

POL: Kommunikation im Stil von QT

Frederik

May 29, 2008

1 Befehle

Slot *in*; Deklariert den Eingang *in* einer Klasse

Signal *out*; Deklariert den Ausgang *out* einer Klasse

Sender.out.connect(Receiver.in); Verbindet das Signal *out* des Objekt *Sender* mit dem Slot *in* des Objekts *Receiver*

Sender.out.disconnect(Receiver.in); Trennt das Signal *out* des Objekt *Sender* von dem Slot *in* des Objekts *Receiver*

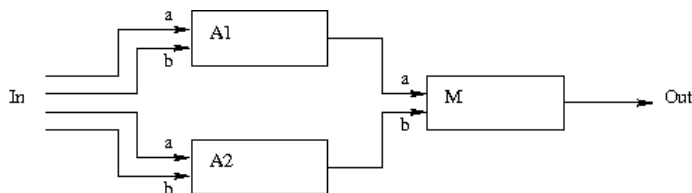
out.emit(value); Sendet eine Nachricht mit Inhalt *value* an alle mit dem Signal *out* verbundenen Slots

in.get() Blockiert, bis eine Nachricht an Slot *in* anliegt und holt diese ab

in.get(default) Holt eine Nachricht von Slot *in* ab oder gibt *default* zurück, falls keine Nachricht anliegt

2 Beispiel: Addierer und Multiplizierer

2.1 Datenfluss



2.2 Code

```
class simplePOL extends ParObj {

  class Multiplier extends ParObj {
    Slot a, b;
    Signal p;

    int result;

    calc() {
      result = a.get() + b.get(); // blocking wait on a and b
      out.emit(result);
    }
  }

  class Adder extends ParObj {
    Slot a, b;
    Signal s;

    int result;

    calc() {
      result = a.get() + b.get();
      s.emit(result);
    }
  }

  Adder A1;
  Adder A2;
  Multiplier M;
  Input In; // Library objects
  Output Out;

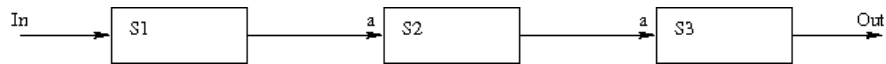
  simplePOL() {
    Out = new Output;
    M = new Multiplier;
    A1 = new Adder;
    A2 = new Adder;
    In = new Input;

    M.p.connect(Out.in);
    A1.s.connect(M.a);
    A2.s.connect(M.b);
    In.out1.connect(A1.a);
    In.out2.connect(A1.b);
    In.out3.connect(A2.a);
    In.out4.connect(A2.b);

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

3 Beispiel: Pipeline

3.1 Datenfluss



3.2 Code

```
class simplePOL extends ParObj {  
    class Stage1 extends ParObj {  
        Slot a;  
        Signal out;  
  
        int p;  
  
        calc() {  
            p = a.get() + 1; // blocking wait on a  
            out.emit(p);  
        }  
    }  
  
    class Stage2 extends ParObj {  
        Slot a;  
        Signal out;  
  
        int p;  
  
        calc() {  
            p = a.get() + 2;  
            out.emit(p);  
        }  
    }  
  
    class Stage3 extends ParObj {  
        Slot a;  
        Signal out;  
  
        int p;  
  
        Stage3(receiver output) {  
            out = output;  
        }  
  
        calc() {  
            p = a.get() + 3;  
            out.emit(p);  
        }  
    }  
  
    Stage1 S1;  
    Stage1 S2;  
    Stage1 S3;  
    Input In;  
    Output Out;  
  
    simplePOL() {
```

```

    Out = new Output;
    S3 = new Stage3;
    S2 = new Stage2;
    S1 = new Stage1;
    In = new Input;

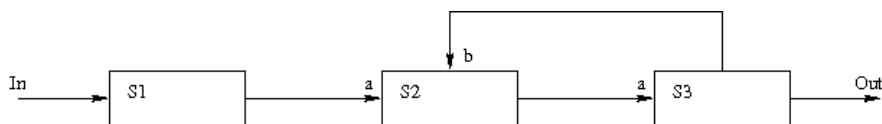
    S3.out.connect(Out.in);
    S2.out.connect(S3.a);
    S1.out.connect(S2.a);
    In.out.connect(S1.a);
}

calc() {
    //empty
}
}

```

4 Beispiel: Pipeline mit Rückführung

4.1 Datenfluss



4.2 Code

```

class simplePOL extends ParObj {

    class Stage1 extends ParObj {
        Slot a;
        Signal out;

        int p;

        calc() {
            p = a.get() + 1;
            out.emit(p);
        }
    }

    class Stage2 extends ParObj {
        Slot a;
        Slot b;
        Signal out;

        int p;

        calc() {
            p = a.get() + b.get(); // blocking wait on a
            out.emit(p);
        }
    }

    class Stage3 extends ParObj {
        Slot a;

```

```

    Signal out;
    int p;

    calc() {
        p = a.get() + 3;
        out.emit(p);
    }
}

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

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

    S3.out.connect(Out.in); // output with 2 inputs assigned
    S3.out.connect(S2.b);
    S2.out.connect(S3.a);
    S1.out.connect(S2.a);
    In.out.connect(S1.a);
}

calc() {
    //empty
}
}

```