Enhancing Semantic Spaces with Event-Driven Context Interpretation

Joo Geok Tan[1], Daqing Zhang[1], Xiaohang Wang[2], and Heng Seng Cheng[1]

[1] Institute for Infocomm Research, Singapore 119613
{tjg, daqing, hscheng}@i2r.a-star.edu.sg
[2] National University of Singapore, Singapore 119260
xwang@i2r.a-star.edu.sg

Abstract. One important functionality provided by a context-aware infrastructure is to derive high-level contexts on behalf of context-aware applications. High-level contexts are summary descriptions about users' states and surroundings which are generally inferred from low-level, explicit contexts directly provided by hardware sensors and software programs. In Semantic Space, an ontology-based context-aware infrastructure, high-level contexts are derived using context reasoning. In this paper, we present another approach to deriving high-level contexts in Semantic Space, event-driven context interpretation. We show how event-driven context interpretation can leverage on the context model and dynamic context acquisition/representation in Semantic Space as well as easily integrate into Semantic Space. Differing from the context reasoning approach, our proposed event-driven context interpreter offers better performance in terms of flexibility, scalability and processing time. We also present a prototype of the event-driven context interpreter we are building within Semantic Space to validate the feasibility of the new approach.

1 Introduction

Smart Spaces are environments where a variety of information sources such as embedded sensors, augmented appliances, stationary computers, and mobile handheld devices, are used to provide contextual information about the environment in which they reside. By making use of this information, applications in Smart Spaces can become context-aware, that is, they are able to adapt automatically and intelligently to the changing situation to provide relevant services to the user [1].

Context-aware applications are typically difficult to build since the developer has to deal with a wide range of issues related to the sensing, representing, aggregating, storing, querying and reasoning of context [2]. To address this, we have proposed Semantic Space [3], an ontology-based context-aware infrastructure where we exploit Semantic Web [4] technologies for explicit representation, expressive querying and flexible reasoning of contexts in Smart Spaces. Since Semantic Space abstracts and provides generic context-aware mechanisms as reusable components, application developers can leverage on Semantic Space to reduce the cost and complexity of building context-aware applications.

H.W. Gellersen et al. (Eds.): PERVASIVE 2005, LNCS 3468, pp. 80 – 97, 2005.

One important functionality provided by a context-aware infrastructure is context interpretation to derive high-level contexts. In context-aware systems, high-level contexts augment context-aware applications by providing summary descriptions about users' states and surroundings. They are generally inferred from low-level, explicit contexts which are directly provided by hardware sensors and software programs [1, 5, 6, 7]. For instance, the implicit high-level context pertaining to a person sleeping in the bedroom may be derived from these explicit low-level contexts provided by various sensors deployed in the bedroom: (1) the person is located in the bedroom, (2) the bedroom light level is low, (3) the bedroom noise level is low and (4) the bedroom door is closed [7].

Various approaches to context interpretation within a context-aware infrastructure have been proposed. In ontology-based context-aware infrastructures [5, 6, 7, 8, 9, 10, 11], context information are modelled with a formal context model using ontologies and kept in a context knowledge base. High-level contexts are derived by applying rule-based reasoning techniques over the context knowledge base. The advantage of this approach is that context reasoning, validation and querying can be easily supported as generic mechanisms in the infrastructure since the context information is explicitly expressed in a machine-interpretable form. Our Semantic Space falls into this category where we use a context reasoner to derive high-level contexts by performing forward-chaining reasoning over a context knowledge base. However, the centralized model of this approach may not scale so well when there is a large number of contexts to be handled. In particular, when the context knowledge base and/or rules set is large, context reasoning over the knowledge base may not perform well enough for time-critical applications [9]. Machine-learning approaches where useful context information are extracted and inferred from sensor data using Bayesian networks in a sensing service infrastructure have also been explored in [12]. Such machine-learning approaches have the advantage that they can be more flexible than static rule-based approaches, however, they may require a fair bit of training before they can become effective.

In this paper, we propose another approach to context interpretation in Semantic Space which is based on an event-driven distributed model of the context-aware system. Our approach, while retaining the benefits of using Semantic Space for context representation, reasoning, validation and query, can offer better performance in terms of flexibility, scalability and processing time when compared to context reasoning. We show how event-driven context interpretation can leverage on the context model and dynamic context acquisition/representation in Semantic Space as well as easily integrate into Semantic Space, co-existing with and complementing other context interpretation mechanisms (such as context reasoning) in the infrastructure, to provide a wide variety of high-level contexts for applications to use. In our event-driven context interpreter, we leverage on the event specification language and composite event detection algorithm found in active database research [13, 14] by extending them from a centralized database environment to a distributed context-aware system. We also extend the context model with an event ontology such that event information can be retrieved from the infrastructure in a consistent and semantic way using semantic queries.

The remainder of this paper is structured as follows. In Section 2, we give an overview of Semantic Space in terms of the various collaborating components present

in the infrastructure. We then describe the context model of Semantic Space and extend it to specify events in Section 3. In Section 4, we show how dynamic context acquisition and representation is supported in Semantic Space and how these functionalities can be leveraged by the event-driven context interpreter. In Section 5, we describe in detail the design of the event-driven context interpreter and compare event-driven context interpretation with context reasoning. We present a prototype we are building to validate event-driven context interpretation in Section 6. Finally, in Section 7, we present our conclusions and future work.

2 Semantic Space Overview

Figure 1 illustrates the architecture of Semantic Space, our ontology-based context-aware infrastructure [3]. As shown in Figure 1, Semantic Space consists of a number of collaborating components, namely *Context Wrappers, Context Aggregator*, *Context Knowledge Base*, *Context Reasoners/Interpreters* and *Context Query Engine*.

