Dynamic Generation of Context Rules

Waltenegus Dargie

Department of Computer Networks, Faculty of Computer Science, Dresden University of Technology
Hans-Grunding-str. 25,
01062 Dresden, Germany
dargie@rn.inf.tu-dresden.de

Abstract. This paper presents a scheme for the dynamic generation of context rules which are useful for modifying the behaviour of mobile devices according to the social and physical settings of their users. Existing context-aware systems employ a pool of predefined rules which will be executed whenever a context of interest is sensed and captured. Defining rules at design time, however, has the following limitations: (1) The system designer should identify the set of context primitives which describe a context of interest as accurately as possible; (2) the various states of each context primitive should be predetermined and mapped to sensory data, which often requires experience or expertise; (3) the availability of mechanisms for capturing the context primitives is presupposed; if instead of the specified context primitives other context primitives are discovered, which may equally describe a similar situation, the system does not react to them, unless of course, all possible alternatives are foreseen; and (4) the desired behaviour itself may not be a priori known, as it is specific to the user. To facilitate a flexible use of context rules, they should be generated dynamically. The scheme proposed in this paper associates decision events – signifying the activities of a user – with a set of context primitives which are acquired at the time the decision events are produced. From the decision-context associations, context rules are generated. The Context-Aware E-Pad will be introduced to illustrate the scheme proposed.

1. Introduction

In context-aware computing, mobile devices and applications adjust their behaviour in accordance with the social and physical settings wherein a computing task takes place [1]. Adjustment in behaviour implies either provision of suitable services or modification of device configurations [2]. Often a context-aware system constantly senses the context of a user and updates a knowledge base. A context reasoning engine receives a set of rules from an application (defined at the time the application is designed) to manipulate the knowledge base; if the captured context satisfies predefined criteria, an action specific to the application will be executed to transform one world model to some other world model [3].
Defining context rules at design time and embedding these rules into the business model of an application has the following limitations:

- The application reacts only to the context primitives which are expressed in a rule; if different context primitives express a situation of interest – thereby leading to the modification of the same behaviour – either the application developer has to foresee all these possibilities at design time or the application will fail to capture the situation and to react to it.
- The application developer specifies a mechanism to detect the specified context; in the absence of the presupposed mechanism, there will be no way for the application to exploit other resources. In a ubiquitous computing environment, the availability of mechanisms for detecting a context may not be foreseen, as resources are highly dynamic.
- Existential quantifiers required for setting criteria for evaluating a context rule must be set at design time, which usually require a priori knowledge, and in some cases, expertise. For example, as in the case[1]: 'When environment loudness is above 4 dB, set ringing tone volume to 2.5'. In order to construct such a rule, the meaning of 4 dB as well as the reference used to measure noise in dB should be known.

To alleviate these prohibitions, we provide support for the dynamic generation of context rules. The way we achieve this goal is by learning the activities of a user, and by mapping his activities onto a multidimensional context space. Assumption to this is that the user makes certain decisions habitually, or with some predictive regularity. Where there is no regularity in the activities of the user, learning cannot take place, or requires substantial computational and storage overhead.

The rest of this paper is organised as follows: in section 2, a brief assessment of related work is given; in section 3, elements of a context rule are introduced; in section 4, the architecture for context processing and rule generation is proposed; in section 5, the event processing components and event expression semantics are discussed; in section 6, the Context-Aware E-Pad is demonstrated; and finally, in section 7, a brief summary is given.

2. Related Work

Wang et al propose Semantic Space [5], a generic infrastructure for reasoning about higher-level contexts. It consists of context wrappers, a knowledge base, an aggregator, a context query engine, and a context reasoner. Context wrappers obtain raw context data from various sources, and transform the data into context markups. Context elements are represented as ontology instances and associated properties that applications can easily interpret. Among various contexts, the researchers identify three classes of real-world objects – user, location, and computing entities, and one class of conceptual objects – activity. These classes of contexts characterise smart spaces. An Aggregator discovers context wrappers and gathers context markups from them, and then asserts the markups into the context knowledge base, which it updates

[1] The example is taken from [4].

whenever a context event occurs. A context knowledge base stores extended context ontology for a particular space and the context markups that are provided by users or gathered from context wrappers. The KB links the context ontology and markups in a single semantic model and provides interfaces for the context query engine and context reasoner to manipulate correlated contexts. The context query engine provides support for applications to query the context KB. Finally, the context reasoner infers abstract higher-level contexts from basic sensed contexts. To employ general purpose logic based reasoning engines, Semantic Space explicitly represents all contexts. Applications submit a set of rules to the context reasoner, which applies them to infer higher-level contexts.

