Context-Awareness and Adaptation in Distributed Event-Based Systems

Eduardo S. Barrenechea, Paulo S. C. Alencar, Rolando Blanco, Don Cowan
David R. Cheriton School of Computer Science
University of Waterloo

Technical Report CS-2011-14

Abstract

Context-aware and proactive technologies have been continuously used over the past years to improve user interaction in areas such as searching and information retrieval, health care and mobile computing. In such systems, context information is typically propagated through events. Although there have been significant advances in distributed context-aware systems, there is still a lack of approaches that model and implement context-aware proactive applications involving the combination of context and distributed events. In our research we plan to address these issues by providing a context-aware event model with support for a new context-aware publish subscribe scheme. Such context-aware model would introduce context as a first class element, allowing components to specify the context in advertisements and subscriptions. The goal of our research is to be able to leverage context as part of our event model and bring behaviour adaptation to publication and subscription of events.

1 Introduction

Context-aware and proactive technologies have been continuously used to improve user interaction in areas such as searching and information retrieval, health care and mobile computing. With the introduction and increased use of smartphones and other mobile computing devices such as netbooks and tablets, context-aware applications can use the environment's information to provide real-time and autonomous adaptation to the user and related context.

Although there have been significant advances in models, frameworks and middleware support for context-aware systems, such approaches still rely mostly on non-distributed event-based interactions (e.g., through the observer pattern [38]) as means for context sources to propagate context information to the context-aware components. However, most of the context-aware systems proposed in the literature are distributed in nature. As such, there is still a lack of approaches that model and implement context-aware proactive applications involving the combination of context and distributed events. First, existing context-aware systems do not rely on efficient context information dissemination schemes provided by distributed event-based systems. Second, distributed event-based models and approaches do not use context as a first class element. Third, the current publish-subscribe approaches that use context, in which publishers generate events and subscribers consume events of interest, only consider the context of external components and not local context. For example, the notification of an event to a subscriber should not depend on context contraints determined by the publisher and a subscriber should be oblivious of the publisher's context when subscribing to an event. Fourth, there is a need to separate and decouple context information from application logic.

In our research we plan to address these issues by providing a context-aware event model with support for a new context-aware publish subscribe scheme. Such a context-aware model would introduce context as a first class element, allowing components to specify the context in advertisements

and subscriptions. This new context-aware publish subscribe scheme would be able to take context constraints into consideration when evaluating the publication of or a subscriber's notification of an event. Such context contraints would be specified by the components themselves, and would identify under which circumstances the component's publications or subscriptions are valid. By allowing a component to control its different publication and subscription behaviours, we can effectively separate context information from the component logic and place it on the publish subscribe scheme. As part of our research, we would model and implement a context-aware distributed event-based framework that provides the necessary infrastructure to publish and deliver events based on a component's context. We also plan to formalize our context-aware event model. This formalization will allow us to verify the behaviour of critical applications that rely on our context-aware publish-subscribe scheme.

In summary, the goal of our research is to be able to leverage context as part of our event model and bring behaviour adaptation to publication and subscription of events.

2 Background

2.1 Context Awareness

Context awareness is defined as the ability to both capture and adapt to context. In order for an application to be context-aware it has to fulfill both those requirements. An application that only catpures context, for example a gps-based location-tracking application, is not context aware since it doesn't adapt its behaviour to the different locations. In contrast, a location-tracking application that alerts the user if he is walking in circles is context-aware since it is able to change its behaviour according to the context input it receives.

When working with context-aware applications, one needs first to define what exactly context is. In [37] Dey defines context as *"...any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application."*

Current research into context-awareness and context-aware systems focuses on three main areas, namely *(i)* context aware programming languages, *(ii)* context models and *(iii)* context-aware middleware and frameworks.

2.1.1 Context-Aware Programming Languages

Context-aware programming languages are an integral part of the emerging *Context-Oriented Programming* (COP) paradigm. These language extensions deal with the lack of explicit support for context and context-awareness that is found in current programming languages[57, 1].

COP tries to address the need that a context-aware application has to adapt its behaviour to different situations or context during run time[2]. There is no context-awareness without behaviour adaptation. Wihtout the proper abstractions context-aware applications often require complex designs and architectures in order to deal with dynamic run-time context changes [27]. This leads to hard to maintain, convoluted code with increased chance of introducing bugs and unspecified behaviour.

Language extensions that support COP provide ways to specify behavioural variations and group these variations into layers [57]. Layers can be dynamically activated or deactivated depending on the run-time context. A scoping mechanism is also provided in order to control the scope of each of the layers of the context-aware application explicitly. COP provides a programming model for context-aware applications that is able to reduce the programming effort and increase interoperability with heterogeneous systems.

2.1.2 Context Models

Context models are used to represent and help understand context information. They provide a common vocabulary used to express, query and share contextual information between different modules or components of a context-aware system.

Many different context models have been proposed in the literature. Context models can be classified by their modelling approach. Modelling approaches that provide hierarchical data structures and partial model or scheme validation are better suited for modelling context [91].

Hierarchical data structures provide context models with the characteristics required to represent higher-level context information. Such information is often the result of reasoning and composition of higher-order context information from primary context elements.

Context models can also be classified by the types of context or context elements they represent [21, 15]. Most importantly, context is highly dynamic and therefore support for dynamic context propagation is highly disereable [21]. Such models are able to capture the intrinsic relationships between different context elements and properly represent how changes to one context element affects the state of other elements in the context model.

A context model that properly represents the different types of context and their relationships is necessary in order to provide proper behaviour adaptation to context-aware applications and systems.

2.1.3 Context-Aware Middleware and Frameworks

Context-aware frameworks provide a set of abstractions and libraries that facilitate the creation of context-aware applications. Context-aware middleware is a fully implemented software solution used to connect context sources and other context-related services with context-aware components and applications. In [38], Dey et al. propose a set of requirements a framework should fulfill in order to deal with context. These requirements are *separation of concerns*, *context interpretation*, *transparent distributed communications*, *contant availability of context acquisition*, *context storage* and *resource discovery*. Such requirements deal with encapsulation of context sensing related code, dissemination of pre-processed context information to interested context-aware components and lifecycle or runtime independence of context source components from context-aware applications. These are the various functional requirements that context-aware framework and middleware implement in order to provide better support for context-aware applications.

Many different context-aware frameworks and middleware have been proposed in the literature [22, 6, 89, 66]. Context-aware frameworks and middleware can be classified according to their main architecture, such as client/server [47, 78, 80], service-oriented [53, 69, 62, 20, 97, 83], agent-based [25, 88], event-based [38], distributed event-based [82, 86]. Even though there are many different architectures in use, most frameworks and middleware still rely on an event-based approach as means for context sources to propagate context information to the framework or middleware internal components.

Context-aware framework and middleware provide all administrative components necessary for storing, managing, querying and distributing context information to registered components. Most frameworks and middleware provide discovery services that allow components to search for different context sources or information. Frameworks and middleware also provide context knowledge bases or context repositories used to store context information. They provide a more stable solution than just relying on events to propagate context to context-aware components and applications [42].

2.2 Distributed Event-Based Systems

Distributed event-based systems (DEBSs) are systems composed of distributed functional components interacting through events. Events are happenings on the system or environment where the component resides. DEBSs use a publish-subscribe interaction scheme [46] to provide communication between producers and consumers of events. Publishers are components that produce events. Subscribers are components that consume events. A subscriber can express interest in a given event through subscriptions. A subscriber will be notified of all events published in the system or environment that match its subscriptions.

2.2.1 Publish-Subscribe Scheme Characteristics

publish-subscribe interaction schemes are very flexible and provide many desirable features that minimize the complexity of the system, greatly improving its overall design and architecture. Such features are full time, space and synchronization decoupling [46].

Space Decoupling

Space decoupling determines that publishers and subscribers do not need to be aware of one another. Publishers publish an event in the system and the system is responsible for delivering this event to all subscribers. Similarly, a subscriber is not required to know any publishers in order to be notified of an event in the system, it simply needs to subscribe to the desired event.

Time Decoupling

Time decoupling determines that publishers and subscribers are not required to participate in publication and notification of events at the same time. A publisher will publish an event even if the subscriber is offline. Conversely, the subscriber may be notified of events even if the publisher is offline.

Synchronization Decoupling

Synchronization decoupling determines that subscribers are asynchronously notified of events. A publisher does not need to wait for a subscriber to process or acknoledge communication when generating events. Similarly, a subscriber is not required to wait for an event in order to continue with its process. Creation and notification of events takes place outside of the main flow control of publishers and subscribers.

2.2.2 Publish-Subscribe Scheme Variations

There are currently three variations of publish-subscribe schemes, namely topic-based, content-based and type-based [46].

