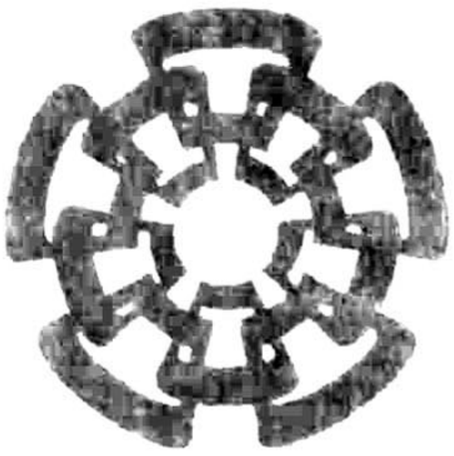
xx(178566.1)



Centro de Investigación y de Estudios Avanzados del I.P.N.
Unidad Guadalajara

Control de Sistemas usando RFID

Tesis que presenta:

Omar Alfredo González Padilla

para obtener el grado de:

Maestro en Ciencias

en la especialidad de:

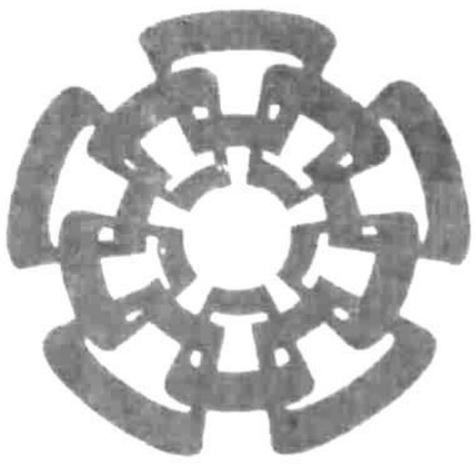
Ingeniería Eléctrica

Directores de Tesis

Dr. Félix Francisco Ramos Corchado
Dr. Herwig Unger

Guadalajara, Jalisco, Agosto de 2008.

ID' 158248-1001



Centro de Investigación y de Estudios Avanzados

del I.P.N.

Unidad Guadalajara

Control Systems Using RFID

A thesis presented by:
Omar Alfredo González Padilla

to obtain the degree of:
Master in Science

in the subject of:
Electrical Engineering

Thesis Advisors:
Dr. Félix Francisco Ramos Corchado
Dr. Herwig Unger

Guadalajara, Jalisco, August 2008.

Control de Sistemas usando RFID

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:

Omar Alfredo González Padilla

Ingeniero en Computación

Universidad de Guadalajara 2002-2005

Becario de CONACYT, expediente no. 203003

Directores de Tesis

Dr. Félix Francisco Ramos Corchado

Dr. Herwig Unger

CINVESTAV del IPN Unidad Guadalajara, Agosto de 2008.

Control Systems Using RFID

Master of Science Thesis
In Electrical Engineering

By:
Omar Alfredo González Padilla
Engineer in Computer Science
Universidad de Guadalajara 2002-2005

Scholarship granted by CONACYT, No. 203003

Thesis Advisors:
Dr. Félix Francisco Ramos Corchado
Dr. Herwig Unger

CINVESTAV del IPN Unidad Guadalajara, August, 2008.

Resumen

Esta tesis propone un nuevo enfoque para desarrollar sistemas RFID. Nuestra propuesta libera a las aplicaciones de negocio de analizar datos RFID buscando patrones relevantes. En lugar de eso, cada aplicación define un conjunto de eventos de interés usando un lenguaje declarativo. Siempre que un evento interesante ocurre en el entorno, la aplicación correspondiente es notificada. Nuestro enfoque hace que las aplicaciones de negocio sean mas faciles de desarrollar y además reduce la cantidad de información comunicada entre el middleware y las aplicaciones, por lo tanto el tráfico de red y la probabilidad de errores disminuye.

La solución propuesta se compone de dos componentes principales: un lenguaje declarativo y una capa de reconocimiento. El lenguaje declarativo le permite a las aplicaciones definir eventos de interés. La capa de reconocimiento analiza el flujo de información RFID en busca de ocurrencias de eventos de interés. Ambos componentes fueron diseñados para trabajar junto con middlewares y dispositivos RFID existentes actualmente.

El lenguaje propuesto permite construir eventos incrementalmente. Los eventos más simples se construyen usando información RFID básica, mientras que eventos más complejos se construyen a partir otros eventos más simples. Nuestro lenguaje permite definir eventos usando: observaciones RFID, relaciones entre observaciones y lugares en el entorno (reuniones, presencias y ausencias), lógica proposicional y ocurrencias previas de eventos más simples (secuencias y combinaciones). La capa de reconocimiento usa Autómatas de Estado Finito y árboles para poder detectar ocurrencias de eventos utilizando flujos de información RFID.

Abstract

This thesis proposes a new approach for developing RFID systems. Our approach releases enterprise applications of analyzing RFID data looking for relevant patterns. Instead, each application defines a set of interesting events using a declarative language. Whenever an interesting event occurs in the physical environment, the corresponding application is notified. Our proposal makes RFID enterprise applications easier to program and reduces the amount of information passed between middleware and applications, therefore network traffic and error probability decrease.

The proposed solution consists of two fundamental components: a declarative language, and a recognition layer. The declarative language allows enterprise applications to define interesting events. The recognition layer analyses the flow of RFID observations, looking for occurrences of interesting events. Both components are designed to work together with existing RFID middlewares and devices.

The language proposed allows constructing events incrementally. Simpler events are constructed using basic RFID information, while more complex events are constructed from simpler events. Our language allows defining events using: RFID observations, relations between observations and locations (i.e. reunions, presences, absences), propositional logic operators, and previous occurrences of events with temporal constraints (sequences, combinations). The recognition layer uses Finite State Automata and threes in order to detect event occurrences within the flow of RFID information.

Acknowledgments

A Dios.

A mis padres, que me han apoyado siempre.

A mis asesores, por aportarme su experiencia.

A CONACYT por apoyar éste trabajo.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem description

Radio Frequency Identification (RFID) [21] is a tool used for automatic identification which most time is utilized for asset tracking. RFID is a concept comparable to bar coding; however using RFID, readings are remote and do not require line of sight. RFID information is generated as a continuous flow of *observations*. An observation is a triplet <object, reader, timestamp> which can be interpreted as the position of an object at a given moment.

An area which takes advantage of RFID technology is Context-Aware Computing [1]. Context-Aware computing comprises applications which must detect particular events happening in the environment and react accordingly. Such applications always require a sensing mechanism in order to gather information about the environment. When Context-Aware Applications must react according to position of objects, RFID seems to be a suitable sensing mechanism.

One of the main components of current RFID architectures is a middleware whose function is to gather and filter observations coming from readers. Despite this filtering process, RFID observations are hard to manage because they are generated at high rates and provide low-level information. The granularity of RFID observations generates two drawbacks:

- Most Context-Aware applications are interested in events which are more complex than simple observations. Hence, Context-Aware applications must perform an analysis over observations. Such analysis detects relevant patterns or relations among observations. Performing this analysis raises the complexity of developing Context-Aware applications.

- A lot of observations which are sent to Context-Aware applications are not useful for

detecting any relevant event. Therefore network resources are unnecessarily wasted, which affects application performance.

1.2 Thesis objectives

The objective of this thesis is to study the previously mentioned drawbacks and to propose an adequate solution. Such solution is implemented by a tool which supports Context-Aware application development. Applications developed using our tool can define a set of interesting high-level events and receive notifications only when relevant events occur, instead of continuously receiving low-level information about localization of objects.

Our proposal satisfies the following requirements:

- Event definition method. To provide a method by means of which Context-Aware applications can define interesting events.

- Event recognition component. To design and implement a component whose task is to recognize occurrences of events by analyzing the flow of RFID observations.

- Compatibility with current RFID infrastructure. The proposed solution should interact with existing middleware and RFID readers.

1.3 Proposed solution overview

The proposed solution was obtained by dividing the problem into three stages, as follows.

- Definition of operators. We have defined a set of operators with sufficient expressivity for defining events. Such set of operators was defined based on Complex Event Processing concepts. However, we have extended such concepts in order to include RFID data.

- Language design. We have designed an XML-Based language for defining events. Context-Aware applications utilize our language in order to specify the set of events they are interested in. We have also developed an XSD schema for validating definitions made using our language.

- Data recognition. We propose a new layer, situated between middleware and Context-Aware Applications. This layer is in charge of analyzing RFID observations for recognizing event occurrences. Each time an event is recognized, the corresponding application is notified. Recognition of events is made using Finite State Automata, trees, sets, and other data structures.

1.4 Thesis outline

This thesis is structured as follows: chapter 2 presents a state of the art overview; chapter 3 illustrates our proposal and all its components; chapter 4 focuses on implementation and includes a study case for a hospital environment; finally, chapter 5 includes conclusions and future work.

Chapter 2

RFID for Context-Aware Computing

2.1 Introduction

In this chapter we present some concepts which are indispensable for understanding our work. At the moment of introducing each concept, we emphasize its importance in our work. Besides the background information, we present some relevant works whose objectives are similar to ours.

2.2 Context-Aware computing

Computer systems often need to be aware about the status of the surrounding environment so they can react accordingly. Such applications are called Context-Aware Applications [1, 14]. Context-Aware Applications gather environmental information through sensors, analyze such information, and adequate their behavior accordingly. This sort of applications uses varied contextual information; for example: temperature, humidity, lighting, noise level, movement, and air pressure. However, location information is by far the most frequently used attribute of context. Therefore RFID is widely used for Context-Aware computing.

2.2.1 Context Information Management

There are different approaches for sensor management and data acquisition used by Context-Aware Applications. Winograd [26] proposes a classification according to such characteristics:

- Widgets. A widget is a kind of driver for accessing sensors. A widget acts as a centralized broker between the set of sensors and a Context-Aware Application. Usage

of widgets hides sensing details and shows only an interface for obtaining data from sensors. Such transparence of low-level details allows applications to operate using different sensing infrastructures. For example, a Context-Aware application which reacts according to location of objects using cameras as sensing mechanism could operate using RFID. This change would be transparent for the Context-Aware Application; the only change required would be on the widget.

- Networked Services. Under this context management, sensors are deployed under a Service-Oriented Architecture. Using networked services each sensor can connect and transfer environmental data to clients. The absence of a centralized component is the main difference of networked services against widgets.

- Blackboard model. This context management is based on a centralized data repository called a *blackboard*. Sensors write their information to the blackboard. Context-Aware applications subscribe with the blackboard in order to be notified when particular events occurs. Such events can be low-level events (simple sensors observations) or high-level events (patterns among several sensor readings). The blackboard is in charge of recognizing event occurrences and notifying the corresponding applications. Current blackboard systems vary in complexity from systems whose events are recognized by just comparing values to systems where usage of artificial intelligence is required.

Current Context-Aware Applications implemented using RFID data, are developed under the widget context management, with the RFID middleware acting as the widget. The purpose of this work is to provide a platform for developing RFID Context-Aware systems under the blackboard context management.

2.3 Radio-Frequency Identification

As mentioned in chapter 1, Radio Frequency Identification (RFID) [8, 21] is a technology used for automatic identification. The foundation of RFID technology is to store data in tags (generally a unique Id [6]), to attach such tags to objects, and to retrieve the content of tags using strategically located readers. Whenever a tag is in the read range of a reader, an observation (containing the tag id, the reader id, and a timestamp) is produced. Communication between RFID tags and readers is performed by means of radio frequency signals, so communication is remote and does not require line of sight.

2.3.1 RFID architectures

Nowadays, most RFID systems are compliant to EPCglobal standards [5]. Such set of standards defines the required components for RFID systems, their behavior and the form of interaction among them. The main objective of EPCglobal architecture and its standards is to increase visibility of RFID-tagged products through the supply chain. When developing Context-Aware systems, only some components of EPCglobal architecture are utilized.

