POL: Kommunikation im Stil von UDP

Frederik Grüll

May 29, 2008

1 Analogien

UDP mit Sockets	POL mit Kommunikationsmatrix
IP-Adresse	Klassen- und Instanz-ID
Port	Eingangsregister-ID
Switch	Multiplexer
Prozess	Task
send(Empfänger, Daten)	Empfänger.send(internes Register)
recv()	Eingangsregister.get()
async. recv()	Eingangsregister.get(default)

2 Gemeinsamkeiten mit UDP

- Jeder Task ist selbst für die Kommunikation mit seinen Partnern verantwortlich.
- Jeder Task kann an jeden anderen Task Daten senden und von jedem anderen Task empfangen.

3 Unterschiede zu UDP

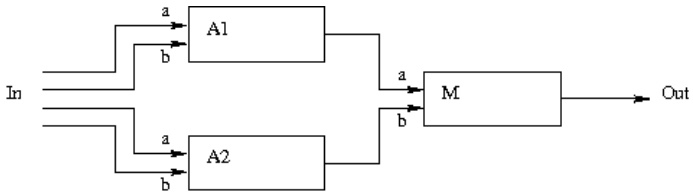
- Der Puffer liegt beim Sender, nicht beim Empfänger. Der "Eingangspuffer" ist nur ein Register groß.
- Daten gehen bei Überlastung des Empfängers nicht verloren, sondern stauen sich über den Multiplexer in der Ausgangsfifo des Senders.

4 Unterschiede zu POL 0.1

- Die calc()-Funktion wird nicht explizit aufgerufen. Sie läuft direkt nach dem Abbarbeiten des Konstruktors ständig.
- Es gibt keine Getter.
- "Port" entspricht "setter"

5 Beispiel: Adder und Multiplizierer

5.1 Datenfluss



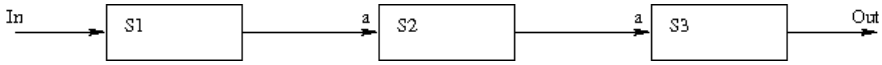
5.2 Code

```
class simplePOL extends ParObj {

  class Multiplier extends ParObj {
    Port a, b;          // Default ist Port<int>
    receiver out;
    int p;

    Multiplier(receiver output) {
      out = output;
    }

    calc() {
      p = a.get() + b.get();  // blockierend auf Port a und b
      out.send(p);
    }
  }

  class Adder extends ParObj {
    Port a, b;

    receiver m;    //Klasse, Instanz, Register von M.a bzw M.b
    int s;

    Adder(receiver mult) {
      m = mult;
    }

    calc() {
      s = a.get() + b.get();
      m.send(s);
```

```
      // evntl: m.send(a.get() + b.get());
    }
  }

  Adder A1;
  Adder A2;
  Multiplier M;
  Input In;        // Objekt aus Bibliothek
  Output Out;

  simplePOL() {
    Out= new Output();
    M  = new Multiplier(Out.out0);
    A1 = new Adder(M.a);
    A2 = new Adder(M.b);
    In = new Input(A1.a, A1.b, A2.a, A2.b)
  }


  calc() {
    //empty
  }
}
```

6 Beispiel: Pipeline

6.1 Datenfluss



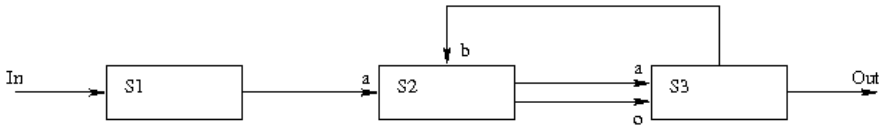
6.2 Code

```
class simplePOL extends ParObj {

  class Stage1 extends ParObj {
    Port a;              // Default ist Port<int>

    receiver out;
    int p;

    Stage1(receiver output) {
      out = output;
    }

    calc() {
      p = a.get() + 1;  // blockierend auf Port a
      out.send(p);
    }
  }

  class Stage2 extends ParObj {
    Port a;

    receiver out;
    int p;
```

```
    Stage2(receiver output) {
      out = output;
    }

    calc() {
      p = a.get() + 2;  // blockierend auf Port a
      out.send(p);
    }
  }

  class Stage3 extends ParObj {
    Port a;

    receiver out;
    int p;

    Stage3(receiver output) {
      out = output;
    }

    calc() {
      p = a.get() + 3;  // blockierend auf Port a
      out.send(p);
    }
  }

  Stage1 S1;
  Stage1 S2;
  Stage1 S3;
  Input In;
  Output Out;

  simplePOL() {
    Out = new Output();
    S3 = new Stage3(Out.out);
    S2 = new Stage2(S3.a);
    S1 = new Stage1(S2.a);
    In = new Input(S1.a);
  }


  calc() {
    //empty
  }
}
```

7 Beispiel: Pipeline mit Rückführung

7.1 Datenfluss



7.2 Code

```
class simplePOL extends ParObj {

  class Stage1 extends ParObj {
    Port<int> a;

    receiver s2_a;
    int p;

    Stage1(receiver output) {
      s2_a = output;
    }

    calc() {
      p = a.get() + 1;
      s2_a.send(p);
    }
  }

  class Stage2 extends ParObj {
    Port<int> a;
    Port<int> b;

    receiver s3_a;              // Stage3.a
    receiver s3_o;              // Stage3.o
    int p;

    Stage2(receiver output, receiver conf_output) {
      s3_a = output;
      s3_o = conf_output;
      s3_o.send(b);
    }

    calc() {
      p = a.get() + b.get(0);   // blockierend auf Port a,
      s3_a.send(p);              // nicht blockierend auf Port b
    }
  }

  class Stage3 extends ParObj {
    Port<int> a;
    Port<receiver> o;

    receiver out;
    receiver s2_b;
    int p;

    Stage3(receiver output) {
      out = output;
    }

    calc() {
      s2_b = o.get(0);
      p = a.get() + 3;          // blockierend auf Port a
      s2_b.send(p)
      out.send(p);
    }
  }

  Stage1 S1;
  Stage1 S2;
  Stage1 S3;
  Input In;
```

```
  Output Out;

  simplePOL() {
    Out = new Output();
    S3 = new Stage3(Out.out);
    S2 = new Stage2(S3.a, S3.o);
    S1 = new Stage1(S2.a);
    In = new Input(S1.a);
  }


  calc() {
    //empty
  }
}
```