Event Processing Support for Cross-Reality Environments

Complex event processing (CEP) is an essential functionality for cross-reality environments. The DejaVu event processing system supports SmartRFLib, a cross-reality application that enables real-time event detection over RFID data streams feeding a virtual library on Second Life.

Nihal Dindar, Çağrı Balkesen, Katinka Kromwijk, and Nesime Tatbul
ETH Zurich

Recent developments in sensor network technologies have paved the way for the widespread use of small, embedded sensing devices in many pervasive application domains. Such sensor networks help integrate physical reality into virtual computing platforms, letting us sense and react to real-world events in an automated fashion. Meanwhile, another class of Web-based computing platforms has recently emerged that enables virtual simulations of the real world over which people can interact online. Cross-reality environments could serve as a bridge across sensor networks and Web-based virtual worlds, improving people's interactions with each other and the physical world. This capability could help many application areas achieve computational pervasiveness.

Sensor networks generate data that often indicate certain complex phenomena in the real world. We can detect these phenomena only by applying higher-level inference techniques to collected sensor data. *Complex event processing* (CEP) systems address this need by enabling pattern-matching queries over primitive sensor events to detect more complex events that carry higher semantic value to the application.[1–3] Common examples include fire in a building, a burglar in a house, theft in a library, or shoplifting in a grocery store.

We see CEP as an essential functionality for cross-reality applications. Through it, we can turn raw sensor data generated in the real world into meaningful information more suitable for representation in the virtual world. For example, rather than simply showing what value each temperature or smoke sensor is reporting, we can show more interesting events, such as fire. Such information is more valuable for users and is certainly easier and more realistic to represent in a virtual world. As such, a CEP system can act both as a transformer and a filter between potentially high-volume raw sensor readings from the physical world and their rich virtual representation: while simple events get transformed into more complex ones, low-level events that would otherwise be regarded as noise or clutter in the virtual world get filtered out.

Cross-reality systems face several challenges, including collecting accurate data from the real world, turning collected data into more meaningful events, and representing and updating events in the virtual world in real time. Furthermore, these systems must provide high performance and scalability to ensure that we can link events across the two environments with the lowest possible latency.

To address these issues, ETH Zurich developed the DejaVu system,[4] an event stream processing system that extends other CEP systems to provide declarative pattern matching over live and archived event streams. We then built the

Published by the IEEE CS ■ 1536-1268/09/$25.00 © 2009 IEEE

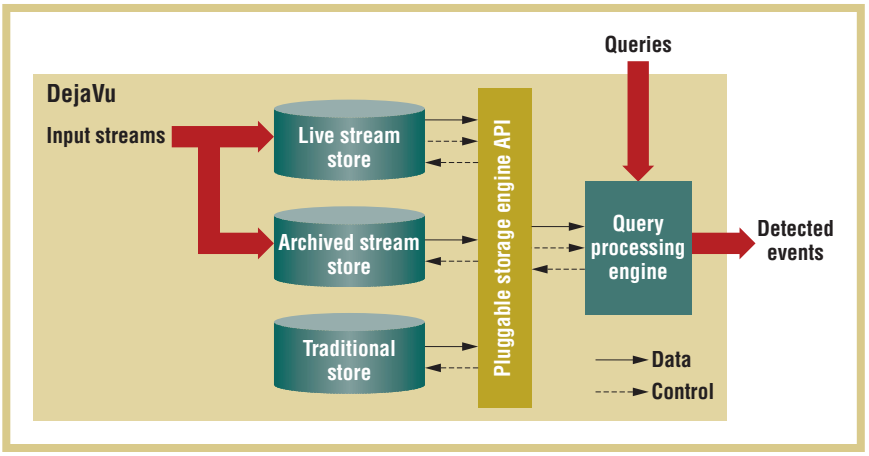


Figure 1. DejaVu system architecture. This architecture builds on and extends the MySQL relational database engine architecture with event pattern matching queries over both live and archived streams of events stored in separate storage engines.

SmartRFLib application around DejaVu to illustrate our event processing system's main features and demonstrate how event processing support can be useful for cross-reality applications.

The DejaVu System

DejaVu targets a broad range of applications, from sensor-based real-time monitoring to financial market data analysis. One of its primary goals is to enable seamless integration of complex event pattern-matching functionality over live and stored event sequences behind a common declarative query interface. This not only enables arbitrary event correlations within or across different time domains but can also help predict future events because an important class of interesting event patterns are those that repeat in time, creating a sense of *deja vu*.