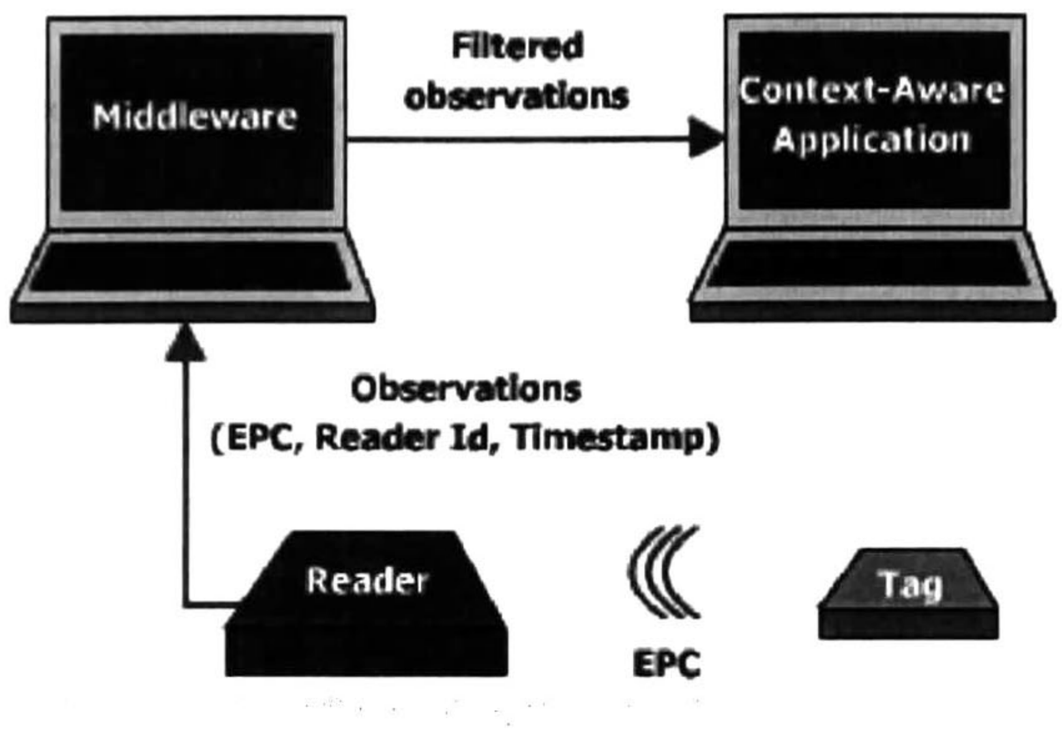


Figure 2.1: EPCglobal system components.

Figure 2.1 shows the principal EPCglobal components for developing Context-Aware applications. Electronic Product Codes (EPC)[4] are identifiers stored in tags. Each EPC must be unique so it identifies a single object. The RFID hardware infrastructure comprises readers and tags. Each time a tag is in the read range of a reader, the corresponding EPC is sent from the tag to the reader. When a reader receives an EPC, an observation (containing the EPC, the id of the reader, and a timestamp) is produced and sent to the middleware. Middleware gathers observations coming from several readers and filters them in order to eliminate duplicated and spurious observations [2, 11]. Afterwards, filtered observations are sent to Context-Aware applications. Finally, Context-Aware applications analyze filtered observations with the purpose of recognizing high-level events happening in the environment.

For developing Context-Aware applications, the usage of the earlier architecture presents two drawbacks. First, Context-Aware applications must analyze observations in order to detect high-level events, so development of Context-Aware applications is difficult. Second, network resources are unnecessarily wasted when middleware sends observations which are not useful for detecting any interesting event.

Application Level Events (ALE) [7] is an interface proposed by EPCglobal for obtaining RFID information and managing RFID readers. ALE allows applications to specify the set of interesting tags and to adjust some parameters for counting, grouping and reporting

observations. ALE also allows to abstract readers into logical places, and to define which logical places are relevant for each application. Using information provided through ALE, middleware knows which tags to report, which readers are relevant, and how to eliminate duplicate and irrelevant observations. ALE also allows applications to choose how they want to receive information; it is possible to receive the complete set of observations, or the differential set of observations (additions or deletions relative to the previous report). Figure 2.2 shows interaction between middleware and applications using ALE.

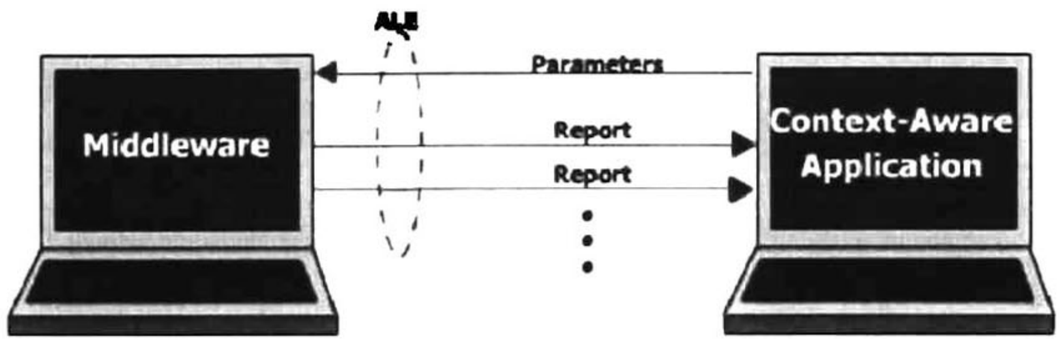


Figure 2.2: Interaction using ALE.

Despite using ALE observations are filtered and network traffic decreases; Context-Aware applications are interested in reacting to events which most of the times are more complex than simple observations. As result, irrelevant observations still overloads the network, and analysis is still performed by Context-Aware Applications.

2.3.2 RFID for Context-Aware Computing

RFID observations provide information about localization of objects at a given moment. Such information is useful for Context-Aware Systems which must react according to localization of people and objects. We call this sort of applications *RFID Context-Aware Applications*; in this section we present some examples. These works exemplify the sort of applications which can take advantage of our proposal.

Dynamic Ubiquitous Mobile Meeting Board (DUMMBO)[3] is a Context-Aware System whose goal is to capture audio from spontaneous meetings around a whiteboard. The whiteboard is equipped with an RFID reader and employees are tagged whit RFID tags. When enough employees are standing around the whiteboard, the application automatically starts recording audio. Later, a user can retrieve the audio and also know the date and time when the respective meeting started. DUMMBO also stores information about which employees were implicated in meetings, and what time did each of them joined and left the meeting.

Under current architectures, DUMMBO is constantly notified about employees nearby to the blackboard. Notifications are received even when insufficient employees are standing in order to determine a meeting. In such cases network traffic is generated, but the information

is not relevant for DUMMBO. In addition, DUMMBO must always analyze information in order to determine the start and finalization of a meeting. However, using our approach, DUMMBO would define in a declarative manner the required environmental features for detecting a meeting start or a meeting finalization. After, DUMMBO would be notified only when a meeting starts or finalizes, which reduces network traffic and development complexity.

In [10], a pervasive hospital bed is presented. Such work is an example of Context-Aware Computing for a hospital environment. Such bed is equipped with an RFID reader, so it is possible to determine who uses the bed and who is near to the bed. The bed is also equipped with a display. Such display adapts its behavior according to nearby people. When the patient is alone, the display can be used as a television; when certain nurses or physicians approach, the display shows information about the patient.

For this example, the Context-Aware application must constantly analyze nearby people in order to determine if there is an authorized nurse or doctor. Observations are sent all the time, even when there is not information about relevant persons. Instead, under our approach, the application would be informed only when a doctor or a nurse approaches to the bed.

In [19] some use cases of RFID usage in a hospital environment are presented. This work is based on current RFID architectures and is based on simple tracking information. However, such use cases would be improved by using high-level event definition and detection. In chapter 4 we present some use cases for a hospital environment.

2.4 Platforms for Context-Aware Computing

Several approaches have been proposed for developing context-aware applications. However, most of them are not intended for RFID systems; instead they can be adapted to different types of sensing methods.

Proactive Activity Toolkit (Proact)[13] is a tool made for inferring activities based on probabilistic inference and data mining. For modeling an activity, first we divide it into stages. For each stage we define the set of involved objects and we assign each object with its probability of being used. For example, if the modeled activity is making tea, the stages involved are: boiling water, putting the tea bag in the water, and flavor the tea. For the first stage, a teapot have a high probability; for the second stage a tea bag have a probability near to one; finally, for the third stage, sugar, milk, lemon, and honey would have a comparable moderate probability. In this approach, inference is made using Bayesian networks. The stages of the actions are considered hidden variables and the sequence of sensed objects are the observable variables.

This tool has been used for inferring several activities of daily living like using a telephone,

preparing a snack, or taking a medication. This tool has showed an average precision of 88%. Despite this approach have showed high accuracy, its probabilistic nature makes it prone to error, which makes it not suitable for critical applications. Another disadvantage is that it is not possible to use historical data neither for defining nor for recognizing events.

RCSM [22, 23, 24] is a middleware for Context-Aware Systems development. RCSM models a Context-Aware System as a set of context-sensitive objects. Each of these objects is composed by an interface and an implementation. Each object is defined as a set of context values. The interface of an object defines a mapping between context values and actions. Such mapping clearly indicates the action or set of actions that should be triggered in response to each interesting event. On the other hand, the implementation contains the actual code of the response actions. RCSM provides a Language for defining interfaces for objects and a compiler in behalf of the automatic generation of customized objects. The generation procedure is the following: first, developers identify their context-sensitive requirements (needed data, meaningful situations, response actions, and scheduling of actions); then use the language to specify the requirements; finally compile the specification in order to generate the objects.

This work considers the usage of any type of sensors. Therefore, its performance can be improved by focusing in a specific type of context-information, like RFID information. Anyway, it is very remarkable of this approach the existence of a grammar for situation definition.

SOCAM [25] is an infrastructure made for developing Context-Aware Systems. Context-Aware Applications developed using SOCAM present a service-oriented architecture. Examples of these services are sensor discovery, data acquisition, and analysis of information. The most remarkable characteristic of SOCAM is its event recognition method. Values of the context are represented as first order logic predicates with the form Predicate(Subject, Value) e.g. Location(John, Bathroom), Temperature(Kitchen, 120), Status(Door, Open). Using this representation, inference of events is done using first order logic, through forward chaining, backward chaining (similar to prolog), or a hybrid execution mode.

The principal disadvantage of using SOCAM is the impossibility of using historical information for recognizing events. This is due to the usage of first order logic inference which limits event expressivity to a reduced set of operators (conjunctions, disjunctions, and negations).

2.5 High-Level Event Management

Event-driven systems [15] are systems where events are generated and consumed between components. Such events may signify a problem, an opportunity, a threshold, etc. Whenever

an event is generated, it is delivered to all interested components; afterwards such components evaluate the event and optionally take actions. RFID systems are a kind of Event-Driven Systems where primitive events are generated by RFID readers. With this work, we extend this idea and include higher-level event management in RFID applications.

Complex Event Processing (CEP) [12, 18] is an emerging technology used for understanding and controlling event-driven systems. The goal of CEP is to identify meaningful high-level events within streams of simpler events. High-level event recognition is performed by looking for relationships between simpler events. Examples of such relationships are causality, timing and membership. Given that RFID information is hard to manage [20, 9], CEP concepts have been applied to RFID in several works.

In [27], a SQL-based language for detecting events along the stream of RFID observations is presented. This work recognizes high-level events looking for specific sequences of observations. However this approach does not support a hierarchy of defined events; all events must be constructed directly from RFID observations, which limits its expressivity and makes tricky the process of constructing events.

PEEX is a system for RFID high-level event management based on probabilities is presented in [17]. The main objective of this approach is to handle noise in RFID observations and ambiguity in high-level event recognition. In this approach, events are probabilistic, i.e. each event definition contains a probability distribution on all its attributes. Given a set of low-level information, such probability distribution models the uncertainty of an actual event occurrence. Event management is performed using SQL queries over a database of observations.

In [16] is presented a system for executing queries over streams of RFID observations. In this approach, each RFID observation is modeled as a primitive event, therefore the input is an infinite sequence of primitive events. In this work, a declarative language called SASE is presented for event high-level definition. SASE allows defining high-level events using simpler events occurrences or non occurrences in a history of events. In [28] an extension of SASE called SASE+ is presented. The base of SASE+ is to use Kleene closure in order to define high-level events as patterns of simpler events. This approach defines events exclusively as occurrences of simpler events; however it is desirable more flexibility for event definition. By augmenting the number of available operators, event definition would become easier and expressivity would increase.

2.6 Discussion

Many Context-Aware Applications take advantage of RFID technology as sensing mechanism; however, directly analyzing RFID information looking for higher level events is hard.

Despite several approaches have been proposed for facilitating Context-Aware Applications development, there are still some drawbacks which must be faced like compliance with currently adopted standards, flexibility for event definition, and historical information usage.

Chapter 3

RFID Composite Event Definition and Detection

3.1 Introduction

In this chapter we present in detail our proposal, which is an extension to current RFID architectures. Such extension allows Context-Aware Applications to use a declarative language in order to define a set of interesting high-level events. Each time an interesting event occurs, the corresponding application is notified.