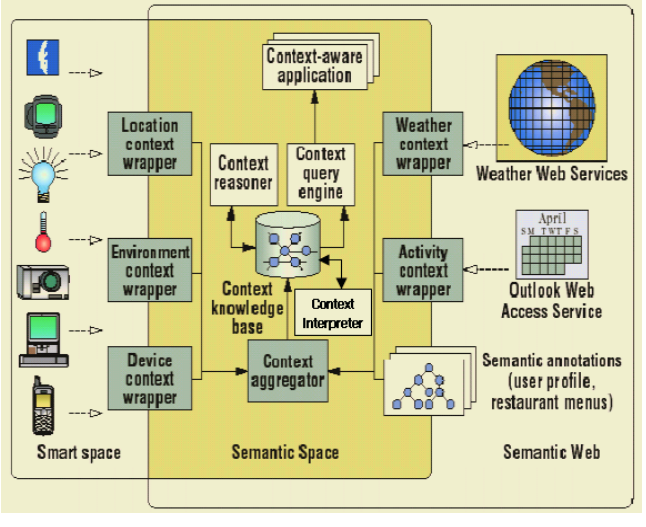

Fig. 1. The Semantic Space Context-Aware Infrastructure

Context Wrappers obtain raw data from software and hardware sensors, transform this information into a semantic representation based on the context model and publish it for other components to access. *Context Aggregator* discovers distributed wrappers, gathers context from them and updates *Context Knowledge Base* asynchronously. *Context Knowledge Base,* which serves as a persistent storage for context information, dynamically links context into a single coherent data model and provides interfaces for *Context Reasoner* and *Context Query Engine* to manipulate stored context. *Context Reasoners/Interpreters* are components that deduce high-level implicit context from low-level, explicit context using reasoning, learning and other techniques for context interpretation. The derived high-level contexts, likewise,

are asserted into the *Context Knowledge Base*, where they can be queried through *Context Query Engine*. *Context Query Engine* is responsible for handling queries about both stored context and inferred, higher-level context.

Our Semantic Space has been built [3] using standard Semantic Web [4] technologies such as RDF (Resource Description Framework) [15] and OWL (Web Ontology Language) [16] for context modelling, and logic inference and semantic query engines from the Jena2 Semantic Toolkit [17] for advanced context reasoning, validation and query. Standard Universal Plug-and-Play (UPnP) [18] is used to support automatic discovery of context providing components.

As shown in Figure 1, Semantic Space can support various ways of performing context interpretation with different context reasoners/interpreters co-existing with each other in the infrastructure, and complementing each other by contributing to the set of available high-level contexts. This wide range of high-level contexts can be accessed from Semantic Space in a uniform way through standard query and subscribe/notify interfaces. The context reasoners/interpreters can leverage on the uniform context model and the generic services provided in the infrastructure (e.g., services from context wrappers, context knowledge base, etc) to support their context interpretation functionality as well as provide generic services to other components in the infrastructure. Context-aware applications, in turn, can retrieve different levels of context from Semantic Space in a uniform manner, without having to know how the different high-level contexts have been derived in the infrastructure, and utilize them to adapt their behaviours accordingly.

3 Context Model

Context representation is an important part of pervasive computing environments and an appropriate context model is the basis for context representation. We use ontologies to model contexts in Semantic Space as this enables context sharing, context reasoning and validation, semantic query and also knowledge reuse.

3.1 Context Ontology

Within the domain of knowledge representation, the term *ontology* refers to the formal, explicit description of concepts, which are often conceived as a set of entities, relations, instances, functions and axioms [19]. Among Semantic Web standards, OWL is used to define and instantiate ontologies that let distributed computing entities exchange and process information based on a common vocabulary.

For Semantic Space, we have defined an upper-level context ontology (ULCO) in OWL to provide a set of basic concepts common across different Smart Space environments. Among various contexts, we have identified three classes of real-world objects (User, Location and Computing Entity) and one class of conceptual objects (Activity) that characterize Smart Spaces (see Figure 2). Linked together, these objects form the skeleton of a contextual environment. They also provide primary indices into other associated contexts. For example, given a location, we can acquire related contexts such as noise, weather, and the number of people inside if we model these objects as top-level classes in ULCO.

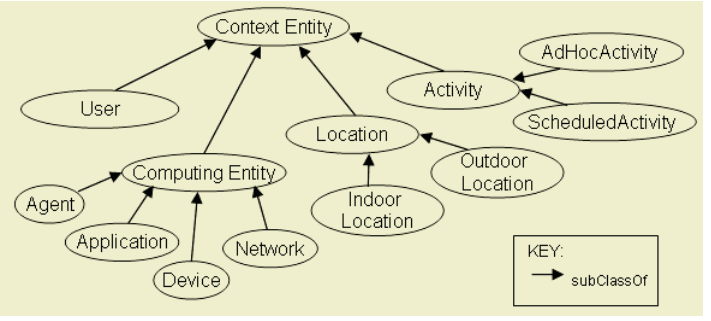

Fig. 2. Upper Level Context Ontology (ULCO) in Semantic Space

Our model represents contexts as ontology instances and associated properties (*context markups*) that applications can easily interpret. Consider the RFID (radio frequency identification) indoor location system that tracks users' location by detecting the presence of body-worn tags. When Grandpa enters the bedroom, the RFID sensor detects his presence and composes the following context markup:

<User rdf:about="#Grandpa"><locatedIn rdf:about="#Bedroom"/> </User>

Each OWL instance has a unique URI, and context markups can link to external definitions through these URIs. For example, the URI www.i2r.s-star.edu.sg/SemanticSpace#Grandpa refers to the user Grandpa, and the URI www.i2r.a-star.edu.sg/SemanticSpace#Bedroom refers to Grandpa's bedroom which have all been defined in the system.

3.2 Context from Events

Context information which are useful to applications fall into many categories [20]. We believe that one very important category of context information are those related to events occurring within the context-aware system. When an event occurs at a computing entity in the system, it can trigger another event at another computing entity or combine with other events to generate a new event. To support this type of context information, we propose another approach to context interpretation within Semantic Space which is based on an event-driven distributed model of the context-aware system.