Gu et al. propose the SOCAM [6] architecture, which shares similar properties with Semantic Space, and consists of a context provider, a context interpreter, a context database and a service locating component. A context provider provides context abstraction to separate the low-level context sensing from the higher-level context manipulation. Each context provider registers at a service registry by using the service locating component. The context interpreter consists of a Context Reasoner and a Knowledge Base to carry out a logic-reasoning to obtain a higher-level context. The Context Reasoner has the functionality of providing deduced contexts based on sensed contexts; the KB contains a context ontology and the instances of (assertions on) this ontology. The researchers distinguish between defined and sensed contexts, which are similar to explicit and implicit contexts as defined by Schmidt et al. [7]. In the case of defined contexts, the user may predefine the instances. A sensed context is an assertion of a context acquired from a context provider into the Knowledge Base. The user-defined rule-based reasoning provides forward chaining, backward chaining and a hybrid execution model. The forward-chaining rule engine employs the standard RETE algorithm [8]; the backward-chaining rule engine employs a logic-programming engine similar to Prolog engines; and a hybrid execution mode performs reasoning by combining both forward-chaining and backward-chaining engines.

One obvious drawback of these frameworks is that they do not deal with uncertainty. The various primitive contexts gathered from context wrappers, context providers, and context acquisition modules are taken as reliable evidences. This however, is quite unrealistic, since uncertainty is always associated with sensed data and the context extracted from these data are prone to be inexact. Furthermore, the reliability of the sources delivering the data may not be ascertained, and the a priori knowledge required to manipulate the sensed data may not be up to date, i.e. it may not reflect the reality represented by the reasoning world-model.

3. Elements of a Context Rule

We define two types of events: context events and decision events. Context events signify the occurrence of an interesting real-world situation. Decision events, on the other hand, correspond to the invocation of services inside mobile devices. Moreover, we distinguish between primitive events and composite events. Primitive events are

those predefined in the system; a mechanism for their detection is assumed to be available. Primitive events include temporal events, atomic context events such as a room temperature being 20°C or the light intensity of a place being 1000 Lux. Events created by the invocation of action subroutines to perform specific actions inside a mobile device give rise to the occurrence of decision events, which are primitive events.

A primitive event, *E*, is expressed as a predicate with four arguments. The event expression includes the event's name, the event's subject (the one which is responsible for the occurrence of the event), the event's value and the time of occurrence (timestamp). Thus, a primitive event is expression as:

$$E(t) \equiv (\exists n)\,(\exists s)(\exists v)(\exists ts)(event(n,\ s,\ v,\ ts) \wedge (t = ts)) \tag{1}$$

In equation (1), the existential quantifiers *n*, *s*, *v*, and *ts* are, respectively, the event type, the subject to which the event refers, the value of the event, and the timestamp; *v* can be a numerical value or an event object. Equation (2) shows a valid instance of a temperature context event.

$$E_T(t) \equiv event(temp,\ RoomA,\ 20°C,\ 10AM) \tag{2}$$

Once a primitive event is specified, it is possible to define an atomic context rule:

$$(\forall t)(\forall i)(\forall k)(\exists j)(\exists l)(\exists m)(E_T(t) \wedge (i < j) \supset heater(k) \wedge (l \le k \le m)) \tag{3}$$

Equation (3) states that if a temperature event is detected the value of which is below the threshold *j*, heater *k* should be adjusted such that the value of *k* should be between *l* and *m*.