In RFID environments, both people and objects can be tagged depending on the requirement of each application. From now on we will use the term *actor* equally for tagged people and for tagged objects.

3.2 Overall proposal

For allowing Context-Aware Applications to define their sets of interesting events, we have developed a declarative language called *RFID-Composite Event Definition Language* (RFID-CEDL).

For performing RFID data analysis and event recognition, we have added a new layer called the *recognition layer* between middleware and Context-Aware Applications. This new layer acts as an intermediary between enterprise applications and middleware: on the one hand it obtains from applications a set of interesting events specified using RFID-CEDL; on the other hand, it uses ALE to interact with the middleware in order to obtain filtered data which is necessary for detecting such events. Figure 3.1 illustrates our overall proposal; in the following sections, we describe in detail its components and their functionality.

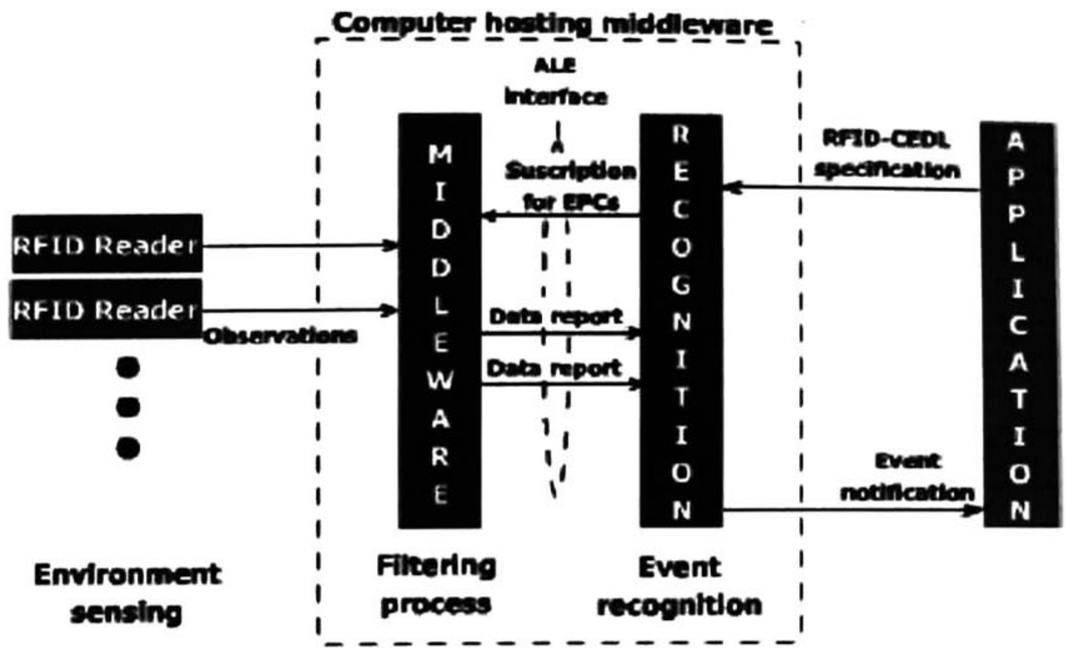


Figure 3.1: Overall proposal.

Notice that the recognition layer is situated in the same host that the middleware. By locating both components in the same host, network traffic decreases substantially.

3.3 Construction and Interpretation of Events

A fundamental part of our work is to define the kind of events manageable by the recognition layer. In chapter 2 several works about composite event definition and detection were presented. We have taken concepts of such works and we have extended them for working with RFID information. In this section, we present our event model and the set of operators available in our approach for constructing events.

3.3.1 Event Categories

We distinguish two categories for events: *primitive events* and *composite events*. Primitive events are automatically generated by the recognition layer using the flow of RFID observations coming from the middleware. Composite events are constructed by combining primitive events and other composite events using a set of predefined operators.

Primitive Events

Primitive events are the starting point for performing event recognition. There are four types of primitive events: *Observation*, *Arrival*, *Departure*, and *Stance*.

Observations are generated by RFID readers and are the basic information unit managed

by the recognition layer. An observation is a triplet (EPC, reader, timestamp) produced whenever an actor is in the read range of a reader. We define an observation event as:

Observation(x, y, t)

where:

x is the id of the observed actor (RFID tag).

y is the identifier of the reader which has detected the actor.

t is the timestamp of the read.

Analyzing directly the flow of observations is hard because RFID readers have high read rates and the volume of observations generated by them is enormous. Instead of performing high-level event recognition by analyzing simple observations, we use *differential information.* Differential information is information about arrivals and departures of actors respecting to read ranges of readers. We define now two primitive events for expressing differential information: *departure* and *arrival.*

A *departure* for the actor x and the reader y is produced when x is observed by y and after that x is not observed by y during a period of time p. The duration of p depends on the read rate and accuracy of readers.

$$(\exists x, y, t_1)((Obervation(x, y, t_1) \wedge (\forall t_2)(Observation(x, y, t_2) \rightarrow ((t_2 \leq t_1) \vee (t_2 \geq t_1 + p)))) \rightarrow Departure(x, y, t_1))$$

The event *Departure(x,y,t_1)* means that actor x has left the read range of reader y at time t_1.

An *arrival* for the actor x and the reader y is produced whenever x is observed by y and for all the previous observations of x by y, there exists a posterior departure. We express:

$$(\exists x, y, t_1)(Obervation(x, y, t_1) \wedge (\forall t_2)((Observation(x, y, t_2) \wedge (t_2 < t_1)) \rightarrow (\exists t_3)(Departure(x, y, t_3) \wedge (t_3 > t_2))) \rightarrow Arrival(x, y, t_1))$$

The event *Arrival(x,y,t_1)* means that actor x has arrived to the read range of reader y at time t_1.

Information provided by arrivals and departures can be further compressed by creating an event *stance.* A *stance* is created for each arrival and its corresponding departure. We define a stance as:

$$(\exists x, y, t_1, t_2)((Arrival(x, y, t_1) \wedge Departure(x, y, t_2) \wedge (t_1 < t_2) \wedge (\neg\exists t_3)(Arrival(x, y, t_3) \wedge (t_3 > t_1) \wedge (t_3 \leq t_2))) \rightarrow stance(x, y, t_1, t_2))$$

The event *Stance(x,y,t_1,t_2)* means that actor x has been located into the read range of reader y during the period comprised between t_1 and t_2.

Composite events

Composite events are events defined using RFID-CEDL. A composite event can be constructed using simpler events, properties of actors, location of actors, and additional data. In the following section we present the set of operators available in RFID-CEDL for expressing such information.

When a composite event is defined using RFID-CEDL, it must be provided with a name n and a set of fields $F:= \{f_1, f_2, ..., f_n\}$. The recognition layer internally generates an analogous type of event which also contains an extra field called t. The attribute t is utilized for registering the timestamp of each event instance. According to this, the general structure of a composite event within the recognition layer is:

$$n(f_1, f_2, ..., f_n, t)$$

If an event instance of type n occurs at time t, their values $v_1, v_2, ..., v_n$ are stored by the recognition layer. We represent a composite event instance as a predicate of the form:

$$n(v_1, v_2, ..., v_n, t)$$

In section 3.4 we describe deeply how to define events by means of RFID-CEDL and the usage of names and fields.

3.3.2 Operators

In this section we present the set of operators available in our platform for composite event construction; however, before presenting operators, it is necessary to define a term called *stays* which evaluates if an actor x is located within the read range of a reader y at a given moment t.

$$(\exists x, y, t_1, t_2, t)((stance(x, y, t_1, t_2) \wedge (t \geq t_1) \wedge (t \leq t_2)) \rightarrow stays(x, y, t))$$

Location Operators

Primitive events provide information about localization of a single actor. Based on this constraint, we have identified the necessity of grouping information about location of several actors. According to this, we have included three operators which take their values by analyzing localization of several actors. These operators are *all*, *any*, and *none*; each of these operators is evaluated as a boolean value. Their values are calculated according to a set of actors $X := \{x_1, x_2, ..., x_n\}$, a reader y, and a time t.

Now we present the three location operators:

- The *any* operator is evaluated as true if any of the actors in X stays within the read range of y at instant t.

$$Any(x_1, x_2, ..., x_n, y, t) \leftrightarrow stays(x_1, y, t) \vee stays(x_2, y, t) \vee ... \vee stays(x_n, y, t)$$

- The *all* operator is evaluated as true if all the actors in X are situated within the read range of y at instant t.

$$All(x_1, x_2, ..., x_n, y, t) \leftrightarrow stays(x_1, y, t) \wedge stays(x_2, y, t) \wedge ... \wedge stays(x_n, y, t)$$

- The *none* operator is evaluated as true if none of the actors in X is situated within the read range of y at instant t.

$$None(x_1, x_2, ..., x_n, y, t) \leftrightarrow \neg stays(x_1, y, t) \wedge \neg stays(x_2, y, t) \wedge ... \wedge \neg stays(x_n, y, t)$$

Occurrence Operators

We now present two operators for defining composite events using a history of occurrences of simpler events. These operators are: *sequence* and *combination*.

- A *sequence* of events is defined specifying a list of event type names, a time t, and an integer i. A sequence will be recognized if all the events in the list of events have happened ordered within the period comprised between t-i and t.

$$Sequence(ev_1, ev_2, ..., ev_n, t, i) \leftrightarrow ev_1(v_{11}, v_{12}, ..., v_{1n}, t_1) \wedge ev_2(v_{21}, v_{22}, ..., v_{2n}, t_2) \wedge ... \wedge ev_n(v_{n1}, v_{n2}, ..., v_{nn}, t_n) \wedge (t - i) <= t_1 < t_2 < ... < t_n <= t$$

- A *combination* of events is defined specifying a list of event type names, a time t, and an integer i. A sequence will be recognized if all the events in the list of events have happened within the period comprised between t-i and t, in any order.

$$Combination(ev_1, ev_2, ..., ev_n, t, i) \leftrightarrow$$
$$ev_1(v_{11}, v_{12}, ..., v_{1n}, t_1) \wedge ev_2(v_{21}, v_{22}, ..., v_{2n}, t_2) \wedge ... \wedge ev_n(v_{n1}, v_{n2}, ..., v_{nn}, t_n) \wedge (t-i) <= t_j <= t$$

where:

$$j = 1,2,...,n$$

Existence Operator

As mentioned before, using RFID-CEDL an event can be defined using additional information which is indirectly related to actors. For example, to recognize an allergic medication in a hospital environment, it is necessary to know to which medicines is allergic each patient.

- *Exists* is an operator provided in RFID-CEDL which searches into additional information looking for occurrences of specific values. Exists returns true if matching information is found, false otherwise.

Boolean Operators

All operators defined previously are evaluated as boolean values. RFID-CEDL allows using boolean operators for combining sub expressions. The supported boolean operators are conjunction, disjunction and negation. We have decided to use this set of operators due to its functional completeness.

Comparison Operators

Events and actors defined in RFID-CEDL are composed by fields. Each instance of an event or an actor provides values for the corresponding fields. When defining an event, it is possible to compare such values using comparison operators (<, <=, =, >, >=).

3.4 A Language for Composite Event Definition

In previous sections we have defined primitive types of events and the set of operators for defining composite types of events. In this section we present the tool which allows composite

event definition. Such tool is a declarative, XML-based language called RFID-CEDL. The principal advantage of using XML syntax is its easy usage and validation.

RFID-CEDL is typified. Validation of data types is easily performed using an XML schema. This feature makes definitions made in our language less error prone. Another advantage of using data types explicitly is that it is easier to read definitions made in our language and to understand the purpose of each element. Primitive types provided in our language are: integer, float, string, boolean, and date; however given its XML nature, the language can be extended to support other primitive types.

For defining the set of interesting events for an application, it is necessary to create a *specification*. A specification is an XML document which must be compliant to our language; such compliance is validated using a special XML-Schema for RFID-CEDL specifications (appendix A).

In this section we present the five main construction sections of a specification, we illustrate the usage of each section with examples of their usage in a hospital environment. Further examples are presented in chapter 4.