We start by defining the various types of events. Events can be broadly classified into (1) Primitive Events and (2) Composite Events [13].

Primitive events are events that are pre-defined in the system and can be detected by a mechanism embedded in the system. In the case of a context-aware system, primitive events are those low-level events which can be directly detected by sensors or other mechanisms embedded in the computing entities (which are typically the context wrappers) in the system. Examples of primitive events are *sensed events* (e.g., when sensor readings exceed a certain threshold), *temporal events* (e.g., at 10 p.m. or 10 seconds after an event *E* occurs), *software events* (e.g., EOF when

retrieving information from a file) and *network events* (e.g., event notification from context wrapper to remote computing entity which has subscribed to the event).

Composite events are events that are formed by applying a set of event operators to primitive and composite events. We leverage on the work on Snoop, an expressive event specification language for active databases [13, 14], by adopting its set of well-defined event operators for our event composition. Table 1 shows a summary of the event operators which can be used and their descriptions.

Table 1. Summary of Event Operators

	Event Operator	Description
1	OR (V)	Disjunction of two events E_1 and E_2, denoted by $E_1 \mathrm{V} E_2$, occurs when E_1 occurs or E_2 occurs.
2	AND (Λ)	Conjunction of two events E_1 and E_2, denoted by $E_1 \Lambda E_2$, occurs when both E_1 and E_2 occur, irrespective of their order of occurrence.
3	ANY	The conjunction event, denoted by ANY(m, E_1, E_2, …, E_n) where $m \leq n$, occurs when m events out of the n distinct events specified occur, irrespective of their order of occurrence. Also to specify *m distinct* occurrences of an event E, the following variant is provided: ANY(m, E^*).
4	SEQ (;)	Sequence of two events E_1 and E_2, denoted by $E_1;E_2$, occurs when E_2 occurs provided E_1 has already occurred. This implies that the time of occurrence of E_1 is guaranteed to be less than the time of occurrence of E_2.
5	Aperiodic Operators (A, A^*)	The Aperiodic operator A allows one to express the occurrences of an aperiodic event within a closed time interval. There are two versions of this event specification. The non-cumulative aperiodic event is expressed as A(E_1, E_2, E_3) where E_1, E_2 and E_3 are arbitrary events. The event A is signalled each time E_2 occurs within the time interval started by E_1 and ended by E_3. On the other hand, the cumulative aperiodic event A^* occurs only once when E_3 occurs and accumulates the occurrences of E_2 within the open time interval formed by E_1 and E_3.
6	Periodic Operators (P, P^*)	A periodic event is a temporal event that occurs periodically. A periodic event is denoted as P(E_1, TI [:*parameters*], E_3) where E_1 and E_3 are events and TI [:*parameters*] is a time interval specification with an optional parameter list. P occurs for every TI interval, starting after E_1 and ceasing after E_3. Parameters specified are collected each time P occurs. If not specified, the occurrence time of P is collected by default. P has a cumulative version P^* expressed as P^*(E_1, TI :*parameters*, E_3). Unlike P, P^* occurs only once when E_3 occurs. Also specified parameters are collected and accumulated at the end of each period and made available when P^* occurs. Note that the parameter specification is mandatory in P^*.

To illustrate composite event formulation, consider the following example of medical management for the elderly within the context of an assistive home environment:

1. *Send an alert to Grandpa to take his medication every 4 hours from the time he wakes up to the time he goes to bed.*

   Using the event operators in Table 1, we formulate the event expressions for this scenario as follows:

   status(Grandpa, isOutOfBed) = locatedIn(Grandpa, Bedroom) ∩
   status(Bed, Nobody)

   status(Grandpa, isBrushingTeeth) = locatedIn(Grandpa, Bathroom) ∩
   (ANY(3, status(Toothbrush, Moved), status(Toothpaste, Moved),
   status(RinsingCup, Moved), status(Comb, Moved), status(Towel, Moved)))

   status(Grandpa, hasWokenUp) = status(Grandpa, isOutOfBed) ;
   status(Grandpa, isBrushingTeeth) ; locatedIn(Grandpa, Bedroom)

   status(Grandpa, isInBed) = locatedIn(Grandpa, Bedroom) ∩
   status(Bed, Somebody)

   status(Grandpa, hasGoneToBed) = status(Grandpa, isBrushingTeeth) ;
   status(Grandpa, isInBed)

   The final high-level composite event we are interested in can be formulated as follows:

   P(status(Grandpa, hasWokenUp), 4 hrs, status(Grandpa,hasGoneToBed))

2. *If Grandpa has not taken his medication within a 10-minute period after the alert is sent, send a SMS message to alert the care-giver.*

   We detect if Grandpa has indeed taken his medication by checking if the pill drawer has been opened and the water flask has been pressed during a 10-minute interval starting from when the status(Grandpa, isAlertedToTakeMedication) alert is sent. The high-level event we are interested in can then be formulated as follows:

   status(Grandpa, hasTakenMedication) = A^*(status(Grandpa,
   isAlertedToTakeMedication, (status(PillDrawer, Opened) ∩
   status(WaterFlask, Pressed)), status(Grandpa, isAlertedToTakeMedication) +
   10 min)

The example show how various primitive and composite events can be combined together, using event operators in the event specification language, to specify a wide range of high-level contextual events.

3.3 Event Ontology

To integrate event-driven context interpretation into Semantic Space context model (see Section 3.1), we propose to extend the context model with an event ontology to specify primitive and composite events. We therefore add another class of conceptual

objects (Event) to the original ULCO. Figure 3 shows a part of the proposed event ontology in graphical format.

