  if (locs.size() != 1)
                        System.exit(1);
                  CTL.Process proc = new CTL.Process(locs.
                        get(0));

            TestPointsCI.use(proc);
```

Client side II

```
int repeat=100000;

DoubleVector[] temp = new DoubleVector[
     repeat];

for(int i=0;i<=repeat-1;i++){
        x = Math.random();
        y = Math.random();
        temp[i]=new DoubleVector(x,y);
}

double z = TestPointsCI.testpoints(temp)
     ;
z=z/repeat;
z *=4;

System.out.println("Pi = "+z);
proc.stopService();

        }
}
```

Build system I

```
[...]
<target name="compile" depends="init" description="Compile.">
        <depend srcdir="${src}" destdir="${build}"
                cache="${cache}" closure="yes"/>

        <javac srcdir="${src}" destdir="${build}" nowarn="true"
                debug="${debug}" excludes="Client.java, Server.
                    java"
                classpath="${classpath}"/>

        <java classname="CodeGen.Main" classpath="${classpath}"
            fork="true">
                <arg value="Impl.TestPoints" />
        </java>
        <javac srcdir="${src}" destdir="${build}" nowarn="true"
                debug="${debug}" classpath="${classpath}"/>
</target>

<target name="run" depends="compile" description="Berechne Pi.">
```

Build system II

```
        <java classname="${RNG}" classpath="${classpath}" fork="
            true">
                <jvmarg value="-Dfile.encoding=ISO-8859-1"/>
        </java>
</target>
[...]
```

- XML file with build rules
- Uses Apache Ant
- Using *-Dfile.encoding=ISO-8859-1* is very important
- Integration into Eclipse/NetBeans/... is left as an exercise for the audience

It works!

```
# Install Sun JDK >= 1.5.0_11
# Install Apache Ant >= 1.7.0

Lenin:~$ mkdir moo
Lenin:~$ cd moo
Lenin:~/moo$ wget -q http://icculus.org/projects/rollercoaster/
    ctl/example/MonteCarloPi-1158.tar.bz2
Lenin:~/moo$ tar xvfj MonteCarloPi-1158.tar.bz2
[...]
Lenin:~/moo$ cd MonteCarloPi/lib/
Lenin:~/moo/MonteCarloPi/lib$ ./get.sh
Lenin:~/moo/MonteCarloPi/lib$ cd ..
Lenin:~/moo/MonteCarloPi$ echo lib >locs.txt
Lenin:~/moo/MonteCarloPi$ ant
Buildfile: build.xml
[...]
     [java] Pi = 3.14596

BUILD SUCCESSFUL
Total time: 5 seconds
```

Outline

Resources

- CTL website: `http://www.wire.tu-bs.de/forschung/projekte/ctl/e_ctl.html`
- CTL4j website: `https://shuya.ath.cx/~neocool/code/CTL/` or `http://www.icculus.org/~boris/projects/CTL/` (mirror)
- Project work *The CTL protocol and its Java implementation* (2006)
- *CTL Manual for Linux/Unix* - for C++
- **Look at the examples**
- Forum: `http://www.wire.tu-bs.de/ctlforum/` (focused on more advanced topics concerning the CTL/C++)