3.4.1 Classes

Within the section *classes*, we create the necessary data structures for defining composite events. Classes are composed by fields; a field is an attribute for the class and is defined with a data type and a name. Classes could either correspond to a kind of actor in the physical world or not. The process of defining classes is similar to defining a database for a system; developers decide which classes are needed and their fields. For example, to recognize allergic medications in a hospital environment, we need to define a patient class, a medicine class, and a patientallergy class which defines to which medicines a patient is allergic. Note that for this example, both patient and medicine correspond to tagged objects in the environment, while patientallergy does not. Next, we show an example of a class definition.

```
<class name="medicine">
  <field>
    <string name="name"/>
  </field>
  <field>
    <date name="caducity"/>
  </field>
  <field>
    <string name="activesubstance" />
  </field>
</class>
```

3.4.2 Instances

Once we have defined the appropriate classes for the application scenario, we must provide information about tagged objects. This task is done through the instances section of our language. Within this section, we link a tag id to an actual object by defining the class to which the object belongs and providing values for the fields defined in such class. For example, we can instantiate a medicine according to the class defined in the previous subsection as follows:

```
<instance epc="urn:upc:id:gid:10.1002.2" class="medicine">
  <attribute>Aspirin</attribute>
  <attribute>2008-12-15</attribute>
  <attribute>Acetylsalicylic acid</attribute>
</instance>
```

The epc attribute is the id stored in the RFID tag of the object we are defining; this attribute is the linkage between specifications made in our language and the physical world. Note that the data types of proportioned values for the attributes must correspond to the data types of the fields defined in the corresponding class.

3.4.3 Data

In the data section we provide additional information which is necessary for event definition but is not directly referent to actors. The data section is similar to the instances section, but given that we are not defining tangible objects, we do not provide a tag id (epc attribute). To illustrate, we will use the example of recognizing allergic mediations; in this event, it is necessary to know which allergies suffers each patient. An example of such information in a specification is presented next:

```
<data class="patientallergy">
  <attribute>John Connor</attribute>
  <attribute>Aspirin</attribute>
</data>
```

Every data definition must correspond to a previously defined class, this correspondence is defined through the attribute named class. For our example, there must be a class named patientallergy in the classes section, and such class must contain two string fields.

3.4.4 Locations

In order to use RFID observations in events definition we must consider the position of RFID readers. Within the locations section, we define relevant places in the environment of our

application. To define a location, we provide its name and a set of RFID reader identifiers. By defining a location as a set of readers, we can manage several physical readers as a single source of observations. Next, we show how to define locations using RFID-CEDL.

```
<location name="room1">
  <reader name="192.168.1.2" />
</location>
<location name="corridor">
  <reader name="192.168.1.2" />
  <reader name="192.168.1.4" />
</location>
```

As shown in the example, the reader named 192.168.1.2 is used to define two different locations, which illustrates that the use of RFID readers for a location definition is not exclusive; we can use the same RFID reader to define two or more locations.

3.4.5 Events

The events section is the most important section in our language because is there where we use the information provided in previous sections to define the events of interest for our application. For each event, we must define *variables*, *fields* and a *condition*.

Variables

The *variables* of an event definition provide the required information for defining such event. There are three types of variables: tagged objects (defined in the instances section), information (defined in the data section), and previously defined simpler events. Each variable should have a name and a type: for tagged objects and information, the type is a class name; for events the type is the name of the event.

Fields

The *fields* of an event provide information related to each event occurrence. Automatically, every time an event is recognized, its name and its timestamp are stored; however if we specify fields for the event, we can store further related information. Fields obtains their values from the variables of the event. As mentioned in section 3.4.5, the fields of an event are useful for defining events from other events. For example, whenever we detect a medication, we can store the medicine name and the patient name as fields; then we could use these values to define potential overdoses of a medicine for a patient.

Condition

The *condition* of an event is a logic expression which is constantly evaluated by the recognition layer in order to determine when an event instance occurs. A condition is constructed using the operators presented in section 3.3.2. We now illustrate an example of a condition; however, several examples will be presented in the next chapter.

Suppose we must detect an erroneous medication. For our example, suppose also that medicines are always provided by a nurse, and a medication is erroneous if is provided to an allergic patient or if the patient has gotten a blood transfusion within the five minutes previous to the medication. The condition for this event is:

```
<event name="wrongmedication">
  <var class="nurse" name="n1"/>
  <var class="medicine" name="m1"/>
  <var class="patient" name="p1"/>
  <var event="transfusion" name="t1"/>
  <fields>
    <string name="patient" value="p1.name"/>
    <string name="medicine" value="m1.name"/>
  </fields>
  <condition>
    <and>
      <all>
        <object>p1</object>
        <object>m1</object>
        <object>n1</object>
        <location>room1</location>
      </all>
      <or>
        <and>
          <sequence within="300">
            <occurrence>t1</occurrence>
          </sequence>
          <equal>
            <value>t1.patient</value>
            <value>p1.name</value>
          </equal>
        </and>
        <exists type="allergicmedicine">
          <value>p1.name</value>
          <value>m1.name</value>
        </exists>
      </or>
    </and>
  </condition>
</event>
```

3.5 A Layer for Composite Event Recognition

In this section we present how the recognition layer performs event recognition. We present the required data structures and their overall functionality. We also describe how information

flows between the recognition layer and the middleware through ALE, and the means for importing event definitions.

3.5.1 Data Management

RFID Data Acquisition

Once the recognition layer has received a specification made in RFID-CEDL, it must use ALE in order to define which data to obtain from the middleware and how to obtain it. The recognition layer must establish three main parameters to accomplish this:

- Interesting readers. The set of interesting readers is defined according to locations defined in the specification.

- Interesting tags. The set of tags which will be reported from the middleware is defined as the set of tags mentioned within the instances section of the specification.

- Type of reports. In order to reduce the volume of information, the recognition layer must be informed only with differential information, i.e. only about arrivals and departures of actors respecting to the read range if interesting readers.

Using these parameters, the middleware informs the recognition layer only about arrivals and departures of interesting tags to read ranges of interesting readers.

RFID Data Storage

The recognition layer stores information obtained from the middleware in a table called t_stance. Each row in t_stance stores an actor id (RFID tag id), a reader id, a start timestamp, and an end timestamp. Each row in this table indicates that an actor was situated in a location during a given period of time.

Every time the recognition layer is notified about the entrance of an object to the read range of a reader, a record with a null end timestamp is created in t_stance; when the recognition layer is notified about an object leaving the read range of a logical reader, the corresponding end timestamp is fulfilled. Figure 3.2 shows example data for t_stance.

Event History Storage

For evaluating *sequences* and *combinations*, it is necessary to maintain a history of recognized events. Such history is kept in a table called t_occurrence. Each row in t_occurrence stores

Actor ID	Reader ID	Start	End
urn:upc:id:gid:10.1002.5	192.168.1.2	1520	NULL
urn:upc:id:gid:10.1002.3	192.168.1.4	1575	1860
urn:upc:id:gid:10.1002.3	192.168.1.4	1930	NULL
urn:upc:id:gid:10.1002.2	192.168.1.2	2860	2930
urn:upc:id:gid:10.1002.7	192.168.1.2	2862	2933
urn:upc:id:gid:10.1002.2	192.168.1.4	4120	5640
urn:upc:id:gid:10.1002.6	192.168.1.4	4128	5639
urn:upc:id:gid:10.1002.2	192.168.1.9	5730	NULL
urn:upc:id:gid:10.1002.7	192.168.1.11	6200	NULL
urn:upc:id:gid:10.1002.6	192.168.1.11	6281	NULL

Figure 3.2: t_stance

an event type name, a timestamp, and linked list of values corresponding to the fields of each event occurrence. The linked list of values is implemented by means of other table called t_value.

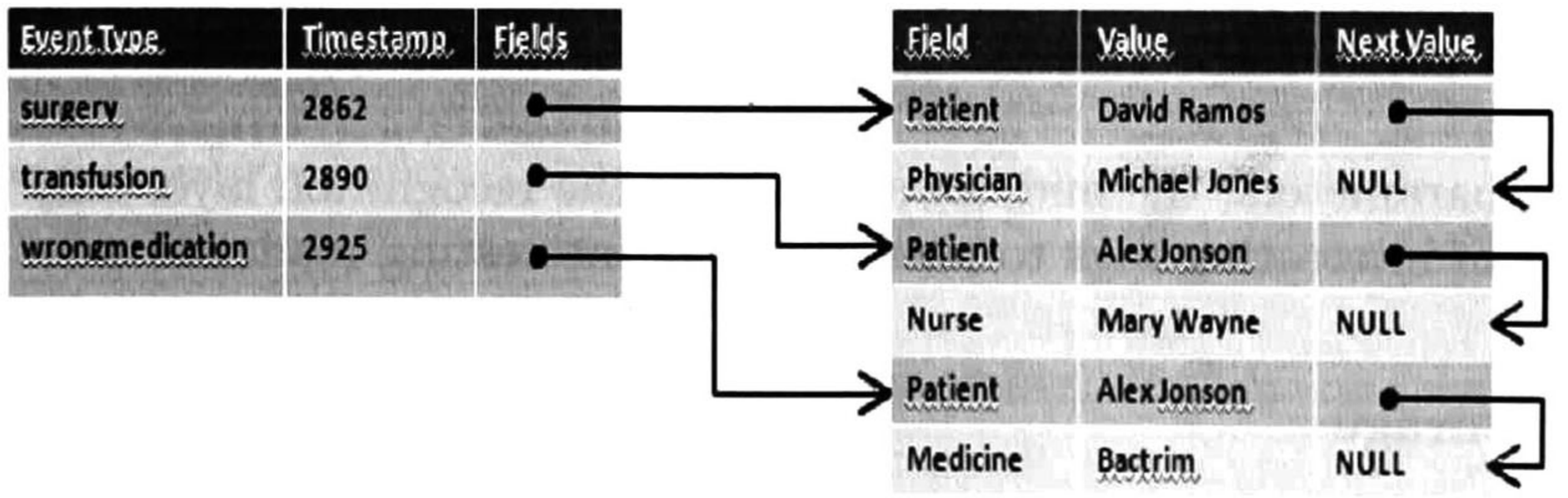


Figure 3.3: t_occurrence and t_value

3.5.2 Event Definition Management

The recognition layer manages each event condition as a syntax tree where the root is the main operator of the condition. The condition is decomposed by levels until the leafs, which are simple operators. The process of importing a RFID-CEDL definition as a syntax tree is straightforward given the inherent tree structure of XML syntax. Figure 3.2 shows an example of the syntax tree corresponding to the event presented in section 3.4.5.

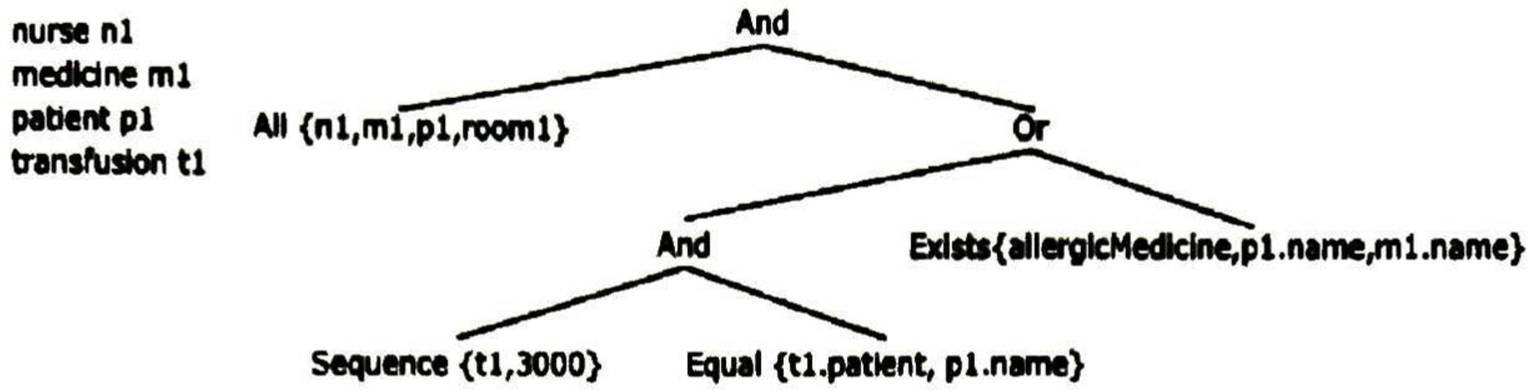


Figure 3.4: Example Syntax Tree

3.5.3 Situation Recognition

Events in our model are triggered by movement of actors. Therefore, the recognition layer evaluates syntax trees whenever an arrival or a departure happens.

When an arrival or departure of actor x is detected, the first step of event recognition is to evaluate all parse trees where x is parameter of a location operator (all, any, none). For each tree evaluated as true, an occurrence of the event type is detected and the respective row in t_occurrence is created.

The second step of event recognition takes place every time a row is created in t_occurrence. When an event of type e happens, all parse trees where an event of type e is parameter of an occurrence operator (sequence or combination) are evaluated. For each tree evaluated as true, the second step is performed again.

We now present how the recognition layer evaluates each operator.