Architecture

With DejaVu, our goal was to develop a coherent system that unifies interface and functionality but lets us customize implementation and optimization methods across live and historical data sequences. To realize this goal, our approach builds on and significantly extends a relational database engine architecture. More specifically, we built DejaVu on top of the MySQL open source database system.[5] As such, we follow MySQL's basic architectural skeleton while making new extensions to support different forms of pattern-matching queries (one-time, continuous, and hybrid) as necessary.

Figure 1 illustrates DejaVu's high-level architecture. A key MySQL feature we exploit in our system design is its pluggable storage engine API. Through this API, we can define and plug in custom storage engines to work seamlessly with the MySQL query-processing engine. In addition to the traditional relational store MySQL provides, which we occasionally need for table lookups, we also created two new storage engines:

- *Live stream storage*. This is an in-memory store that accepts push-based inputs. It essentially acts like a queue, feeding live events into the query-processing engine as they arrive. The query processor can access the live stream store either in pull mode (that is, MySQL-style) or push mode. We added the latter option in our design to enable an adaptive switch between these two modes and to flexibly cope with unpredictable bursts in input load inside the storage engine. In other words, under high load, we switch to pull mode and let the storage handle the problem.
- *Archived stream storage*. DejaVu can also fully or selectively materialize live input events into an archive store, which respects the events' predefined order and allows updates only in the form of appends. It also provides features such as data compression and efficient access methods for historical pattern-matching queries. Furthermore, because the archive is a persistent store, it can support the live store in dealing with bursts and failures.

In addition to creating two new storage engines, we also made an important extension to the MySQL query processor. More specifically, we introduced a finite state machine (FSM) implementation to drive the evaluation of the pattern-matching queries; the FSM runs as an integral part of the MySQL query plan. We present an example FSM later in the article.

As with most relational database systems, MySQL allows only one-time queries. So, we added new mechanisms into the MySQL query-processing engine for query life-cycle management and continuous result reporting for long-running queries over streaming data.

A key design decision in DejaVu was to build the pattern-matching engine on top of MySQL mainly so that we could exploit MySQL's extensible storage engine API for the two different stream stores that we wanted to create. An additional advantage was that extending MySQL's parsing and execution engine was rather convenient and let us implement the pattern-matching extensions in our SQL-based CEP language.

Language

Any event processing system needs a powerful language for expressing events. In DejaVu, we express complex event patterns through an SQL-like declarative query language. This language is based on a recent language standard proposal for pattern matching on row

Related Work in Event Processing and Virtual Worlds

The work we describe in the main text builds on and extends previous work on sensor data acquisition, data stream and complex event processing, and sensor data integration into online virtual worlds.

Earlier sensor querying systems such as TinyDB[1] focused on energy-efficient data acquisition and query-processing techniques over sensor networks, such as in-network aggregation. Model-driven data-acquisition systems such as BBQ[2] used learning-based probabilistic models to improve efficiency further by exploiting temporal and spatial correlations across sensor readings. More recently, researchers have proposed statistical techniques for adaptive cleaning of RFID data.[3,4] Our SmartRFLib application directly implements the data-cleaning algorithms this work proposes. As we discuss in the main text, we've found that this approach has some limitations in the compression step of RFID data acquisition, for which we devise a flexible solution.

A body of previous work also exists on data stream and complex event processing systems that closely relates to our DejaVu system. First of all, DejaVu is a stream-processing system and, as such, builds on the fundamental concepts that data stream management systems such as Aurora,[5] the Stanford Stream Data Manager (Stream),[6] and TelegraphCQ[7] propose. DejaVu differs from these systems in that in addition to managing live streams, it can also store and process archived streams. Additionally, its architecture extends a relational database architecture, and it focuses on pattern-matching queries for complex event processing. DejaVu is closer in spirit to complex-event processing (CEP) systems such as HiFi,[8] Stream-based and Shared Event Processing (SASE),[9] and Cayuga.[10] Its fundamental difference from these systems is its support for integrated query processing over live and archived event streams. (The HiFi project demonstrated a similar library scenario in earlier work, but its main focus was on the RFID event acquisition problem, and it used a different architecture and a more restricted event language.[11])

Finally, SmartRFLib isn't the first cross-reality application to integrate sensor data into Second Life. The Plug ubiquitous networked sensing platform developed at the MIT Media Lab provides a virtual representation within Second Life for browsing sensor data streaming from the Plug network,[12] called the Shadow Lab. Its users can interact with the Plug network through the Second Life interface. As another example, Emiliano Musolesi and his colleagues have proposed the CenceMe platform, a personal sensing system that lets social network members share presence and activity information via mobile phones.[13] The sensor readings from mobile phones that reflect people's activities are virtually represented in Second Life. This work targets a broad set of problems from activity recognition and visualization to intermittent cell phone connectivity and privacy issues.

