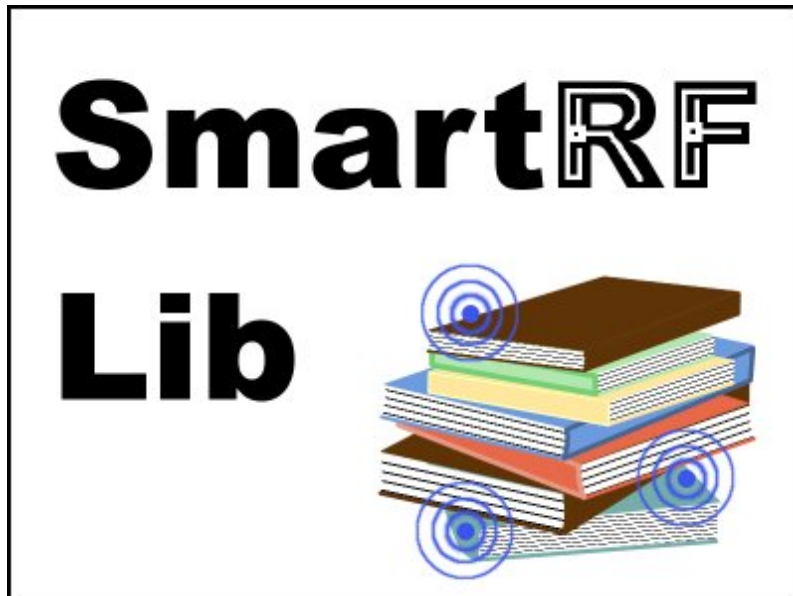


Progress Report 2



SmartRF Lib

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

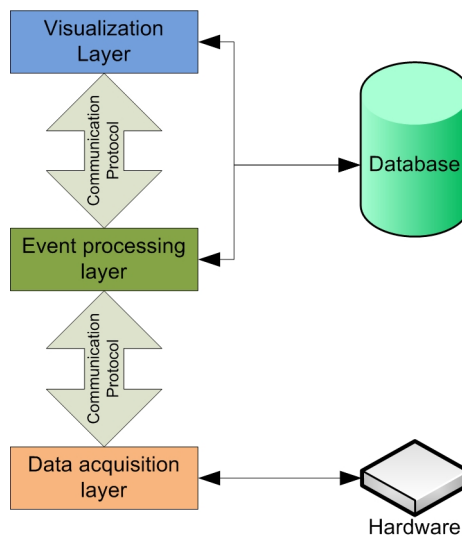
November 28, 2007

Contents

1	Introduction	3
2	Communication	3
	2.1 Visualization layer - event processing layer	3
	2.2 Event processing layer - data acquisition layer	4
	2.3 Communication sketches	5
3	Visualization Layer	6
	3.1 Second Life	6
	3.2 Web interface	9
	3.3 Timetable and further planning	11
4	Event Processing Layer	11
	4.1 Finite state automaton	12
	4.2 Stream Data and Hybrid Queries	12
	4.3 Assumptions	13
	4.4 Future directions	14
	4.5 Expression tree	14
	4.6 Some adaptation in our EBNF	14
	4.7 Events	17
	4.8 time table update	17
5	Data Acquisition layer	17
	5.1 Adaptive Cleaning	17
	5.2 Adaptive Cleaning Algorithm	19
	5.3 Interface to upper layer	20
	5.4 Reading Tests	20
6	Conclusions	23

1 Introduction

Our smart library is now at an interesting stage: the single layers have already some basic functionality and the goal is now to make them work together. We therefore not only describe the single advancements made on the different layers, but also define a communication protocol which enables them to communicate with each other. There are two of these communication points: one is between the data acquisition layer and the event processing layer and one is between the event processing layer and the visualization layer. We also redefined slightly the overall schema of our application by sharing the database between the visualization and event processing layer:



2 Communication

An important task in our library system is the communication between the 3 layers. The event processing layer has connections with both other layers. This section describes the how these layer communicate.

2.1 Visualization layer - event processing layer

General functionality

The communication with the event processing layer is done by Java RMI method calls. This method allows us to trigger events in real time and therefore always be up to date. Some of the methods are defined by the event processing layer so that the visualization layer can call them. These are typically the methods which need only to be executed on request of the user or once a day, for example. The other methods are implemented on the visualization layer and are triggered by the event processing layer when an event has been detected. These events can be alarms such as illegal checkouts, book thefts or too many books notifications, but can also be simple events such as checkouts and books taken out of a shelf.

Implementation by the visualization layer

We define several methods for the lower layer to call:

- **checkIn(String bookId):** Is called when a book is checked in. This results in a movement in second life from wherever the book is to it's own position in the shelf.
- **checkOut(String bookId):** Is called when a book is checked out. The book is now moved from wherever it is to a free position on the checkout wall.
- **tooManyBooks(String personId):** When someone borrows more books than it is allowed this method is called. An alarm is triggered in second life which makes the book and the gates blink in light blue.
- **illegalCheckout(String bookId):** When someone checks out a book which is destined to in library use only, the same type of alarm is triggered in second life as the tooManyBooks method. Just the text above the book and gates is different.
- **bookTheft(String bookId):** When a book is stolen, this method triggers an alarm in second life which makes the book and the gate blink in red.
- **bookRemove(String bookId):** When a book is lost, this method sets the book to transparent.
- **bookBack(String bookId):** When a book is found again, this method sets the book back to its normal transparency.

Of course, all these events will also trigger some visualization on the web interface, but this is not implemented yet.

Implementation by the event processing layer

The communication with the visualization layer is done by remote procedure call. We use java as our implementation languages so we also decided to take RMI. Both layers provide now a service where to call remote procedures.

The event processing layer actually has a service which provides access to the database. This may not be necessary at the end. We plan to merge the database later on, so that both layer have direct access to the database. Our layer needs access to the database to evaluate the events and to do some action. The visualization layer needs access basically to store the books position in the shelf in the virtual second life world.

Our layer needs possibilities to inform the upper layer when an event occurred. We can do this with the remote procedures we get from the visualization service. So the event processing layer takes the service given by the visualization layer to notify about detected events. This means for example if we detect the event 'theft' then we can call a remote procedure bookTheft() with some parameters to inform the upper layer of this event.

2.2 Event processing layer - data acquisition layer

This communication relies on TCP sockets.

Implementation

Here we communicate by TCP Sockets. The event processing layer is the server, the data acquisition layer is the client of this connection.. The client from the data acquisition layer then connects to the server using it's ip and port. The event processing layer can get the data by an ObjectOutputStream. The objects that are sent by the connection are of the class SendObject.java. A such sendObject contains the following data:

String tagID: This field contains the RFIDTag that is measured at the reader.