Topic-Based Publish-Subscribe Scheme

A topic-based publish-subscribe scheme defines subscription through the use of topics or channels. All components that subscribe to a specific topic will receive all events published within this topic. Subscribers can subscribe to more than one topic at a time. Subscribing to a topic is similar to becoming member of a group. A component will be notified of all communication intended for that group. The topic-based publish subscribe scheme does not provide any type of filtering capability. The component itself has to analyze and filter all events that it receives through the topic subscription.

Content-Based Publish-Subscribe Scheme

A content-based publish-subscribe scheme introduces the concept of content filtering. Events are no longer classified according to external terms, such as a topic, but according to the contents or properties of the event. Components can subscribe to events by specifying filters through a subscription language. Filters are constraints on event properties, specifying a name-value tuple expression with comparison operators. A component will be notified of an event if the event properties match the filtering expression defined in the subscription. Filters can be logically combined to create more complex subscriptions patterns.

Type-Based Publish-Subscribe Scheme

A type-based publish-subscribe scheme introduces the concept of event type. Events are no longer classified according to topics or its internal attributes, but according to its kind. This publish-subscribe scheme follows more closely the object-oriented programming paradigm which leads to desirable features such as data encapsulation and inheritance. Different event types result in different events even if the internal event properties are the same. Subscription works by subscribing to a specific event type. A subscriber will be notified of an event if said event is of the same type as the one specified in the subscription. Type-based publish-subscribe schemes support content filtering as well. A component can specify filters based on the values of the internal properties of an event type. In this case, a component will be notified of an event if and only if the event type matches both the event type defined in the subscription and if the values of the event properties match the filtering expression.

3 Related Work

Work related to our approach fall under two categories: *(i)* context-aware frameworks and middleware and *(ii)* context-awareness in publish-subscribe schemes and distributed event-based systems.

3.1 Context-Aware Frameworks and Middleware

The Context Toolkit [38, 40, 39] is one of the first approaches to a context-aware framework. It applies a graphical user interface approach, where the context widgets are used to provide access to context information. A context widget is able to abstract the complexity of dealing with context sources and provides a reusable component with a specific interface that can be easily customized for creating context-aware applications.

The context toolkit advocates the use of events as an optimal solution to disseminate context information. It does not, however, use a distributed event approach. Each component that requires a specific type of context has to contact the context widget individually and subscribe to context updates. This approach is similar to the observer pattern. A context widget will notify the registered component if a change in context takes place.

SOCAM [52, 53] is a Service-Oriented Context-Aware Middleware that enables rapid prototyping of context-aware services. It provides a semantic space where different context-aware services can have access to context information. The SOCAM architecture consists of context providers, context interpreters, context database, service location service and context-aware mobile services.

Context providers provide abstractions for low-level context sensing components, hiding the complexity and exposing the context information. Each context provider needs to register with the service location service in order to be made available for context-aware mobile services. The context interpreter also acts as a context provider. It provides higher-order context information by combining lower-order context gathered from multiple context providers. The context database is a central location used to store context information. It provides a simple interface for services to query, add, delete or modify context information. The service location service is a directory of context providers that allows other components and services to locate context providers. The context-aware mobile services are the ones that provide actual context-awareness for the application. They are able to make use of context information and adapt their behaviour accordingly.

SOCAM uses Java's RMI technology to provide component communication and context information dissemination. This approach is similar to the context toolkit one, where a component registers for context information updates with a context provider similar to the observer pattern. The context-aware mobile service registers itself with a context provider and is notified through a RMI callback. SOCAM also provides a way for services to query the context information, without the need to register for context updates manually.

The Context Broker Architecture (CoBrA for short) [24, 25] is an approach that provides a middleware to create context-aware software agents. CoBrA divides the context model in different domains. Each domain has an autonomous agent, namelly, the domain context broker. The broker is responsible for sharing context information with other context-aware agents. The broker maintains an instance of the context model of the domain to ensure a consistent and coherent context knowledge base.

The broker is also responsible for context acquisition and the completeness of the context information. Context acquisition in CoBrA is accomplished in the same fashion as in the context toolkit and SOCAM. The broker is able to register with context sensors and context interpreters and be notified of changes and updates to context information.

In [68], Korpipaa et al. introduce a blackboard-based context management framework. A context manager is responsible for storing context and notifying clients of any context changes. Clients can be resource servers, context recognition services, security components and context-aware applications. Notification takes place in an event-based manner, where clients subscribe to changes in a specific context type. Such approach also uses the observer pattern method and not distributed events. Furthermore, clients are notified of context changes, and still have to implement behaviour adaptation for different context states at code level, within the application logic.

In all of the systems described each component is responsible for searching for and registering with a context source. All of the context information is disseminated by the context source directly

to the registered components. This approach is different from a distributed event-based system. In a distributed event approach, the component would subscribe to a specific event schema and would be notified of any changes in context by receiving an event implementing said schema published by a context source. A distributed event approach is able to decouple the component and the context provider, resulting in a simpler, more effective design. Both publishing and subscribing component have no knowledge of one another. All of the event notification is handled by the middleware.

All of the systems mentioned also provide a context database or repository. Even though context information is being stored, it is not being used by the middleware to determine any adaptation in behaviour due to context changes. All of the adaptation is being accomplished by the component. This requires that both application logic and context information be stored in the component. The ability to separate and decouple context information from application logic would have a very positive result in both the design and understanding of a context-aware component.

3.1.1 Distributed Event-Based Context-Aware Middleware

STEAM [73] is a proximity-based distributed event-based system designed for use in ad-hoc networks. It introduces the concept of location and proximity filters, which help in determining geographical areas where events are valid, effectively bounding event propagation to a desired region.

A publisher is able to advertise the event type along with the bounding geographical area in which the published event is valid. Any subscriber within the event's bounding geographical area will be notified of the event.

The sentient object model [9] was later introduced on top of STEAM to create a framework for developing mobile, context-aware applications. It applies to real world systems and hardware interfaces.

The sentient object is a context-aware entity which uses sensors and actuators as its interface. Sensors are components used to capture context information, while actuators are used to modify the system's state according to the captured context information. Both sensors and actuators are hardware interfaces. Sensors publish events based on changes in context sensed from the real world and actuators subscribe to events and react by changing the state of the real world in some predetermined way through a physical device.

MoCoA [86], a customisable middleware for context-aware mobile applications, uses both STEAM and the sentient object model. MoCoA introduces the notion of event channels, which are used to specify the constraints on both propagation and notification of events in a geographical area. The publisher is the one responsible for defining the event channel for a given advertisement.

All of these related systems focus mostly on real world spaces, such as smart environments, and interfaces, such as self-driving cars, than on software systems.

Gaia [82] is a context-aware meta-operating system that supports the development and execution of portable active space applications. It is a distributed middleware infrastructure that coordinates different networked software components and devices in a physical space. Changes in an active space, such as adding or removing components and users as well as context updates, are handled through the use of distributed events. Gaia uses a topic-based publish subscribe scheme, where producers and consumers are connected through a channel. Gaia's use of events is only to disseminate information to the many components in the system. Its event model does not provide any type of context-aware adaptation to publishers or subscribers.

3.2 Context-Oriented Programming

Other approaches [77, 3] also focus on the use of events to trigger different behaviours thus achieving behaviour adaptation. Even though such approaches advocate the use of events, they are not considered frameworks or middleware. They do not use distributed events and their focus is on lower-level solutions dealing with language support for context-aware applications and Context-Oriented Programming.

In [54] Henricksen et al. introduce a set of conceptual models to facilitate the development of context-aware applications. One of such models is the Triggering programming model, which follows the event-condition-action model. It allows asynchronous implicit invokation calls in response

to context changes. Context changes are considered changes in state with six distinct possible values ranging from *TrueToFalse, TrueToPossiblyTrue, EnterFalse, TrueToFalse, PossiblyTrueToFalse, Changed*. It is important to note that this approach does not specify any event model and changes in state are strictly confined to transitions of boolean parameters. Such triggers are limited to relationships already explicit in the model and do not support context parameters with non-boolean values.

3.3 Distributed Event-Based Systems

In [29] Cugola et al. propose the introduction of context as a first class element in event-based systems. This modification allows each node to set its current context state and allows for subscriptions with both content and context filters as well as publication with context filters.

Context filters work by filtering the context of either the publisher or the subscriber with the specified context filter expression. A subscription specifies a content filter for the event and a context filter for the context state. If the event content matches the content filter and the context of the publisher matches the context filter then the subscriber will be notified of the event. Similarly, a publisher publishes an event by specifying the event content and a context filter. All components subscribing to that event whose subscription context filter matches the publication context filter will be notified of the event.

The approach proposed by Cugola et al. differs from our approach in various points. First it is concerned mainly with content-based event models while our approach focuses on a type-based event model. Second, even though the paper states that context was introduced as a first class citizen it does not clearly identify the relationships between context, events and components, with a brief mention on how to set the node's context. Third, their approach also focuses more on efficiency of routing and context propagation than on models and middleware.