REFERENCES

1. S. Madden et al., "The Design of an Acquisitional Query Processor for Sensor Networks," *Proc. ACM SIGMOD Conf.*, ACM Press, 2003, pp. 491–502.
2. A. Deshpande et al., "Model-Driven Data Acquisition in Sensor Networks," *Proc. 30th Int'l Conf. Very Large Databases Conf.*, Morgan Kaufmann, 2004. pp. 588–599.
3. S.R. Jeffery, M. Garofalakis, and M.J. Franklin, "Adaptive Cleaning for RFID Data Streams," *Proc. 32nd Int'l Conf. Very Large Databases Conf.*, ACM Press, 2006, pp. 163–174.
4. S.R. Jeffery, M.J. Franklin, and M.N. Garofalakis, "An Adaptive RFID Middleware for Supporting Metaphysical Data Independence," *Very Large Databases J.*, vol. 17, no. 2, 2008.
5. D. Abadi et al., "Aurora: A New Model and Architecture for Data Stream Management," *Very Large Databases J.*, special issue on best papers of VLDB 2002, vol. 12, no. 2, 2003.
6. A. Arasu, S. Babu, and J. Widom, "The CQL Continuous Query Language: Semantic Foundations and Query Execution," *Very Large Databases J.*, vol. 15, no. 2, 2006.
7. S. Chandrasekaran et al., "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World," *Proc. Conf. Innovative Data Systems Research* (CIDR 03), 2003.
8. M.J. Franklin et al., "Design Considerations for High Fan-In Systems: The HiFi Approach," *Proc. Conf. Innovative Data Systems Research* (CIDR 05), 2005.
9. D. Gyllstrom et al., "SASE: Complex Event Processing over Streams (Demo)," *Proc. Conf. Innovative Data Systems Research* (CIDR 07), 2007, pp. 407–411.
10. A. Demers et al., "A General Purpose Event Monitoring System," *Proc. Conf. Innovative Data Systems Research* (CIDR 07), 2007, pp. 412–422.
11. S. Rizvi et al., "Events on the Edge (Demo)," *Proc. ACM SIGMOD Conf.*, ACM Press, 2005, pp. 885–887.
12. J. Lifton et al., "A Platform for Ubiquitous Sensor Deployment in Occupational and Domestic Environments," *Proc. 6th Int'l Conf. Information Procesing in Sensor Networks* (ISPN 07), ACM Press, 2007, pp. 119–127.
13. M. Musolesi et al., "The Second Life of a Sensor: Integrating Real-world Experience in Virtual Worlds using Mobile Phones," *Proc. Workshop on Embedded Networked Sensors* (HOTEmNets 08), ACM Press, 2008.

sequences in relational tables using a new MATCH RECOGNIZE clause.[6] In the standard SQL syntax, this clause follows a table name in the FROM part and enables us to match the specified pattern on that table. Thus, the original language proposal assumes that pattern-matching queries will be applied over contiguous rows in a given relational table. In DejaVu, we've extended the MySQL language parser to follow a similar syntax, but we also allow users to attach the MATCH RECOGNIZE clause to both archived as well as live stream tables. In fact, we interpret that in the latter case, the MATCH RECOGNIZE clause defines a "semantic window" over the live stream. This interpretation is in perfect agreement with the traditional SQL syntax for time and count-based windows (see www.streamsql.org).[7] We illustrate the MATCH RECOGNIZE clause's details in the following section.

SmartRFLib on Second Life

Consider a typical library with many books and registered users. In an RFID-based scenario, all books and library users would need passive RFID tags that could uniquely identify them. Additionally, multiple RFID reader devices would be located at key points in the library, each with a well-defined role. Readers placed at the library exit, for example, would keep track of books and people leaving the library, whereas readers installed near each bookshelf would monitor automatic book check-ins. Through continuous reports from these readers, we could track each book object's current location as users removed it from its designated shelf. More important, we could automatically detect important library events such as book check-ins, check-outs, illegal check-outs (such as reference books or quota violations), and thefts. We'd want to detect these events in real time so that library personnel could take any necessary action immediately.

