

SmartRFLib - Progress Report 2

Cagri Balkesen, Ali Sengül
Nihal Dindar, Florian Keusch
Catharina Kromwijk, Gautier Boder

Overview

- Adaptive Cleaning Part
- Communication with upper layer
- Compression Part
- Simple Visualization of Cleaning
- Next tasks

Adaptive Cleaning of RFID Readings

- Epoch = basic reading time unit
- Window size = some number of epochs
- Adaptive cleaning algorithm from SMURF Paper adapted
- Window size changes according to read probabilities

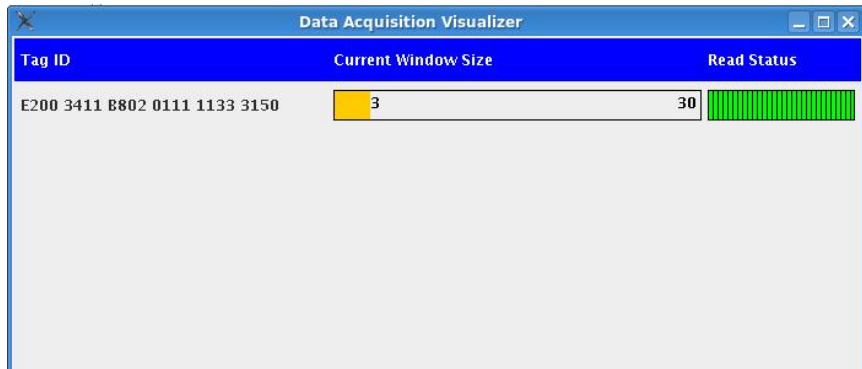
Communication

- Communication with Object streams over TCP sockets
- We will have different channels for each reader
- Data push rate will be slower than raw readings

Compression

- Two types of compression
- For shelf: time based
- For checkout and exit: range exit based
- This layer will diminish data rate that we'll push
- For shelf it's parametric and currently operational

Simple Visualization of Cleaning



Next Work

- Cleaning improvement: Mobil tag detection
- Adding all readers and sending readings for all
- Compression for checkout and exit
- Some experiments to understand reading behaviors

Outline

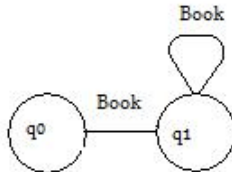
- Finite State Automata
- Communication with both layers
- What's next?

Finite State Automata

```
NAME book_theft ON R3  
PATTERN (Book + b[])  
MATCH INCREMENTAL  
WHERE NOT borrowed(b[i].id)  
ACTION theft(b[],id)  
RETURN True
```

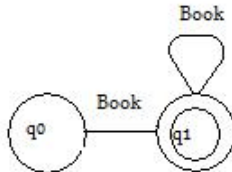
Finite State Automata

```
NAME book_theft ON R3  
PATTERN (Book + b[])  
MATCH INCREMENTAL  
WHERE NOT borrowed(b[i].id)  
ACTION theft(b[],id)  
RETURN True
```



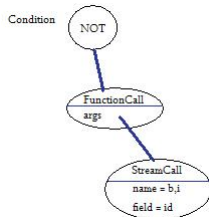
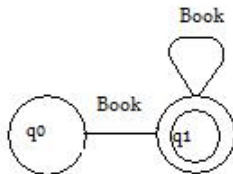
Finite State Automata

```
NAME book_theft ON R3  
PATTERN (Book + b[])  
MATCH INCREMENTAL  
WHERE NOT borrowed(b[i].id)  
ACTION theft(b[],id)  
RETURN True
```



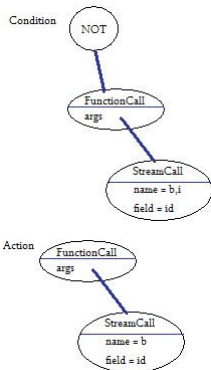
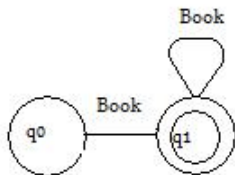
Finite State Automata

NAME book_theft **ON** R3
PATTERN (*Book* + *b*[])
MATCH INCREMENTAL
WHERE NOT borrowed(*b*[*i*].*id*)
ACTION theft(*b*[],*id*)
RETURN True



Finite State Automata

NAME book_theft **ON** R3
PATTERN (*Book* + *b*[])
MATCH INCREMENTAL
WHERE NOT borrowed(*b*[*i*].*id*)
ACTION theft(*b*[],*id*)
RETURN True