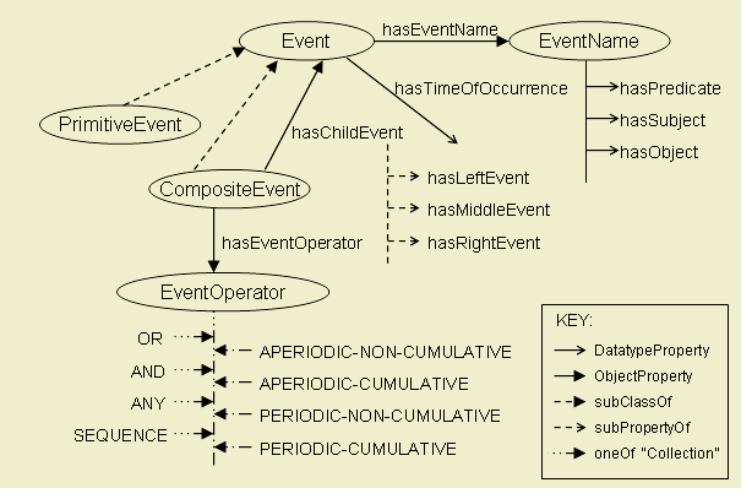

Fig. 3. Event Ontology (Partial)

The event ontology captures the key concepts of events described in Section 3.2. For example, primitive and composite events are modelled as subclasses of the *Event* class, thereby inheriting all the properties generic to events such as *hasEventName*, *hasTimeOfOccurrence*. Composite events, on the other hand, have additional properties such as *hasEventOperator*, *hasChildEven*t, *hasLeftEvent*, *hasMiddleEvent*, and *HasRightEvent* which enables the event expression associated with the composite event to be specified in the ontology.

With an event ontology as part of the context model, information about events can then be retrieved in a consistent and semantic way using semantic queries. For example, to find out all the parent child relationships of the events or to find out all the events which have occurred at a certain time, the following semantic queries coded in RDQL [21] format can be issued to the context query engine:

```
SELECT ?X, ?Y WHERE (?X, <owl:hasChildEvent>, ?Y)
SELECT ?X WHERE (?X, <owl:hasTimeOfOccurrence>, "20.05.04 13:00:41")
```

For a particular Smart Space environment where the ULCO has been further extended to model the specific Smart Space, additional linkages between various event objects to other objects in the context model can be defined. In our example, the User class is linked to the Event class by the ObjectProperty status, thereby enabling the various contexts such as status(Grandpa,isOutofBed), status(Grandpa, hasWokenUp), status(Grandpa, hasTakenMedication) to be modelled.

4 Dynamic Context Acquisition and Representation

The dynamism of a pervasive computing environment where sources of context can come and go warrants the need to support discovery and configuration of context sources (or their software wrappers). When a new context source joins the contextual environment, the context-aware infrastructure and applications should be able to locate and access it, and when the context source leaves the environment, applications should be aware of its unavailability to avoid stale information.

In Semantic Space, we employ standard UPnP as the mechanism for dynamic discovery of context sources. We provide a standard re-usable software wrapper with UPnP functionality which can be used to wrap context sources into context wrappers. Our context aggregator is then used to discover and aggregate the context information dynamically as context wrappers come and go or change their contextual information. In the following sections, we describe context wrappers and context aggregator in more detail.

4.1 Context Wrappers

Context wrappers obtain raw context information from various sources such as hardware sensors and software programs and transform them into context markups (see Section 3.1). Some context wrappers such as the RFID location context wrapper and the environment context wrapper (which gathers environmental information such as temperature, noise and light from embedded sensors) work with hardware sensors deployed in the Smart Space. Software-based context wrappers include the activity context wrapper, which extracts schedule information from Microsoft's Outlook 2000 and the weather context wrapper which periodically queries a Weather Web Service (www.xmethods.com) to gather local weather information.

To support explicit and uniform representation of context from a variety of context sources, wrappers in Semantic Space transform raw data (e.g. sensor data) into context markups based on shared ontologies used in the context model. As described in Section 3.1, context markups flowing within the Semantic Space are described in ontology instances, which can be serialized using alternative concrete syntaxes including RDF/XML and RDF/N-Triple. For example, a piece of context markup expressing the weather forecast of a city is serialized into XML (Figure 4(a) and triple format Figure 4 (b)).

```
<City rdf:id=http://...#Singapore>
<highTemperature>36</highTemperature>
<lowTemperature>28</lowTemperature>
<weatherType rdf:resource="http://...#Sunny"/>
```

(a)

```
(<http://...#Singapore> <http://...#highTemperature> "36")
(<http://...#Singapore> <http://...#lowTemperature> "28")
(<http://...#Singapore> <http://...#weatherType> <http://...#Sunny)
```

(b)

Fig. 4. XML and triple serialization of weather context

In Semantic Space, context wrappers publish context markups in the form of triples and other components can search for wrappers based on the matching of triple patterns. A triple pattern is a (subject, predicate, object) comprising named variables and RDF values (URIs and literals). To explicitly describe the wrapper's capability, each wrapper is associated with one or more triple patterns to specify the types of provided context. These triple patterns will be used as service description in wrapper advertisement and discovery. An example triple pattern for the location wrapper is:

```
(?user, http://...#locatedInRoom, ?room)
```

Once a wrapper is started, it periodically sends advertisement messages (with triple patterns) on the local network. Due to multicast and the periodic messages, other components in the system (e.g., context aggregator, context interpreter, context-aware applications etc) are notified about the presence of a wrapper, followed by the process of triple pattern matching and context subscription.

Our context wrappers have been implemented as UPnP services that can dynamically join a Smart Space, obtain IP addresses, and multicast their presence for others to discover. Context wrappers use the UPnP general event notification architecture (GENA) to publish context changes as events to which consumers (e.g., event-driven context interpreters) can subscribe. Since all context wrappers in Semantic Space are self-configuring components that support a unified interface for acquiring contexts from sensors and providing context markups, event-driven context interpreters in Semantic Space can easily leverage on context wrappers to provide the primitive events used in event-driven context interpretation. Since these primitive events can be dynamically added (and discovered) during runtime, the system is therefore able to evolve dynamically with new primitive and composite events being added at runtime.