We developed our SmartRFLib application to perform a similar function as in the scenario we just described. We implemented SmartRFLib on the DejaVu prototype system and ran it on a simplified library setup that we physically created in our ETH lab space. In this setup, we installed three Alien ALR 8800 RFID readers (representing shelf, checkout station, and exit) in distant corners of the lab room. We tagged several of our existing textbooks with RFID labels (Electronic Product Code Class 1). We also labeled several name tags to represent library users' ID cards. We demonstrated SmartRFLib to a few different audiences using this lab setup, and we'll present it as a use-case scenario for DejaVu at an interactive demonstration session at the upcoming Special Interest Group on Management of Data (SIGMOD) conference.[4] In the meantime, we've initiated contact with ETH Zurich Library Services, which recently started using an RFID-based self-checkout setup in a few department libraries (for example, the "Green Library" in the Environmental Sciences Department; www.ethbib.ethz.ch/dez/gruen_e.html). In the future, we're hoping to collaborate more closely with them to improve our application and install it in a real library setup.

This RFID-based sensing infrastructure helps us track objects' physical locations, but we also want to be able to view them in the virtual world. Thus, we built a virtual representation of our library in Second Life (see www.secondlife.com), a 3D virtual world on the Web that's entirely user-created. It's organized into islands further divided into parcels, which users (represented by avatars) can buy and modify. As Figure 2 shows, the SmartRFLib virtual library resides at the ETH Island in Second Life. Through this virtual library, we can visualize the detected library events in real time. As such, SmartRFLib connects the real-world events DejaVu detects using real-time readings from an RFID sensor network with their representations in Second Life. Through this real-time visualization interface, library management or building security personnel can monitor important library events more easily. Furthermore, once this real-world virtual connection is established, users could reflect virtual happenings into the real world in different forms, such as notification messages, database updates, or even physical device actuations, as we discuss later.

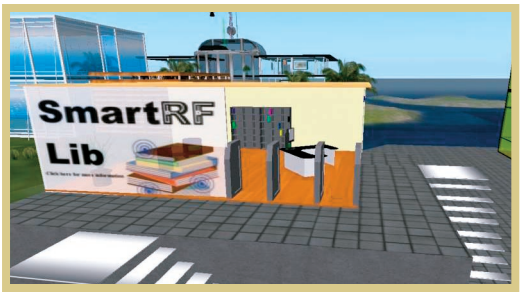


Figure 2. SmartRFLib on Second Life. We built this virtual library on the ETH Island in Second Life to visualize important library events as they happen in the real world.

The entire SmartRFLib implementation follows a three-tier system architecture, similar to the one in Figure 3 (although the arrows in the figure run only from the physical world toward the virtual one, the reverse is also possible, though we focus primarily on the former in this article). At the bottom, the data acquisition layer provides the actual link with the real world. It takes in raw RFID readings from the readers, turns them into "primitive events," and sends these events to the event processing layer. This layer (DejaVu) processes these primitive events to detect a set of more complex events in the library. When it does so, the event processing layer communicates with the visualization layer, which then updates the Second Life interface to display the corresponding alerts in the virtual world.