String readID: This field contains the unique readerID. In our whole library setup as it was proposed we will place 3 different readers.

int type: This field contains the type of the RFIDTag. It's set to 0 if the type is a book. It's set to 1 if RFIDTag corresponds to a book.

String typeString: This string is basically nothing else than the type, but in words. If a type is book then this string should be set to "Book". If it's a person then the string should be "Person".

Date time: This is the time when the tag is measured by the reader.

When a new sendObject arrives in the event processing layer then we have to do preprocess the data before feeding it into the processing engine. First the sendObject is classified. The type can't be set by the data acquisition layer since this layer does not have access to our database. So first we lookup the database if the tagId is a person and then the tagID and the typeString are set correctly. The next step is to put the sendObject into a queue (SendObjectQueue.java). Adding the sendObject into the queue is not trivial. We have to guarantee the order of the tags. In the assumptions of the event processing layer we stated (see section finite automaton) that the processing strongly relies on the ordering. In an event first the person tag and then the book tags follow. For forcing this we have to add the sendObject to the correct position in the queue. We also may have to add some locking mechanisms to ensure that this order is guaranteed. Adding and getting from this queue has to be synchronized in order not to mess up the queue's consistency.

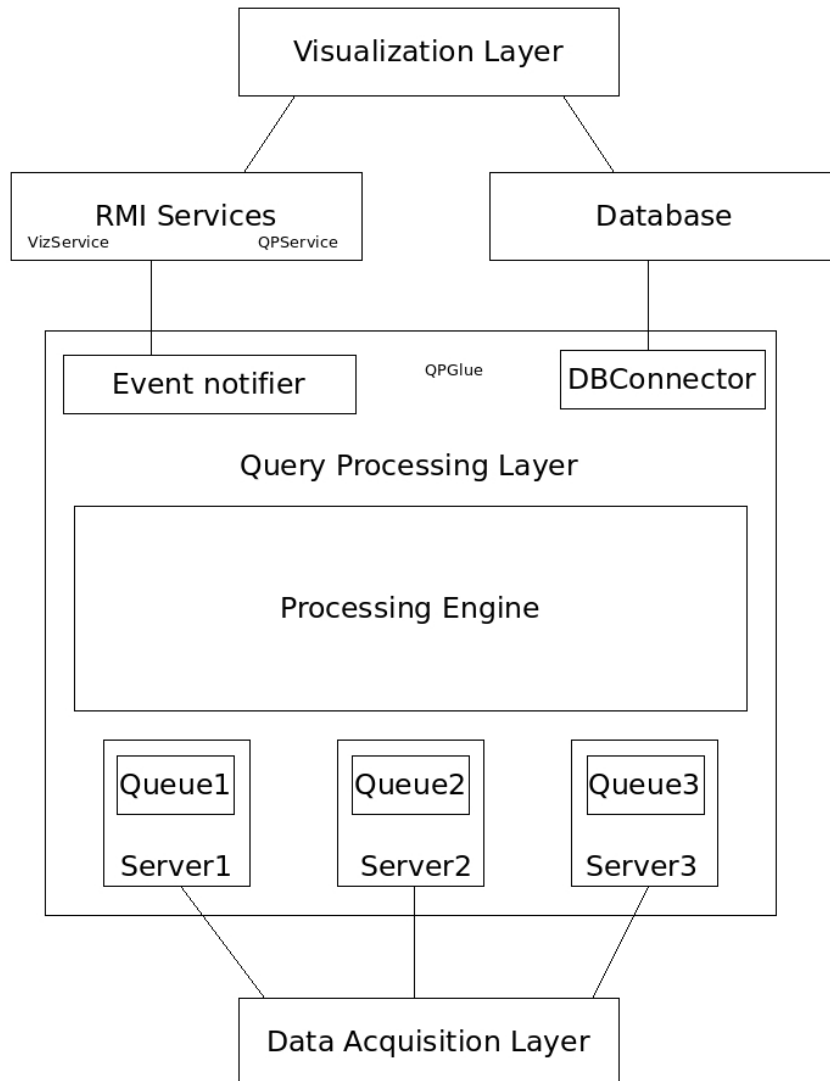
For the communication with the data acquisition layer the event processing layer opens 3 server connections. We get an objectStream connection for each different reader. We will have 3 different queues simulating streaming data from the lower layer. These queues are the feeders of the corresponding finite automata.

2.3 Communication sketches

Event processing layer communication sketch

The event processing layer has to communicate with both other layers. See in the Section communication.

The communication between the 2 layers the database and the event processing layer is illustrated in the following sketch:



3 Visualization Layer

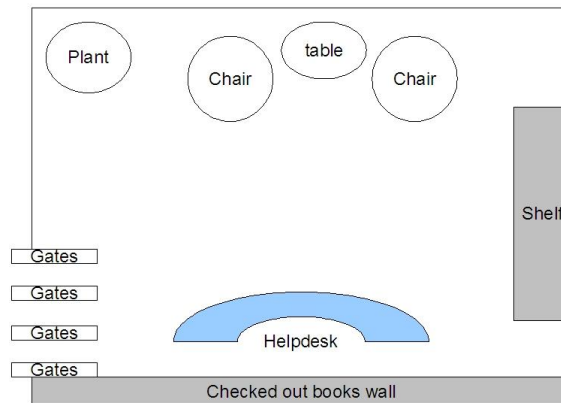
3.1 Second Life

The second Life library is currently very basic and for this progress report we have made it more interactive. The changes are as well in the design of the library as in the communication with our Java program.