Primitive events discussed so far are useful for modelling simple behaviours. For many context-aware applications, however, it is necessary to detect certain combinations of events in order to capture a dynamic real-world situation and perform suitable actions. Three composite event expressions are adopted from [9] to express composite events. These are: **ANY**, **SEQ**, and **APeriodic** events. The **ANY** event expression is a conjunction event which occurs when *m* out of a pool of *n* distinct events occurs, regardless of their order of occurrence The **SEQ** event expression is a sequence of events where the order of occurrence is preserved. The **APeriodic** (**A**) event expression is the occurrence of an event, E_2, within a closed time interval [E_1, E_3]. This composite event is a non-cumulative event; i.e., the event ***A*** is signalled every time E_2 is detected within the time interval started by E_1 and ended by E_3. If instead of E_2, **ANY** is used to express the **APeriodic** composite event, ***A*** will be signalled every time one or a combination of the events expressed in the **ANY** composite event is detected.

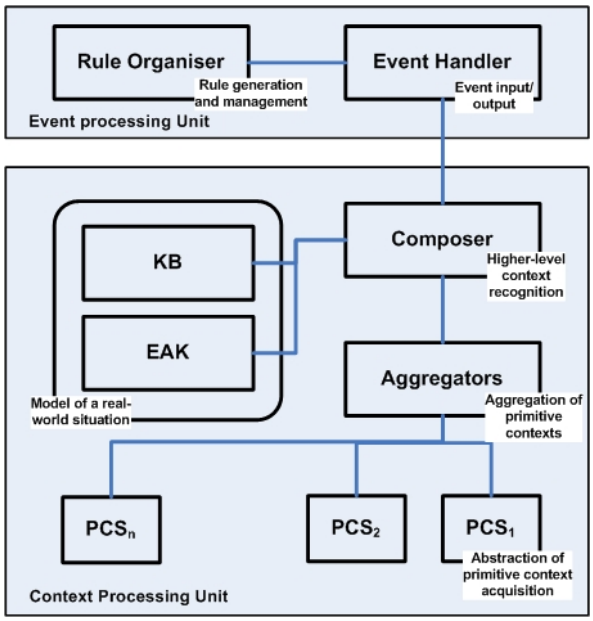

Figure 1: Architecture for context acquisition and processing.

4. Architecture

Two basic operations are required in order to dynamically generate context rules: capturing of a dynamic real-world situation and understanding of the user's preference in that situation. Consequently, the architecture we propose comprises two main units: the context acquisition and processing unit and the event processing unit.
The former is responsible for interacting with a variety of sensors measuring some physical parameters of a real-world situation, translating the sensory data into a desirable format, and performing various manipulations and comparisons in order to produce an expressive, higher-level context, which is an abstraction of a dynamic real-world situation. This unit consists of Primitive Context Servers (PCS), Aggregators, a Knowledge Base (KB), an Empirical Ambient Knowledge component (EAK), and a Composer. The second unit, the event processing unit, is responsible for generating context rules by associating a user's actions with a context in which these actions are carried out. This unit consists of an Event Handler (EH) and a Rule Organiser (RO). Figure 3 shows the complete architecture.
Although the main goal of this paper is to explain the dynamic generation of context rules, we will briefly introduce the context processing unit.

4.1. Primitive Context Server (PCS)

A PCS abstracts from other components, such as Aggregators, the details and complexities of extracting data from physical sensors. It transforms the raw sensory data into a meaningful context atom. In some cases, multiple features (context atoms) can be extracted from a single sensor. An atomic context maps directly to a real world object. The primitive context provided by a PCS are delivered along with a description of the sensing element, for the components employing the context to decide how to rate the context.

4.2. Aggregator

Often a piece of context primitive is not sufficient to appropriately model a real-world situation [10]. The real world is far too complex to be captured in complete detail by a single primitive. Therefore, several aspects of entities should be captured to reason about a dynamic real world situation. Moreover, the implication of inexact sensory data should also be taken into account. Subsequently, an aggregator is responsible for enhancing the quality of sensed data.