RFID Data Acquisition

The RFID data acquisition layer's main responsibility is to interact with

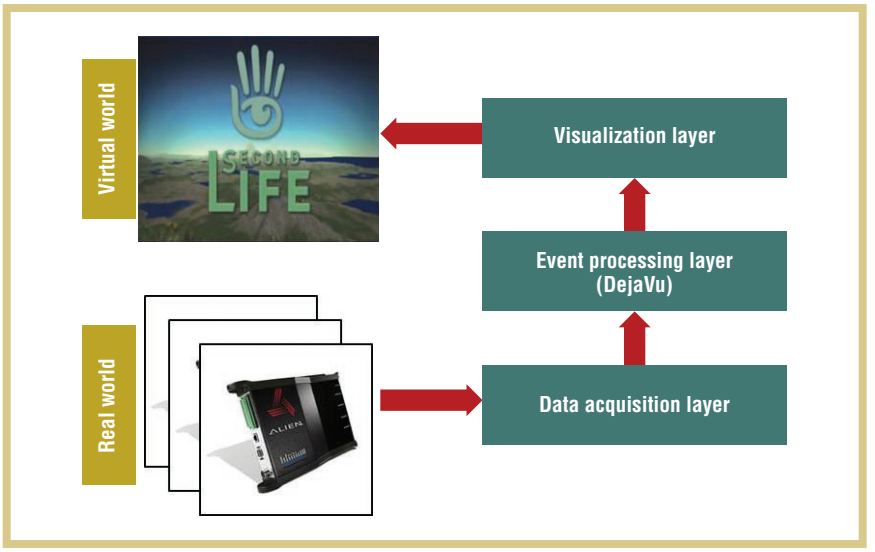


Figure 3. A three-tier architecture for cross-reality environments. The data acquisition layer interfaces with the sensing infrastructure to ensure that the system can correctly collect primitive sensor events. The event processing layer turns the primitive events into complex ones of interest for the virtual world. The visualization layer interfaces with users via the virtual world.

the physical world via the RFID hardware—which captures the presence of people and objects in its vicinity—and accordingly provide event tuples representing these observations to the event processing layer. The main challenge in data acquisition is dealing with the RFID data, which is typically inaccurate (so-called "dirty data") and large in volume. Inaccuracy can be due to missed and unreliable readings or inconsistent readings across multiple readers. The system must clean and transform dirty data into an event stream before it goes to the query-processing layer. Furthermore, the system must compress redundant readings so they don't unnecessarily overwhelm the upper system layers with high data volumes.

This layer accomplishes data acquisition in three steps: first, it captures the RFID readings and puts them into a raw data sink. Then, it cleans the raw readings using a probabilistic data-cleaning algorithm. Finally, it reduces the cleaned data via an application-aware data-compression algorithm.

The data cleaning step is critical because it ensures that the system passes along only correct readings to the upper layers. We based our data-cleaning approach on the adaptive data-cleaning method Shawn Jeffery and his colleagues proposed.[8] In this method, the system collects a window of RFID readings over several reading cycles and, if it observes a tag with enough confidence within this window, reports that tag as a reading (see Figure 4). Furthermore, the system adaptively adjusts the reading window size based on the detected objects' moving patterns, leading to lower error rates.

Jeffery proposed an abstraction layer that separates physical RFID devices and applications to provide *metaphysical data independence* (MDI).[9] Without this separation, applications would need to handle any errors in the sensing devices, making these applications complex, brittle, and difficult to change due to their dependence on the sensor devices. MDI essentially shields applications from the underlying problems that occur when dealing with physical devices directly. In the SmartRFLib application, we also followed an MDI-based approach for data cleaning, which was quite effective. However, we identified a limitation of this general approach when we were dealing with the data compression problem.

In short, cleaning RFID data improves correctness but doesn't avoid the problem of large data volumes, which is critical for system performance. To deal with this issue, we compress cleaned tuples by representing certain readings with fewer tuples. As Figure 4 shows, we've developed two alternative compression techniques: the first is based on periodic responses from tags and the second is based on momentary changes in tag status. The event processing layer can choose one of these techniques for each complex event that it's trying to detect. For example, a theft-detection event might care only about the first-time reading of an RFID tag (the second technique), whereas the shelf event might be interested in getting periodic lists of all currently present books on a particular shelf (the first technique). The key idea here is to customize the compression method according to each complex event's semantic need in the application.

The MDI approach can support application requirements in a generic way but isn't so easy to customize based on different application needs; we can compress data more effectively if we take application semantics into account. So, we extended the MDI approach to allow a parametric interface between the application layer and the physical device layer, thus enabling users to customize desired data acquisition features.

Event Processing with DejaVu

The event processing layer is situated in the middle of our architecture and must thus communicate with both the data-acquisition layer and the visualization layer. On one side, it collects the incoming input streams from the lower layer; on the other, when it detects an important event, it informs the upper layer to update relevant ap-

plication state and associated real-time displays. The event processor's main responsibility is to run complex event-detection queries over the input to process and transform primitive RFID events into semantically richer ones that represent alerts of interest to the modeled application.

In our SmartRFLib application, the data acquisition layer feeds RFID readings for books and people into the DejaVu query-processing engine via live stream store instances. We modeled five library events in SmartRFLib: book check-in, book checkout, illegal checkout of reference books, illegal checkout due to exceeding borrowing quotas, and book theft. We express each of these as a continuous query written in our SQL-based language. DejaVu then parses each continuous query into a corresponding MySQL query plan, augmented with a finite state machine that represents the pattern clause in the query. The DejaVu query-processing engine then executes these query plans continuously, reporting the detected events as real-time streams of alerts.