World architecture

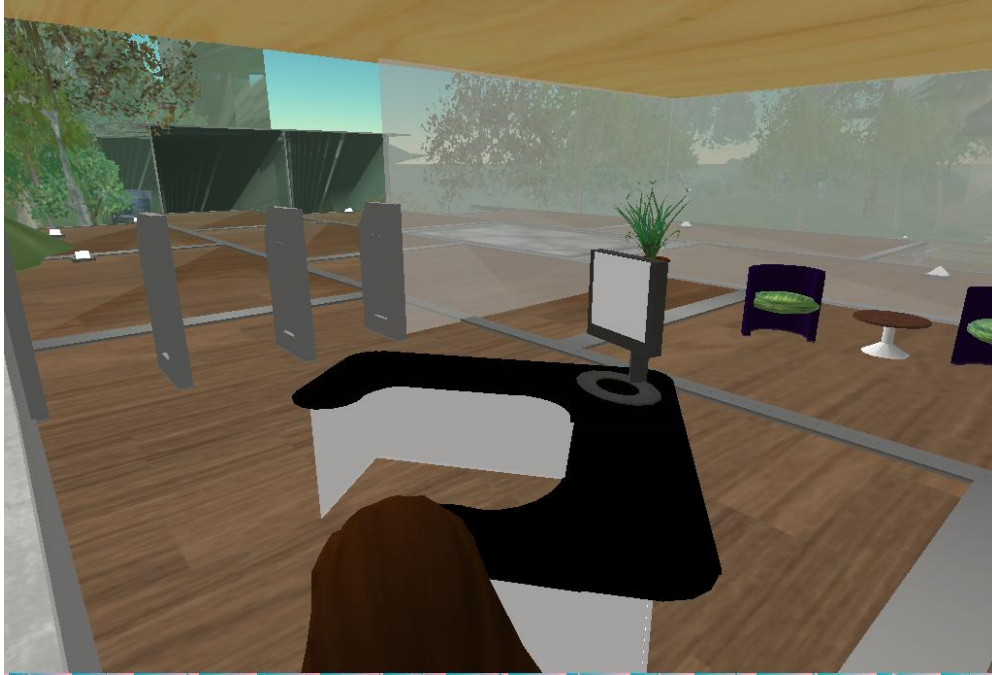
Our library now has a new architecture to make it more realistic, and therefore, more intuitive. One of the changes has been to add gates such as we find at the exits of most stores. These gates will start to blink when there is a theft alarm or there has been an illegal checkout. We also moved the checked out books wall from outside the library to behind the help desk. Finally we added some chairs where you can sit down with you Second Life avatar and have an overall view on the situation.

This is the new map of the second life world:



These two views which you could have:



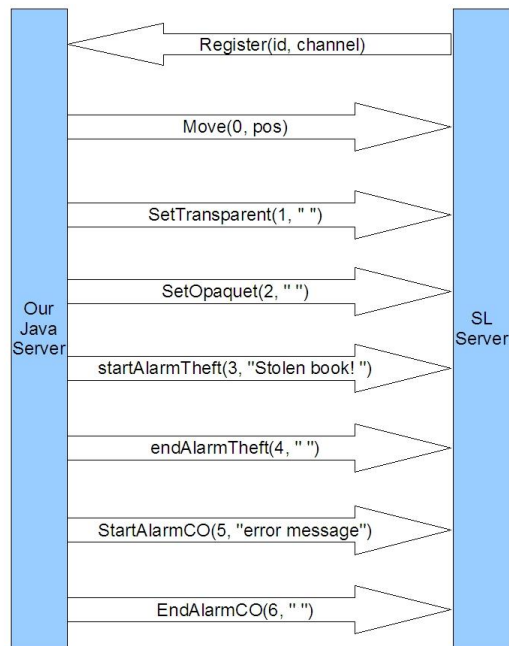


Second Life events

The events still come in with the XML-RPC protocol provided by second life. Sadly enough, this protocol has still a very big delay. Therefore, we still have to have a second interface (see web interface section). We have several types of events which can trigger events in second life:

- set a book transparent and not transparent: when we loose sight of a book it is set slightly transparent without disappearing from the shelf.
- turn on / off a theft alarm: this is the alarm which makes the books and the gates flash red
- turn on / off a checkout alarm: this alarm is triggered when an illegal checkout occurs (for example: too many books are checked out). With this alarm, the books and the gates also flash, but not in red.
- move a book: this allows us to move a book from one position to another. this is used when a book is checked out or checked in, for example.

Here we can see the exact communication protocol between our java server and our second life world books. (the gates are basically the same, but implement just the two alarm functions)



3.2 Web interface

The web interface is used to allow user inputs and queries. It allows users to request for a certain books and displays their location. If a searched books should be on shelf A, but it is located on the shelf B it will be marked as mistakenly placed. Books that are taken out of the library without authorization or that aren't authorized to leave the library need also to be listed. In addition to querying the current status of our system the web interface will also provide a real-time view of all books' status. This is due to the fact that we need to show some events, like alarms, quickly to the end-user. However this cannot be achieve using Second Life due the high propagation delay. The last but not the least functionality is the management of books. As the lower layers use only the rfid tags the visualization layer need to know more on books. For the real-time view of the system we need to have their title, author.... However we also need to store their current location inside SL as spoken in the corresponding section. The user can enter all this information using the management part of the web interface. The web interface communicates with the core of the Visualization Layer using RMI. All requests sent by the user through the web interface are forwarded to the Visualization Layer's core for processing. Real-time books' statuses are shown in the web interface using a Java applet running on the client. This applet is registered inside the VL core and receives every event right the way. In a first release it will only show the books status and description inside a table. However this could be expanded to a more graphical visualization using Java2D.

(to be done: view of the home page of the web interface)

functionalities The web interface functionalities are defined to a strict minimum for this first release. As mentioned in the introduction the user can query the system using pre-defined queries like `;;where is that book; ;` or `;;show me every misplaced books; ;`. In addition to these fixed queries the user has also the possibility to enter it proper SQL query to watch the content of the database.

- Start the live status Java applet

- Search for books using
 - their status
 - their author, title, isbn
- Manage the books
- Execute a SQL-Query directly

Live Status (real-time visualization) The live status is provided with a Java applet that is linked through RMI with the Visualization Core. This liaison ensures that every event received by the VLs core that needs to be shown to the user is forwarded to the applet. The first release of the applet only displays a table with all the books inside the library and their status / location. Later on we could expand it by adding some query functionalities to reduce the amount of books shown inside the table. Another advantage of using the applet technology together with RMI is that we can reuse this code inside a hypothetical SwingGUI. The data flow is driven by the VLs core to the applet and at no time, inside the first release, the applet can decide to get some information.

(to be done: view of the applet)

Search This is one of the most important functionality of the Visualization Layer. It allows the user to look for books inside the library. Queries can be based on different criteria such as the book's title, author or ISBN. The returned information from the VL's core is the list of all books that match the given request with their current status and location. In addition a link to the SL world will be provided to allow users that have SL installed on their computer to see the book inside the Virtual World.

Management of books and persons This is the most important part of the visualization. Without it nothing could be displayed or edited by a regular user for the whole application. It allows users to add/edit/remove books or persons from the system. These entered records will be used by the Query Processing layer to determine whether the rfid tag is owned by a book or a person. The visualization layer will use the description of every book to show a clear description to the user. The interface allows creation, modification and deletion of entries. In the first release there is no login required it means that everybody can step into this part of the web interface and change some values. This needs to be improved for a production application.

Execute raw SQL-Query This will be the first to be implemented inside the web interface. It allows user to paste a MySQL query inside the application and run it against the database. This allows advanced users or administrators to run very specific queries for maintenance of advance data management. The eventual result of the query is shown whether as an HTML table or as CSV raw text.

