An Active Architecture for Managing Events in Pervasive Computing Environments

Edwin Wong, Lisa Burnell, Charles Hannon

Department of Computer Science, Texas Christian University
Box 298850, Fort Worth, TX, 76129
{e.s.wong, l.burnell, c.hannon}@tcu.edu

Abstract

Many dramatizations have depicted a fully automated home living environment, where actions and events are understood or even anticipated. While the realization of such environments requires innovations on many fronts, our current research focuses on the development of an active database subsystem to respond to events in a home. The architecture implements the well-known event-condition-action (ECA) paradigm within an active data layer that is abstracted from the detection and processing of raw data sources. Our goals in this work are to develop and analyze the effectiveness of the active database system in a smart home system being developed as part of a larger collaborative effort. The knowledge encoded in the system is based on significant domain modeling, including analysis of inhabitant-device data collected from actual and simulated home environments, and standard knowledge acquisition techniques. The current system is composed of a Java framework for event management, an underlying data model to store transient and persistent data, and a JESS rule base to represent condition and actions. Interaction between the subsystems is maintained through an event manager responsible for interpreting the semantics of event execution and recognition.

Introduction

Smart home environments have the potential to enhance home living whether through reducing cost and effort or supporting the elderly or handicapped. While the realization of such environments requires innovations on many fronts, our focus here is on the analysis of event types and the development of an active database subsystem to capture and respond to events in a home. To date, the application of such systems to smart homes has been limited.

The Event-Condition-Action (ECA) paradigm in active databases detects event patterns, evaluates conditions under which the event occurs and selects appropriate actions. A relational database trigger is perhaps the most familiar use of the ECA paradigm. Rules are defined that monitor specific data updates and take appropriate actions. For instance, a bank database trigger might raise a flag when the current funds available drop below a certain value. In such a case, a withdrawal would constitute the triggering event, the condition would consist of a comparison of values and the action could be something as simple as sending an alert to the bank manager.

Our system employs a Java control framework, a relational database, and a rule base for representing complex conditions and decisions. Active behavior is based on the Event-Condition-Action model and is implemented in JESS (Java Expert System Shell). A Java event dispatcher is used to manage event information, communicate with the rule base, and execute appropriate actions when necessary. The event dispatcher takes advantage of the Java Reflection API to reference objects, fields and methods at runtime. Objects and methods can be reflected (initiated) at runtime from within the rule base to dynamically construct appropriate actions. Event processing is performed independently of underlying hardware. A simulator that currently supports a subset of Smart Home events provides the input required to test the system. SOAP and CORBA provide interfaces between both Java and C++ components of the smart home system.

In this article, we report on the development of our active database system that is one component of a collaborative effort to create a distributed multi-agent smart home. First, we discuss related work in smart homes and active databases, and summarize the architecture of the smart home within which our system fits. Next, the active data model is presented, including event types, representation, and persistence. Architecture and implementation issues are described, followed by conclusions and discussion of future work.

Related Work

Smart Homes

Smart home architectures typically adopt a layered architecture approach, with the database separating lower-level processing (e.g., sensors) from decision-making components. Database implementations may be centralized or distributed, and are often based on a relational or object-relational model.

In the Microsoft Easy Living Project (Microsoft, 2002), a world model database describes computing devices, people and their personal preferences, services and the

Copyright © 2002, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

home geometry such as rooms and doorways. The database serves as an abstraction layer between sensors and applications that use data from sensors. The disadvantage of the relational DBMS used is speed and the awkwardness of representing some knowledge as relations.

Rule bases are common in intelligent environments. The Intelligent Home Project (Lesser, et al, 1999) achieves goals by analyzing the methods of achieving them and taking the best action, based on a set of rules. Ambiente (i-LAND, 2002) uses rules to allow a room to reconfigure itself to the state that the user last had it in.

Active Databases

Active database capability most frequently uses triggers that execute actions based on the Event-Condition-Action (ECA) Model (Elmasri and Navathe, 2003). Events are specified actions detected by the system that are tested for specified conditions that, when true, trigger one or more actions to be taken. In most relational DB systems, rules are defined to detect certain data manipulation operations such as inserts, updates and deletes. Object-oriented database management systems (OODBMS) such as Sentinel (Chakravarthy, 1997) consider any operations on an object as a possible event. Snoop (Chakravarthy and Mishra, 1993) defines an expressive language for representing active capability, most prominently in Sentinel. Ode (Gehani, 1992) is another active database developed by AT&T Bell Laboratories.