Let's examine a library event. Given a live stream of book readings **Books(Tstamp, ReaderId, TagId)** from the RFID readers, we express the book-theft complex event in our SQL-based language as follows:

```
SELECT notify_theft(tstamp, book_tag)
FROM Books MATCH_RECOGNIZE(
  PARTITION BY TagId
  MEASURES B.Tstamp AS tstamp,
       B.TagId AS book_tag
  ONE ROW PER MATCH
  AFTER MATCH SKIP PAST LAST ROW
  INCREMENTAL MATCH
  PATTERN(A* B)
  DEFINE A AS ((A.ReaderId != Checkout) AND
         (LAST(A.Tstamp)-FIRST(A.Tstamp)
          < 5 MIN))
      B AS (B.ReaderId = Exit)
  );
```

In this example, the **MATCH RECOGNIZE** clause consists of an event pattern definition (that is, **PATTERN ... DEFINE**) together with some modifiers that indicate how the system should perform the match. We define the book-theft event pattern as zero or more book readings that haven't been reported at a checkout reader within a five-minute time window, followed by a reading detected at an exit reader. The system applies this pattern separately to each book stream partitioned by a book tag identifier (**PARTITION BY TagId**), and in an incremental fashion (**INCREMENTAL MATCH**). The **INCREMENTAL MATCH** clause specifies that the system can trigger a matching event every time it receives a new input data element (as opposed to **MAXIMAL MATCH**, in which the system would trigger an event only after it finished matching the longest sequence of all events satisfying the pattern). **ONE ROW PER MATCH** indicates that for each match the system finds, it will output only one result tuple. **AFTER MATCH SKIP PAST LAST ROW** indicates that after the system finds a match, it will continue the pattern-matching operation with the next tuple that follows the end of the previously matched pattern. **MEASURES** shows further transformations in the output that the user can then use in the **SELECT** clause. Finally, the system invokes a user-defined function (**notify theft**) as the query result for each match found.

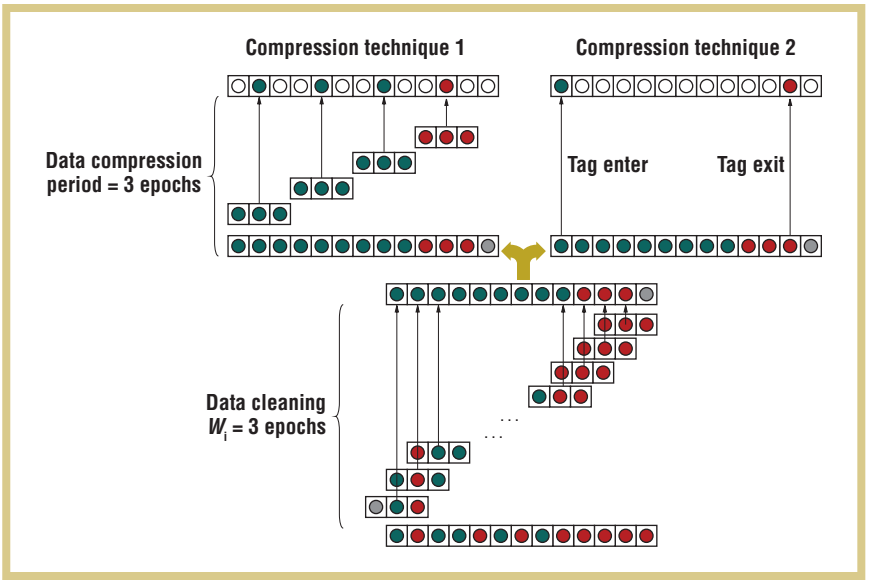


Figure 4. RFID data cleaning and compression. Raw RFID readings are cleaned and compressed before they're passed on to the higher layers of the system.

Figure 5 shows the FSM that corresponds to our example. It's essentially an internal representation of the complex event pattern specified in **PATTERN ... DEFINE** in the book-theft query. Every time the FSM reaches the final state, it invokes the **notify theft** function, raising an alert message that DejaVu will send to the virtual library.