Technical background The main technologies used for the web interface are listed below:

- Java Enterprise 5
- RMI
- Struts Framework 1.3.8
- MySQL 5.1

3.3 Timetable and further planning

Previous objectives

Date	Description	Status
24.10.2007	Starts a LSL script on a outside events	completed!
24.10.2007	Create a script that move objects around	completed!
24.10.2007	Define the web application functionalities	in progress
31.10.2007	Design the whole visualization layer	in progress
31.10.2007	Find a place where to build	completed!
31.10.2007	Build our virtual library	completed!
07.11.2007	Write the first progress report	completed!
14.11.2007	Implement the SL communication component for VL	completed!
14.11.2007	Finalize the design of the functionalities of the web interface	completed!
14.11.2007	Finalize the whole design of the VL	completed!
21.11.2007	Finalize the communication protocol with the other layers	completed!

Futures objectives

Date	Description	Status
21.11.2007	Begin implementation and testing	in progress
28.11.2007	Write the second progress report	in progress
05.12.2007	Final testing with the other layers	-
05.12.2007	Add / remove / change books through the web interface	-
05.12.2007	Book registers itself with the server	-
12.12.2007	First presentation to restricted public	-
17.12.2007	Second working demo	-

4 Event Processing Layer

For this layer we made progress in different parts. We adapted our event definition language that is now called SRFL (SmartRF Lib Language). The parser can now fully parse the now defined events in the event folder located at trunk.src.qp.parse.events. If the parser fails it throws a ParseException. We had meetings with the other 2 layer groups and defined the communication between the layers. These parts have been implemented so far and are ready. (see in the section communication for descriptions). The TCP connections with the data acquisition has been tested together on 2 different

workstations (client and server) and it seemed to be working. The RMI service of the visualization layer has also been tested. All the provided methods could be called on the remote dedicated server that we use for testing.

Right now we're on constructing the Finite state automata from a given defined element file. This task is very challenging. We managed to detect certain patterns in an event stream. We have extended the finite state automata and the parser and are still doing progress there.

4.1 Finite state automaton

Finite state automaton is used in order to detect the pattern. Since pattern is represented as regular expression in our event language, using finite state automaton to detect it is a natural choice. In our implementation of finite state automaton, it has states, inputs and transitions functions. Besides these, our automaton has condition which is the where clause of the event definition. Automata has dummy starting state and final state as default. Having dummy final state is our design choice in order to get longest match. Our automaton is designed to catch the longest match by ignoring the irrelevant streams. For example, assume input stream is the following: $\{p, a, b, s, c, d, b\}$ where p,s are the people and a,b,c,d are books and pattern is defined person followed by one or many books. Beside the regular expression, also some conditions related to the detected book is defined. An assume that, while book b and c satisfy the condition, book a and d do not. In such a case, finite automaton detects p and b as a match by ignoring the book a, since it does not satisfy the condition and reports the match when it reads the input person s. Reading unexpected type of input is one termination criteria of the automaton. For the rest of the input, it will store person s and books c and b as match but it will not reach final state since it has not read unexpected input yet. In this case after certain time, automaton reports match of $\{s, c, b\}$. So to sum up, in case of longest match setting which is the default setting for the automaton, automaton ignores the irrelevant inputs which do not satisfy the condition and can reach the final state when it reads unexpected input or does not read any input for certain time.

Besides the longest match, incremental match can also be defined for automaton. In case of incremental match, finite automaton does not have dummy final state and it reports the match whenever it reads the expected inputs. If we consider the previous example again, in case of incremental match finite automaton reports the following matches: $\{p, b\}$, $\{s, c\}$, $\{s, c, b\}$ This approach is necessary since at some scenarios reporting the match as soon as possible is crucial like theft detection.

Another issue in pattern recognition in streams is the behavior of the system after match. After a match is detected, our automaton continues to scan the stream beginning with the next input after match. It is also possible to let user to define the beginning input of scan operation like [DIG07], but by considering the time limit our system starts scanning with the next input after match.

4.2 Stream Data and Hybrid Queries

In our library application, there is a need to access the stored data for detecting events. For example while detecting checkout event, we have to check whether the number of books that borrowed by the user exceeds his/her limit. Beside stored data, streams which are Book and Person in our application also have fields like tag id, reader id and time stamps. In our system we are providing both stored data access and stream data access. Access to stored data is provided by user defined functions. Beside some functions we provided, user can also define new functions in order to provide stored data access. So far, we can run hybrid queries, by sending stream data as argument.

In our system, we are differentiating access to stream data and stored data syntactically. While using `.` notation to access stream data like `b.tagId`, function call is used to access stored data like `borrowedBy(book, person)`.

4.3 Assumptions

During our implementations we made the following assumptions:

ordering of SendObjects We assume that the data we get from the data acquisition layer is ordered by time stamp. There may be different SendObjects with the same time stamp, but when we get a SendObject with time stamp 3 then we won't get a SendObject with time stamp 1 in future.

filtering and compression We assume that the data coming from the lower layer is compressed and filtered. This means that we don't expect it to have duplicates. It should be smoothed to interpolate detection misses. Some adaptive time windows are placed in order to get meaningful data.

different data modes For the query processing layer we need different modes of data transmission from the data acquisition layer. For example from the exit reader we immediately want a response if a tag is detected. And only sent once. For the shelf for example we want this stream to give us the data every 5 minutes. The checkout reader should give us all the measured tags but shouldn't send duplicates, so it waits until the tags are removed.

match order Our finite state automaton matches the incoming data stream always with the same order. First comes the 'Person' if any then it matches the 'Books'. We assume that we can get the stream in this order from our data Queue. The Queue receives the SendObjects from the Client and orders them in a such way.

SendObject types We assume that in our library we can only have the type 'Person' or 'Book'. Our finite state automaton does not accept other types to process.

Hybrid queries Our query processing engine does not support hybrid queries so far. New event queries can be written in SRFLL and added to the system. To do so we have predefined user functions. Hybrid query would mess up our parser since the return value of such database accesses could be of any type. Basically we have 3 different types in the expression tree evaluation: boolean, numeric, set. It may be possible to add a function to support SQL queries over a separate interface, maybe they can be placed from the visualization layer.

SRFLL All the events are correctly and clearly defined in SRFLL. If this is not the case there will be errors while parsing.

reader grouping The events can be grouped by reader. This means that we will in our library scenario have for every reader a data stream, means 3 TCP connections with the lower layer. For every connection exists a queue that has the items for the corresponding events that can also be grouped by reader.

1 input stream So far our query processing engine assumes that an event can only have input from 1 stream. This means for example that an event only takes data from the exit reader.

real-time We assume that we can process our incoming data streams more or less in real-time. Our queues are not meant to explode. Which means that we assume that the processing engine can deal with the incoming amount of data.

Complex events So far it's not possible to build complex events.

We rely on these assumptions to make our life easier and to develop in our layer. Violating some of these assumptions may cause errors in our event processing layer.