Fourth, a major difference between our approach and the approach of Cugola et al. is in how the context filters affect publications and subscriptions. Cugola et al. focuses on using context to filter either publishers, from a subscriber's perspective, or subscribers, from a publisher's perspective. Our approach focuses on the component's context and how it affects its subscriptions and publication of events.

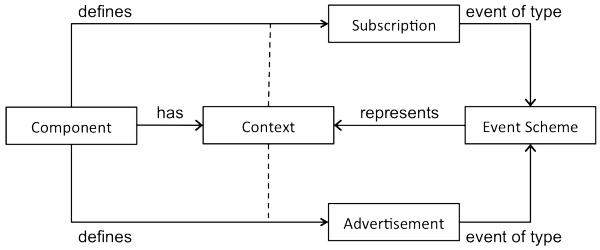
In [49] Frey et al. propose a context-aware publish-subscribe scheme. The approach of Frey et al. focuses on limiting the subscription scope through context of relevance and context of interest. Context of relevance is used by a publisher to represent either the context that is affected by the event or the context in which the event is relevant. Context of interest is used by subscribers to identify the context of a publisher.

The combination of context of relevance and context of interest is able to limit the subscription scope of a system effectively. As in the approach proposed by Cugola et al. the approach by Frey et al. allows a subscriber to filter events by the publisher's context. It also allows publishers to determine in which context their events should be valid. A component will only be notified of an event if the context of interest in the subcription matches the context of relevance in the event's publication.

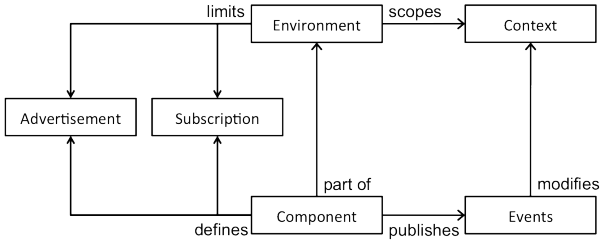
The approach of Frey et al. is very similar to symmetric subscriptions [81], although it uses context instead of the information not contained in an event. Frey et al. do not introduce context into the event model and there is no clear relationship between context, events and components. The approach also focuses on efficient routing, performance and scalability instead of on models and middleware. Lastly, their approach focuses on using context to filter either publishers, from a subscriber's perspective, or subscribers, from a publisher's perspective while our approach focuses on how the context of a given component affects its own subscriptions and publication of events.

In [41] Dittrich et al. propose AGILE, an adaptive index for context-aware information filters, introducing the concept of a context-aware information filter and their architecture. A context-aware information filter (CIF) is a component that keeps the subscriber's profile and context information. It uses the context information to filter out messages that are not valid in the subscriber's context. A CIF contains two input streams, the first is the stream of events or messages being relayed to the system while the second is the stream of context updates to the component's context state.

Dittrich et al. focus on indexing and filtering of information and not on event-based systems models and middleware. There is also a differentiation between messages or events and context



(a) Publish-Subscribe View



(b) Environment View

Figure 1: Context-aware event model.

updates, which, in an event-based system, are not necessarily different.

Cayuga [35] is a stateful publish subscribe system. It allows a component to define subscriptions spanning multiple events with support for parametrization and aggregation. A stateful subscription is capable of specify sequences of related or unrelated events to be used as subscription filters. This ensures that an event will only be delivered to a subscribing component if certain conditions are met or if some event sequence has taken place. Cayuga is able to use the state of the system and its networked components when determining the validity of a subscription. It does not, however, provide a context-aware event model nor takes context into consideration. Also, the work in Cayuga focuses on publish-subscribe implementations and algorithms and not on context-aware models.

In summary, both the approaches by Cugola et al. and Frey et al. focus on the context of both publishers and subscribers as external entities to the component. Said context is used to validate if the publisher of the event is relevant to the component with respect to its subscription or, in the case of publication, if the subscriber to the event is relevant. This emphasis on factors external to the component for publish-subscribe systems is undesireable since it causes a higher coupling between components. A subscriber must be made aware of specific details of a publisher's context states in order to subscribe to an event. AGILE introduced the notion of context-aware information filters and focuses on indexing strategies but does not introduce context into distributed event-based systems. Finally, Cayuga provides a stateful publish-subscribe system, but also focuses on low level indexing strategies and algorithms.

4 Context-Aware Event Model

Our proposed approach consists of introducing context as a first class element into the model. This requires an explicit representation of both syntax and semantics and how context interacts with both events and components. Once these relationships are established we can identify the impact context has on the event model and, subsequently, on the publish-subscribe scheme. Figure 1(a) shows the publish-subscribe view of the event model, while Figure 1(b) shows the environment view of the event model.

The system is composed of independent *Adaptive Components* (components for short), that interact

with one another through events. Components are uniquely identified and each component executes in its own program space. Components have the ability to publish and/or subscribe to events.

Components are grouped into *contextual environments* (environments for short). An environment represents a set of context parameters. All components in the same environment share the these context parameters and their states. Components grouped in the same environment are considered siblings. Each environment has a number of specialized components in charge of administrative and component interaction controlling activities.

4.1 Events

An event is a data representation of a happening in the system. Events are published by components. An event has a schema that determines data attributes associated with the event. Attributes can be primary types or data structures defined from primary types. An event has exactly one event schema.

4.2 Event Schema

Every event in the system has an event schema. An event schema specifies the data attributes all events implementing the event schema must have. The name of an event schema is assumed to be unique. All events inherit from a base event schema. Advertisement and subscription requests are also handled through events.

4.2.1 Base Schema

All event schemas inherit from a base schema. This base event schema specifies the following attributes: *(i)* event time, *(ii)* publication time and *(iii)* time-to-live (TTL).

Event time refers to the time the happening was observed by the component. *Publishing time* refers to the time when the event was published. *TTL* is set by the component publishing the event and determines the amount of time that the event remains relevant to the system.

4.2.2 Advertisement Schema

The advertisement schema is used by components to announce its intention of publishing events. It inherits from the base schema and therefore has all of the attributes declared in it. It also specifies the following attributes: *(i)* identifier, *(ii)* event schema, *(iii)* context filtering expression and *(iv)* event scope.

Identifier is used to identify the publishing component, that is, the source of the event. *Event schema* is a reference to the event schema being implemented by events published by this component. *Context filtering expression* is used to determine the context in which publication of the event should take place. *Event scope* (defined in Section 4.3) indicates the scope of the published event.

A component's advertisements are identified by the event schema. Any advertisement can be updated by the component by publishing an advertisement event matching the same event schema with different context filtering expressions or event scope.

4.2.3 Subscription Schema

The subscription schema is used by components to announce its interest in a specific event schema. It also inherits from the base schema and therefore has all of the attributes declared in it. The subscription schema also specifies the following attributes: *(i)* identifier, *(ii)* event schema, *(iii)* content filtering expression and *(iv)* context filtering expression and *(v)* subscription scope.

Identifier is used to identify the subscribing component, that is, the component that should notified of an event implementing the given event schema. *Event schema* is a reference to the event schema of interest for this subscriber. *Content filtering expression* is used to filter events based on the values of its attributes. *Context filtering expression* is used to filter events based on the context of the environment in which the subscription is defined. An event is only delivered to a subscribing component if both the content filtering expression and the context filtering expression evaluate to `true`. *Subscription scope* (defined in Section 4.3) indicates the desired scope of the subscription.

4.2.4 Context Schema

The context schema is used as the base schema for events carrying context information. It inherits from the base schema and adds the following attribute: *(i)* context identifier.

Context identifier refers to the type of context the event is being used to propagate. It should refer to a unique identifier within the context model and properly identify which context value or structure the information contained in the event relates to.

4.3 Event and Subscription Scope

The notion of event and subscription scoping refers to a form of event privacy control. Grouping components into environments allows publishers and subscribers to control their publication and subscription range in various ways. An event scope limits the reach of a published event. A subscription scope limits the set of valid publishers in a subscription.

An event can be allowed to be consumed only by a publisher's sibling components, only by outside components or by both. A *local* scope limits the event range to sibling components within the same environment as the publisher. A *nonlocal* scope limits the event to outside components only. An *any* scope allows an event to reach sibling components within the same environment as the publisher as well as outside components.

Similarly, a subscriber is able to specify if it is interested in events originating from within the environment, from outside the environment or from both. A *local* scope refers to events originating from sibling components within the same environment as the subscriber. A *nonlocal* scope refers to events originating from components outside the subscriber's environment. An *any* scope refers to events originating both from inside and outside the subscriber's environment.

4.4 Event Advertisement, Publishing and Subscribing

Before publishing an event, a component must advertise the event to the system. Advertisement is accomplished by publishing an event that implements the advertisement schema. The advertisement schema provides all the necessary information to identify a component, the event schema, the context filter and the event scope. An advertisement can be removed by publishing an unadvertisement event for the desired event schema.

