ALG-Seminar Sensornetze

Benutzung der Testbedsoftware in den Fluren

Tobias Baumgartner

TU Braunschweig

27. April 2011

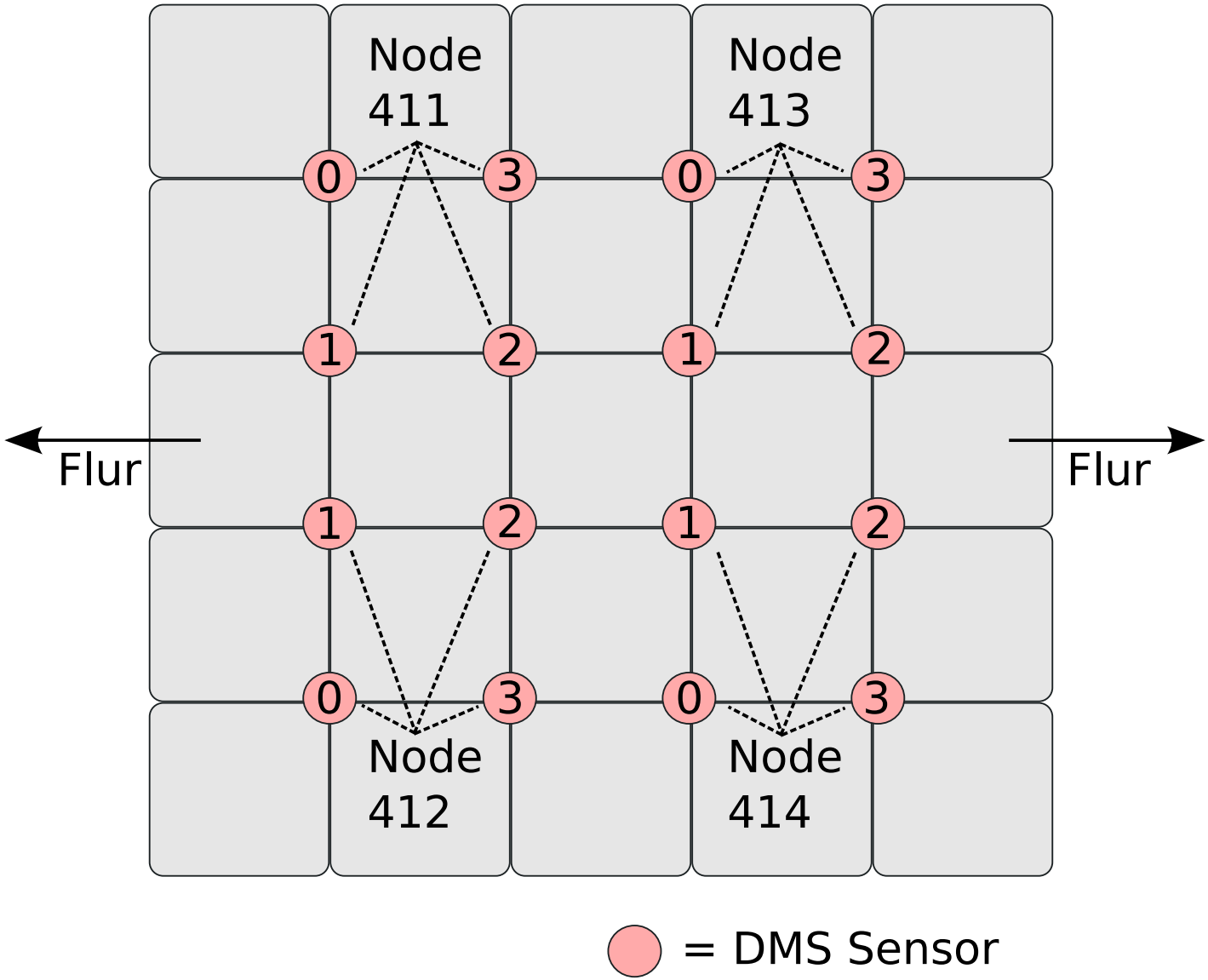
Outline

Outline

Flure

- Großer Flur
  - Zugriff über `wbportal.ibr.cs.tu-bs.de` (im fensterlosen Raum)
  - 30 iSense-Knoten, 3 Alix-Boards unterm Boden
  - Alix-Boards: `wbhallway{1,2,3}.ibr.cs.tu-bs.de`
    - 401..410 an `wbhallway1`
    - 411..420 an `wbhallway2`
    - 421..430 an `wbhallway3`
- Miniflur
  - Zugriff über `algminiflur.ibr.cs.tu-bs.de`
  - 4 iSense-Knoten, Alix-Board eingebaut
  - Knoten-IDs 101..104

Sensorplatzierung



Outline

testbed-runtime aka Daniel-Backend

- Java Framework aus Lübeck
- **iWSN-API**: Kommunikation mit Sensorknoten
- **Reservierungssystem (RS)**: Reservieren von Sensorknoten für einen bestimmten Zeitraum
- **Authentifizierung/Autorisierung (SNAA)**: Verwalten von Benutzern/Passwörtern/ggf. Rechten
- **Experimentation scripts**: Skriptbasierte Steuerung von Experimenten (Flashen, Debugausgaben sammeln, ...)

Ablauf Testbed-Zugriff

- Anfrage eines Benutzers mit user/pass sowie gewünschte Zeit bei RS
- RS fragt bei SNAA nach, ob user/pass auf das gewünschte Testbed zugreifen darf
- Wenn erfolgreich, gibt RS einen Reservierungsschlüssel (KEY) zurueck
- Mit dem KEY kann der Benutzer beim SM eine Instanz einer WSN-API beantragen
- Ist der KEY korrekt (SM fragt RS), gibt SM die URL einer WSN-API zurueck
- Mit dieser URL kann der Benutzer auf das Testbed zugreifen

iWSN API