4.4 Future directions

So far, we implemented function call with only stream arguments and boolean return values. We are going to implement also different return values number, string and set. Because of time limit, we might not succeed to implement set operations.

So far, we are not providing operations on kleene closure streams like indexing or aggregate function, we are planning to implement kleene closure by extending our symbol table in a way to store index values as well. Because of time limit, we might not succeed to implement aggregate operations on kleene closure.

So far, our system has not provide time windows yet, we are planning to use java timers to implement this functionality.

We are planning to implement the new keywords in our language, MATCH LONGEST and MATCH INCREMENTAL

So far, we tried our system with one input stream and tried to detect one event, which means one finite state automaton. Our final system will work with three input streams, one for each reader and multiple event detection. We are planning to detect these events concurrently.

We have to connect all the components together to get a working system.

The order of the incoming objects from the data streams has to be guaranteed by putting them in a correct order into the SendObjectQueue.

Optimizations

We believe that there are a lot of optimization opportunities in our system, but some of them are like the following.

Because of ease of implementation, we are evaluating the conditions of a event detection, where clause, at each matching state. Actually, conditions can be separated and related ones can be attached to each states. In this way, the implementation can be speed up.

4.5 Some adaptations in our EBNF

SRFLL - EBNF

In the meantime we adapted SRFLL basically in 2 points. The actionDecl that has been suggested in the last progress report has been added. It's optional and can call user defined functions.

We suggest here another keyword 'MATCH' that specifies the way how our finite state automaton matches the patterns (see section finite state automaton). This is not fully implemented yet.

```

eventDecl ::= NAME string onDecl PATTERN patternDecl matchDecl
           WHERE whereDecl ACTION actionDecl RETURN returnDecl

matchDecl ::= | MATCH LONGEST

           | MATCH INCREMENTAL
onDecl    ::= | ON IDENTIFIER:i
returnDecl ::= boolExpr
actionDecl ::= functionCall | functionCall , actionDecl

patternDecl ::= SEQ [ qItemList ]
qItemList  ::= qItem | qItem , qItemList

qItem      ::= string string | string+ string []
           | string* string []
whereDecl  ::= expr | expr AND whereDecl
expr       ::= numExpr | setExpr
           | boolExpr | NOT boolExpr
numExpr    ::= numTerm = numTerm | numTerm <> numTerm
           | numTerm < numTerm | numTerm <= numTerm
           | numTerm > numTerm | numTerm >= numTerm
setExpr    ::= setItem IN setTerm | setItem NOT IN setTerm
           | numTerm = ANY setTerm | numTerm = SOME setTerm
           | numTerm = ALL setTerm | numTerm <> ANY setTerm
           | numTerm <> SOME setTerm | numTerm <> ALL setTerm
           | numTerm < ANY setTerm | numTerm < SOME setTerm
           | numTerm < ALL setTerm | numTerm <= ANY setTerm
           | numTerm <= SOME setTerm | numTerm <= ALL setTerm
           | numTerm > ANY setTerm | numTerm > SOME setTerm
           | numTerm > ALL setTerm | numTerm >= ANY setTerm
           | numTerm >= SOME setTerm | numTerm >= ALL setTerm

setItem    ::= functionCall | number | bool | string
setTerm    ::= functionCall
boolExpr   ::= boolExpr OR boolExpr | boolExpr AND boolExpr
           | functionCall
numTerm    ::= numTerm + numTerm | numTerm - numTerm
           | numTerm * numTerm | numTerm / numTerm
           | functionCall | number
functionCall ::= string.string() | string.string (arglist)
           | string[number].string() | string[number].string(arglist)
           | string (arglist)
arglist    ::= string | string , arglist
number     ::= integer | real
integer    ::= digit | digit,digit
real       ::= integer . digit
digit      ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
string     ::= letter | letter letter | letter digit
letter     ::= A .. Z | a .. z

```

SRFLL - Events

Not borrowable: User wants to take book which has to be used in library only

```
NAME not_borrowable ON R1
PATTERN SEQ(Book+ b[ ])
WHERE NOT borrowable(b[i])
ACTION alarm(), b[].id
RETURN True
```

Too much books: User wants to take books more than defined book limit

```
NAME too_much_book ON R1
PATTERN SEQ(Person p, Book+ b[ ])
WHERE getBookLimit(p.id) < countBorrowedBooks(p) + b[].size
ACTION alarm(), p.id, getBookLimit(p.id)
RETURN True
```

Book theft: A book that is not yet borrowed leaves the library

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

Checkout of book: A normal event when a user borrows a book

```
NAME book_checkout ON R1
PATTERN SEQ(Person p, Book+ b[ ])
MATCH LONGEST
WHERE NOT not_borrowable(b) AND NOT too_much_book(p,b)
ACTION checkout(p,b)
RETURN True
```

Return of book: A user returns his borrowed books

```
NAME book_checkin ON R1
PATTERN SEQ(Person p, Book+ b[ ])
MATCH LONGEST
WHERE isBorrowedBy(b,p)
ACTION checkin(p,b)
RETURN True
```

4.6 Time table update

Note: many of our task are continous developped further. Mostly we started simple and then enhanced the constructs or concepts from week to week.

date	description	status
17.10.07	Proposal Report	done
24.10.07	database sample fake data db setup sase event parsing	done done done
31.10.07	database testing start SASE implementation design read many papers	in progress parser and lexer done done
07.11.07	progress report I many other things	done done
11.11.07	added 'ACTION' to SRFL clearer design of SRFL started with parser extension definition of finite state automata	done done done done
14.11.07	finite state automata expression tree	in progress in progress
21.11.07	database sample fake data db connections extended, junit testing started finite state automata expression tree	done in progress in progress in progress
28.11.07	progress report II communication with DAL TCP communication with VIZL RMI extended SRFL	done done done adding 'MATCH' keyword
05.12.07	proper interfaces db merge setup extend finite state automata compose whole system	already done - - -
12.12.07	layer testing communication testing bugfixing	- - -
17.12.07	working demo report ?	- -

5 Data Acquisition layer

5.1 Adaptive Cleaning

Sensor devices provide raw data about the physical world. This raw data cannot be directly used by application because they have lots of noise therefore they are unreliable. However most of the application requires reliable data. For this reason sensor data should be cleaned before incorporating them into application [JGF06]. Cleaning raw data produced by sensor requires the knowledge of the environment, knowledge of the device and data produced by the sensor. Therefore converting sensor data into reliable application data has significant complexities. A new layer of metaphysical data independence (MDI) is proposed in [JFG07] to cope with these complexities. MDI separates application and physical sensing device. Application layer and sensing device layer is not separated in the current sensing device. Any errors in current sensing device need to be handled by the application therefore these applications are complex, brittle and difficult to change due to their dependence on the physical sensor device.