Active database research has examined both integrated solutions and adding an active layer that resides above underlying data sources. Alert (Schreier, et al., 1991) is a simple architecture that provides active capabilities to SQL queries. ECAAgent (Li and Chhakravarthy, 1999) provides a seamless approach that emphasizes the ECA paradigm. Finally, (Desari, 1999) reports on an event detection framework on which our work is based.

While most research concentrates on centralized models with data and events in a single domain, complex applications such as smart homes can benefit from a distributed architecture. The Global Event Detector (GED) (Chakravarthy and Liao, 2001) detects events in a distributed environment through a client/server architecture that minimizes message communication and allows for event registration and notification. GED extends the active capabilities of the Sentinel active OODBMS. Other systems include SRI's Open Agent Architecture (OAA) (Martin, 1999) that supports cooperation of software services within autonomous agents, actively matching agents with user requests, and DeeDS (Andler, 1996) that implements active features in a real-time distributed database system.

An active data layer in a smart home needs to not only react, but also predict events and inhabitant behaviors. Many of the rules and events in a home environment and in active systems in general are dependent upon time. For recording a favorite television show or starting the sprinkler system, techniques from temporal databases (Dittrich and Gatziu, 1993; Ramamritham, 1996) can be employed.

Smart Home Architecture

The smart home effort for which we are exploring active DB issues is focused on developing a distributed, multi-agent, adaptable home environment (Figure 1) (Cook, 2001). Sensors monitor the environment and, if necessary, transmit data to other agents through the Communication layer. The data is recorded in the Information layer, learned concepts and predictions are updated as needed, and the Decision layer is alerted of the presence of new data. During action execution, information flows top down. The Decision layer selects an action and relates the decision to the Information layer to update the database and send a message to the Communication layer that routes the action to the appropriate effector to execute. If the effector is actually another agent, the agent receives the command as perceived information and must decide upon the best method of executing the desired action. A specialized interface agent provides interaction capabilities with users and with external resources such as the Internet.

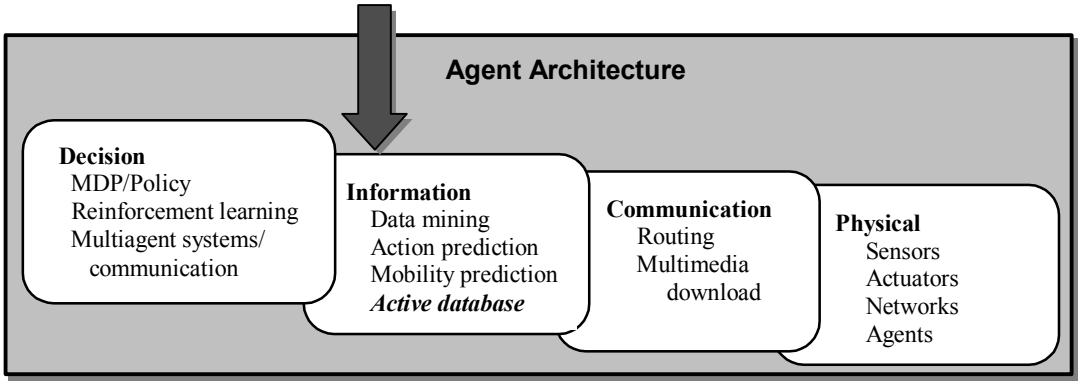

Figure 1. Agent Architecture.

Example Scenario

The smart home needs to detect, predict and respond to inhabitant behaviors. Consider an example in which an inhabitant wakes up at 6:30, checks the weather on the computer, showers, dresses, gets a cup of coffee while watching local news, gets into the car, and leaves for work. The following actions could be taken by the smart home:

- At 6:30, turn on the shower so water is warm by the time inhabitant is done reading the weather.
- Start the coffee pot 5 minutes into inhabitant's shower so coffee is fresh and ready by the time inhabitant is done dressing.
- Turn on TV to local news as soon as the inhabitant enters the kitchen.
- Turn off all lights in the house once inhabitant has left for work and set the security system.