All

The *all* operator is evaluated looking into t_stance. $All(x_1, x_2, ..., x_n, y)$ is evaluated as true if for each actor $(x_1, x_2, ..., x_n)$ there exists a row with the reader set to y and a null end timestamp.

Any

The *any* operator is evaluated looking into t_stance. $Any(x_1, x_2, ..., x_n, y)$ is evaluated as true if for any actor $(x_1, x_2, ..., x_n)$ there exists a row with the reader set to y and a null end timestamp.

None

The *none* operator is evaluated looking into t_stance. $None(x_1, x_2, ..., x_n, y)$ is evaluated as true if for each actor $(x_1, x_2, ..., x_n)$ there does not exist a row with the reader set to y and a null end timestamp.

Sequence

The recognition layer evaluates sequences using Non-deterministic Finite Automata (NFA). For each sequence, a NFA is constructed by linking successive states for successive elements in the sequence. Each transition is labeled with the corresponding event type in the sequence. The last state is the only accepting state of the automata. Figure 3.5 illustrates the NFA corresponding to sequence ev_1, ev_2, ev_3.

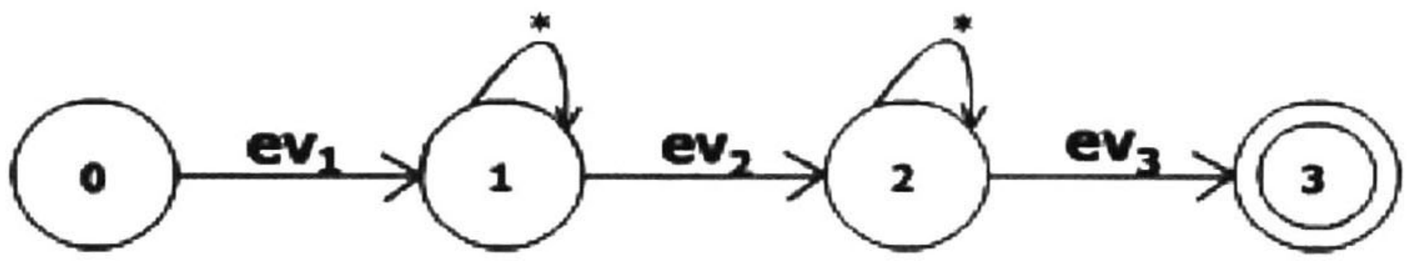


Figure 3.5: NFA for sequence ev_1, ev_2, ev_3

A *sequence* is evaluated looking into t_occurrence. For evaluating the sequence of events represented by $Sequence(ev_1, ev_2, ..., ev_n, i)$ the recognition layer filters the set of event occurrences registered within the last i seconds. The filtered set of events is ordered by timestamp and provided as input for the corresponding NFA. All paths from the initial state to the acceptation state are satisfactory sequences of events.

Combination

A *combination* is also evaluated looking into t_occurrence. For evaluating the combination of events represented by $Combination(ev_1, ev_2, ..., ev_n, i)$ the recognition layer filters the set of event occurrences registered within the last i seconds. The filtered set of occurrences is divided into equivalence classes according to the event type. Let such equivalence classes be the subsets $EV_1, EV_2, ..., EV_n$. The set of satisfactory combinations is EV_1 X EV_2 X EV_n. If an event type is included more than once as a parameter of the combination, it should be considered the same number of times in the Cartesian product.

Exists

The recognition layer uses a table called t_data for managing information provided through the *data* section of specifications. Each row in t_data stores the class name of the data element, and a linked list of values corresponding to the attributes of each data element. The linked list of values is implemented by means of other table called t_attribute.

To evaluate an *exists* operator, the recognition layer filters all the rows with the corresponding class name, and searches sequentially for a matching list of attribute values.

3.5.4 Discussion

In this chapter, we have presented our approach for event management using RFID information. We have presented the sort of manageable events and the set of operators available for their construction. We have deeply described a language for composite event definition. Finally we have included the means for obtaining basic RFID information and recognizing events from such information.

Chapter 4

A RFID-Environment Simulator

In this chapter we describe the implementation of the proposal described in chapter three. Such implementation is a tool which generates simulated RFID environments from given RFID-CEDL specifications. The simulator also allows defining RFID-CEDL specifications using graphic user interfaces. We also present some study cases for a hospital environment which where simulated using our simulator.

4.1 Simulator

In this section we present a tool called RFID-Composite Event Simulator (RFID-CES). RFID-CES allows constructing and simulating specifications compliant with RFID-CEDL. After a specification is constructed, it can be simulated; the simulation of a specification generates logical places and a set of actors according to the specification. At simulation time, users can move actors within the simulated environment. The simulation tool recognizes and displays recognized events derived from such movements.

4.1.1 Constructing Specifications

Given the simplicity of RFID-CEDL, RFID-CES allows constructing specifications by means of graphic user interfaces. In this section we present how to use RFID-CES for providing information corresponding to each of the five main construction areas of RFID-CEDL.

Classes

A class must be provided with a name and a set of fields. Each field has a name and a type. Figure 4.1 shows an interface for adding classes to a specification. Input provided in figure

4.1 generates the following fragment of RFID-CEDL code:

```
<class name="medicine">
  <field>
    <string name="name"/>
  </field>
  <field>
    <date name="caducity"/>
  </field>
  <field>
    <string name="substance" />
  </field>
</class>
```

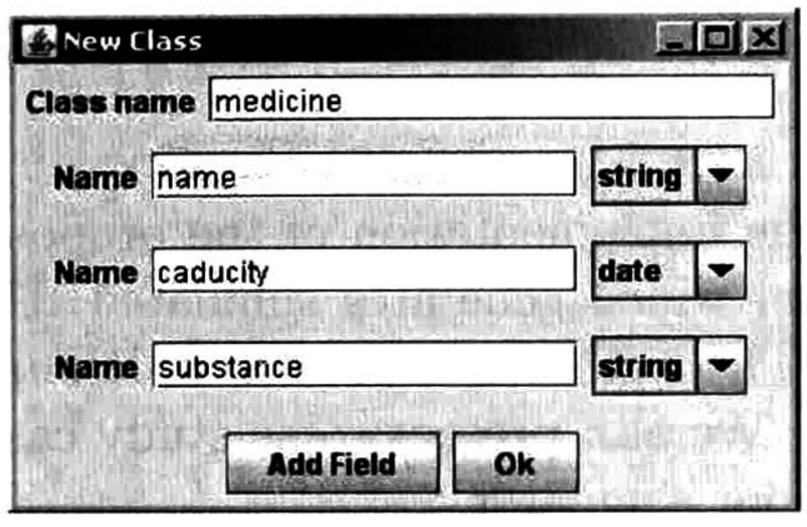


Figure 4.1: Interface for adding classes.

Instances

When defining an instance, the user must indicate to which class the instance belongs, which is the EPC stored in the corresponding RFID tag and also must provide values for the fields of the corresponding class. Figure 4.2 shows the interface for adding instances to a specification. Input provided in figure 4.2 generates the following instance definition:

```
<instance epc="gid:10.1002.2" class="medicine">
  <attribute>Aspirin</attribute>
  <attribute>2008-12-15</attribute>
  <attribute>Acetilsalicylic acid</attribute>
</instance>
```

Data

When defining additional data, the user must indicate to which class does the data belongs and provide values for the fields of the corresponding class. Figure 4.3 shows the interface for adding data to a specification. Input provided in figure 4.3 generates the following data definition:

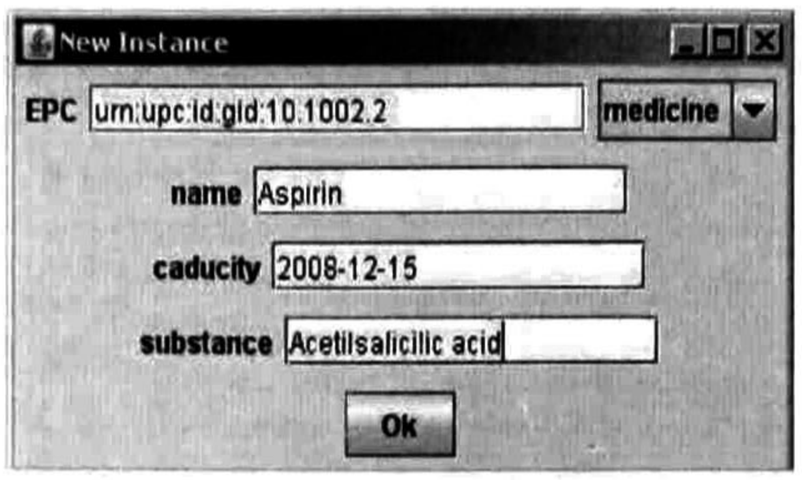


Figure 4.2: Interface for adding instances.

```
<data class="allergy">
  <attribute>Omar Gonzalez</attribute>
  <attribute>Acetilsalisilic Acid</attribute>
</data>
```

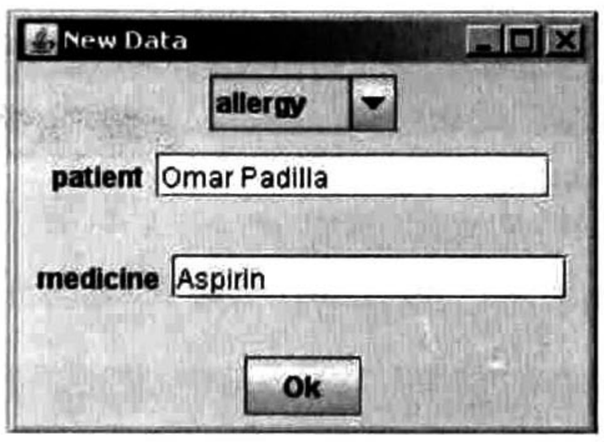


Figure 4.3: Interface for adding data.

Locations

For defining locations, the user must provide a name for the location and also the set of identifiers of the RFID composing the location. Figure 4.4 shows the interface for adding locations to a specification. Input provided in figure 4.4 generates the following code defining a location:

```
<location name="room1">
  <reader name="198.146.83.24" />
  <reader name="198.146.83.25" />
</location>
```

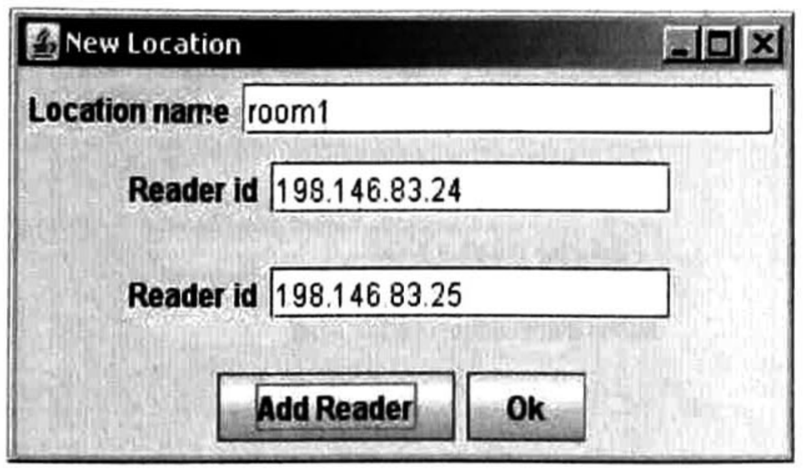


Figure 4.4: Interface for adding a location.

Events

Defining an event using RFID-CES requires several steps. The first of them is to provide a name for the event, which is illustrated in figure 4.5.

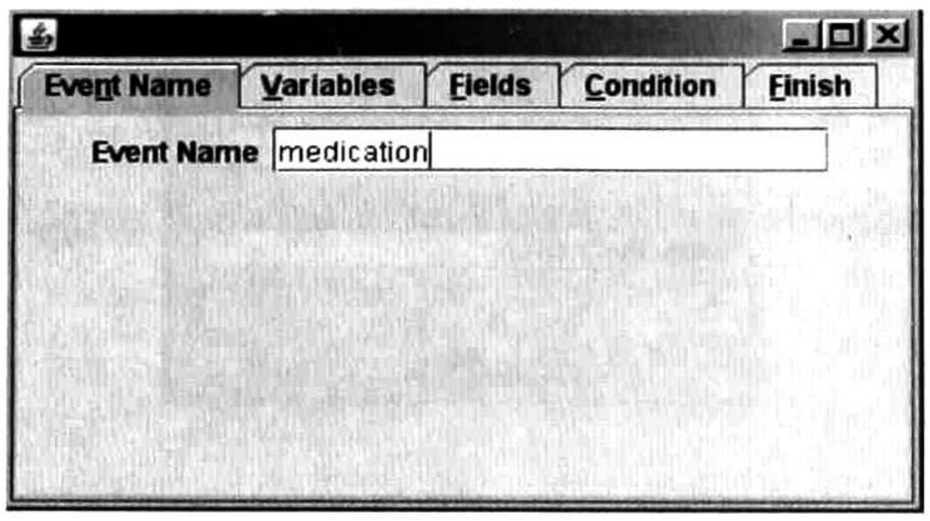


Figure 4.5: Interface for defining the name of an event.