4.3. Knowledge Base

A priori knowledge of entities (places, devices, persons, etc) playing role in dynamic computing environments is useful both for modelling a real world situation and for appropriately interpreting sensor measurements. We distinguish between factual knowledge and knowledge based on beliefs. Facts reflect objective reality; they do not change or if they do, change only slowly over time. Beliefs, on the other hand, are not necessarily true, and they change quite frequently. The KB comprises a collection of facts constituting the vocabulary of an application domain, and a list of assertions about individual named entities in terms of this vocabulary. The vocabulary consists of concepts, which denote sets of entities, and relations, which denote binary relationships between these entities. In addition to atomic concepts and relations, the KB allows the building of complex descriptions of concepts and relations.

4.4. Empirical Ambient Knowledge

The KB accommodates facts only, however incomplete. There is little support to encode uncertain knowledge. On the other hand, empirical and heuristic knowledge of situations and people's perception of them is helpful to reason about dynamic real-world situations. However, this type of knowledge is rather based on beliefs established on past experiences and observations which cannot be described as facts, but as uncertain knowledge. The EAK quantitatively describes various aspects of an entity in terms of numerous atomic contexts (mapping to sensor data). The choice of a particular context atom depends on its capability in capturing a relevant (physical) aspect of the entity it describes. Other criteria include feasibility of measuring or recognising the context atom as accurately and unambiguously as possible.

4.5. Composer

Even with reliable sensory data, we may still not be able to capture a real world situation. The reason for this is a gap between what sensors can provide and what applica-

tions may need. Composition deals with a single higher-level context as an abstraction of more numerous context atoms. Once entities are modelled using the facts and beliefs stored in the KB and EAK, respectively, the composer accesses these components to retrieve useful knowledge which can serve as a reference to manipulate reports from sensors. The result is a higher-level context representing a real world situation.

A composer has three components. The logic based reasoning component (LBR) manipulates the facts, relations, and assertions in the KB. The probabilistic reasoning component (PR) deals with imprecise and possibly erroneous data in the EAK. We call it a probabilistic reasoning component for convenience of expression, although it may or may not be such. Our own implementation of the Composer is a probabilistic reasoning scheme based on Bayesian Networks. The decision unit reconciles the decisions of the LBR with that of the PR.

At times, the PR may not be able to make decision because two or more propositions have equal posterior probabilities of the differences in posterior probabilities are not significant enough. Consider a scenario of reasoning about the whereabouts of a person in the absence of any localisation sensor. Suppose the outcome of the PR indicates that it is equally probable for the person to be either on a CORRIDOR or inside a ROOM. Knowing from the KB that a CORRIDOR and a ROOM are mutually exclusive concepts, but a BUILDING subsumes both concepts, it is possible for the decision unit to decide that a person is inside a BUILDING instead of randomly selecting for either a COORIDOR or a ROOM.

5. Event Processing Components

In this section, we will introduce the Event Handler (EH) and the Rule Organiser (RO). These two components are required for the dynamic generation and execution of context rules which modify the behaviour of mobile devices or induce context-aware applications to execute suitable services.

5.1. Event Handler (EH)

The EH is a bridge between various context-aware applications and the context acquisition and processing block. It receives decision events from the applications and associates them with contexts which are acquired from the computing environment. These decision events are produced when an interesting context is captured. As mentioned earlier, context rules are generated when decision events are associated with a context wherein the events are produced. Decisions are the basic elements of context-aware applications with which useful services are executed and the behaviour of applications (devices) is modified. Each decision corresponds to an action routine the execution of which causes a system to take certain actions, transforming one world model to some other world model.

To associate decisions with a context, the EH subscribes to context-aware applications running on the mobile devices owned by a user to be notified of the occurrence

of decision events. When a decision event arrives, it checks whether the same event has previously occurred; if it has not occurred, it tags the event by giving it a universally unique identifier (UUID) and associates the event with a context describing the situation of the application, the user, the device, and/or the place. If the event has already occurred, the context association will occur in the same way, and additionally, the EH performs aggregation of decision-context associations. Aggregation deals with the merging and filtering of decision-context associations referring to the same decision. The aim is to identify the types and states of primitive contexts which best represent a situation of interest in which a decision event occurs.