Using this SQL-based CEP language provides several advantages. First, its declarative nature helps programmers focus on the pattern they want to match rather than on specifying a detailed FSM directly. Second, as with traditional databases, we can compile a SQL-based specification into an algebraic query plan for optimization and execution. This also applies when the query plan contains FSMs. Finally, its syntax is easy to learn and use due to widespread familiarity with SQL. However, the CEP language we use also presents a trade-off between how much the programmer can express (for example, various matching modes) and the degree of query complexity exposed to the programmer, which risks lessening the language's

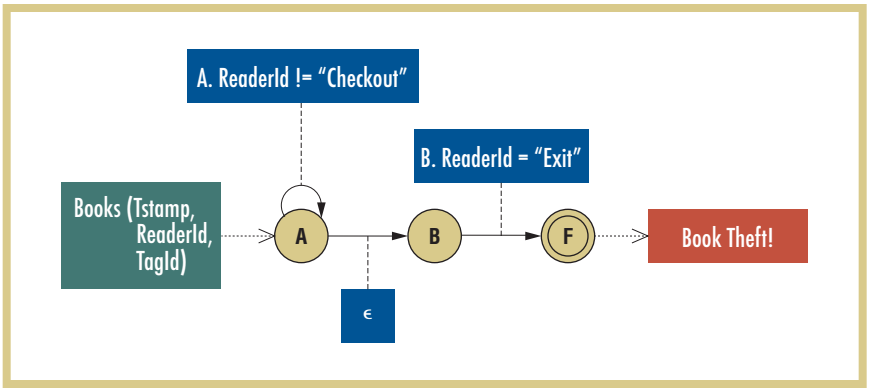


Figure 5. Finite state machine (FSM) for the book-theft event. The FSM represents the complex event pattern specified in the book theft query.

Figure 6: Book-theft event. This event causes exit gates to turn red and an alarm to sound in the virtual library.

declarativeness. However, this isn't specific to only the language that we use; other alternative pattern-matching languages also carry similar risk.[2,10]

Virtual Representation in Second Life

The visualization layer constitutes our architecture's topmost layer. In addition to certain traditional GUI tasks (such as browser-based result displays for specific queries over the library database), this layer is responsible mainly for interacting with the virtual library that we've created in Second Life. Second Life operates on the basis of a client-server architecture, and the most important servers are those running the simulator software. Each simulator server is responsible for a 256 × 256 meter parcel. The simulator runs the physics engine, conducts collision detection, tracks where all the objects are, and sends their locations to a viewer program running on the client side.

Visualizing the real-time library events based on the alerts the event processing layer generates requires the system to control objects' states in the virtual world. To initiate a change for a Second Life object, we must run a small script written in the Linden Scripting Language (LSL) for it. We can define such scripts individually for each object and program them to apply one or more of the following state changes in the virtual world: changing an existing object's position, color, or transparency, attaching a given text to an existing object, playing a sound, or creating a new object.

Figure 6 shows how Second Life visualizes a book-theft alert from our earlier example (exit gates turn red and an alarm sounds).

We faced a couple of important challenges when building our virtual library in Second Life. When representing real events in the virtual world, for example, changes that these events cause should look realistic; at the same time, they must also be informative and intuitive. We found this difficult to achieve for some of our library events. For example, we had to create a special wall for visualizing checked-out books because simply making them invisible wasn't a viable option.

A second challenge was ensuring low-latency communication with the virtual world. As a design goal, we wanted to implement this communication without any modification to the Second Life viewer software. We considered two methods: XML RPC requests coming from outside Second Life, and HTTP requests invoked from within it. We can implement the RPC-based approach more elegantly, but it was unpredictably slow (up to a few minutes of latency). Currently, for most applications deployed on Second Life, this delay is probably acceptable, but for our virtual library, even a few seconds delay would be unacceptable because an event such as a book theft must be communicated in real time.

To deal with the latency issue, we developed an HTTP-based approach on the basis of attaching a script to each object that can receive an event. This script then checks periodically whether the object should change its state. These are actually simple HTTP requests to a Web address. We chose this method because it let us have relatively short response times (less than one second). This poll-based approach puts considerable load on the server handling requests, especially if the library has many books. However, we can adjust the poll period for each object according to the latency requirements of each event associated with that object (for example, two seconds for book objects and one second for gate objects), which makes the issue less critical. We can also imagine using a hybrid approach—that is, having the real-time alarm on just the gates and thus having only the gate working with HTTP requests while the books receive information through XML RPC.