4.2 Context Aggregator

Context aggregator discovers context wrappers and gathers context markups from them. The need for aggregation comes in part from the distributed nature of context, as context must often be retrieved from distributed sensors via various context wrappers. Aggregation is also critical for supporting knowledge-based management and processing tasks, such as expressive query and logic inference of context.

We implemented the context aggregator as an *UPnP control point* which inherits the capability to discover wrappers and subscribe to context changes. Once a new wrapper is attached to the Smart Space, context aggregator will discover it and register to published context. Whenever a wrapper detects the change in context, context aggregator is notified and then asserts the updated context markups into the context knowledge base.

To support event-driven context interpretation, we extend the context aggregator with an interface whereby information on the set of context wrappers (and their associated context) currently available can be retrieved. Rather than discovering individual context wrappers, primitive events available for composite event formulation can be made available from a single point, thereby simplifying the implementation of composite event formulation in event-driven context interpretation (see Section 6).

5 Event-Driven Context Interpretation

In this section, we look at the design of the event-driven context interpreter. We start by describing event graphs which are data structures representing composite events and explain the composite event detection algorithm used in event-driven context interpretation. We assert that event-driven context interpretation has a place in Semantic Space as another approach to context interpretation by discussing the advantages of event-driven context interpretation over context reasoning in Semantic Space.

5.1 Event Graphs

Event expressions contained in composite event specifications are converted into a collection of event graphs which are used in the composite event detection process.

An event graph is effective on a per computing entity basis. An event graph comprises non-terminal nodes (N-nodes), terminal nodes (T-nodes) and edges. N-nodes represent composite events and may have several incoming and outgoing edges. T-nodes represent primitive events and have no incoming edges, except those from remote computing entities which have been subscribed to for incoming network events, and possibly several outgoing edges. Figures 5(a) and 5(b) show the event graphs for the medical management example described in Section 3.2.

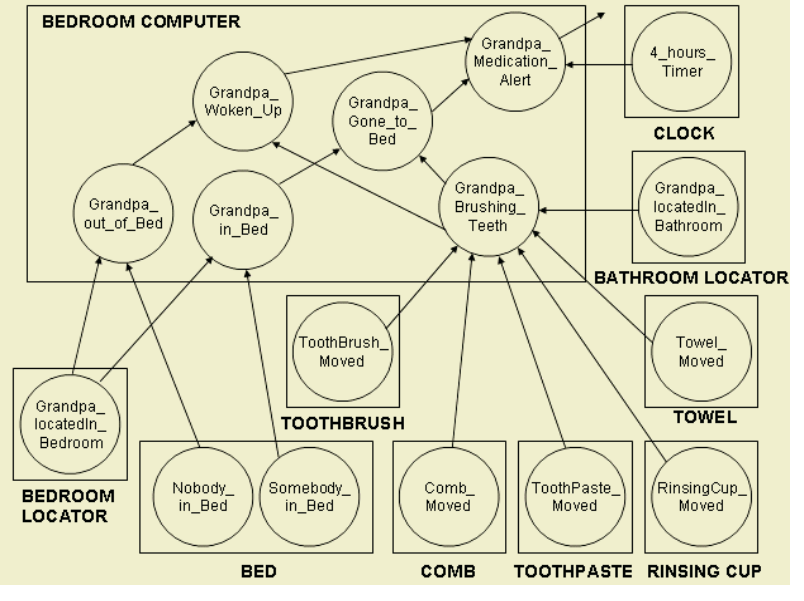

Fig. 5(a). Event Graphs for Example Context Interpreter 1

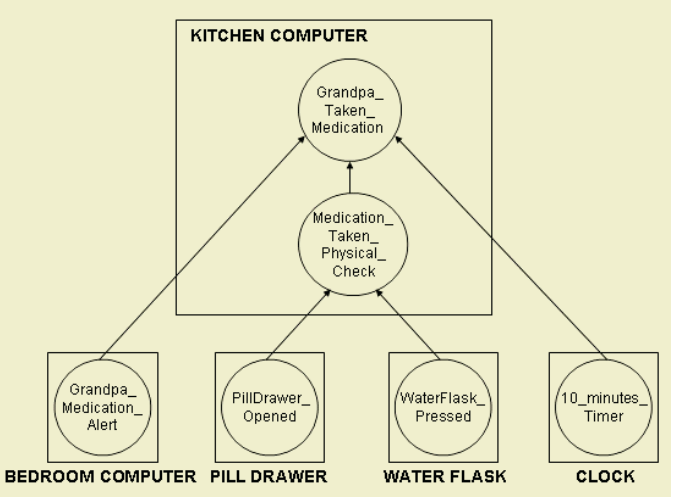

Fig. 5(b). Event Graphs for Example Context Interpreter 2

5.2 Composite Event Detection Algorithm

The composite event detection algorithm is an extension of the composite event detection algorithm in Snoop [14] so as to cater to a distributed environment where computing entities can subscribe to other computing entities in the system for events of interest to them.

When a primitive event occurs, it activates the terminal node that represents the event in the event graph. This in turn activates all nodes attached to it via the outgoing edges. When a node is activated, it evaluates the incoming event using the operator semantics of the node and if necessary, activates one or more nodes connected to it by propagating the event to them. An example of how the composite event detection algorithm operates for the "Or" and "Aperiodic" operators is shown in the flow chart in Figure 6.