To handle these problems, physical sensing device and application layer is separated as in MDI to hide the complexity of sensing device from application [JFG07]. An MDI based cleaning mechanism is demonstrated in [JFG07, JFG07]. Smoothing window is the standard data cleaning mechanism for today's RFID systems. The idea in the smoothing window is to give more time to read for a tag in order to avoid missed readings. The size of the smoothing window depends on the character of the environment and the configuration of the application [JFG07]. Therefore window size needs to be changed over time. For example in library environment, some tagged books maybe on the shelves (static) while the others are moving as in the checkout. Smoothing window size for these two different tags are different due to different tag dynamics. But setting the window size is not an easy task. Window size should balance two opposing application requirement such as ensuring the completeness which means that capturing all the tags in the vicinity of the reader and tag dynamics which means that detecting the tag movement within the window. In [JFG07, JFG07] statistical properties of data is used to adjust the size of the window automatically instead of application is setting the window size. This kind of behavior made it possible to hide the window size from the application as mentioned in MDI.

Readers communicate with the tags by sending an RF signal and wait for tags response. Tags in the vicinity of the reader area respond to these signals with their unique identifier code. One iteration of the readers for determining all the tags in the readers vicinity is called interrogation cycle. Readers interrogation cycles are grouped into epoch. An epoch is basically a number of interrogation cycles. Readers keep track of all identified tags' number of interrogation responses, last read time and tagid for each epoch in a taglist.

Assume that window size W_i has w_i epochs. Each epoch is seen as independent Bernouli trial with the success probability p_i . Number of successful observation of tag in the window is random variable with a binomial distribution. Assume that tag i is seen only subset S_i of all epoch in W_i . Average probability p_{avg} in the observation epoch can be calculated as $p_{avg} = \frac{p_i}{|S_i|}$ where p_i is the success probability. P_i can be calculated following. Assume that reader has a configuration with a total number of 10 interrogation cycles per epoch. And assume that number of interrogation responses of the tag is 9. Then p_i is basically $\frac{9}{10}$ which is 0.9. Expected value and variance of the seen epoch in W_i can be calculated as:

$$E[|S_i|] = w_i \cdot p_{avg}$$

$Var[|S_i|] = w_i p_{avg} (1 - p_{avg})$ Window size should have enough epochs to ensure that all tag in the vicinity is seen by the reader. According to a lemma in [JFG07, JFG07] setting the smoothing window to be $w_i \frac{\ln(\frac{1}{\delta})}{p_i}$ ensures that tag i is observed within W_i with probability $> 1 - \delta$ Setting δ to small value such as 0.1 guarantees the completeness with high probability 0.99. However this can miss the tag movements. In order to avoid false positiveness, window size should be decreased in the process of transition detection. A binomial sampling model is employed in [JFG07, JFG07] to detect the transitions. From the central limit theorem it is assumed that no transition occurs when the the value of $|S_i|$ is within $\pm 2Var[|S_i|]$ of its expectation with probability close to 0.98. According to this observation, transition is assumed to occur when the number of observed reading is less than the expected number of readings and the following condition is true. $||S_i| - w_i p_{avg}| > 2\sqrt{w_i p_{avg} (1 - p_{avg})}$

5.2 Adaptive Cleaning Algorithm

Algorithm starts with an initial window size of one epoch for each tag and then adjust the size dynamically based on readings. During each new epoch algorithm starts processing the readings and it determines the number of epoch to ensure completeness. If required window size is more than current window size, it incrementally increases

the window size. If the current window size larger than complete window size and if detect transition is accoured then the algorithm decreases the current window size by dividng with 2.

```

Enumeration seenTagIDs = tagWindowData.keys(); //All tag IDs
    seen so far
while( seenTagIDs.hasMoreElements() ){
    String tempID = (String)seenTagIDs.nextElement();
    TagWindowData wData = tagWindowData.get( tempID );
    //If the tag has reading in the this epoch than
    //put this data as PerEpochData else put PerEpochData with
    //minus prob.
    //probEpoch : sampling probability inside this epoch
    double probEpoch = -1.0;
    //Also put the timestamp of the reading if any
    long renewTime = 0;
    if( tagsInThisEpoch.containsKey( tempID ) ){
        Tag tempTag = tagsInThisEpoch.get( tempID );
        renewTime = tempTag.getRenewTime(); //Maybe we should
        //user getHostRenewTime()
        probEpoch = (double)tempTag.getRenewCount() /
            SetupReaders.CYCLES;
    }
    PerEpochTagData epochData = new PerEpochTagData( probEpoch,
        renewTime );
    //In any case tags window will be slided and this epoch
    //will be included in this window
    wData.addToWindow( epochData );
    wData.processWindow( comprLayer, visualizer );
    int completeWindowSize = calculateCompleteSize( wData.
        getAvgSamplingProb() );
    int currentWindowSize = wData.getTagWindowSize();
    int newWindowSize;
    //If the tag has not been seen in any of the epochs of this
    //window then
    //reset window size to 1 epoch
    if( wData.getCountTagSeen() == 0 ){
        newWindowSize = 1;
        wData.setTagWindowSize( 0, 1 ); //TODO check if it is
        //working
        visualizer.addOrUpdateTag(tempID, 1, true); //Set window
        //size to 1
    }
    else if( completeWindowSize > currentWindowSize ){
        newWindowSize = Math.max( Math.min( currentWindowSize
            +1, completeWindowSize ), 1 );
        if( newWindowSize != currentWindowSize+1 )
            System.err.println("Window Size Increment
                Assumption Wrong!!!");
        wData.setTagWindowSize( newWindowSize, 0 );
        visualizer.addOrUpdateTag(tempID, newWindowSize, true);
    }
    else if( detectTransition( wData.getCountTagSeen(),
        currentWindowSize, wData.getAvgSamplingProb() )){
        newWindowSize = Math.max( Math.min( currentWindowSize /
            2, completeWindowSize ), 1 );
        wData.setTagWindowSize( newWindowSize, 1 );
        visualizer.addOrUpdateTag(tempID, newWindowSize, true);
    }
}

```

```
    }  
    else  
        wData.setTagWindowSize( currentWindowSize, 2 );
```

5.3 Interface to upper layer

Cleaning of dirty data in the data acquisition layer will be done with using adaptive method. After our data is cleaned, it will be send to upper layer using an interface to the upper layer. In order to provide an interface to the upper layer to access the data we willgenerate primitive event tuples to the upper layer which are more reliable than the raw readings. A client and server programm is implemented to make the connection with the upper layer. In this client and server system, upper layer is thought at server, capture layer which gets the reading and makes the cleaning is thought to be client. Object streams over TCP/IP sockets for upper layer is created to access the data. Time, ObjectID and ReaderID ordered by time and grouped by ReaderIDobject streams is provided to the upper layer.