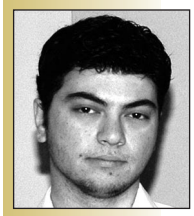
Although we focused here on the information flow from the real world toward the virtual one, as mentioned, we believe that our architecture can also work in the other direction. For ex-

ample, in SmartRFLib, you could walk through the virtual library and borrow virtual books, which would automatically be checked out from the real library and shipped to your real-world address. Such interactions could have many advantages compared to today's text-based Web interfaces. As in a similar real-life situation, users in the virtual world could easily inspect other books similar to a known book by looking through the nearby shelves. In general, a Second Life-like virtual world interface makes human-computer interaction closer to the real experience, and, as we've shown, event processing can contribute to this process from many perspectives.

DejaVu can already run a few application scenarios, including SmartRFLib and a financial pattern-matching application.[4] We're currently working on measuring and improving its query-processing performance based on a few optimization ideas, including one that exploits recurring patterns in FSM optimization. Future work also includes a more detailed investigation of adaptive techniques that are built into our system architecture, such as switching between pull and push models for the flow of data in the system.

the AUTHORS

Nihal Dindar is a doctoral student and a research assistant in ETH Zurich's Computer Science Department. Her research interests are in complex event processing and stream data management. Dindar has an MS in computer science from ETH Zurich. Contact her at dindarn@student.ethz.ch.



Çağrı Balkesen is a doctoral student and a research assistant in ETH Zurich's Computer Science Department. His research interests are in performance optimization of data stream management systems. Balkesen has an MS in computer science from ETH Zurich. Contact him at bcagri@student.ethz.ch.

Katina Kromwijk is a master's student in ETH Zurich's Computer Science Department. Her research interests are in Web-based information systems. Kromwijk has a BS in computer science from EPF Lausanne. Contact her at katinka@student.ethz.ch.

Nesime Tatbul is an assistant professor of computer science at ETH Zurich. Her research interests are in data management systems, with a recent focus on data stream processing and networked data management. Tatbul has a PhD in computer science from Brown University. She is a member of the IEEE, IEEE Computer Society, the ACM, and ACM's Special Interest Group on Management of Data (SIGMOD). Contact her at tatbul@inf.ethz.ch.

ACKNOWLEDGMENTS

We thank Gautier Boder, Florian Keusch, and Ali Sengül for their contributions to an earlier version of the Smart RFID Library application, and Michele De Lorenzi and Julien Vocat for their help with Second Life. This work has been supported in part by grants Swiss NSF NCCR MICS 5005-67322 and Swiss NSF ProDoc PDFMP2-122971/1.

REFERENCES

1. D. Gyllstrom et al., "SASE: Complex Event Processing over Streams (Demo)," *Proc. Conf. Innovative Data Systems Research* (CIDR 07), 2007, pp. 407–411.
2. A. Demers et al., "A General Purpose Event Monitoring System," *Proc. Conf. Innovative Data Systems Research* (CIDR 07), 2007, pp. 412–422.
3. M. Akdere, U. Çetintemel, and N. Tatbul, "Plan-Based Complex Event Detection across Distributed Sources," *Proc. Very Large Databases Conf.*, ACM Press, 2008, pp. 66–77.
4. N. Dindar et al., "DejaVu: Declarative Pattern Matching over Live and Archived Streams of Events (Demo)," to appear in *Proc. ACM Sigmod Conf.*, ACM Press, 2009.
5. S. Pachev, *Understanding MySQL Internals*, O'Reilly, 2007.
6. F. Zemke et al., *Pattern Matching in Sequences of Rows*, tech. report, ANSI standard proposal, July 2007.
7. A. Arasu, S. Babu, and J. Widom, "The CQL Continuous Query Language: Semantic Foundations and Query Execution," *Very Large Databases J.*, vol. 15, no. 2, 2006, pp. 121–142.
8. S.R. Jeffery, M. Garofalakis, and M.J. Franklin, "Adaptive Cleaning for RFID Data Streams," *Proc. Very Large Databases Conf.*, AMC Press, 2006, pp. 163–174.
9. S.R. Jeffery, M.J. Franklin, and M.N. Garofalakis, "An Adaptive RFID Middleware for Supporting Metaphysical Data Independence," *Very Large Databases J.*, vol. 17, no. 2, 2008, pp. 265–289.
10. J. Agrawal et al., "Efficient Pattern Matching over Event Streams," *Proc. ACM Sigmod Conf.*, ACM Press, 2008, pp. 147–160.