Once an event schema has been advertised, the component can publish the event. An event will only be published if the `context` parameter specified in the `advertisement` event matches the state of the environment when the `publish` request is made.

A component wishing to be notified of events of type $\texttt{schema}_E$ advertised in the system must subscribe publishing a subscription event matching the event schema $\texttt{schema}_E$. A subscription can be cancelled by publishing an unsubscribe event.

4.5 Contextual Environment

A context environment is a logic region used to set bounds to an instance of the context model. All of the components inside the same environment share the same context parameters and values. A component can be part of more than one environment at a time. Subscription and advertisement of events are submitted to a particular environment. If a component belongs to more than one environment its subscription and advertisements are distinct between environments. For example, if a component belongs to both environment ENV_1 and ENV_2 and subscribes to event e_A in environment ENV_1 and e_B in ENV_2, it will not be notified of an event e_A published in ENV_2.

Any event that affects the values of context parameters in an environment will also affect the components contained inside the environment. Context filtering in both event advertisement and subscription is based on the context parameters of the environment of which the component is part. An event will only be published if the context filter expression specified in the advertisement matches the context parameters of the environment. In the case that the event being published represents a context change in an environment, updating of the environment context parameters takes precedence over context filter expression evaluation. Similarly, an event will only be delivered to a component if the context filter expression specified in the subscription matches the context of the environment.

4.6 Adaptive Components

An adaptive component is a self-contained software entity that provides all of the necessary modules to accomplish some specific task or function. Any internal communication between a component's modules are handled by the modules themselves and not through system events.

A component can be both a publisher and a subscriber concurrently. Components are the only system entity that is able to publish or subscribe to events, in other words, there is no entity that is capable of generating or subscribing to events other than a component.

All context sources in the system are components. A component that is a context source should encapsulate all of the necessary modules to capture and process the context information. Context information is then disseminated through an event implementing a pre-defined event schema that corresponds to the context being captured.

All context sinks in the system are components. A component that is a context sink subscribes to the event schema corresponding to the context information it requires. Not all components are context aware, and not all context-aware components are context sources or sinks. This distinction is relevant since a component can still be context aware, i.e. adapt its behaviour to the current context state, without the need of keeping track of the context information.

Context awareness is accomplished transparently and autonomically through the use of context filters in publications and subscriptions. An event will either be published by the system or delivered to a given component if and only if the context filtering expression matches the context of the environment. This simple rule allows a component to associate different behaviours with notification of different events. A component can achieve behaviour adaptation by controlling under which context it is notified of an event.

Adaptation of the publication of events is also easily accomplished. A component specifies a context filtering expression with the advertisement of an event publication. Upon receiving the publication request from the component, the system will evaluate the context filtering expression against the current context of the environment. If the context matches, the event will be published to all subscribing components. A component can achieve publication adaptation by controlling under which context its events will be published.

Such approach removes the burden of determining the validity and applicability of the event notification or publication from the component and places it on the system. A component is no longer in need of providing its own copy of the context model or contacting an external context database. It is also not required to keep track of any context changes in the environment. With context aware adaptation being part of the publish-subscribe scheme in the form of context filtering expressions, a component is effectively a collection of behaviours that are combined to provide a specific functionality. All of the application logic is contained inside the component and clearly differentiated from context information.

5 A Proposed Context-Aware Event-Based Architecture

The architecture of the framework is shown in Figure 5. The framework is composed of three different layers, the context-aware application layer, where custom context-aware components and application are placed. These components are the publishers and subscribers in an environment. The middleware layer contains the administrative components responsible for managing advertisements, subscriptions along with content and context filtering and scoping. The bottom layer refers to the underlying distributed event-based system used to support our system.

5.1 Context-Aware Application Layer

All different custom components in the system are placed in this layer. A component may relate to an entire self-contained application or a simple module or service providing some specific functionality. Components are responsible for all generation and consumption of events in the system.

Components can be either context providers or context consumers. A context provider is a component that generates context events and propagates them through the system, either inside or outside

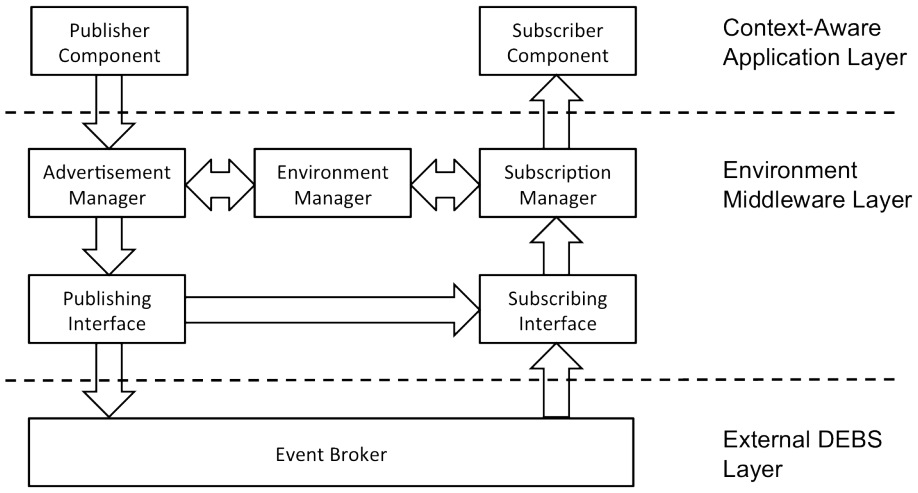


Figure 2: Architecture of the context-aware event-based framework.

the environment. Context consumers are components that use context information in order to provide behaviour adaptation.

Events without context can also be propagated to environments and components throughout the system. Components can be classified into plain components and context-aware components.

5.1.1 Plain Components

A component is considered plain if it is completely oblivious to context. In other words, plain components do not subscribe to context events or use any form of context filtering in their subscriptions.

Plain components can still be context sources and publish context events. Being able to sense some type of context, in other words, having knowledge of some type of context is not sufficient to characterize context awareness. Both knowledge and adaptation are required for a component to be considered context-aware.

5.1.2 Context-Aware Components

A component is considered context-aware if and only if it has knowledge of some type of context and if it is able to change its behaviour according to changes in said context. A context-aware component doesn't necessarily need to receive context information by subscribing to context events. Knowledge of context can be expressed through context filters in its subscription. Behaviour adaptation follows from specifying under which circumstances it is to be notified of a specific event.

This approach guarantees that a component is aware of some type of context and that changes to this type of context may affect its behaviour. Any component that uses context filters in its subscriptions is considered context-aware.

5.2 Environment Middleware Layer

The environment middleware layer contains many different administrative components that are used to enable advertisement, publication and subscription to take place in the system. It is also responsible for context filtering and event and subscription scoping as well as communication with other peers or event brokers in a distributed event-based system.

5.2.1 Environment Manager

The environment manager is the administrative component responsible for keeping track of the state of context in the environment. It has its own instance of the context model which is updated according to events published both from inside and outside the environment.

The environment manager is responsible for context disambiguation. It provides the necessary logic to analyze context events, extract the context information and ensure that the context model instance is always in a coherent state.

The environment manager is also accessible to components in the context-aware application layer. This access provides a simple query interface service that can be used by components to access the context information in a way that is similar to a context knowledge base. The environment manager is not limited only to the role of a context repository. It can contain other context-related components such as context interpreters and context reasoning capabilities that are used to extract higher-order context information from the context model.

5.2.2 Advertisement Manager

The advertisement manager is responsible for handling all context filtering in advertisements from components in the environment. When a component publishes an event, the advertisement manager checks if the event has any context information pertaining to the environment. The environment manager is notified of any context information present in events being published from inside the environment.

After the context state is updated, the advertisement manager processes the event publication according to the parameters specified in the components advertisement of the event. The advertisement manager checks if the event matches any event schema in the component's advertisements, and if the current context state matches the specified context filtering expression. The event is forwarded to the publishing interface if those conditions hold.

5.2.3 Subscription Manager

The subscription manager is responsible for handling all context filtering in subscriptions from components in the environment. It also forwards context information from events originating outside the environment to the environment manager. Such events may still affect the context state of the environment and should be considered by the environment manager.

The subscription manager is able to match components with their subscriptions, request context information from the environment manager, evaluate context filtering expressions and subscription scope values. An event is delivered to a subscriber if the event schema matches the one specified in the subscription, and if current environment context state matches the context filtering expression and if the subscription scope matches the event origin.

5.2.4 Publishing Interface

The publishing interface is an event forwarding component. It handles dissemination of both private and public events. The publishing interface contains all of the necessary logic to publish an event both to the environment's subscribing interface and to any underlying distributed event-based system.

Events are allowed to leave the environment if and only if a publishing component explicitly states the correct privacy scope in its event advertisement. Whenever a component publishes an event with a privacy scope of *nonlocal* or in its advertisement, the publishing component forwards the event to the underlying distributed event-based system. If a component publishes an event with a privacy scope of *local* in its advertisement, the publishing component forwards the event to the environments subscribing interface. If the advertisement states a privacy scope of *any*, the event will be forwarded both to the environment's subscribing interface and to the underlying distributed event-based system.