5.2. Rule Organiser (RO)

The main task of a rule organiser is the generation of context rules from an aggregate of decision-context associations. Besides this, the RO manages context rules. Managing context rules is necessary if two or more contextual states trigger the same decision event; or if two or more decision events can be mapped to the same context. Example to the second case is switching a mobile phone to a vibration mode while loading an appropriate document whenever a user attends a meeting or a lecture. The sequential event operation is employed to describing decisions events referring to the same context.
There are times when a set of rules no longer represent a user's preferences. To cope with this situation, human intervention is required. Human intervention is followed by a process called unlearning – a process referring to the disintegration of context rules.

5.3. Observation Time

Decision-context aggregation takes place during an observation time, a time required by the EH to learn the behaviour of a mobile user. The EH subscribes to a single or multiple applications to be notified of the occurrence of decision events. The applications produce desirable events when a mobile user interacts with them by modifying their behaviour or by invoking certain services. An observation time is expressed as an ***Aperiodic*** composite event – it begins and ends with temporal events. Between these two events a number of decision events, and the association of these decision events with a context, occur.
The E_1 of an observation time can be either an absolute or a relative temporal event. An absolute temporal event corresponds to a unique time span on the time line with a clearly defined reference time and an offset time. A relative temporal event corresponds to a unique time span on the time line, but the reference event can be other than a temporal event. An example for an absolute temporal event is when a user specifies a time span to train a system; an example for a temporal event is when a user specifies the frequency of the occurrence of a particular decision event. During an observation time, the event flow is from applications to the EH down to the available Primitive Context Servers.

5.4. Execution Time

An observation time is followed by an execution time. During the execution time, context rules are fired. The condition for firing a rule is the occurrence of a desirable context signifying a situation of interest. During an execution time, the EH has already learned the behaviour of the user in a situation of interest, and therefore subscribes to the Composer to be notified of the detection of a specific context. The event flow is from Primitive Context Servers to the Composer and the EH and up to context-aware applications. The final event to be produced is a decision event.

6. Application

This section introduces the Context-Aware E-Pad (CAEP). The motivation behind the design and implementation of CAEP is to demonstrate (1) how applications can employ a higher-level context without the need to directly deal with its composition; and (2) the dynamic generation of context rules by associating a context with actions (decision events) performed by a mobile user. The reason for choosing CAEP as a demonstrator is its richness in the diversity of decision events which can be produced; hence, it is possible to associate each decision event with different situations.
CAEP is a context-aware application intended to be used in a mobile environment; it takes into account the context of a mobile user to create, load, send, receive, and delete confidential documents. Figure 2 shows the main components constituting CAEP.

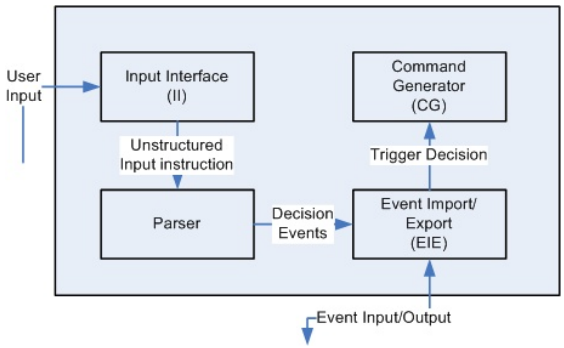

Figure 2: Components of the Context-Aware Electronic Document

The design concept of CAEP enables the dynamic generation of context rules from a user's activity. The essential features of CAEP are summarised as follows:

- During an observation time, CAEP publishes the user's activities (in the form of decision events) to the Event Handler (EH). This is useful for

associating the decision events with a set of contexts describing the situation in which the activities are taking place.

- From the association, the Rule Organiser generates a context rule or a subpart thereof, and stores the result.
- During an execution time, CAEP receives instructions from the EH to create, load, receive, send, and delete documents. Instructions from the EH are results of the occurrence of a context of interest.
- Moreover, CAEP offers a semantic-based input interface to a user. The interface enables a user to enter instructions from which decision events and context rules can be constructed.

6.1. Components of CAEP

The components of CAEP are: the Input Interface (II), the Parser, the Event Import/Export component (EIE), and the Command Generator (CG). Figure 3 shows the components of CAEP.