For this scenario, the active DB must receive notification of a number of events and respond appropriately. To predict future behaviors, inhabitant behaviors must be recorded.

Knowledge acquisition and domain analysis of the required events, conditions and actions has been performed in three ways: 1) analysis of actual and simulated inhabitant behaviors and device interactions, 2) interviews and observations of typical inhabitant behavior patterns, and 3) analysis of output from inhabitant behavior predictors developed within the overall smart home project. (Cook, et al, 2003).

The Active Database

The active database design is patterned after that described by Li and Chakravarthy (1999) in which a mediator uses ECA rules to interface with a standard DBMS. Such an approach overcomes restrictions of trigger capabilities in existing systems, such as lack of composite event specification and restriction of a trigger to a single table. A relational DBMS affords many built-in benefits for maintenance, concurrency control, and transaction management. The separation of active capability from the underlying database system provides a number of benefits including extensibility, scalability, and portability (Li and Chakravarthy, 1999).

In this section we describe event types, data representation, and storage, with further details of the current version given in the implementation section. The phrase “external agent” used in this section refers to smart home agents within the complete smart home software that interact with the active database system.

Events and Action Types

Typical home events can be categorized. Four key event types are 1) data manipulation, 2) temporal, 3) exception, and 4) behavioral. Data manipulation events correspond to standard update queries in relational databases. For example, in predicting television-viewing patterns by an inhabitant, an external agent (e.g. a smart remote control) would gather data such as channel, duration and genre of what was viewed. Temporal events constitute events that occur based on a specified pattern with time. This category may include events that occur periodically within a schedule, or events that occur at a specific time. For example, inhabitants may want the living room temperature set at 70 degrees at 7:00 a.m. when inhabitants awaken every morning. In contrast, an event where recording of a television show occurs at 8:00 p.m. on Sunday, February 10, 2003 describes an instant event. The first example relies on a specified schedule, while the second event relies on recognizing a specific instance in time. In the event of failures or unexpected behavior, certain exception events are raised, and actions must be carefully designed to handle these conditions. One example of an exception event is a power failure. What actions should the DBMS take when power is restored? Finally, behavioral events include actions taken by a particular inhabitant. Events in this category frequently include events that require recording for interpretation by the decision layer. For instance, the system may record when the children arrive home from school, or when dinner for a particular household is typically served.

Temporal and behavioral events and the associated actions that should be taken have been our primary focus. Critical issues are the ordering and timing of action execution. Actions may be a single device change or a composition of actions for which an order of execution must be guaranteed. In either case, how long an action takes to complete and whether it was successful must be considered.

Persistent and Transient Data

The data and storage types are based on whether the data needs to be retained over time. Persistent data such as object location and name are maintained in a relational database. Transient data regarding current state, including sensor readings, on/off device status, and inhabitant locations, is maintained in the knowledgebase, with notifications sent to appropriately registered agents. However, specific events that produce transient data may have a useful purpose after a particular session has ended. For example, an inhabitant turning the television on in the late afternoon is useful for predicting future behaviors and thus needs to persist.

Device and Inhabitant Representation

An entertainment class contains devices such as televisions, VHS recorders, and DVD players that generally allow direct control assuming a reliable physical layer for transmitting, for example, IR signals, between the software and these devices. JINI (Jini, 2003) and JavaTV (JavaTV, 2003) could be used to support interaction with this class of devices.