5.2.5 Subscribing Interface

The subscribing interface is an event aggregator component. It aggregates both private and public events directed at the environment. The subscribing interface is responsible for issuing and updating

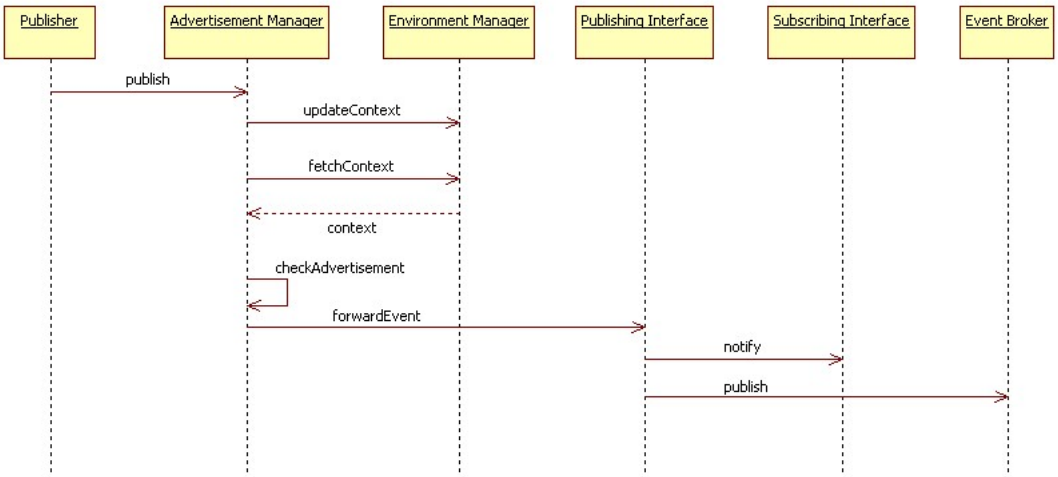


Figure 3: Publication of events.

subscriptions to the underlying distributed event-based system.

Outside events are forwarded to an environment if and only if a subscriber component explicitly states the correct privacy scope in its subscription to an event schema. Whenever a component subscribes to an event with a subscription scope value of *nonlocal* or *any*, the subscribing interface forwards the event subscription to the underlying distributed event-based system. This allows outside events to reach the environment.

If, on the other hand, a component publishes an event with privacy scope of either *local* or *any* and there are no components in the environment that subscribe to this event's schema, the even is not forwarded to the subscription manager. This limits the number of events the subscription manager has to process. It is important to note that if this event is a context event, the context information has already been used in updating the environment manager during the processing done by the advertisement manager.

5.2.6 Publication and Notification of Events

The framework provides two very specific operations. The first operation relates to internal components publishing events through the system. Figure 3 shows the sequence diagram with the process used in publishing an event. Whenever an event publication takes place in an environment, the published event is first received by the Advertisement Manager. If the event carries any context information or update, this data is sent to the Environment Manager in order to update the current context state. After any context updates have taken place, the Advertisement Manager fetches the necessary context information from the Environment Manager and processes the advertisement parameters associated with the published event. If the event is cleared for publication, it is forwarded to the publishing interface. The publishing interface determines the event scope and notifies the Subscribing Interface and/or publishes the event to the underlying distributed event-based system.

Figure 4 shows the sequence diagram with the process used in notifying a subscriber of a published event. The Subscribing Interface is the main entry point for event produced either inside or outside the environment. After receiving an event, the Subscribing Interface checks if there are any subscriptions associated with the schema of the event being published into the environment. If there are no subscriptions matching the schema, the event processing is aborted and no other action is taken. If there are components subscribing to the schema, the event is forwarded to the Subscription Manager.

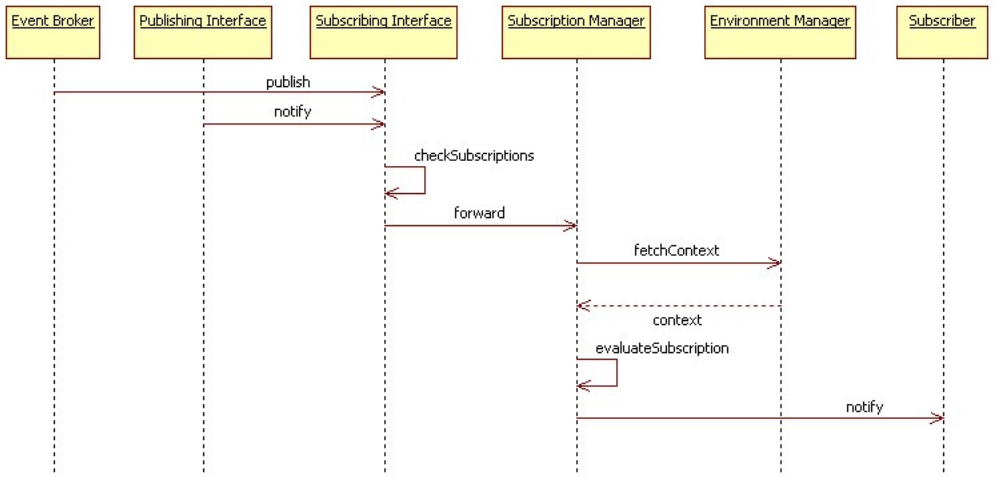


Figure 4: Notification of events.

5.3 DEBS Layer

The DEBS layer refers to the external distributed event-based system to which this environment is connected. Such systems can be other distributed event-based middleware systems, which can be represented by an event broker node, or other environments connected in a peer-to-peer fashion. Such an approach is very flexible in terms of network layout and control.

6 Evaluation

We plan to evaluate our approach in three ways: *(i)* framework implementation, *(ii)* application case studies and *(iii)* formal verification of the event model and context-aware publish-subscribe scheme.

We will model and implement a context-aware distributed event-based framework that supports the new context-aware publish-subscribe scheme enabled by our context-aware event model. This framework will also implement the functional requirements described in Section 2.1.3, making it a complete solution for the design and implementation of mobile context-aware applications. We have conducted a preliminary design of the framework that is based on our proposed context-aware event model and publish-subscribe scheme (Figure 5). The framework architecture consists of three layers: the context-aware application layer, the environment middleware layer and the external distributed event-based layer. The external DEBS layer represents the underlying distributed event-based system infrastructure. The environment middleware layer implements our novel context-aware publish-subscribe scheme, which goes beyond a type-based scheme by providing local context-aware control over publications and subscriptions of events, decouples context-aware information from application logic and supports a context-model agnostic approach. The context-aware application layer is where the context-aware applications interact with the system through our context-aware publish-subscribe scheme.

The framework will then be used to design and implement various case studies dealing with behaviour adaptation in context-aware applications. These case studies will enable us to demonstrate, explain and verify how changes in context result in changes in behaviour, enabling specific components to adapt to the context at hand. As part of our case studies we will also conduct a scenario-based evaluation.

Finally, we will validate our context-aware event model and subsequently our context-aware publish-subscribe scheme. Validation will be accomplished by extending the distributed event-based system formalization based on the kell-m process calculus [10, 12]. Many critical applications, such as e-

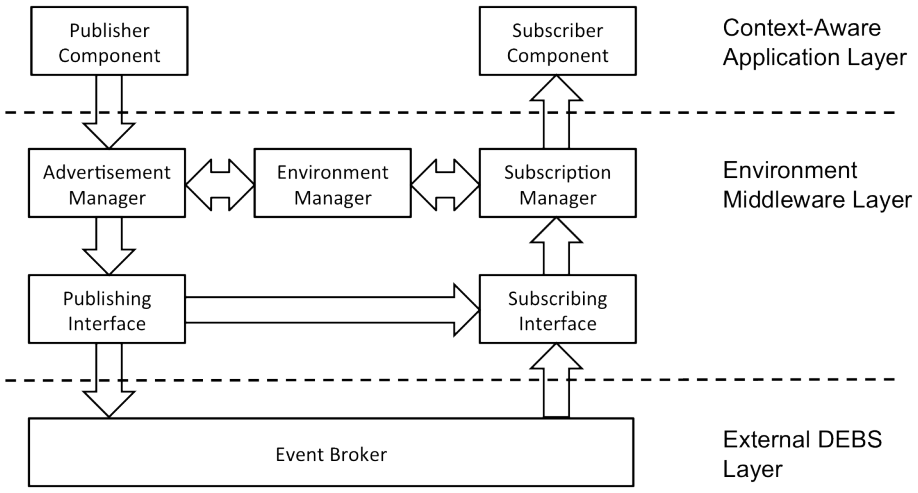


Figure 5: Preliminary architecture design of the context-aware event-based framework.

health, e-commerce and emergency services, are dependent on events that must happen under specific contextual circumstances. Such crucial event notification must have a guaranteed delivery to the interested subscribers. In this sense, our proposed validation approach will provide the necessary quality of service assurance in handling such applications.