6.1.1. The Input Interface (II)

The input interface enables a user to enter instructions. To enable the dynamic generation of context rules, input instructions are managed centrally by the input interface. Instructions which are provided via the GUI produce decision events directly, so they need not be further processed. However, besides the graphical user interface, CAEP provides a semantic-based input interface for a user to enter textual, visual or audio instructions from which decision and context states are constructed. Such instructions require further processing, such as the conversion of audio and visual inputs into text, and the parsing and processing of the textual instructions. The II is responsible for transforming inputs of other modalities into textual instructions.

6.1.2. The Parser

The parser receives unstructured textual instruction from the II to generate decision events. Any instruction from a user should therefore consist of the decision event to be produced and the object (a document's name) to which the decision refers; additional elements of a user's instruction include contextual elements which can be associated with the decision events and persons to or from whom a note (document) should be sent or received.
The Parser employs the KB of the context processing unit in order to produce a meaningful decision event. This is carried out by passing a parsed instruction to the KB. Words such as *create*, *load*, *send*, and *receive*, when appear in an instruction, are automatically interpreted as decision event generators; as a result, the KB searches within the collection of parsed words for associated objects, namely, documents and

notes. A document refers to a self contained file, while a note refers to a part of a document. *Receive* and *send* instructions, besides being decision event generators, are associated with recipients and senders, respectively, i.e., to a human entity type. To illustrate decision event creation, consider the following instruction from a user.

Receive last week's DB2 note from Kathy.

When the II receives this instruction, it forwards it to the Parser without producing any decision event. The Parser parses the instruction word by word and issues a query request to the KB, which manipulates the concepts and roles it stores to create associations with the collection of words from the Parser. This is displayed in figure 3.

$$\begin{bmatrix} receive\cdots\ from \rightarrow event(decision); \\ last\ week \rightarrow context(time(between\ (27.09.2004,\ 01.10.2004))); \\ DB2 \rightarrow entity(document); \\ note \rightarrow partOf\ (document) \\ kathy \rightarrow entity(person) \end{bmatrix}$$

Figure 3: A Query result from the KB for producing a *receive* decision event

The Parser receives the result and produces the decision event shown in figure 4.

$$event\begin{bmatrix} receive[from = 'kathy';], \\ CAEP; \\ \begin{bmatrix} content = note; \\ subcontentOf = document; \\ createdOn = between(27.09.2004,\ 01.10.2004) \end{bmatrix}, \\ 05.10.2004, 10:10 \end{bmatrix}$$

Figure 4: A decision event generated by the Parser

6.1.3. The Event Import/Export Unit (EIE)

The event import/export (EIE) is responsible for dispatching decision events and receiving context events. Decision events are exported to the EH during an observation time, so that they can be associated with a context; a context event – or more precisely, the notification of its occurrence – is received from the EH during an execution time, so that CAEP performs an action associated with the context.

6.1.4. The Command Generator (CG)

All decision events which can be generated inside CAEP are registered at the Command Generator. This is useful for monitoring the activities of CAEP in a centralised manner. Only registered events can be triggered or associated with a context. A re-

quest to trigger an action should therefore arrive from the EIE referring to one of the registered decision types. When a user performs an action referring to one of the decision events, the CG notifies the EIE, which in turn notifies the EH; and the association of a decision event with a context ensues.

6.2. Decision-Context Association Semantics

This section illustrates how we trained CAEP to autonomously *load* a document suitable for a particular lecture a user attended. Besides loading a document, CAEP autonomously collected notes from one of the user's friends and insert them[2] in the appropriate document if the user happened to miss a previous lecture. An observation time of one month was set during which time four lecture sessions were conducted. The user loaded a DB2 document four times, but one of these decisions was made during a makeup class rather than the normal lecture sessions. The set of primitive contexts which were collected at the time the loading decisions have been made are shown in table 1.