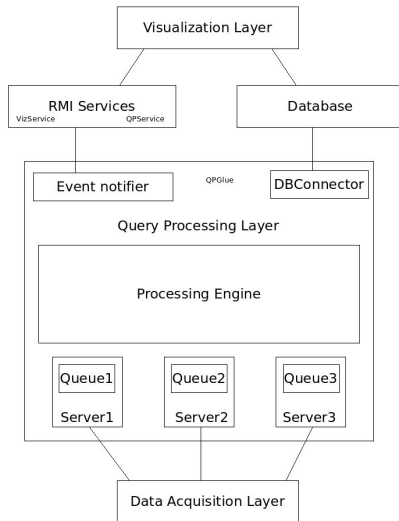
Communication with DA layer

- Use **TCP Sockets** as connection to the lower layer
- **3 Servers**, for every reader one
- ObjectsStream sends a '**SendObject**'
- fill missing data in SendObject
- add SendObject to the **SendQueue** in correct **order**
- SendQueues are the input for the processing engine

Communication with VIZ layer

using RMI (see section Visualization)

Communication sketch



What is next?

- Stream data: operation on kleene closure
- Function call: different return values
- Time Window
- New Keywords: MATCH LONGEST, MATCH INCREMENTAL
- Multiple input stream, multiple finite state automata
- Collate all the components

Communication between Query processing Layer and Visualization Layer

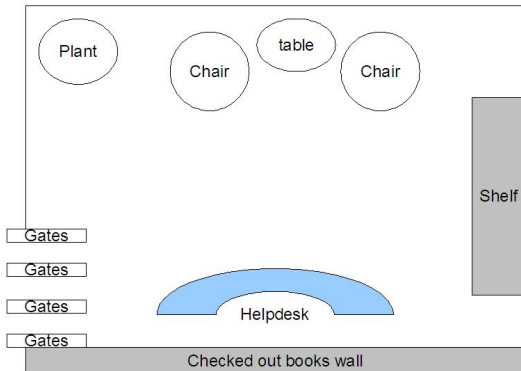
- The communication is based on RMI calls
- No more RMI calls from the visualization layer to the query processing layer since we now share the database access
- All the RMI methods are implemented on the visualization layer and the query processing layer calls them when an event is triggered
- Methods: Checkout(...), Checkin(...), TooManyBooks(...), illegalCheckout(...), bookTheft(...), bookRemove(...), bookBack(...)

Overview

- Redesign of the Second Life world
- Web interface
- Timetable

The new Second Life world

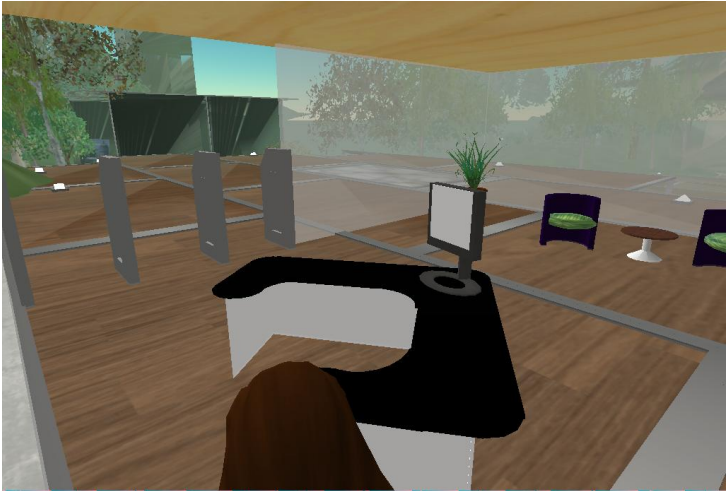
- We wanted to make the library more realistic
- Added chairs from where everything can be observed



The new Second Life world



The new Second Life world



The new Second Life world: Alarms

- The books and gates each have LSL code included which can show visual alarms
- Sound alert is possible, but we have to pay for upload of a sound file: for next time?
- The books and gates blink on alarm and display a suitable text to be as clear as possible
- Delay:TODO

Functionalities

- Finally defined!
- **Users interrogations** through
 - Fixed queries
 - Parametrized queries
 - User defined queries
- Show the **real-time system's status**

Fixed queries

- List all **stored books** and their related information
- List all **stored persons** and their related information
- List all **borrowed books**
- List all **lost books**
- List all **misplaced books**

Parametrized queries

- Search for a specific book
 - By author,
 - title,
 - tagId

User defined queries

- Run any MySQL queries and shows the result set

Real-time system's status

- Uses a **Java Applet**
- Shows all books' status within a table
- Connected through RMI with the VL's core system
- Received any events

T timetable

Currently: implementation and testing of the web interface

Next:

- Final testing with the other layers
- Add / remove / change books through the web interface
- Book registers itself with the server
- Finalize applet and web interface