7 Preliminary Scenario Based Evaluation

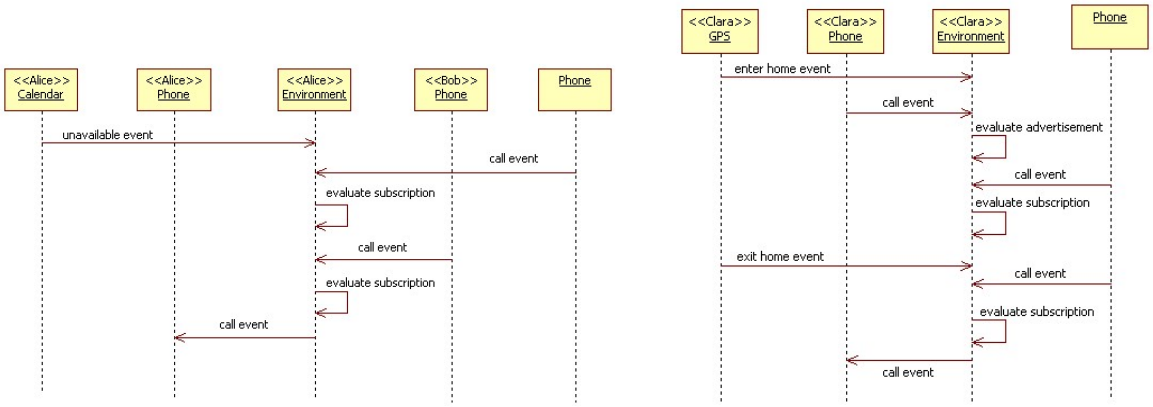
In this section, we provide a scenario-based evaluation [64] of our context-aware distributed event-based framework by walking through different scenarios and demonstrating how different parts of the system work together. For our sample use cases, we consider the realm of mobile devices. The various applications in a mobile device can relate to different components in our distributed event-based system. Each component has its own function. For example, the Phone component is responsible for making and receiving calls, the AddressBook component holds the user's contact information, the Calendar component holds the user's schedule and the GPS component is responsible for keeping track of the user's location.

Context in mobile devices is very dynamic since these types of devices are constantly with the user, changing environments according to the user's actions. Mobile devices are also very constrained in terms of memory, processing power and may be very inconvenient to use when excessive user input is required. Processing power and memory constraints encourage the different components to focus on providing a single functionality instead of being all encompassing applications like some desktop applications. Context awareness can be achieved simply by providing a clean and effective way to share information between these different components.

7.1 Use Case: Scheduled Meeting

Suppose in our use case that Alice is in a meeting with her co-workers and only wants to receive calls from Bob, her boss. Current approaches in smartphones do not allow for easy automation of such scenario. Alice needs to be aware of her schedule and switching her phone to silent upon entering a meeting. Even if this is the case, Alice still needs to check any calls she might receive during her meeting and manually screen the calls from Bob. Such simple adaptation can be easily achieved through our approach.

The pieces of context that are represented in our scenario are Alice's status, i.e. *unavailable* when in a meeting and *available* otherwise, and her relationship with Bob, i.e. calls from her boss are



(a) Scheduled meeting use case sequence diagram.

(b) Context-aware calls use case sequence diagram.

Figure 6: Use case sequence diagrams.

important to her. Alice's smartphone Calendar component is responsible for publishing events relating to her status whenever there's a change in her schedule. By creating the following subscription, the Phone component can be made aware of Alice's status and adapt its behaviour to ring only if she is available, *subscription(IncomingCall, Status='available', scope='local')*. If Alice's status is *unavailable* then the subscription to incoming calls is not valid and the *IncomingCall* event will not be propagated to the Phone component. When Alice's status is *available* then the subscription is valid and the *IncomingCall* event gets propagated. In the case of allowing Bob's call to go through, another subscription *subscription(IncomingCall, Caller='Bob')* can be appended. If either of the subscriptions is valid the event will be propagated to the Phone component. As a result, even if Alice's status is *unavailable* then she can still receive Bob's call. Figure 6(a) shows the sequence diagram for this use case.

7.2 Use Case: Context-Aware Calls

Suppose now that Alice doesn't want her teenage daughter Clara to use her cell phone at home. Currently there's no simple way to guarantee such functionality other than by taking the device away from Clara. With our approach it becomes a matter of having the correct subscriptions and publication expressions.

Clara's current location is provided by the GPS component in her cell phone. Whenever Clara's location changes, the GPS component publishes a *LocationChange* event that results in the Context Broker updating the context knowledge base for this environment. By using the following advertisement expression in the Phone component *advertisement(OutgoingCall, Location!='home')* we can prevent Clara from making outgoing calls when at home. If the Phone component publishes an *OutgoingCall* event the Context Broker checks its current location and validates the context state of the mobile device. If the location is different from home then the *OutgoingCall* event gets propagated to the service provider, requesting a connection to the target phone. If the location is home, this event publication is not propagated and the outgoing call connection will not be requested. Similarly, by adding the subscription *subscription(IncomingCall, Location!='home')* we can prevent Clara from receiving incoming calls while at home. Figure 7.2 shows the sequence diagram for this use case.

7.3 Discussion

In previous sections we have presented the primary building blocks of an adaptive context-aware distributed event-based framework. Our insights with scenario-based evaluation have unveiled three

design issues, which are discussed next.

A. Context availability. As discussed in the use case described in Section 6(a), the first design concern is related to context availability. In current context-aware approaches, a component must have access to a context source that provides the information it requires in order to adapt its behaviour to the user's context. In other words, Alice's Phone component is required to have access to context information being provided by Alice's Calendar component. A proposed solution using current context-aware frameworks would require the Phone component to search for a schedule context provider and register with it to receive updates to Alice's schedule. This component would be Alice's Calendar component. The Calendar component would contact the Phone component whenever a change in Alice's schedule takes place. Alice's Phone component would be required to deal with said context information on the source code level, that is a hardcoded solution. This affects the overall design and implementation of the Phone component. It is now required to keep track of schedule changes, which is not it's intended purpose. Whenever the Phone component receives a call notification, it is then required first check if it's stored status allows it to ring.

B. Incremental Context-Aware Publications and Subscriptions. A second design concern relates to extending context-aware publications and subscriptions. As seen in Section 6(a), Alice's Phone component should not be dependent only on Alice's status, but also depend on the callers identity. Alice's Contacts components holds all the information relating to the people close to Alice and their relationship with her. If Alice's status is unavailable the Phone component is required to check if the caller is allowed to go through. In order for Bob's call go through, the Phone component has to access Alice's Contacts component. This once again requires changes to the Phone component and adds extra functionality that does not relate to the Phone component's main objective of placing and receiving calls. Such hardcoded solution is in clear contrast with the straightforward approach provided by our context-aware publish subscribe scheme.

C. Local versus Foreign Context. A third design concern relates to local versus foreign context information. The use case described in Section 7.2 shows a clear example of using local context in publications and subscriptions. Current context-aware publish subscribe approaches limit the use of context only for entities foreign to the publisher or subscriber. In Clara's case it would be required to modify the Phone components of all those who call her in order to enforce the location restrictions. A Phone component calling Clara would be required to specify that the Phone component receiving the call event would only be notified of the event if its location is different from Clara's home. Similarly, a Phone component receiving a call event from Clara's Phone component would be required to specify that the location of the event publisher be different from Clara's home. Through this example it is clear that Clara's Phone component has no control over its behaviour of placing, accepting or rejecting a call events. It is entirely dependent on external components and their knowledge of a foreign entity's context.

8 Conclusion

In this research proposal we have claimed that distributed event-based models and approaches should use context as a first class element, where the local context of its components should be taken into consideration in publications and subscriptions of events. To this end, we aim at introducing a context-aware event model and a new publish-subscribe scheme. In addition we plan to define a context-aware distributed event-based framework that provides support for mobile context-aware user interactions through real time subscription and publication behaviour adaptation. We have also discussed the utility of our proposed research through a scenario-based evaluation.

Further, we are working on designing and implementing an object-oriented version of the proposed framework. We will also work on the formalization of our models to be able to support the verification of critical context-aware applications based on the our proposed context-aware publish-subscribe scheme. Overall, we believe that an approach that combines context and distributed events and supports a publish-subscribe scheme that is affected by local context will contribute to further research exploration in mobile and context-aware systems.

References

[1] M. Appeltauer, R. Hirschfeld, and T. Rho. Dedicated programming support for context-aware ubiquitous applications. In *Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008. UBICOMM '08. The Second International Conference on*, pages 38 –43, 10 2008.