TABLE 1: A Decision-Context Associations

1	[[Context: time: 21.09.2004, 9:57:13M] [Context: temperature: 20] [Context: sound_pressure: 13 dB] [Context: light_intensity: 720 Lux] [Context: RH: 50%]]
2	[[Context: time: 01.10.2004, 10:10:23 AM] [Context: RH: 46%] [Context: temperature: 22]]
3	[[Context: time: 05.10.2004, 10:06:47 AM] [Context: RH: 52%] [Context: temperature: 22] [Context: light_Intensity: 700 Lux]]
4	[[Context: time: 12.10. 2004, 10:01:07 AM] [Context: temperature: 23] [Context: sound_pressure: 11.98 dB] [Context: RH: 48%]]

RH = relative humidity; temperature is measured in degree centigrade. 20 micro Pascal is used as a reference to measure sound pressure.

TABLE 2: Summary of decision-context associations.

Decision	**Context**		
	Place	Time	Day
Load	ROOM	10:02 AM{±3}	Tuesday

[2] Duplicate documents were not allowed. If multiple notes refer to a similar context-decision association, they were considered to be duplicates.

The syntax to associate decisions with a context is given by:

$$decision(\ description\ )\ associatedWith$$
$$context(\ description\ )$$

Figure 5: A syntax for decision-context association

The Composer of the context processing unit computed a higher-level context by taking the primitive contexts listed in table 1; thus, what appeared in a context-decision association was the higher-level context *Room* instead of the set of primitive contexts (see figure 5), as a result of which the dynamics of primitive contexts is totally abstracted from CAEP. The Composer employed a self-organising Bayesian Network to reason about various places (rooms, corridors, buildings and outdoors)[3]. The Rule Organiser accumulated the decision-context associations until the observation time was over, and merged and filtered associations to generate an aggregate association. The summary is shown in table 2. The context rule generated out of the aggregate association is given by equation (4). This rule was used during an execution time to autonomously load the DB2 document whenever the user attended a DB2 lecture. Notice that existential quantifiers were dynamically identified from the primitive contexts.

$$decision\ (\ load\ (DB2)\ )\ associated\ With$$
$$context\ ((location\ (Room\ ));(time\ (September\ \ 21,\ 2004,9:57:23\ AM\ )))$$

Figure 6: A loading decision associated with contexts

$$(\forall d)(\forall t)(\forall l)\left(\begin{array}{l} day(d)\wedge time(t)\wedge loaction(l)\wedge \\ (d='Thursday')\wedge(09{:}59\le t\le 10{:}05)\wedge(l='Room') \end{array}\right)\supset(decision(load(pad\ (DB2)))) \quad (4)$$

6.2.1. Mapping Multiple Decisions

Two or more decision events can be associated with a similar context, signifying the occurrence of multiple activities in the same situation. For example, during the time of training CAEP to load the DB2 document, the user switched the ringing tone of his mobile phone to a vibration mode. Since the EH has subscribed to the application managing the mobile phone's ringing style, it was notified of the change in behaviour of the mobile phone while the user attended a lecture. The two decision events, namely, the *loading* event and the ringing style adjustment event, occurred together frequently; hence, they were associated to the same sets of contexts. The expression for associating multiple decisions with a similar context is given by:

[3] Composition of a Higher-Lever context is treated elsewhere [10].

$$decision(\ description\ )\ followedBy\ \ decision(\ description\ )\ \ associatedWith\ \ context(\ description\ )$$

Figure 7: Syntax for associating multiple decisions with a similar context

$$decision\ (\ ringerStyle(vibration)\ )\ \ followedBy\ \ \ decision\ (\ load\ (DB2))\ associated\ With\ context\ ((location\ (Room));(time(September\ \ 21,\ 2004,9{:}57{:}23\ \ AM\ )))$$

Figure 8: Multiple Decisions associated with a similar context

For the scenario we just described, the merged association looked like the one depicted in figure 8. The context rule which was generated from the aggregated decision-context association is given in equation 5.

$$(\forall d)(\forall t)(\forall l)\left(\begin{array}{l}(day(d)\wedge time(t)\wedge loaction(l)\wedge(d='Thursday')\wedge(09{:}59\le t\le 10{:}05)\wedge(l='Room'))\supset\\(decision\ (load\ (pad\ (DB2)))\wedge decision(ringerStyle(mobilePhone(vibration\ ))))\end{array}\right) \quad (5)$$

6.2.2. Managing Temporal Events

Temporal context could directly reveal a user's habit. However, due to the unpredictable nature of the user's actions, there is uncertainty in dealing with time. Additional knowledge is required to resolve this uncertainty, since inconsistency in time is application specific. For example, a user may not arrive at a lecture session exactly on a set time; he may arrive later or earlier than the time at which a lecture begins; but he cannot be earlier or later than the starting time by more than 90 minutes, considering 90 minutes the duration of a lecture. Therefore, this extra knowledge has been explicitly made known to the RO.