- Besteht aus WSN-API (Testbedzugriff) und Session-Management
- Konfiguration per XML-Datei (`/opt/testbed-runtime/tr.iwsn-testbed.xml`)
- Auf Alix-Boards im Flur
  - Weiterleiten von Sensoroutput an Portal Server
  - Weiterleiten von Kommandos vom Portal Server an Knoten
- Portal Server und Miniflur
  - Webservices (SM, WSN-API)
  - Protobuf-Interface (*Webservices per Sockets*)
  - CoCoS
  - Socket Connector

Reservierungssystem

- Reservieren von Sensorknoten aus Testbed
- Konfiguration:
  `/opt/testbed-runtime/tr.rs-testbed.properties`
- Eintrag in Google Kalender
- TUBS-Testbed-Account
  - `tubs.testbed@googlemail.com`

SNAA

- Authentifizierung und Autorisierung
- Konfiguration:
  `/opt/testbed-runtime/tr.snaa-testbed-shibb.properties`
- Shibboleth
  - Wisebed SNAA System
  - Account in BS erlaubt z.B. auch Zugriff in Lübeck (nach expliziter Freischaltung dort)
- htpasswd
  - User/pass in Datei
  - Schnell gemacht, aber nicht übertragbar
- Derzeit in Benutzung: Shibboleth

Outline

Starten/Stoppen der testbed-runtime (1 von 2)

- Drei Startskripte verfügbar
  - `/etc/init.d/tr.{iwsn|rs|snaa} {start|stop|restart|force-reload|stats|isrunning}`
  - Logs in `/var/log/testbed-runtime/`
  - Prozess-ID in `/var/run/testbed-runtime/`
- Portal Server (`wbportal`) und Miniflur (`algminiflur`)
  - iWSN, RS, SNAA
- Alix-Boards im Flur (`wbhallwayX`)
  - iWSN
- Starten der Scripte
  - Jeweils als Benutzer `testbed` oder `root`
  - Auf Miniflur derzeit auch Benutzer `hhh1` und `hhh2`

