Nachrichten an den RCX schicken

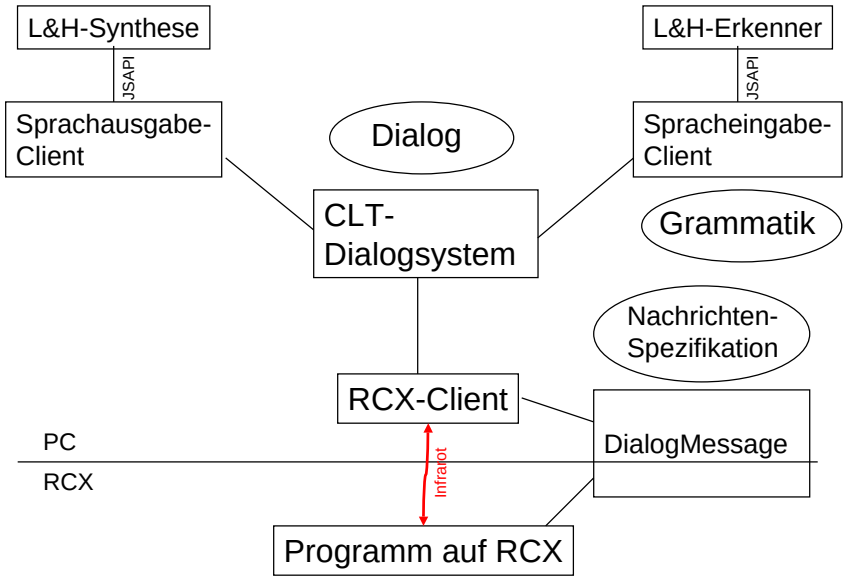
Alexander Koller

Softwareprojekt "Sprechende Roboter"
14. Mai 2004

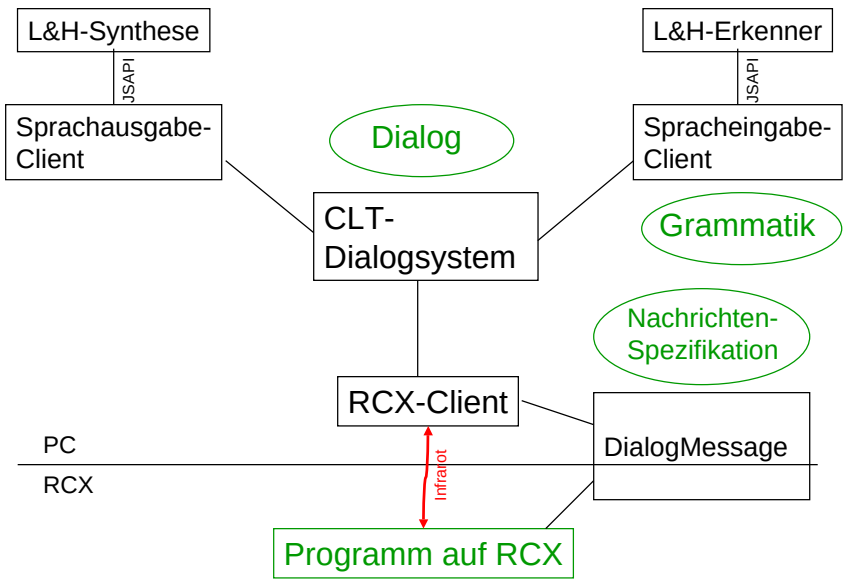
Überblick

- Struktur des Systems auf dem PC
- DialogMessage: Austausch von Nachrichten über Infrarot
- Spezifikation von Nachrichtentypen
- Verwenden auf PC- und RCX-Seite
- Kurzer Überblick über Tools

Struktur des Systems auf dem PC



Struktur des Systems auf dem PC



Zu einem neuen Projekt gehören:

- Dialog-Spezifikation
- Programm für RCX
- Erkenner-Grammatik
- Spezifikation der Nachrichten

DialogMessage

- Spezifikation von Nachrichtentypen in XML-Datei.
- XML-Datei wird in Quelltext für Klasse DialogMessage kompiliert.
- Erzeuge Versionen für PC und RCX: Fast gleich, aber RCX viel kleiner (und eingeschränkte Funktionalität).
- Binde in RCX-Client auf PC und in Programm auf dem RCX ein.