5.4 Reading Tests

To have sense of how RFID readers are performing with distance we made some test using the Alien ALR-9880. Our experimental methodolgy is as follows. We put 5 tagged books in the same plane as the antennas at varying distances. We measure the success probability p_i for 500 epochs for the 10 cm, 100 cm and 150 cm distances. In the plots we set the succes probability to -1 when the reader cannot read the tagged book.

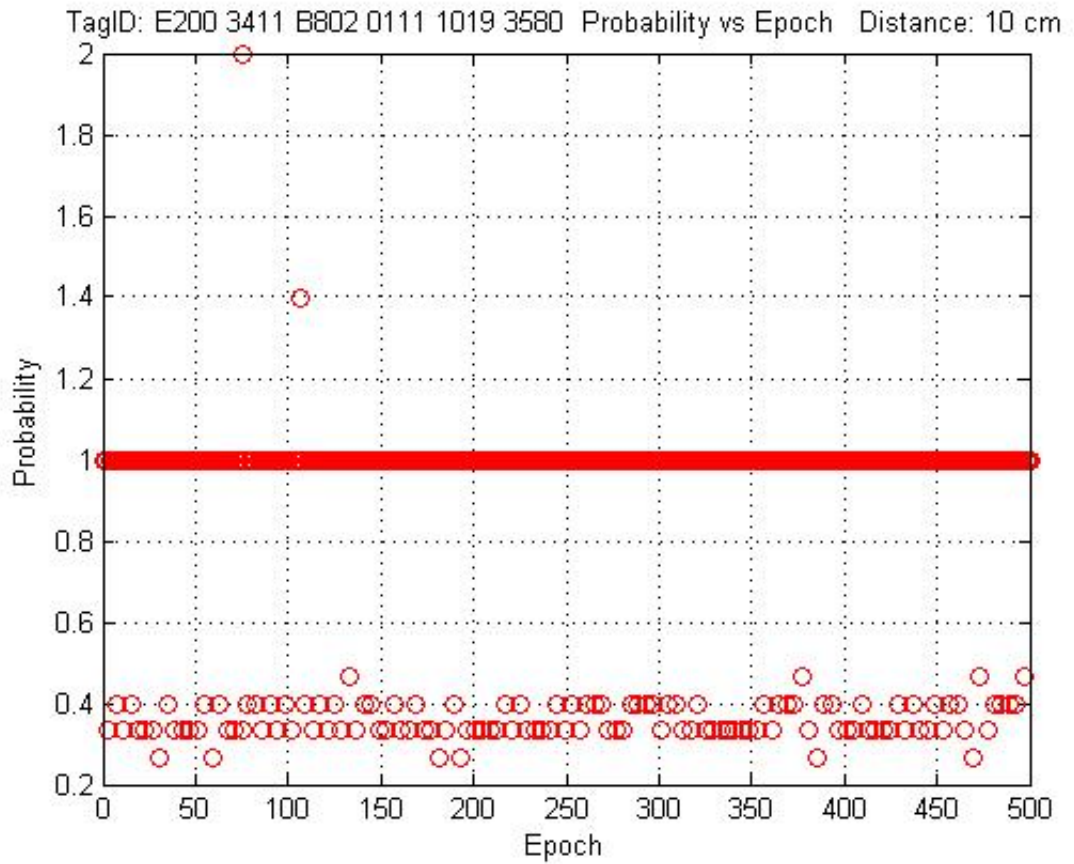


Figure 1 depicts the result from 10 cm distance. We saw that the reader read tagged book for all epochs. For two of the epoch we saw that probability value is bigger than 1 which can explain that reader is reader a longer epoch duration than the one we set. We can ignore these measurements as being noise values because they only 2 measurements in 500 measurements.

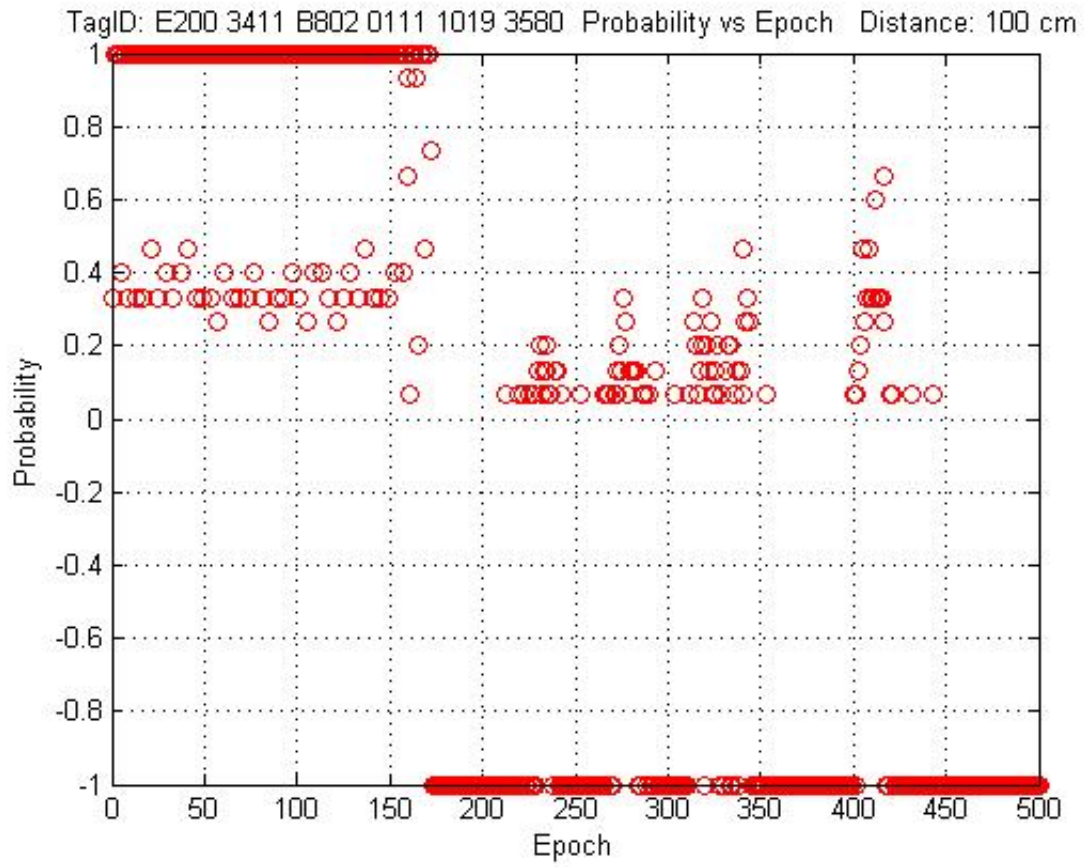


Figure 2 depicts the result from 100 cm distance. We say that reader read tagged book less than half of the all epochs. This can be explain as the decrease of read rate with increasing distance.