7. Discussion

Existing context-aware applications integrate rules which are defined at design time. It has been shown that defining context rules at design time has certain limitations including the need to identify the actions and the context wherein these actions are invoked as well as the existential quantifiers for setting criteria for triggering the context rules.

This paper presented a distributed architecture for dynamically generating context rules. The architecture consists of two main units: the context acquisition and recognition unit and the event detection and processing unit. The first unit is responsible for gathering data from various sensors, transform the data into a desirable format and performs various manipulations to reason about a higher-level context, which is an abstraction a dynamic real world situation, while the second unit is responsible for detecting the occurrence of decision events inside context-aware applications, and for associating these events with a context of interest. The time required to perform context-decision association is called an observation time, and association takes place

inside the Event Handler. When the observation time is over, the Rule Organiser aggregates decision-context associations and generates context rules from the aggregation. Afterwards, the Event Handler begins to listen to the occurrence of a context of interest to fire the decision events stored in the Rule Organiser.
The Context-aware E-Pad was designed and implemented to demonstrate our approach. The design concept of CAEP frees both the application developer and the user from setting rules to perform desirable actions. It has been shown how CAEP employed the higher-level context PLACE (corridor, room, building, or outdoors) without the need to directly deal with its composition. Moreover, CAEP has been trained to load suitable documents during lecture sessions.

Reference

1. Dourish, P. 2004. What we talk about when we talk about context. *Personal Ubiquitous Comp*. 8, 1 (Feb. 2004), 19-30.
2. Dargie W., Loeffler, T., Droegehor, O., David, K. 2005. Composition of Reusable Higher-Level Contexts. In *Proceedings of the 14th Mobile and Wireless Communication Summit*, IST, Dresden, Germany (June 2005).
3. Dargie W., Loeffler, T., Droegehor, O., David, K. 2005. Architecture for Higher-Level Context Composition. In *Proceedings of the Workshop on Context-Aware Proactive Services (CAPS05)*, Helsinki (June 2005).
4. Mäntyjärvi, J. and Seppänen. 2003. T. Adapting Applications in Handheld Devices Using Fuzzy Context Information. *Interacting with Computers J.*, vol. 15, no. 4, 2003, pp. 521–538.
5. Wang, X., Dong, J. S., Chin, C., Hettiarachchi, S., and Zhang, D. 2004. Semantic Space: An Infrastructure for Smart Spaces. *IEEE Pervasive Computing* 3, 3 (Jul. 2004), 32-39.
6. Gu, T., Pung, H. K., and Zhang, D. Q. 2004. Toward an OSGi-Based Infrastructure for Context-Aware Applications. *IEEE Pervasive Computing* 3, 4 (Oct. 2004), 66-74.
7. Schmidt, A., Beigl, M., and Gellersen, H.-W. 1999. There is more to context than location. *Computers and Graphics* **23** (1999), no. 6, 893–901.
8. Forgy, C. L. 1982. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17-37, 1982
9. Chakravarthy, S., Krishnaprasad, V., Anwar, E., and Kim, S. 1994. Composite Events for Active Databases: Semantics, Contexts and Detection. In *Proceedings of the 20th international Conference on Very Large Data Bases* (September 12 - 15, 1994).
10. Dargie, W. and Hamann, T. 2006. A Distributed Architecture for Reasoning about a Higher-Level Context. (To Appear) In *Proceedings of the 2nd IEEE International Conference on Wireless and Mobile Computing, Networking and Communications* (WiMob 2006), IEEE Press (June 19-16 2006).