Ein Beispiel

```
<?xml version='1.0' encoding='utf-8'?>

<messages>
 <package name="de.saar.coli.lego.clients.rcxclient" />

 <argdecl>
  <name>floor</name>
  <datatype><int/></datatype>
  <description>The target floor</description>
 </argdecl>

 <message>
  <name>GO</name>
  <description>Instructs RCX to move to that
        floor.</description>
  <argument>floor</argument>
 </message>
</messages>
```

Struktur der Spezifikation

- Spezifikation enthält
  - Deklaration der Ziel-Package:
    (muss de.saar. ... .rcxclient sein!)
    `<package name="..." />`
  - Deklarationen von Nachrichten:
    `<message>` ... `</message>`
  - Deklarationen von Datenfeldern:
    `<argdecl>` ... `</argdecl>`
- Jede Deklaration muss einen Namen und kann eine Beschreibung enthalten.

Deklarationen von Datenfeldern

- Zusätzlich muss jedes Datenfeld einen Datentyp definieren:
  - Integer (1 Byte): `<int/>`
  - Atome:

```
<atoms>
 <atom>MOVING</atom>
 <atom>STOPPED</atom>
</atoms>
```

Deklarationen von Nachrichtentypen

- Die Deklaration der Nachricht gibt an, welche Datenfelder die Nachricht enthält:

```
<message>
    <name>STATUS_IS</name>
    <description>Description of a current
            status.</description>
    <argument>status</argument>
    <argument>floor</argument>
    <argument>direction</argument>
  </message>
```

Kompilation der Spezifikation

- Die Spezifikation wird durch folgenden Aufruf kompiliert:

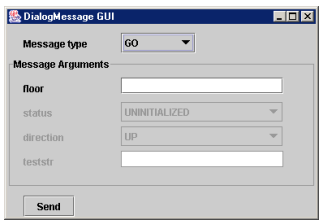
```
ant -Dproj.name=<...>
    -Dproj.messagespec=<...> project-pc
```

- In proj.messagespec wird der Name der XML-Datei angegeben; in proj.name ein eindeutiger Name.
- Erzeugt build/lib/DialogMessage-XXX.jar, wobei XXX der eindeutige Name ist.
- Kann in den RCX-Client eingebunden werden (siehe nächste Woche).

Ein GUI für DialogMessages

- Zum Test will man manchmal Nachrichten an den RCX schicken, ohne DialogMessage zu kompilieren und in Client einzubinden.
- Graphisches Interface:

```
java -jar build/lib/DialogMessageWin.jar
          [-dummy] <Name XML-Datei>.xml
```



Kompilation für den RCX

- Für den RCX muss eine eigene, abgespeckte Version von DialogMessage erzeugt werden.

```
ant -Dproj.path=<...> -Dproj.main=<...>
    -Dproj.messagespec=<...> project-rcx
```

- Dies erzeugt DialogMessage.java und kompiliert es zusammen mit dem Lejos-Programm in `${proj.path}/RCX`. Die Klasse, die die main-Methode enthält, wird in proj.main angegeben.
- Kompilierte Klassen liegen dann in `build/src/${proj.path}/RCX`.
- Upload auf den RCX:

```
ant -Dproj.path=<...> -Dproj.main=<...>
    project-rcx-upload
```

Nachrichtenbehandlung auf dem RCX

- Auf dem RCX soll ein eigener Thread laufen, der Nachrichten über Infrarot annimmt und verarbeitet.
- Implementiert in Klasse `MessageProcessor`.
- Übergebe an ihren Konstruktor eine Instanz von `MessageHandler`; kapselt Methode, die mit Nachrichten aufgerufen wird.