The second step is to define the required variables for defining the event. Each variable has a name and a type. The type of a variable can be either a class name or a previously defined event name. Figure 4.6 shows how to define variables for an event using RFID-CES.

The third step is to define the set of fields which will be stored with each occurrence of the event type. For each field, the user must specify a name and its value. The value of a field is obtained from the variables of the event. Figure 4.7 shows how to define fields for an event using RFID-CES.

The last step for defining an event is to specify the condition required in order to recognize the event. RFID-CES allows constructing a condition by combining RFID-CEDL operators in a condition three. Figure 4.8 illustrates the interface available in RFID-CES for constructing an event condition.

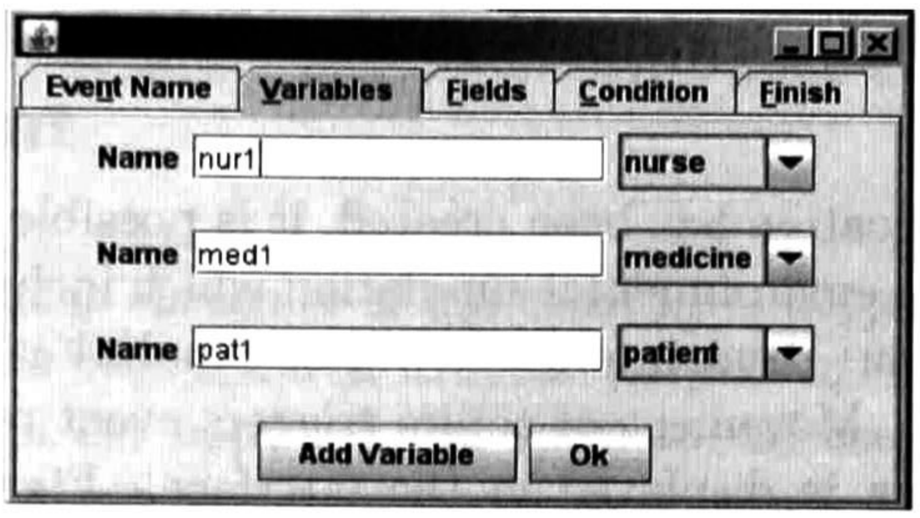


Figure 4.6: Interface for defining variables for an event.

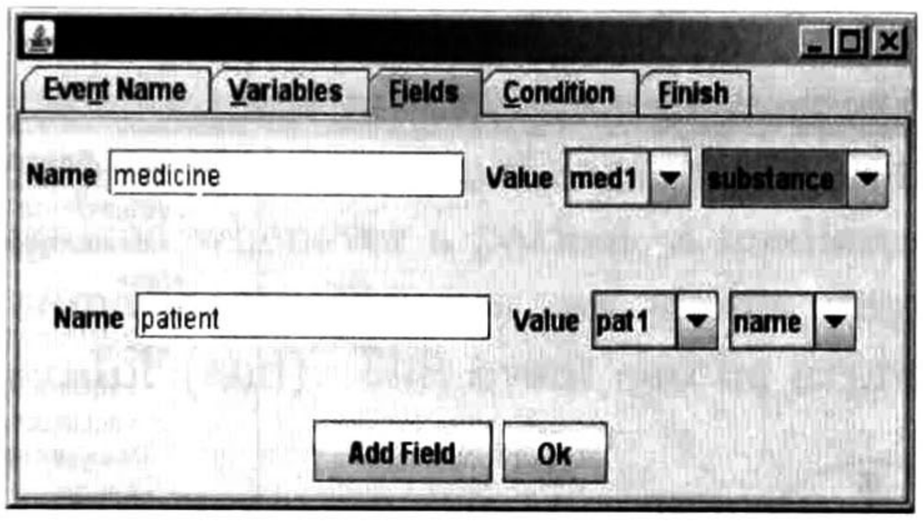


Figure 4.7: Interface for defining fields for an event.

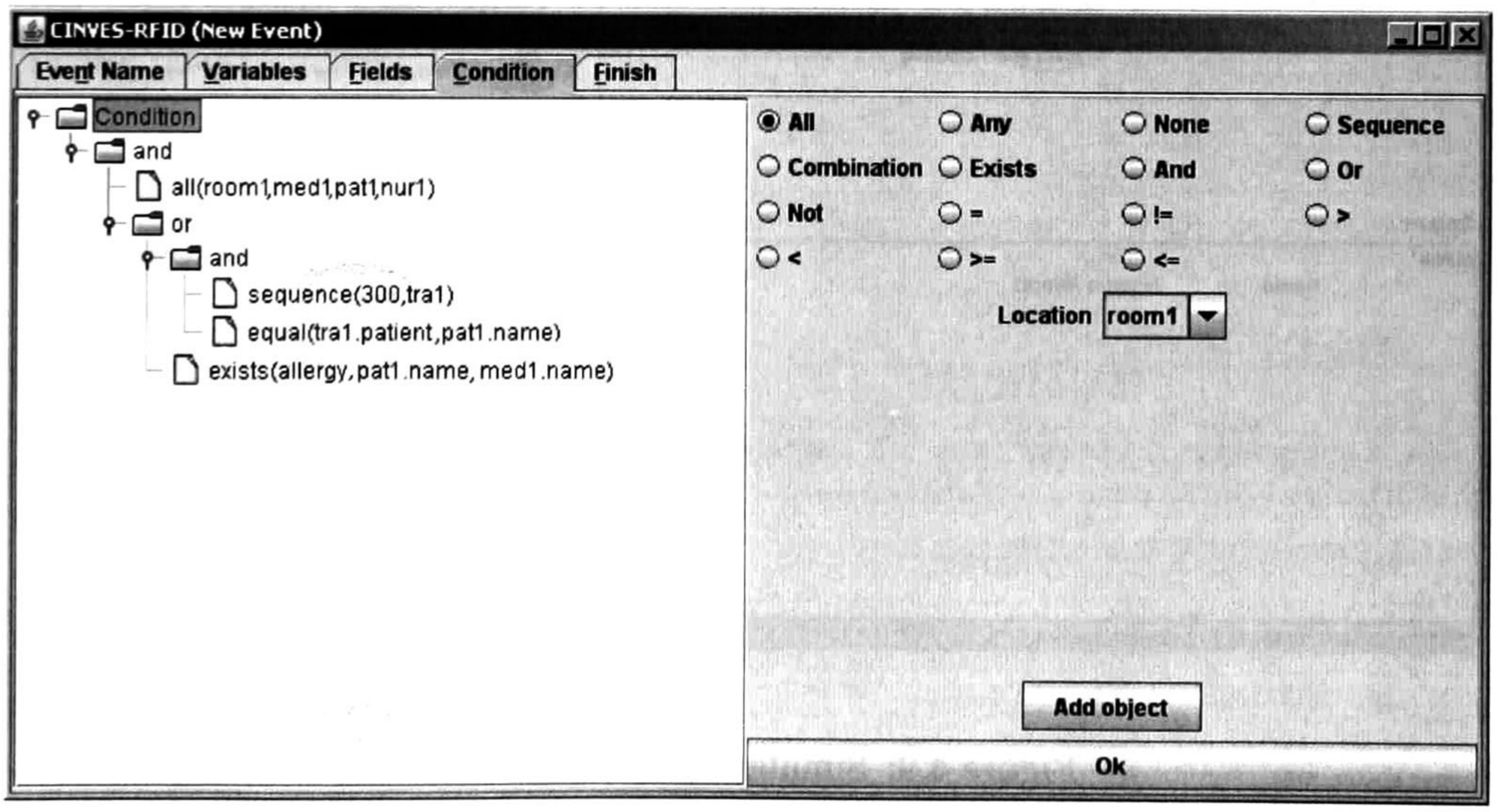


Figure 4.8: Interface for defining an event condition.

4.1.2 Environmental Simulation

Once a RFID-CEDL specification has been created, it is possible to simulate it using RFID-CES. RFID-CES creates an environmental simulation which includes locations and instances according to the specification. Users interact with this simulation by moving instances within the simulated environment. Movement of actors triggers event recognition procedures. The history of recognized events is displayed in the interface. Figure 4.9 shows a simulation example.

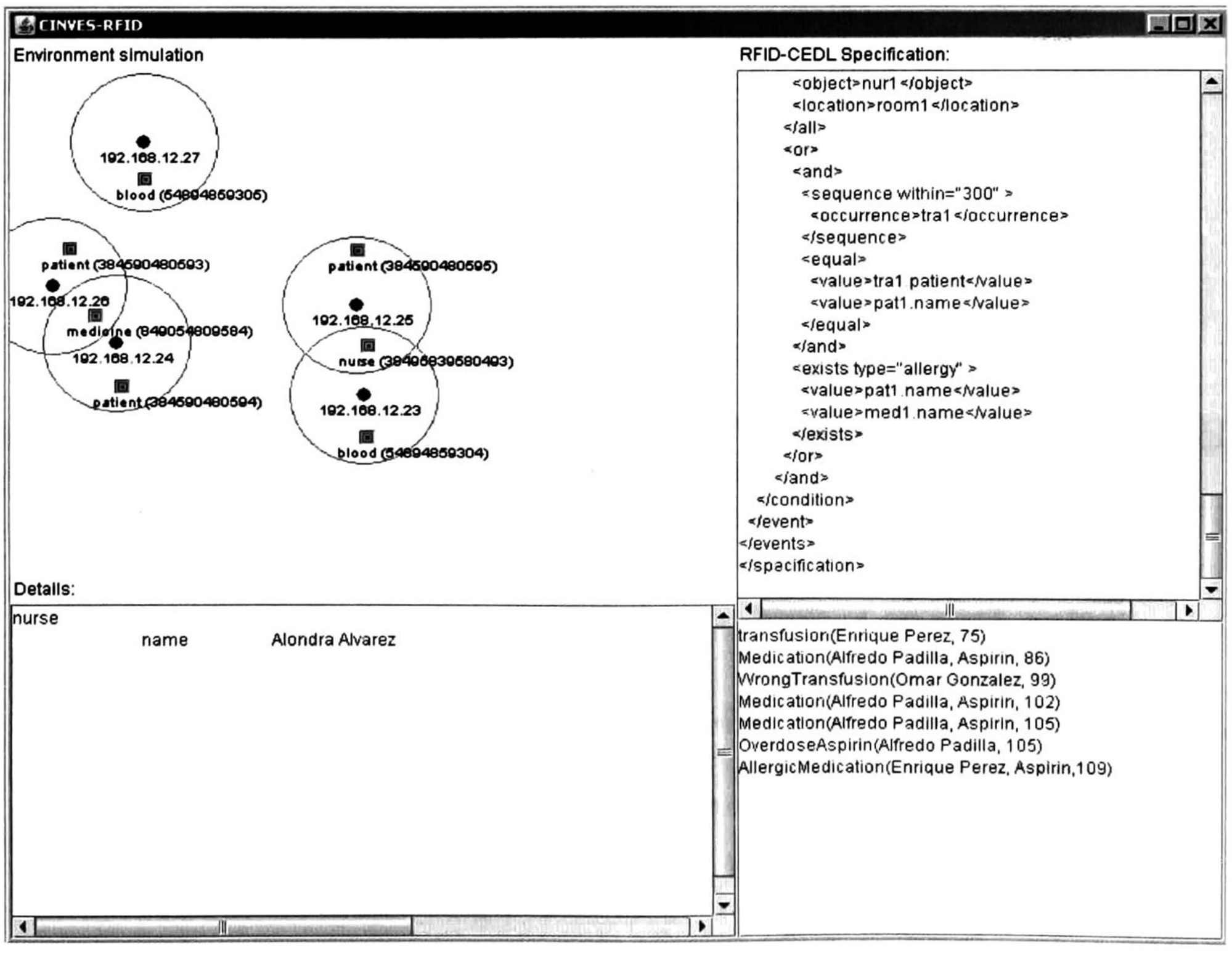


Figure 4.9: Simulation in RFID-CES.