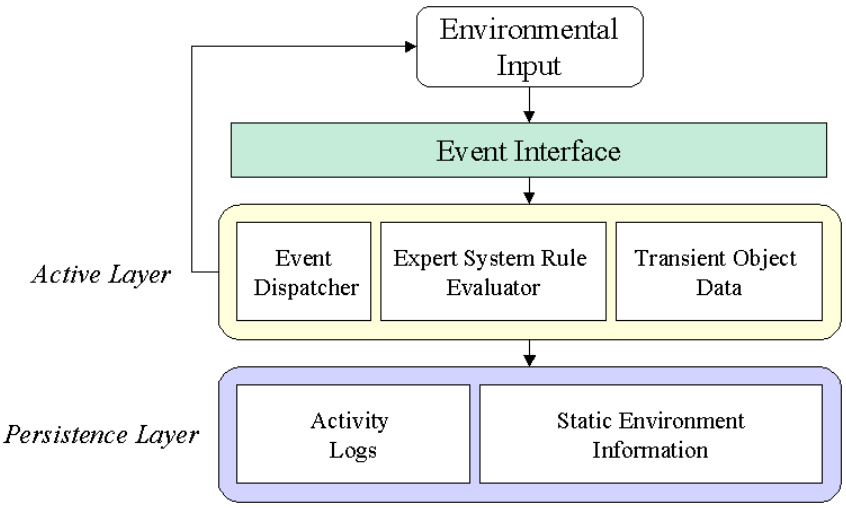

Figure 2. Active Database Architecture.

The inhabitant layer is perhaps the most difficult to model, since human behaviors are not as easily classified and detected as electronic devices. Efforts to date have minimized this challenge by modeling relatively simple behaviors. A generic Person class defines a general movement action and location manipulators, which alter the person's location in the environment. Although inhabitants are technically physical objects, they do not conform to the mappings of standard devices (e.g., no on/off functions), the Person class extends the general *PhyObject*, primarily for use of extended attributes such as id and location

The utilities class primarily consists of devices that aid action detection. Sensors of varying types, such as motion, temperature and air are implemented. Active sensors, such as motion, monitor pre-defined events and trigger alarms when actions are detected. In contrast, passive sensors remain in the background, collecting information about the status of the environment, and must be periodically checked.

Other classes contain basic devices used primarily for categorization, and include kitchen appliances, light and temperatures. For the most part these devices have simple actions such as changing state from on to off.

Two interfaces are the *Pollable* and *Identifyable* interfaces respectively. The latter enforces ID and name uniqueness, while the former provides an interface for device readings, which often appear as conditions in rule antecedents.

In the next section, we describe the implementation, including the persistence mechanism, events and rules, and mention future plans where appropriate.

Implementation

The architecture for event management (Figure 2) relies on a layered approach relating to the two fundamental types of data encountered. First, the event interface is responsible for event processing, whether the input is derived from sensors or software triggers. Furthermore, the event interface resolves semantics of an event such as the event type (composite/primitive), coupling mode and execution mode (sequential, repetition, etc). Another significant responsibility of the event interface is proper management of Jess facts, retracting and updating when necessary.

From the event interface, data moves toward the active layer, which maintains object and environment status data, evaluates Jess rules and dispatches additional events accordingly. Operations from the active layer not only encompass internal data manipulation events, but also external environment operations such as switching off lights.

Below the active layer resides the persistence layer, which storages long-term data. Any updates such as logging of a television may be pre-specified and executed, or may be a single data manipulation operation to long-term data such as location.

The active layer is implemented in Java and Jess and the persistent layer in Postgres. Java supports multi-platform development and deployment, including handheld (Java ME) and entertainment (JavaTV) devices. Jess provides the rules to express conditions and actions, and is expressly designed to integrate with Java. Postgres is robust, readily available, and supports object extensions to base relational model.

The persistence mechanism, which provides seamless interaction between the relational database and the active architecture, uses the Castor API. Castor provides binding of Java objects with data from relational database tables. Although Castor has a built in mechanism to deduce object information using reflection, an XML mapping file is used to bind object fields to relational database columns. Persistent device objects are currently bound at system

initialization using Castor. Device updating, using something like JINI, is planned for a future version.

Events

Primitive event handling depends largely upon the capabilities of the Java virtual machine and closely resembles an earlier approach (Dasari, 1999). Events that are available to the Smart Home are loaded through the classpath environment variable, which includes the supported objects home devices. Each class is recursively traversed to determine supported methods, regardless of scope. The goal is to select appropriate method objects available given a message. Primitive event input can occur directly through a user interface, while composite events is accomplished through XML input. Generation of events occurs through a testing user interface, where method signatures are selected, followed by the specification of input parameters if applicable. An approach to interfacing with underlying hardware components would require detecting event occurrences at the hardware level, triggering the appropriate Java method, which is then recognized by the expert system.