5.3 Event-Driven Context Interpreter

Figure 7 shows the overall architecture of the event-driven context interpreter. A packet filter filters packets containing event graphs to the *Event Graph Processor* module and packets containing events to the *Primitive Event Detector* module. Upon receiving the event graphs, the *Event Graph Processor* merges new event graphs with existing event graphs in the *Event Catalog* and updates the *Event Catalog*. The *Event Graph Processor* then proceeds to subscribe to the context wrappers, context reasoners and/or other context interpreters for the events corresponding to the T-nodes in the event graphs, if the required subscriptions do not already exist. Upon receiving network events or when a local event is detected, the *Primitive Event Detector* sends an event notification (with its associated event information) to the *Event Queue*.

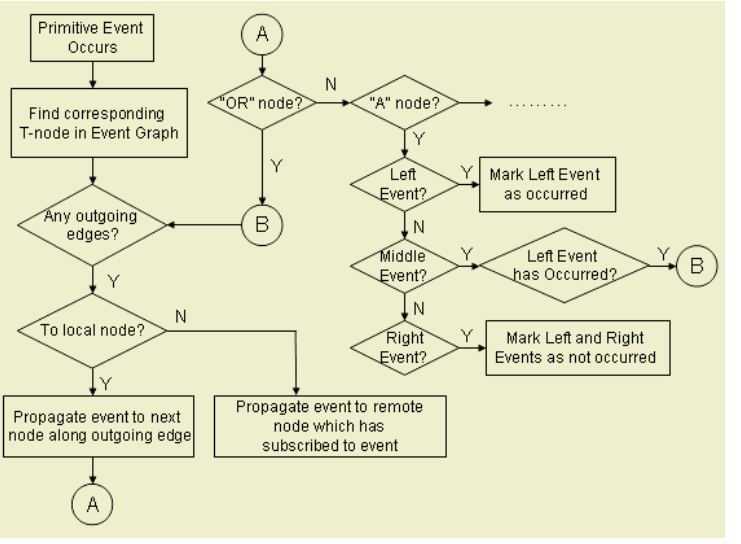

Fig. 6. Composite Event Detection for the "Or" and "Aperiodic" Operators

The *Composite Event Detector* retrieves each event from the *Event Queue* and processes it using the composite event detection algorithm (see Section 5.2) based on the event graphs stored in the *Event Catalog*. As the event graph is transversed, the *Event Dispatcher* will be signalled when graph nodes with event subscriptions are encountered. The *Event Dispatcher* will in turn send event notifications to the computing entities (e.g., other context interpreters or ECA Rule Service (see Section 6)) which have subscribed to the events.

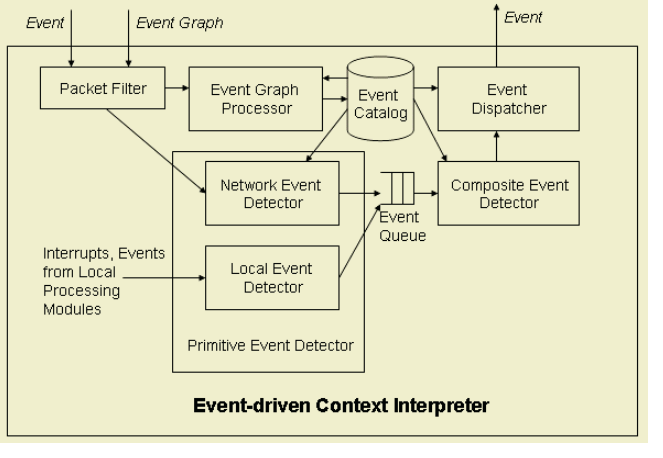

Fig. 7. Architecture of Event-driven Context Interpreter

5.4 Comparison of Event-Driven Context Interpretation with Context Reasoning

Figure 8(a) shows the model for our event-driven context interpretation. In this model, events corresponding to low-level contexts 1 and 2 generated by context wrappers 1 and 2 are propagated to context interpreter 1 where high-level context 1, specified in terms of low-level contexts 1 and 2 using the event specification language (see Section 3.2), is interpreted using the composite event detection algorithm (see Section 5.2). High-level context 1 is then propagated as an event to context interpreter 2. Low-level context 3 event from context wrapper 3 propagates to context interpreter 2. At context interpreter 2, high-level context 2 specified in terms of high-level context 1 and low-level context 3 is then interpreted.

Figure 8(b) shows a different model where context reasoning is used to derive high-level contexts. In this model, context wrappers propagate their low-level contexts to a context aggregator which stores the context information in a context knowledge base. The context reasoner, through the context knowledge base, can then derive high-level contexts 1 and 2 by reasoning over the knowledge base.

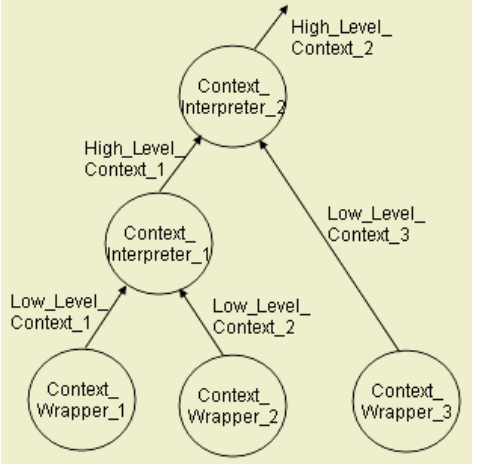

Fig. 8(a). Event-driven Context Interpretation

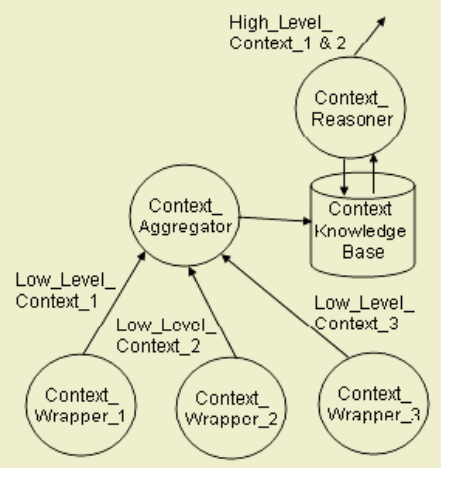

Fig. 8(b). Context Reasoning

By comparing the two models, event-driven context interpretation offers better performance in terms of flexibility, scalability and processing time. Being event-driven, processing can be triggered on demand in a flexible manner, thereby saving resources. The type of processing activated can also be selected based on previously received events. For example, in an environment where there are many sensors with power constraints, certain sensors can be active while the others are in sleep mode. When the active sensors detect a certain event, the others can be awakened to more accurately classify the situation. Distributed processing, which is more scalable than a centralized model, can be easily supported through the use of a number of context interpreters. The context interpreters, not only enable the processing load to be distributed, by having context interpreters close to its related context wrappers, the