4.2 Study Case. Managing RFID Events in a Hospital Environment

In this section we include some examples of use cases for a hospital environment. Use cases presented in this section have been selected with the purpose of showing usage and effectivity of the operators proposed in our approach. In addition, all use cases have been simulated using RFID-CES and all of them have been correctly recognized by the simulator.

4.2.1 Medication

Suppose a medication can be provided by a physician or by a nurse. Under this assumption, a medication would be recognized whenever a patient, a medicine, and either a physician or a nurse are situated in the same room. This use case shows usage of two location operators (all, any) and a boolean operator (and). This event can be expressed using RFID-CEDL as follows:

```
<event name="medication">
  <var class="nurse" name="nur1" />
  <var class="patient" name="pat1" />
  <var class="medicine" name="med1" />
  <var class="physician" name="phy1" />
  <fields>
  <string name="patient" value="pat1.name" />
  <string name="medicine" value="med1.name" />
  </fields>
  <condition>
      <and>
        <all>
          <object>pat1</object>
          <object>med1</object>
          <location>room1</location>
        </all>
        <any>
          <object>nur1</object>
          <object>phy1</object>
          <location>room1</location>
        </any>
      </and>
  </condition>
</event>
```

For this example it is necessary a previous definition of some classes (nurse, patient, medicine, physician), and a location (room1).

4.2.2 Allergic Medication

For this use case we will make the same assumption that in the previous use case. However, for showing the usage of the existence operator, we define an allergic medication extending the condition of a simple medication. Such extension is a search for an allergy into additional information. This event can be expressed using RFID-CEDL as follows:

```
<event name="wrongmedication">
  <var class="nurse" name="nur1" />
  <var class="patient" name="pat1" />
  <var class="medicine" name="med1" />
  <var class="physician" name="phy1" />
  <fields>
  <string name="patient" value="pat1.name" />
  <string name="medicine" value="med1.name" />
  </fields>
  <condition>
      <and>
        <all>
          <object>pat1</object>
          <object>med1</object>
          <location>room1</location>
        </all>
        <any>
          <object>nur1</object>
          <object>phy1</object>
          <location>room1</location>
        </any>
        <exists type="allergy">
          <value>pat1.name</value>
          <value>med1.substance</value>
        </exists>
      </and>
  </condition>
</event>
```

4.2.3 Overdose

With this use case we show the usage of the sequence operator and of a comparison operator. Suppose that an overdose of a substance called "digoxin" can be mortal, and that an overdose happens if the patient is medicated two times within 24 hours. This event can be expressed as follows:

```
<event name="digoxinoverdose">
  <var class="medication" name="med1" />
  <var class="medication" name="med2" />
  <fields>
    <string name="patient" value="med1.patient" />
  </fields>
  <condition>
    <and>
```

```
    <sequence within="86400">
      <occurrence>med1</occurrence>
      <occurrence>med2</occurrence>
  </sequence>
    <equal>
      <value>med1.medicine</value>
      <value>digoxin</value>
    </equal>
    <equal>
      <value>med2.medicine</value>
      <value>digoxin</value>
    </equal>
    <equal>
      <value>med1.patient</value>
      <value>med2.patient</value>
    </equal>
  </and>
 </condition>
</event>
```

Notice that periods of time in RFID-CEDL must be provided in seconds, thus the period of 24 hours is expressed as 86400.

4.2.4 Dangerous drug interaction

This use case illustrates the combination operator. Suppose there are two medicines called "digoxin" and "verapimil" which are dangerous if provided to the same patient within one hour. This situation can be detected by defining an event as follows:

```
<event name="dangerouscombination">
  <var class="medication" name="med1" />
  <var class="medication" name="med2" />
  <fields>
    <string name="patient" value="med1.patient" />
  </fields>
  <condition>
    <and>
      <combination within="3600">
        <occurrence>med1</occurrence>
        <occurrence>med2</occurrence>
      </combination>
      <equal>
        <value>med1.medicine</value>
        <value>digoxin</value>
      </equal>
      <equal>
        <value>med2.medicine</value>
        <value>verapimil</value>
      </equal>
      <equal>
        <value>med1.patient</value>
        <value>med2.patient</value>
      </equal>
    </and>
```

```
    </condition>
  </event>
```

4.2.5 Wrong Transfusion

Suppose that blood transfusions are performed exclusively by nurses. A wrong blood transfusion could be recognized if the blood type of the patient does not match with the type of the blood bag. This use case illustrates the usage of the not operator, and of a comparison operator.

```
<event name="wrongtransfusion">
  <var class="blood" name="blo1" />
  <var class="nurse" name="nur1" />
  <var class="patient" name="pat1" />
  <fields>
    <string name="patient" value="med1.patient" />
  </fields>
  <condition>
    <and>
      <all>
        <object>blo1</object>
        <object>nur1</object>
        <object>pat1</object>
        <location>room1</location>
      </all>
      <not>
        <equal>
          <value>blo1.type</value>
          <value>pat1.bloodtype</value>
        </equal>
      </not>
    </and>
  </condition>
</event>
```

This example could be expressed using the operator *different* ($\neq$), however it was espressed as *not equal* for showing the *not* operator.

4.3 Comparison

In this section, we compare out proposal with other approaches with similar objectives. Such approaches have been presented in chapter 2. For comparing approaches we take in account the following features:

- RFID. The approach is focused in RFID information management.

Table 4.1: Comparison of approaches.

Approach	RFID	History	Compatibility	Error	Language	Functions
Proposed approach	Yes	Yes	Yes	No	Yes	No
Proact	Yes	No	No	No	No	No
RCSM	No	Yes	No	No	Yes	Yes
SASE	Yes	Yes	No	No	Yes	No
PEEX	Yes	No	No	Yes	Yes	No

- History. Allows using past information for constructing complex events from simpler ones.
- Compatibility. Compatibility with current RFID architectures and components.
- Error. Management of radio frequency communication errors.
- Language. Includes a language for event definition.
- Functions. Includes general purpose built-in functions (e.g. average, current date, current time, etc).

4.4 Conclusions

In this chapter we have presented a tool which implements our proposal by simulating RFID-Environments. We have also presented some use cases for a hospital environment which shows the capability of our language of being used in real scenarios. Finally, we have defined a set of attributes and we have compared our proposal with similar approaches using such attributes.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

In this thesis we have presented a new approach for high-level event management in RFID systems. We have proposed a new form of interaction between enterprise applications and existing RFID components. Enterprise applications developed under our proposal defines a set of interesting high-level events and are notified only when such events occurs. This interaction releases enterprise applications of performing RFID data analysis and also reduces network traffic between middleware and enterprise applications.

We have presented and formalized a set of operators for defining events starting with simple RFID observations and we have proposed a declarative language called RFID-CEDL for expressing such set of operators. Such set of operators allows defining events using relations between positions of several objects, a history of event occurrences, boolean operators, and comparison operators.

We have also designed a new layer in charge of event recognition. We have described the interaction between this layer and components compliant with currently adopted standards. We have also presented the set of techniques and data structures utilized by this layer for recognizing events by evaluating each of the proposed operators.

In addition, we have developed a tool for RFID environment simulation. Such tool, called RFID-CES, allows the generation of specifications compliant to RFID-CEDL. We have used such tool for applying our concepts in use cases for a hospital application. Such use cases have showed the suitability of our proposal for real scenarios.

5.2 Future Work

There are several aspects which can be improved in this work.

- Take in account errors. Include a model for management of communication errors, which are usual while working with RFID.
- Filtering of evaluated threes. To propose a more efficient mechanism for filtering the set of events which are evaluated each time an actor changes its position.
- Include built-in functions. Include built-in functions in our language, and also a mechanism for including functions developed by the user.

Appendix A

RFID-CEDL Schema

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- Start of the specification structure -->

  <xs:element name="specification">
    <xs:complexType>
      <xs:sequence>

<!-- Section for defining classes -->

      <xs:element minOccurs="0" maxOccurs="1" name="classes">
        <xs:complexType>
          <xs:sequence>

<!-- Structure of each class -->

            <xs:element minOccurs="1" maxOccurs="unbounded" name="class">
              <xs:complexType>
                <xs:sequence>

<!-- Each class is composed by zero or more fields  -->

                  <xs:element minOccurs="0" maxOccurs="unbounded" name="field">
                    <xs:complexType>

<!-- Each field must be defined with a type and a name -->

                      <xs:choice minOccurs="1" maxOccurs="1">
                        <xs:element minOccurs="1" maxOccurs="1" name="int">
                          <xs:complexType>
                            <xs:attribute name="name" type="identifier" use="required" />
                          </xs:complexType>
                        </xs:element>
                        <xs:element minOccurs="1" maxOccurs="1" name="string">
                          <xs:complexType>
                            <xs:attribute name="name" type="identifier" use="required" />
                          </xs:complexType>
                        </xs:element>
                        <xs:element minOccurs="1" maxOccurs="1" name="float">
                          <xs:complexType>
```

```
    <xs:attribute name="name" type="identifier" use="required" />
  </xs:complexType>
</xs:element>
<xs:element minOccurs="1" maxOccurs="1" name="date">
  <xs:complexType>
    <xs:attribute name="name" type="identifier" use="required" />
  </xs:complexType>
</xs:element>
<xs:element name="class">
                        <xs:complexType>
                          <xs:attribute name="type" type="identifier" use="required" />
                          <xs:attribute name="name" type="identifier" use="required" />
                        </xs:complexType>
                      </xs:element>
                    </xs:choice>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
              <xs:attribute name="name" type="identifier" use="required" />
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

<!-- Section for defining instances -->

    <xs:element minOccurs="0" name="instances">
      <xs:complexType>
        <xs:sequence>

<!-- Structure of each instance -->

          <xs:element minOccurs="1" maxOccurs="unbounded" name="instance">
            <xs:complexType>

<!-- Each instance contains a list of zero or more attributes-->

              <xs:sequence>
                <xs:element minOccurs="0" maxOccurs="unbounded" name="attribute" />
              </xs:sequence>

<!-- Each instance must contain an epc(unique id) -->

              <xs:attribute name="epc" type="xs:string" use="required" />

<!-- Each instance must be defined as an instance of a predefined class -->

              <xs:attribute name="class" type="identifier" use="required" />
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

<!-- Section for defining additional information -->

    <xs:element minOccurs="0" name="information">
      <xs:complexType>
        <xs:sequence>
```

```
<!-- Structure of each data -->

          <xs:element minOccurs="1" maxOccurs="unbounded" name="data">
            <xs:complexType>

    <!-- Each data contains a list of zero or more attributes -->

              <xs:sequence>
                <xs:element minOccurs="0" maxOccurs="unbounded" name="attribute" />
              </xs:sequence>

        <!-- Each data must be defined as an instance of a predefined class -->

                <xs:attribute name="class" type="identifier" use="required" />
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>

<!-- Section for defining locations -->

      <xs:element minOccurs="0" name="locations">
        <xs:complexType>
          <xs:sequence>

    <!-- Structure of each location -->

            <xs:element minOccurs="1" maxOccurs="unbounded" name="location">
              <xs:complexType>

        <!-- Each location must contain at least one reader -->

                <xs:sequence>
                  <xs:element minOccurs="1" maxOccurs="unbounded" name="reader">
                    <xs:complexType>
                      <xs:attribute name="name" type="ip" use="required" />
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>

        <!-- Each location has a name -->

                <xs:attribute name="name" type="identifier" use="required" />
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>

<!-- Section for defining events -->

    <xs:element minOccurs="0" name="events">
      <xs:complexType>
        <xs:sequence>

<!-- Structure of each event -->

          <xs:element minOccurs="1" maxOccurs="unbounded" name="event">
            <xs:complexType>
```

```
<!-- Each event has zero or more variables -->

          <xs:sequence>
            <xs:element minOccurs="0" maxOccurs="unbounded" name="var">
              <xs:complexType>

    <!-- Each variable has a class -->

                  <xs:attribute name="class" type="identifier" use="optional" />

    <!-- Each variable has a name -->

                  <xs:attribute name="name" type="identifier" use="required" />
              </xs:complexType>
            </xs:element>

<!-- Each event has zero or more fields -->

            <xs:element minOccurs="0" maxOccurs="1" name="fields">
              <xs:complexType>

    <!-- Each field has a name, a value and a type -->

                <xs:choice minOccurs="1" maxOccurs="unbounded">
                  <xs:element name="string">
                    <xs:complexType>
                      <xs:attribute name="name" type="identifier" use="required" />
                      <xs:attribute name="value" type="xs:string" use="required" />
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="int">
                    <xs:complexType>
                      <xs:attribute name="name" type="identifier" use="required" />
                      <xs:attribute name="value" type="xs:int" use="required" />
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="float">
                    <xs:complexType>
                      <xs:attribute name="name" type="identifier" use="required" />
                      <xs:attribute name="value" type="xs:float" use="required" />
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="date">
                    <xs:complexType>
                      <xs:attribute name="name" type="identifier" use="required" />
                      <xs:attribute name="value" type="xs:date" use="required" />
                    </xs:complexType>
                  </xs:element>
                </xs:choice>
              </xs:complexType>
            </xs:element>