Events, both primitive and composite, are represented as standard Java objects with a corresponding Jess fact that is asserted when an event runs. The *PrimitiveEvent* must maintain a reference to the invoking object, the method object to be invoked and an array of parameters, which are cast appropriately before execution. Events are prioritized based on coupling mode and execution order, which will be discussed further. The Jess template for a simple event is shown below.

```
(deftemplate PrimitiveEvent
  "The basic template for a primitive event"
  (slot event)  ;Java Method object that is a primitive event
  (slot parent) ;object reference to the calling object
  (multislot params); store the parameters
  (slot firedTime); when the method was fired
  (slot priority) ;priority of the event ))
```

An event is processed by the *EventDispatcher*, which analyzes the semantic properties of the event. Data is extracted to generate a Jess fact. For *PrimitiveEvents*, fact assertions simply set the appropriate slot values to the corresponding object attribute value. *CompositeEvent* facts are generated dynamically by the expert system. Event patterns, which constitute composite events require detection in the form of a Jess rule, with the right hand side of the rule asserting a *CompositeEvent* fact with references to other event IDs. Parameter passing is modeled after that described in (Dasari, 1999). Parameters are maintained in arrays as general Java objects obtained from input. Primitive data types are represented as literals, while object parameters are referenced through a unique ID.

Composite events, which are composed of other events both primitive and composite, inherit other attributes such as name and id from the event superclass. Specification of composite events and the semantics of composite events have been well researched and generally many different composite event patterns exist. However, for the purpose of home events, typical events can be represented in the categories presented earlier.

Whereas most database transactions occur instantaneously, events in home living do not necessarily terminate in a reasonable period of time. For instance, a channel surf event, which cycles through the channels in a loop may not end until some sort of user intervention has occurred. Hence, each event received by the event dispatcher for execution is arranged such that all expert system fact updates, database transactions and method execution occur in a separate thread. The effect of this is platform dependent and may pose problems in single processor systems (Holub, 2000).

Scheduled tasks are implemented as Java timer tasks, and the job of scheduling is left to the virtual machine. Polling temperature sensors is one example of such a task. Although this works for simple types of polling patterns, not all events run on a regular, scheduled pattern used by a Java *TimerTask*. For instance, consider the monitoring of gas levels in the house, where the carbon monoxide levels were taken every ten minutes. However, suppose an increase occurred, causing the responsible agent to ask for readings at faster intervals than ten minutes. Hence, the scheduling is a function of the results of the event itself. In particular, the monitoring of human behavior, such as human motion and location is difficult to monitor in a regular fashion. This is an area still under investigation.

Rules

Rule evaluation and event detecting are accomplished using the Jess API (Jess, 2003). The rules left hand side corresponds to triggering events and conditions, which cause invocation of actions and possibly new events to be generated. Rules can be predefined or, via the reinforcement learning component in the decision layer, modified dynamically.

A recent update to Jess includes rule listeners that provide the ability to listen for rule firings when the corresponding events and conditions have occurred. The current system takes advantage of this feature, and actions are written in Java code. All actions to be executed must extend the generic *ECAAction* class, which provides the framework for all actions that appear on the RHS of Jess rules. Furthermore, these actions are themselves a type of event, either primitive or composite, and can trigger the firing of other rules. Again, *ECAActions* are executed using the event dispatcher.

We are exploring two improvements in this area that would enhance usability and performance. First, the association between rule and action is hard coded within a generic *JessListener*, which listens for the firing of any rule. A better method of association between rules and actions is desired. Secondly, and outside the scope of the current project is the ability to compile actions into Java code. For example, many actions involve turning on or off something.

Preliminary evaluation of system performance for simple and composite event execution has been performed on a standard MS Windows 2000 PC. For simple events, the time between event invocation and recognition rule was measured using a maximum of 1000 rules and 250,000 facts. Peak target rule recognition rate was 20 milliseconds. For composite events, the maximum number of child events was 500. Similar to the first experiment, these limits are affected by hardware specifications.

Conclusions and Future Work

We have presented an active database subsystem to respond to events in a home that is based on the event-condition-action (ECA) paradigm. The main contributions of this paper are the domain analysis and identification of issues peculiar to intelligent environment domains, and the development of a prototype active database system for a multi-agent smart home that is based on inhabitant behaviors in actual homes. The active database is a necessary component to support reasoning and learning within the smart home that we are developing in partnership with another university. Storage, retrieval and relatively simple decision making is supported with the active database subsystem.