[2] Malte Appeltauer, Robert Hirschfeld, Michael Haupt, Jens Lincke, and Michael Perscheid. A comparison of context-oriented programming languages. In *International Workshop on Context-Oriented Programming*, COP '09, pages 6:1–6:6, New York, NY, USA, 2009. ACM.

[3] Malte Appeltauer, Robert Hirschfeld, Hidehiko Masuhara, Michael Haupt, and Kazunori Kawauchi. Event-specific software composition in context-oriented programming. In Benoît Baudry and Eric Wohlstadter, editors, *Software Composition*, volume 6144 of *Lecture Notes in Computer Science*, pages 50–65. Springer Berlin / Heidelberg, 2010.

[4] Marco Autili, Paolo Di Benedetto, and Paola Inverardi. A programming model for adaptable java applications. In *Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java*, PPPJ '10, pages 119–128, New York, NY, USA, 2010. ACM.

[5] Engineer Bainomugisha, Wolfgang De Meuter, and Theo D'Hondt. Towards context-aware propagators: language constructs for context-aware adaptation dependencies. In *International Workshop on Context-Oriented Programming*, COP '09, pages 8:1–8:4, New York, NY, USA, 2009. ACM.

[6] Matthias Baldauf and Schahram Dustdar. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2004.

[7] Eduardo S. Barrenechea, Rolando Blanco, and Paulo Alencar. Modeling context-aware distributed event-based systems. Technical report, David R. Cheriton School of Computer Science, University of Waterloo, 4 2010.

[8] Paolo Bellavista, Antonio Corradi, Rebecca Montanari, and Cesare Stefanelli. A mobile computing middleware for location- and context-aware internet data services. *ACM Trans. Internet Technol.*, 6:356–380, November 2006.

[9] Gregory Biegel and Vinny Cahill. A framework for developing mobile, context-aware applications. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, PERCOM '04, pages 361–, Washington, DC, USA, 2004. IEEE Computer Society.

[10] Rolando Blanco and Paulo Alencar. Modelling distributed event-based systems using the kell-m calculus. Technical report, David R. Cheriton School of Computer Science, University of Waterloo, 1 2001.

[11] Rolando Blanco and Paulo Alencar. Towards modularization and composition in distributed event based systems. Technical report, David R. Cheriton School of Computer Science, University of Waterloo, 2009.

[12] Rolando Blanco and Paulo Alencar. Semantics and encoding of the kell-m calculus. Technical report, David R. Cheriton School of Computer Science, University of Waterloo, 1 2011.

[13] Rolando Blanco, Jun Wang, and Paulo Alencar. A metamodel for distributed event based systems. In *Proceedings of the second international conference on Distributed event-based systems*, DEBS '08, pages 221–232, New York, NY, USA, 2008. ACM.

[14] Rolando M. Blanco. *Process Models for Distributed Event-Based Systems.* PhD thesis, David R. Cheriton School of Computer Science, University of Waterloo, 2010.

[15] Cristiana Bolchini, Carlo A. Curino, Elisa Quintarelli, Fabio A. Schreiber, and Letizia Tanca. A data-oriented survey of context models. *SIGMOD Rec.*, 36:19–26, December 2007.

[16] Andre Bottaro, Johann Bourcier, Clement Escoffier, and Philippe Lalanda. Context-aware service composition in a home control gateway. In *Pervasive Services, IEEE International Conference on*, pages 223 –231, 7 2007.

[17] Licia Capra, Wolfgang Emmerich, and Cecilia Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Trans. Softw. Eng.*, 29:929–945, October 2003.

[18] Stefano Ceri, Florian Daniel, Federico M. Facca, and Maristella Matera. Model-driven engineering of active context-awareness. Technical report, Dipartimento di Elettronica e Informazione Politecnico di Milano, 2005.

[19] Stefano Ceri, Florian Daniel, Maristella Matera, and Federico M. Facca. Model-driven development of context-aware web applications. *ACM Transactions on Internet Technology (ACM TOIT*, 7:2007, 2007.

[20] Alvin T. S. Chan and Siu-Nam Chuang. Mobipads: A reflective middleware for context-aware mobile computing. *IEEE Trans. Softw. Eng.*, 29:1072–1085, December 2003.

[21] Yu-Ling Chang. A dynamic user-centric mobile context model. Master's thesis, David R. Cheriton School of Computer Science, University of Waterloo, 2010.

[22] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical report, Department of Computer Science, Dartmouth College, 2000.

[23] Guanling Chen and David Kotz. Solar: An open platform for context-aware mobile applications. In *In Proceedings of the First International Conference on Pervasive Computing (Pervasive 2002*, pages 41–47, 2002.

[24] Harry Chen. An intelligent broker architecture for context-aware systems. Technical report, University of Maryland, 2003.

[25] Harry Chen, Tim Finin, and Anupam Joshi. Semantic web in the context broker architecture. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, PERCOM '04, pages 277–, Washington, DC, USA, 2004. IEEE Computer Society.

[26] Denis Conan, Romain Rouvoy, and Lionel Seinturier. Scalable processing of context information with cosmos. In Jadwiga Indulska and Kerry Raymond, editors, *Distributed Applications and Interoperable Systems*, volume 4531 of *Lecture Notes in Computer Science*, pages 210–224. Springer Berlin / Heidelberg, 2007.

[27] Pascal Costanza and Robert Hirschfeld. Language constructs for context-oriented programming: an overview of contextl. In *Proceedings of the 2005 symposium on Dynamic languages*, DLS '05, pages 1–10, New York, NY, USA, 2005. ACM.

[28] Marcel Cremene, Michel Riveill, and Christian Martel. Autonomic adaptation solution based on service-context adequacy determination. *Electron. Notes Theor. Comput. Sci.*, 189:35–50, 7 2007.

[29] G. Cugola, A. Margara, and M. Migliavacca. Context-aware publish-subscribe: Model, implementation, and evaluation. In *Computers and Communications, 2009. ISCC 2009. IEEE Symposium on*, pages 875 –881, 7 2009.

[30] Gianpaolo Cugola and H. arno Jacobsen. Using publish/subscribe middleware for mobile systems. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6:2002, 2002.

[31] Gianpaolo Cugola and Matteo Migliavacca. On context-aware publish-subscribe, 2008.

[32] Ricardo Couto A. da Rocha, Markus Endler, and Thiago Senador de Siqueira. Middleware for ubiquitous context-awareness. In *Proceedings of the 6th international workshop on Middleware for pervasive and ad-hoc computing*, MPAC '08, pages 43–48, New York, NY, USA, 2008. ACM.

[33] Cléver R. G. de Farias, Marcos M. Leite, Camilo Z. Calvi, Rodrigo M. Pessoa, and José G. Pereira Filho. A mof metamodel for the development of context-aware mobile applications. In *Proceedings of the 2007 ACM symposium on Applied computing*, SAC '07, pages 947–952. ACM, 2007.

[34] R.C.A. de Rocha and M. Endler. Middleware: Context management in heterogeneous, evolving ubiquitous environments. *Distributed Systems Online, IEEE*, 7(4):1 –1, 2006.

[35] Alan Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker White. Towards expressive publish/subscribe systems. In *In Proc. EDBT*, pages 627–644, 2006.

[36] Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5:4–7, 2001.

[37] Anind K. Dey, Gregory D. Abowd, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *HUC*, pages 304–307, 1999.

[38] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, 2001.

[39] Anind K. Dey and Jennifer Mankoff. Designing mediation for context-aware applications, 2005.

[40] Anind K. Dey, Daniel Salber, and Gregory D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 434–441, New York, NY, USA, 1999. ACM.

[41] Jens-peter Dittrich. Agile: Adaptive indexing for context-aware information filters. In *In Proc. of the 24th ACM SIGMOD Intl. Conf. on Management of Data*, pages 215–226, 2005.

[42] Simon Dobson, Paddy Nixon, and Lorcan Coyle. Hybridising events and knowledge as a basis for building autonomic systems. *IEEE TCAAS Letters*, 2007.

[43] Weichang Du and Lei Wang. Context-aware application programming for mobile devices. In *Proceedings of the 2008 C3S2E conference*, C3S2E '08, pages 215–227, New York, NY, USA, 2008. ACM.

[44] Hector A. Duran-limon, Gordon S. Blair, Adrian Friday, Paul Grace, George Samartzidis, Thirunavukkarasu Sivaharan, and Maomao Wu. Context-aware middleware for pervasive and ad hoc environments. Technical report, Computing Department, Lancaster University, 2003.

[45] R. Etter, P.D. Costa, and T. Broens. A rule-based approach towards context-aware user notification services. In *Pervasive Services, 2006 ACS/IEEE International Conference on*, pages 281 –284, 6 2006.

[46] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe, 2003.

[47] Patrick Fahy and Siobhan Clarke. Cass – a middleware for mobile context-aware applications. In *Workshop on Context Awareness, MobiSys*, 2004.