latency associated with deriving high-level contexts from low-level contexts can also be reduced. Event-driven context interpretation is also expected to perform better in terms of processing time when compared to reasoning since it is working with only a very small specific subset of the context information. In addition, with the proposed event specification language (see Section 3.2), temporal contexts, which are a very important category of context information [20], can be easily supported.

The tradeoff in event-driven context interpretation is increased communication overheads since event-driven context interpretation is based on a distributed model of the system whereas in context reasoning, communication is limited to accessing a centralized server.

6 Prototype

We are currently developing a prototype for validating event-driven context interpretation in Semantic Space (see Figure 9). Besides the event-driven context interpreter, the prototype encompasses 2 other components, namely *User Interface* and *ECA Rule Service* which are designed to enable users to benefit from the ease and flexibility of configuring events and Event-Condition-Action (ECA) rules [22] associated with the events. We leverage on Semantic Space by making use of these generic components provided within the infrastructure: *Context Query Engine*, *Context Knowledge Base*, *Context Aggregator* and standard event subscribe/notify interfaces.

The *User Interface* component serves as the front-end to the user/programmer of event-driven context interpretation. Using this front-end, the user/programmer is able to (1) view all or selected events present in the system (both primitive and composite) and their corresponding context wrappers and context interpreters/reasoners, (2) formulate composite events using drag-and-drop of events and event operators and (3) associate an ECA rule to a selected event. The typical sequence of user interactions is as follows. The user/programmer issues a semantic query about events in the system. The *Display Events* module retrieves the information by querying the *Context Query Engine* for event descriptions and the *Context Aggregator* for the network addresses of the context wrappers and/or context reasoners/context interpreters generating the events. The user/programmer formulates new composite events by drag-and-drop of selected events and event operators. The *Compose Events* module creates the event ontology associated with the new events and updates the *Context Knowledge Base*. The event ontology is also passed to the *Event Graph Generator* module which parses the ontology and translates each event expression there into an event tree. The *Event Graph Generator* exploits commonalities in event expressions and coalesces event trees into event graphs. The output of the *Event Graph Generator* is a set of event graphs for the various event-driven context interpreters in the system. Each individual event graph is then sent to its corresponding context interpreter. If the user/ programmer wants to activate an action on the occurrence of an event, he/she can associate an ECA rule with the event by specifying the event handler and the ECA Rule Service which has the event handler installed. The *Associate ECA Rule* module will then update the selected ECA Rule Service with the ECA rule information.

The *ECA Rule Service* component is intended to ease development of event-driven context-aware applications and can be considered as an instance of a context-aware application. With reference to the medical management example given in Section 3.2, the ECA rules can be formulated as follows:

On P(status(Grandpa, hasWokenUp), 4 hrs, status(Grandpa,hasGoneToBed))
Condition True
Action Alert Grandpa and generate status(Grandpa, isAlertedToTakeMedication) event

On status(Grandpa, hasTakenMedication)
Condition False
Action Send SMS message to care-giver

A packet filter at the ECA Rule service component filters packets containing ECA rules to the *ECA Rule Processor* module and packets containing events to the *Event Processor* module. Upon receiving ECA rules, the *ECA Rule Processor* subscribes to the context wrappers and context reasoners/context interpreters for the specified events and updates its *Rule Catalog* with the ECA rules. Upon receiving an event, the *Event Processor* will invoke the event handler for the event based on the information in the *Rule Catalog*. We plan to provide some common basic event handlers with the ECA Rule Service such as "Call X" or "SMS Y" which the basic users of the system can exploit. The more advanced programmer can implement a new event handler and install it in the computing entity running the ECA Rule Service, if the existing event handlers do not meet his/her requirements.