Starten/Stoppen der testbed-runtime (1 von 2)

- DEMO

Outline

Experimentation Scripts (1 von 2)

- Teil der testbed-runtime
- Skripte basierend auf Webservices und Sockets
  - reserve
  - check-alive[-proto]
  - reset[-proto]
  - flash[-proto]
  - listen[-proto]
  - show-wiseml

Experimentation Scripts (2 von 2)

- DEMO

CoCoS

- DEMO

Tarwis

- `https://wbportal.ibr.cs.tu-bs.de/portal/TARWIS/Welcome/welcomeIndex.php`
- DEMO

Wisebed SNAA

- https://wbident.ibr.cs.tu-bs.de/idpadmin/
- https://wbident.ibr.cs.tu-bs.de/idpadmin/password
- DEMO

Outline

iSense-Knoten (thx Winnie!)

- Kommunikation mit CoCoS und DMS-Board
  - DMS-Werte lesen
  - LED-Kommandos senden
  - ...
- Grundlegende *Lib* vorhanden
  - Communicator: CoCoS $<->$ iSense $<->$ DMS-Board
  - SensorDataTask: Bearbeiten von DMS-Paketen, Weitergeben an CoCoS
- Einbinden in eigene Anwendung
  - CoCoS funktioniert wie gewohnt
  - Verteilte Algorithmenentwicklung möglich

Pakettypen

- Jede Nachricht *CoCoS* $<->$ *iSense* $<->$ *DMS-Board* wird durch Type im ersten Byte identifiziert
- Basistypen

```
1     PCK_DMS_VALUES_ID = 'D',
2     PCK_DMS_VALUES_PAYLOAD = DMS_COUNT * sizeof(uint16),
3     PCK_TEMPERATURE_ID = 'H',
4     PCK_TEMPERATURE_PAYLOAD = 2,
5     PCK_SENSOR_EVENT_ID = 'I',
6     PCK_SENSOR_EVENT_PAYLOAD = 2,
7     PCK_FLOOR_TILE_EVENT_ID = 'J',
8     PCK_FLOOR_TILE_EVENT_PAYLOAD = 2,
9     PCK_LEDS_ID = 'L',
10    PCK_LEDS_PAYLOAD = 3,
11    PCK_PIR_STATE_ID = 'P',
12    PCK_PIR_STATE_PAYLOAD = 0,
```

Eigene Anwendung

- Ableiten von `HallwaySensorHandler`
- Methoden, die überschrieben werden **können**

```
1   /// Rohdaten der DMS (derzeit gesendet mit 250Hz)
2   void dms_stream( uint8 id, uint16 value );
3   /// Mittelwerte der DMS: (derzeit gesendet mit 5Hz)
4   void dms_event( uint8 id, uint16 shortterm, uint16 longterm );
5   /// Wird aufgerufen, sobald ein PIR ausschlaegt
6   void pir_event();
```

Beispiel (1 von 2)

- Ableiten von isense::Application und HallwaySensorHandler

```
class FloortileSensor
    : public isense::Application,
      public HallwaySensorHandler
```

- Berabeiten von Sensorwerten

```
#ifdef USE_DMS_STREAM
    virtual void dms_stream( uint8 id, uint16 value );
#else
    virtual void dms_event( uint8 id, uint16 shortterm,
                                      uint16 longterm );
#endif
```

- Private Member

```
    SensorDataTask sensor_data_task_;
    Communicator comm_;
```

Beispiel (2 von 2)

- Implementierung

```
1 FloortileSensor::
2 FloortileSensor( isense::Os& os )
3     : isense::Application( os ),
4       sensor_data_task_( os, comm_ ),
5       comm_( os, &sensor_data_task_, 0 ),
6       os_( os ),
7       [...]
```

#fin

- thx!