Beispiel

```
class Test {
  class MyMessageHandler implements MessageHandler {
    public void process (DialogMessage in, DialogMessage out) {
      if( in.Type == DialogMessage.TYPE_BEEP )
        for( int i = 0; i < in.howoften; i++ ) Sound.beep();

      out.Type = DialogMessage.TYPE_OK;
    }
  }

    // ...

  RCXPort port;
  try { port = new RCXPort(); } catch(Exception e) {}

  MessageProcessor p = new MessageProcessor(port,
                                  new MyMessageHandler());
  p.start();
}
```

MessageHandler und DialogMessage

- `MessageHandler` ist ein Interface, dessen Implementierungen die `process`-Methode überladen müssen.
- `process()` bekommt die eingehende `DialogMessage` übergeben und soll eine Antwort-Nachricht zurück übergeben. Dazu Felder von `out` überschreiben.
- `DialogMessage` ist hier die RCX-Version der aus der XML-Spezifikation erzeugten Klasse.

DialogMessage auf dem RCX

- Der Typ der Nachricht steht im Feld `Type`.
- Es gibt statische Konstanten `TYPE_FOO` für jeden Nachrichtentyp FOO.
- Für jedes mögliche Argument gibt es ein gleichnamiges Feld in `DialogMessage`.
- Für ein Atom `FOO` im Argument `bar` gibt es eine statische Konstante `ATOM_bar_FOO`.
- Achtung: Typüberprüfung selbst implementieren!

Struktur eines Projekts

- In Euer Projektverzeichnis gehören:
  - XML-Nachrichtenspezifikation
  - In Unterverzeichnis "RCX": RCX-Quelldateien, incl. Kopie von `MessageProcessor.java` und `MessageHandler.java`.
  - Später: Dialogspezifikation und Erkennergrammatik.
- Zum Kompilieren verwendet Ihr ant-Buildfile aus dem CVS.

Tools

- Wir verwenden "Ant", um Java-Programme zu kompilieren.
- Außerdem benötigen wir das Versionskontroll-system CVS, um Quelltexte zu verwalten.
- Dafür lohnt es sich, das Cygwin-System zu installieren. Vorsicht: Dann muss cygwin1.dll im Lejos-Verzeichnis gelöscht werden.
- Weitere Tools und ihre Installation sind auf der CD beschrieben.

CVS

- CVS ist ein System, mit dem mehrere Leute gleichzeitig an (Quell-)Texten arbeiten können, ohne sich ins Gehege zu kommen.
- Es gibt ein zentrales Repository, das immer konsistent gehalten wird.
- Jeder Benutzer hat eine lokale Kopie der Dateien im Repository.
- Konflikte, die bei gleichzeitiger Änderung auftreten, müssen lokal behoben werden.

CVS: Die wichtigsten Befehle

- Eine neue lokale Kopie des Repository erzeugen:
  ```
  cvs -d ...@gnome:/proj/lego/cvs co lego
  ```
- Lokale Dateien auf den neuesten Stand bringen:
  ```
  cvs update
  ```
- Lokale Dateien ins Repository zurückschreiben:
  ```
  cvs commit
  ```
- Neue Dateien für CVS registrieren:
  ```
  cvs add <Dateiname>
  ```
- CVS funktioniert wunderbar übers Netzwerk. Dazu muss aber Umgebungsvariable CVS_RSH gesetzt sein:
  ```
  export CVS_RSH=ssh
  ```

Struktur des Lego-CVS

- Das CVS-Modul "lego" enthält die Dateien für die Lego-Tools:
  - Verzeichnis "de" enthält Quelltexte.
  - Verzeichnis "build" enthält Kompilate.
  - Verzeichnis "projects" enthält ältere Projekte.
  - Datei "build.xml" ist Ant-Buildfile.
- Eure eigenen Dateien gehören nicht ins Modul "lego"; wir legen noch ein eigenes Verzeichnis für euch im Modul "teams" an.

In der nächsten Sitzung

- Dialogsystem
- Spracherkenner-Grammatik
- Integration DialogMessage und Dialogsystem