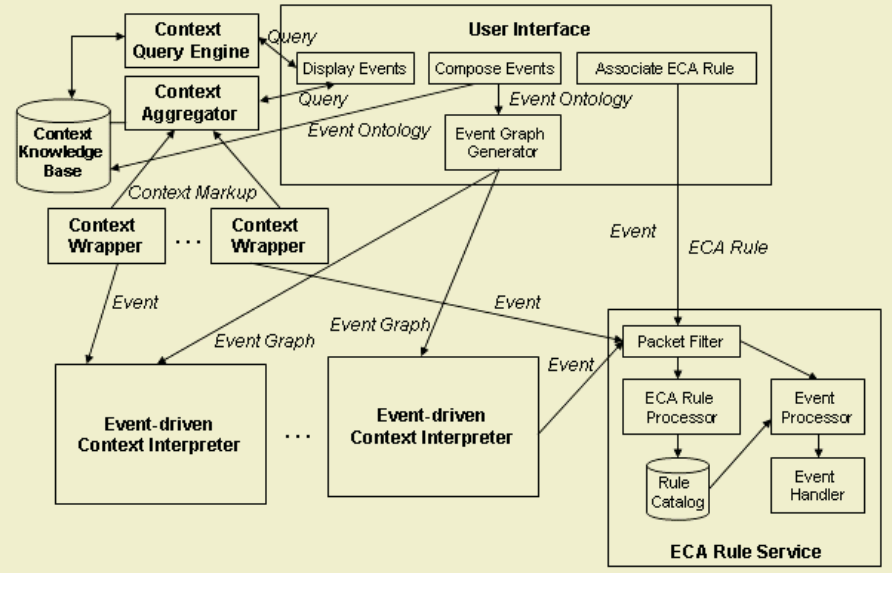

Fig. 9. Prototype for Event-driven Context Interpretation

7 Conclusions and Future Work

In this paper, we present event-driven context interpretation which is another approach to deriving high-level contexts in Semantic Space, a ontology-based context-aware infrastructure for Smart Spaces. We show how event-driven context interpretation can leverage on the context model and dynamic context acquisition/representation in Semantic Space as well as easily integrate into Semantic Space. Differing from the context reasoning approach, our proposed event-driven context interpreter offers better performance in terms of flexibility, scalability and processing time. We also present a prototype of the event-driven context interpreter we are building within Semantic Space to validate the feasibility of the new approach.

We intend to evaluate event-driven context interpretation with our prototype within Semantic Space in two dimensions: (1) Ease of development of event-driven context-aware applications using our generic event mechanisms and (2) Performance evaluation of event-driven context interpretation versus context reasoning. Our current model of event-driven context interpretation assumes that events can be detected with certainty. This is usually not the case in reality, particularly when sensors are involved. We plan to investigate further how composite event specification and composite event detection can be enhanced when events are detected in a more probabilistic manner.

We are providing event-driven context interpretation as a specific context interpretation mechanism within Semantic Space, which by design supports various context interpretation mechanisms co-existing within its infrastructure. We believe that event-driven context interpretation can be used to derive a wide range of useful high-level contexts, particularly when (1) the desired high-level context can be easily specified in terms of our event operators and (2) the desired high-level context is made up of many distributed contexts which can be derived in a hierachical fashion. As part of our future work, we would also like to further investigate and elaborate the specific scenarios in which event-driven context interpretation would be the preferred choice of context interpretation mechanisms.

References

1. Dey, A. K., and Abowd G. D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Anchor article of a special issue on Context-Aware Computing, Human-Computer Interaction (HCI) Journal, Vol. 16 (2001)
2. Hong, J. I., and Landay, J. A.: An infrastructure approach to context-aware computing. Human-Computer Interaction (HCI) Journal, 16(2-3) (2001)
3. Wang, X., Zhang, D., Dong, J., Chin, C., Hettiarachchi, S. R.: Semantic Space: A Semantic Web Infrastructure for Smart Spaces. In IEEE Pervasive Computing, Vol. 3, No. 2 (2004)
4. Berners-Lee, T., Hendler, J. and Lassila, O.: The Semantic Web. Scientific American, May (2001)
5. Chen, H., Finin, T., and Joshi, A.: Semantic Web in the Context Broker Architecture. IEEE Conference on Pervasive Computing and Communications (PerCom) (2004)

6. Ranganathan, A., and Campbell, R. H.: A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. In ACM/IFIP/USENIX International Middleware Conference (2003)
7. Wang, X.: The Context Gateway: A Pervasive Computing Infrastructure for Context Aware Services. Research proposal, http://www.comp.nus.edu.sg/ ~wangxia2, (2003)
8. Chen, H., Finin, T., and Joshi, A.: An ontology for context aware pervasive computing environments. Knowledge Engineering Review, Special Issue on Ontologies for Distributed Systems (2004)
9. Wang, X., Gu, T., Zhang, D., and Pung, H. K.: Ontology Based Context Modeling and Reasoning using OWL. Workshop on Context Modeling and Reasoning (CoMoRea) at IEEE International Conference on Pervasive Computing and Communication (PerCom'04) (March, 2004)
10. Gu, T., Pung, H. K., and Zhang, D.: A Service-Oriented Middleware for Building Context-Aware Services. Journal of Network and Computer Applications (JNCA), Vol. 28, Issue 1 (2005) 1-18
11. Gu, T., Pung, H. K., and Zhang, D.: Towards an OSGi-Based Infrastructure for Context-Aware Applications in Smart Homes. IEEE Pervasive Computing, Vol. 3, Issue 4 (2004)
12. Castro, P., and Richard, M.: Managing Context for Smart Spaces. IEEE Personal Communications, Vol. 7, no. 5 (October 2000) 21-28
13. Chakravarthy, S., Krishnaprasad, V., Anwar, E., and Kim, S.-K.: Composite Events for Active Databases: Semantics, Contexts and Detection. In VLDB (1994)
14. Chakravarthy, S., and Mishra, D.: Snoop: An Expressive Event Specification Language For Active Databases. Technical Report UF-CIS Technical Report TR-93007, University of Florida (1993)
15. Klyne, G., and Carroll, J. J., (eds.): Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation (2004)
16. McGuinness, D. L., and van Harmelen, F.: OWL Web Ontology Language Overview. W3C Recommendation (2004)
17. Jena2 Semantic Web Toolkit. http://www.hpl.hp.com/semweb/jena2.htm
18. Microsoft Corporation: UPnP Device Architecture Specification. Technical Report of Microsoft Corporation (November 1999)
19. Gruber, T.: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, Vol. 5, no. 2 (1993) 199-220
20. Chen, G., and Kotz, D.: A survey of context-aware mobile computing research. Technical Report TR2000-381, Darmouth College, Computer Science, Hanover, NH (November 2000)
21. Miller, L., Seaborne, A., Reggiori, A.: Three Implementations of SquishQL, a Simple RDF Query Language. In Proceedings of First International Semantic Web Conference, Italy (2002)
22. Beer, W., Christian, V., Ferscha, A., and Mehrmann, L.: Modeling Context-aware Behavior by Interpreted ECA Rules. Proceedings of Euro-Par 2003, Springer Verlag, Vol. LNCS 2790, ISBN: 3-540-40788-X, (August 2003) 1064-1073