Current work indicates promise in the approach. In particular, the seamless interaction between the rule base and the event dispatcher has generally led to accurate evaluation of rules and execution of events. It remains to be seen how the system will react as temporal sequences and composite events are incorporated. A simulator that currently supports a subset of Smart Home events provides the input required to test the system.

Three possible avenues for further work are 1) moving from a simulated to physical data environment, 2) exploring the use of a distributed processing environment designed to support inference systems, as in (Hannon, 2002), and 3) adding an emotion-based control mechanism (Hannon, 2003) to model inhabitant's emotional states and respond appropriately via biologically inspired decision making.

Acknowledgements

This work was partially supported by the National Science Foundation under Grant EIA-0203499.

References

Andler, S., et al. 1996. DeeDS Towards a Distributed and Active Real-Time Database System. *SIGMOD Record*, 25(1).

Chakravarthy, S. 1997. Sentinel: An Object-Oriented DBMS with Event-Based Rules. *ACM SIGMOD Conference*, Tucson.

Chakravarthy, S. and Liao, H. 2001. Asynchronous Monitoring of Events for Distributed Cooperative Environments. *Third International Symposium on Cooperative Database Systems for Advanced Applications* (codas), Beijing.

Chakravarthy, S. and Mishra, D. 1993. Snoop: An Expressive Event Specification Language For Active Databases. Technical Report, University of Florida.

Cook, D. 2001. MavHome Smart Home Project Description. http://ranger.uta.edu/smarthome/sh/.

Cook, D., Youngblood M., Heierman, E., et al. 2003. MavHome: An Agent-Based Smart Home, *IEEE Int. Conf. on Pervasive Computing and Communications*, Fort Worth, TX, Mar. 23-26, 521-524.

Dasari, R. 1999. *Events and Rules for Java: Design and Implementation of a Seamless Approach*. Master's Thesis, University of Florida, December 1999.

Dittrich, K. and Gatziu, S. 1993. Time Issues in Active Database Systems. *International Workshop on an Infrastructure for Temporal Databases*, Arlington, TX.

Elmasri, R. and Navathe, S.B. 2000. *Fundamentals of Database Systems 3/e.* Addison Wesley Longman, Inc.: Reading, MA.

Gehani, N. H. 1992. Composite Event Specification in Active Databases: Model and Implementation. *Proc. of the 18th Int. Conference on Very Large Databases*.

Hannon, C. 2002. A Geographically Distributed Processing Environment for Intelligent Systems, *Proc. of the 15th International Parallel and Distributed Systems Conference*: 355-360.

Hannon, C.2003. Emotion-based Control Mechanisms for Agent Systems. *Proc. of the International Conference of Information Systems and Engineering*, 10-15.

Holub, A. 2000. *Taming Java Threads*. Apress, Berkeley, CA.

i-LAND, 2002. Ambiente. http://www.darmstadt.gmd.de/ambiente/i-land.html

JavaTV, 2003. http://java.sun.com/products/javatv/.

Jess, the Rule Engine for the Java Platform. 2003. Available at http://herzberg.ca.sandia.gov/jess/.

Jini. 2003. http://www.jini.org/.

Lesser, V.; Atighetchi, M.; Benyo, B.; et al. 1999. The UMASS intelligent home project. In *Proc. of the 3rd Annual Conf. on Autonomous Agents*, 291--298.

Li, L., and Chakravarthy, S. 1999. An Agent-Based Approach to Extending the Native Capability of Relational Database Systems. *Proc. of the 15th International Conference on Data Engineering*, Sydney, March 23-26, 384-391.

Martin, D. L., et al. 1999. The Open Agent Architecture: A Framework for Building Distributed Software Systems. *Applied Artificial Intelligence*.

Microsoft. 2002. *Microsoft Easy Living Project*. http://research.microsoft.com/easyliving/.

Ramamritham, K., et al. 1996. Integrating Temporal, Real-Time, and Active Databases. *SIGMOD R*ecord, 25(1).

Schreier, U., et al. 1991. Alert: An Architecture for Transforming a Passive DBMS into an Active DBMS. *Proc. of the 17th International Conference on Very Large Databases*, Barcelona.