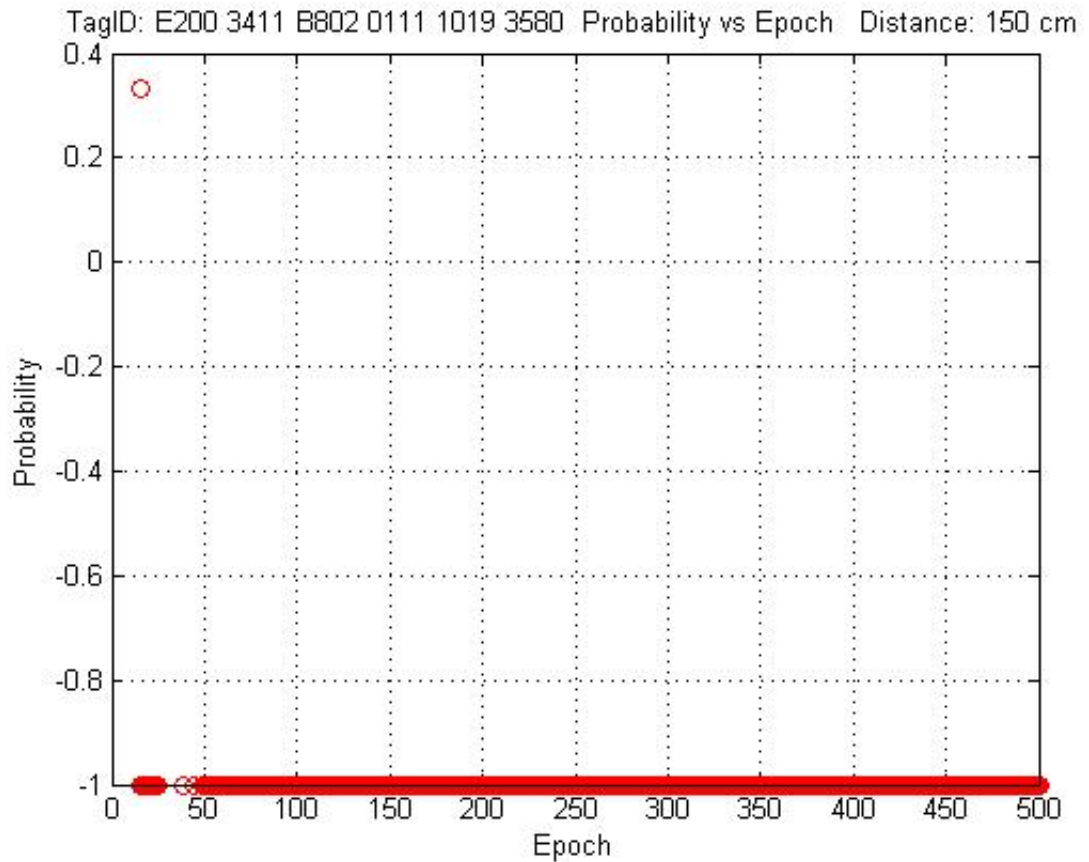


Figure 3 depict the reading results from 150 cm distance. We saw that reader cannot read the tagged books for all the epochs.

As we saw from the above figures, read rate of the reader decreases with the increasing distance.

6 Conclusions

We now have reached a state where we have a program which can pass information from the bottom layer to the top layer. We can now assume that the communication protocol won't change anymore, or just with slight adjustments. The next steps will be to complete each layer so that the program becomes really usable. The visualization layer still needs to complete the web interface as well as the input of new books and persons. The database also has to be put in common between the two top layers. The next step of the query processing layer will be to implement multiple input streams, add some more keywords to differentiate the pattern matches, complete the function calls and extend the functionality of the finite automata. Finally, the query processing layer has to connect all it's components and the layers to ensure proper communication. The next step will already be an unofficial demo and at that stage we will have a more or less completely functional program.

Bibliography

- [BDG⁺07] Lars Brenna, Alan Demers, Johannes Gehrke, Mingsheng Hong, Joel Osher, Biswanath Panda, Mirek Riedewald, Mohit Thatte, and Walker White. Cayuga: a high-performance event processing engine. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1100–1102, New York, NY, USA, 2007. ACM.
- [CEF⁺04] Owen Cooper, Anil Edakkunni, Michael J. Franklin, Wei Hong, Shawn R. Jeffery, Sailesh Krishnamurthy, Fredrick Reiss, Shariq Rizvi, and Eugene Wu. Hifi: a unified architecture for high fan-in systems. In *vldb'2004: Proceedings of the Thirtieth international conference on Very large data bases*, pages 1357–1360. VLDB Endowment, 2004.
- [CKRS04] Sudarshan S. Chawathe, Venkat Krishnamurthy, Sridhar Ramachandran, and Sanjay Sarma. Managing rfid data. In *vldb'2004: Proceedings of the Thirtieth international conference on Very large data bases*, pages 1189–1195. VLDB Endowment, 2004.
- [DCS07] Yanlei Diao, Richard Cocci, and Prashent Shenoy. Spire: Scalable processing of rfid event streams. 2007.
- [DIG07] Yanlei Diao, Neil Immerman, and Daniel Gyllstrom. Sase+: An agile language for kleene closure over event streams. 2007.
- [GWC⁺07] Daniel Gyllstrom, Eugene Wu, Hee-Jin Chae, Yanlei Diao, Patrick Stahlberg, and Gordon Anderson. Sase: Complex event processing over streams. 2007.
- [JFG07] Shawn R. Jeffery, Michael J. Franklin, and Minos Garofalakis. An adaptive rfid middleware for supporting metaphysical data independence. 2007.
- [JGF06] Shawn R. Jeffery, Minos Garofalakis, and Michael J. Franklin. Adaptive cleaning for rfid data streams. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 163–174. VLDB Endowment, 2006.
- [Pal04] M. Palmer. Seven principles of effective rfid data management. 2004.
- [RJK⁺05] Shariq Rizvi, Shawn R. Jeffery, Sailesh Krishnamurthy, Michael J. Franklin, Nathan Burkhart, Anil Edakkunni, and Linus Liang. Events on the edge. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 885–887, New York, NY, USA, 2005. ACM.
- [WBBB07] Evan Welbourne, Magdalena Balazinska, Gaetano Borriello, and Waylon Brunette. Challenges for pervasive rfid-based infrastructures. In *PERCOMW '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 388–394, Washington, DC, USA, 2007. IEEE Computer Society.

- [WDR06] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 407–418, New York, NY, USA, 2006. ACM.
- [WL05] Fusheng Wang and Peiya Liu. Temporal management of rfid data. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 1128–1139. VLDB Endowment, 2005.