<!-- Structure of the event condition which is a boolean expression -->

              <xs:element name="condition" type="boolExp" />
            </xs:sequence>

    <!-- Each event has a name -->

              <xs:attribute name="name" type="identifier" use="required" />
          </xs:complexType>
```

```
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:simpleType name="identifier">
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z]([a-zA-Z0-9])*" />
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="value">
  <xs:simpleContent>
    <xs:extension base="xs:string" />
  </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="ip">
  <xs:restriction base="xs:string" />
</xs:simpleType>

  <!-- Each boolean expression is defined as an operator (which is the main operator of the condition)-->

<xs:complexType name="boolExp">

      <!-- Such operator is taken from the set of operators included in our approach -->

  <xs:choice>
    <xs:element name="and" type="and" />
    <xs:element name="or" type="or" />
    <xs:element name="not" type="not" />
    <xs:element name="greater" type="greater" />
    <xs:element name="greaterequal" type="greaterequal" />
    <xs:element name="less" type="less" />
    <xs:element name="lessequal" type="lessequal" />
    <xs:element name="equal" type="equal" />
    <xs:element name="different" type="different" />
    <xs:element name="all" type="all" />
    <xs:element name="any" type="any" />
    <xs:element name="sequence" type="sequence" />
    <xs:element name="combination" type="combination" />
    <xs:element name="exists" type="exists" />
    <xs:element name="none" type="none" />
  </xs:choice>
</xs:complexType>

<!-- Structure of the all operator -->

<xs:complexType name="all">
  <xs:sequence>

  <!-- Must contain at least one object(variable name) -->

    <xs:element minOccurs="1" maxOccurs="unbounded" name="object" type="identifier" />

  <!-- Must be specified a location-->

    <xs:element name="location" type="identifier" />
  </xs:sequence>
</xs:complexType>
```

```
<!-- Structure of the any operator -->

<xs:complexType name="any">
  <xs:sequence>

  <!-- Must contain at least one object(variable name) -->

    <xs:element minOccurs="1" maxOccurs="unbounded" name="object" type="identifier" />

  <!-- Must be specified a location-->

    <xs:element name="location" type="identifier" />
  </xs:sequence>
</xs:complexType>

<!-- Structure of the none operator -->

<xs:complexType name="none">
  <xs:sequence>

  <!-- Must contain at least one object(variable name) -->

    <xs:element minOccurs="1" maxOccurs="unbounded" name="object" type="identifier" />

  <!-- Must be specified a location-->

    <xs:element name="location" type="identifier" />
  </xs:sequence>
</xs:complexType>

<!-- Structure of the sequence operator -->

<xs:complexType name="sequence">

  <!-- Must contain at least one event(variable name) -->

  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element minOccurs="1" maxOccurs="1" name="occurrence" type="identifier" />
  </xs:sequence>

  <!-- Must be specified a period of time (in seconds) for the sequence to be recognized -->

  <xs:attribute name="within" type="xs:int" />
</xs:complexType>

<!-- Structure of the combination operator -->

<xs:complexType name="combination">

  <!-- Must contain at least one event(variable name) -->

  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element minOccurs="1" maxOccurs="1" name="occurrence" type="identifier" />
  </xs:sequence>

  <!-- Must be specified a period of time (in seconds) for the combination to be recognized -->

  <xs:attribute name="within" type="xs:int" />
</xs:complexType>

<!-- Structure of the exists operator -->
```

```
<xs:complexType name="exists">

  <!-- Must contain at least one value to search -->

  <xs:sequence>
    <xs:element maxOccurs="unbounded" name="value" type="value" />
  </xs:sequence>

  <!-- Must be defined a class for searching into corresponding data -->

  <xs:attribute name="type" type="identifier" />
</xs:complexType>

<!-- Structure of comparison operators -->

<xs:complexType name="greater">
  <xs:sequence>
    <xs:element name="value1" type="xs:string" />
    <xs:element name="value2" type="xs:string" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="greaterequal">
  <xs:sequence>
    <xs:element name="value1" type="xs:string" />
    <xs:element name="value2" type="xs:string" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="less">
  <xs:sequence>
    <xs:element name="value1" type="xs:string" />
    <xs:element name="value2" type="xs:string" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="lessequal">
  <xs:sequence>
    <xs:element name="value1" type="xs:string" />
    <xs:element name="value2" type="xs:string" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="equal">
  <xs:sequence>
    <xs:element minOccurs="2" maxOccurs="unbounded" name="value" type="xs:string" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="different">
  <xs:sequence>
    <xs:element name="value1" type="xs:string" />
    <xs:element name="value2" type="xs:string" />
  </xs:sequence>
</xs:complexType>

<!-- Structure of the boolean operators -->

<xs:complexType name="and">

<!-- Each operator can be a subexpression (another condition) -->

  <xs:choice minOccurs="2" maxOccurs="unbounded">
    <xs:element name="and" type="and" />
    <xs:element name="or" type="or" />
```

```
<xs:element name="not" type="not" />
<xs:element name="greater" type="greater" />
<xs:element name="greaterequal" type="greaterequal" />
<xs:element name="less" type="less" />
<xs:element name="lessequal" type="lessequal" />
<xs:element name="equal" type="equal" />
<xs:element name="different" type="different" />
<xs:element name="all" type="all" />
<xs:element name="any" type="any" />
      <xs:element name="sequence" type="sequence" />
      <xs:element name="combination" type="combination" />
      <xs:element name="exists" type="exists" />
      <xs:element name="none" type="none" />
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="or">
    <xs:choice minOccurs="2" maxOccurs="unbounded">
      <xs:element name="and" type="and" />
      <xs:element name="or" type="or" />
      <xs:element name="not" type="not" />
      <xs:element name="greater" type="greater" />
      <xs:element name="greaterequal" type="greaterequal" />
      <xs:element name="less" type="less" />
      <xs:element name="lessequal" type="lessequal" />
      <xs:element name="equal" type="equal" />
      <xs:element name="different" type="different" />
      <xs:element name="all" type="all" />
      <xs:element name="any" type="any" />
      <xs:element name="sequence" type="sequence" />
<xs:element name="combination" type="combination" />
      <xs:element name="exists" type="exists" />
      <xs:element name="none" type="none" />
    </xs:choice>
  </xs:complexType>
  <xs:complexType name="not">
    <xs:choice minOccurs="1" maxOccurs="1">
      <xs:element name="and" type="and" />
      <xs:element name="or" type="or" />
      <xs:element name="not" type="not" />
      <xs:element name="greater" type="greater" />
      <xs:element name="greaterequal" type="greaterequal" />
      <xs:element name="less" type="less" />
      <xs:element name="lessequal" type="lessequal" />
      <xs:element name="equal" type="equal" />
      <xs:element name="different" type="different" />
      <xs:element name="all" type="all" />
      <xs:element name="any" type="any" />
      <xs:element name="sequence" type="sequence" />
<xs:element name="combination" type="combination" />
      <xs:element name="exists" type="exists" />
      <xs:element name="none" type="none" />
    </xs:choice>
  </xs:complexType>
</xs:schema>
```

Bibliography

[1] R. Want B. Schilit, N. Adams. Context-aware computing applications. *IEEE Workshop on Mobile Computing Systems and Applications*, 1994.

[2] M. Lampe C. Floerkemeier. Rfid middleware design - addressing application requirements and rfid constraints. *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*, pages 219 – 224, October 2005.

[3] Gregory D. Abowd Daniel Salber, Anind K. Dey. The context toolkit: Aiding the development of context-enabled applications. *Proceedings of CHI'99*, pages 434–441, May 1999.

[4] EPCglobal. *EPCglobal Tag Data Standards Version 1.3.1*, March 2006.

[5] EPCglobal. *The EPCglobal Architecture Framework Version 1.2*, September 2007.

[6] EPCglobal. *EPCglobal Tag Data Standards Version 1.3.1*, September 2007.

[7] EPCglobal. *The Application Level Events (ALE) Specification Version 1.1*, February 2008.

[8] Bill Glover. *RFID Essentials*. O'Reilly, January 2006.

[9] Xiaolei Li D. Klabjan H. Gonzalez, J. Han. Warehousing and analyzing massive rfid data sets. *Proceedings of the 22nd International Conference on Data Engineering*, page 83, 2006.

[10] A. Lykke-Olesen R. Nielsen K. Halskov J. Bardram, C. Bossen. Virtual video prototyping of pervasive healthcare systems. *Proceedings of the 4th conference on Designing interactive systems: processes, practices, methods, and techniques*, pages 167 – 177, June 2002.

[11] H. Sheng L. Dong, D. Wang. Design of rfid middleware based on complex event processing. *Cybernetics and Intelligent Systems, 2006 IEEE Conference on*, pages 1 – 6, December 2004.

[12] David Luckham. *The Power of Events: an Introduction to Complex Event Processing in Distributed Enterprise Systems.* Addison-Wesley, August 2007.

[13] M. Perkowitz-D. Patterson D. Fox H. Kautz D.Hahnel M. Fishkin, K. Philipose. Inferring activities from interactions with objects. *Pervasive Computing*, 3:50 – 57, October 2004.

[14] Florian Rosenberg Matthias Baldauf, Schahram Dustdar. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing 2007 - Vol. 2, No.4*, pages 263–277, 2007.

[15] Brenda M. Michelson. *Event-Driven Architecture Overview*, February 2006.

[16] D. Suciu N. Khoussainova, M. Balazinska. High-performance complex event processing over streams. *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 407 – 418, June 2006.

[17] D. Suciu N. Khoussainova, M. Balazinska. Peex: Extracting probabilistic events from rfid data. *Proceedings of the 24th International Conference on Data Engineering*, 2008.

[18] C. Laudy-H. Soubaras N. Museux, J. Mattioli. Complex event processing approach for strategic intelligence. *Information Fusion, 2006 9th International Conference on*, pages 1 – 8, July 2006.

[19] D. Guinard P. Fuhrer. Building a smart hospital using rfid technologies: Use cases and implementation. *Proceedings of European Conference on eHealth 2006*, pages 131 – 142, October 2006.

[20] Xue Li R. Derakhshan, M. Orlowska. Rfid data management: Challenges and opportunities. *IEEE International Conference on RFID 2007*, pages 175 – 182, March 2007.

[21] C. M. Roberts. Radio frequency identification (rfid). *Computers & security*, pages 18–26, February 2006.

[22] D. Huang-P. In S. Yau, Y. Wang. Situation-aware contract specification language for middleware for ubiquitous computing. *Distributed Computing Systems, 2003. FTDCS 2003. Proceedings*, pages 93 – 99, May 2003.

[23] D. Huang-Y. Yao S. Yau, H. Gong. Support for situation awareness in trustworthy ubiquitous computing application software. *Software-Practice & Experience*, 36:893 – 921, July 2006.

[24] Y. Wang-B. Wang S. Gupta S. Yau, F. Karim. Reconfigurable context-sensitive middleware for pervasive computing. *Pervasive Computing*, 1:33 – 40, July 2002.

[25] D. Zhang T. Gu, K. Pung. Toward an osgi-based infrastructure for context-aware applications. *Pervasive Computing*, 3:66 – 74, October 2004.

[26] T. Winograd. Architectures for context. *Human-Computer Interaction*, 16 No. 2:401–419, 2001.

[27] P. Liu-C. Zaniolo S. Liu Y. Bai, F. Wang. Rfid data processing with a data stream query language. *IEEE 23rd International Conference on Data Engineering, 2007*, pages 1184 – 1193, April 2007.

[28] D. Gyllstrom Y. Diao, N. Immerman. Sase+: An agile language for kleene closure over event streams. *UMass Technical Report*, 2007.

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL I.P.N.
UNIDAD GUADALAJARA

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó la tesis

Control de Sistemas usando RFID - Control Systems Using RFID

del (la) C.

Omar Alfredo GONZÁLEZ PADILLA

el día 22 de Agosto de 2008.

Dr. Luis Ernesto López Mellado
Investigador CINVESTAV 3B
CINVESTAV Unidad Guadalajara

Dr. Félix Francisco Ramos Corchado
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Mario Angel Siller González
Pico
Investigador CINVESTAV 2A
CINVESTAV Unidad Guadalajara