[48] Patrik Floréen, Michael Przybilski, Petteri Nurmi, Johan Koolwaaij, Anthony Tarlano, Matthias Wagner, Marko Luther, Fabien Bataille, Mathieu Boussard, Bernd Mrohs, and Sianlun Lau. Towards a context management framework for mobilife. In *In IST Mobile and Wireless Communications Summit*, 2005.

[49] Davide Frey and Gruia-Catalin Roman. Context-aware publish subscribe in mobile ad hoc networks. In Amy Murphy and Jan Vitek, editors, *Coordination Models and Languages*, volume 4467 of *Lecture Notes in Computer Science*, pages 37–55. Springer Berlin / Heidelberg, 2007.

[50] Serena Fritsch, Aline Senart, and Siobhan Clarke. Addressing dynamic contextual adaptation with a domain-specific language. In *Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*, SEPCASE '07, pages 2–, Washington, DC, USA, 2007. IEEE Computer Society.

[51] K. Geihs. Middleware challenges ahead. *Computer*, 34(6):24 –31, June 2001.

[52] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A middleware for building context-aware mobile services. In *Vehicular Technology Conference, 2004. VTC 2004-Spring. 2004 IEEE 59th*, volume 5, pages 2656 – 2660 Vol.5, 5 2004.

[53] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A service-oriented middleware for building context-aware services. *J. Netw. Comput. Appl.*, 28:1–18, 1 2005.

[54] Karen Henricksen and Jadwiga Indulska. Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, 2(1):37 – 64, 2006.

[55] Karen Henricksen, Jadwiga Indulska, Ted McFadden, and Sasitharan Balasubramaniam. Middleware for distributed context-aware systems. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, volume 3760 of *Lecture Notes in Computer Science*, pages 846–863. Springer Berlin / Heidelberg, 2005.

[56] Annika Hinze, Petra Malik, and Robi Malik. Interaction design for a mobile context-aware system using discrete event modelling. In *Proceedings of the 29th Australasian Computer Science Conference - Volume 48*, ACSC '06, pages 257–266, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[57] Robert Hirschfeld, Pascal Costanza, and Oscar Nierstrasz. Context-oriented programming. *Journal of Object Technology, March-April 2008, ETH Zurich*, 7(3):125–151, 2008.

[58] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. Context-awareness on mobile devices - the hydrogen approach. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9 - Volume 9*, HICSS '03, pages 292.1–, Washington, DC, USA, 2003. IEEE Computer Society.

[59] Jason I. Hong and James A. Landay. An infrastructure approach to context-aware computing. *Hum.-Comput. Interact.*, 16:287–303, December 2001.

[60] Peizhao Hu, Suan Khai Chong, J. Indulska, and S. Krishnaswamy. Context-aware and resource efficient sensing infrastructure for context-aware applications. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, pages 492 –498, 4 2010.

[61] Yongqiang Huang and Hector Garcia-molina. Publish/subscribe in a mobile environment. *MobiDE*, 2001.

[62] Rui José, Adriano Moreira, Helena Rodrigues, and Nigel Davies. The around architecture for dynamic location-based services. *Mob. Netw. Appl.*, 8:377–387, 8 2003.

[63] Georgia M. Kapitsaki, George N. Prezerakos, Nikolaos D. Tselikas, and Iakovos S. Venieris. Context-aware service engineering: A survey. *Journal of Systems and Software*, 82(8):1285 – 1297, 2009. SI: Architectural Decisions and Rationale.

[64] Rick Kazman, Gregory Abowd, Len Bass, and Paul Clements. Scenario-based analysis of software architecture. *IEEE Softw.*, 13:47–55, November 1996.

[65] Issa Khalil, Saurabh Bagchi, Yunhua Koglin, Elisa Bertino, and Xukai Zou. Publish/subscribe systems, 2006.

[66] Kristian Ellebæk Kjær. A survey of context-aware middleware. In *Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering*, pages 148–155, Anaheim, CA, USA, 2007. ACTA Press.

[67] Johan Koolwaaij, Anthony Tarlano, Marko Luther, Petteri Nurmi, Bernd Mrohs, Agathe Battestini, and Raju Vaidya. Context watcher: Sharing context information in everyday life. In *in Proceedings of the IASTED conference on Web Technologies, Applications and Services (WTAS). IASTED*, pages 12–21, 2006.

[68] P. Korpipaa, J. Mantyjarvi, J. Kela, H. Keranen, and E.J. Malm. Managing context information in mobile devices. *Pervasive Computing, IEEE*, 2(3):42 – 51, 2003.

[69] Hui Lei, Daby M. Sow, John S. Davis, II, Guruduth Banavar, and Maria R. Ebling. The design and applications of a context service. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6:45–55, October 2002.

[70] Jinjiao Lin, Chengxiang Song, and Haiyang Wang. A Rule-based Method for Improving Adaptability in Pervasive Systems. *The Computer Journal*, 53(2):177–190, 2010.

[71] Marco Mamei and Franco Zambonelli. Programming pervasive and mobile computing applications with the tota middleware. In *PerCom 2004. Proceedings of the Second IEEE Annual Conference on*, pages 263–273, 2004.

[72] Hidehiko Masuhara and Gregor Kiczales. Modeling crosscutting in aspect-oriented mechanisms. pages 2–28. Springer-Verlag, 2003.

[73] René Meier and Vinny Cahill. Exploiting proximity in event-based middleware for collaborative mobile applications. In Jean-Bernard Stefani, Isabelle Demeure, and Daniel Hagimont, editors, *Distributed Applications and Interoperable Systems*, volume 2893 of *Lecture Notes in Computer Science*, pages 285–296. Springer Berlin / Heidelberg, 2003.

[74] Daniel Coutinho De Mir, Marco Tulio, and Oliveira Valente. A flexible and extensible component-oriented middleware for creating context-aware applications, 2006.

[75] Oscar Nierstrasz, Marcus Denker, and Lukas Renggli. Model-centric, context-aware software adaptation. In Betty Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 128–145. Springer Berlin / Heidelberg, 2009.

[76] Angel Núñez and Jacques Noyé. An event-based coordination model for context-aware applications. In *Proceedings of the 10th international conference on Coordination models and languages*, COORDINATION'08, pages 232–248, Berlin, Heidelberg, 2008. Springer-Verlag.

[77] Angel Núñez, Jacques Noyé, and Vaidas Gasiūnas. Declarative definition of contexts with polymorphic events. In *International Workshop on Context-Oriented Programming*, COP '09, pages 2:1–2:6, New York, NY, USA, 2009. ACM.

[78] Tapio Pitkäranta, Oriana Riva, and Santtu Toivonen. Designing and Implementing a System for the Provision of Proactive Context-aware Services. *Workshop on Context Awareness for Proactive Systems (CAPS'05)*, 16-17 June 2005.

[79] Anand Ranganathan and Roy H. Campbell. A middleware for context-aware agents in ubiquitous computing environments. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, Middleware '03, pages 143–161, New York, NY, USA, 2003. Springer-Verlag New York, Inc.

[80] Oriana Riva and Santtu Toivonen. A hybrid model of context-aware service provisioning implemented on smart phones. In *In Proc. ICPS'06*, pages 47–56, 2006.

[81] Walid Rjaibi, Klaus R. Dittrich, and Dieter Jaepel. Event matching in symmetric subscription systems. In *Proceedings CASCON*, 2002.

[82] Manuel Román, Christopher Hess, Renato Cerqueira, Roy H. Campbell, and Klara Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, 1:74–83, 2002.

[83] V. Sacramento, M. Endler, H.K. Rubinsztejn, L.S. Lima, K. Goncalves, F.N. Nascimento, and G.A. Bueno. Moca: A middleware for developing collaborative applications for mobile users. *Distributed Systems Online, IEEE*, 5(10):2 – 2, 2004.

[84] Sriram Sankaranarayanan, Henny Sipma, Ting Zhang, David Dill, and Zohar Manna. Event correlation: Language and semantics. In *In Embedded Software, Third International Conference, EMSOFT 2003, volume 2855 of Lecture Notes in Computer Science*, pages 323–33. Springer, 2003.

[85] Bill N. Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *IN PROCEEDINGS OF THE WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS*, pages 85–90. IEEE Computer Society, 1994.

[86] Aline Senart, Raymond Cunningham, Mélanie Bouroche, Neil O'Connor, Vinny Reynolds, and Vinny Cahill. Mocoa: Customisable middleware for context-aware mobile applications. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4276 of *Lecture Notes in Computer Science*, pages 1722–1738. Springer Berlin / Heidelberg, 2006.

[87] F. Seyler, C. Taconet, and G. Bernard. Context aware orchestration meta-model. In *Autonomic and Autonomous Systems, 2007. ICAS07. Third International Conference on*, pages 17 –17, 6 2007.

[88] A. Shchzad, Hung Quoc Ngo, S.Y. Lee, and Young-Koo Lee. A comprehensive middleware architecture for context-aware ubiquitous computing systems. In *Computer and Information Science, 2005. Fourth Annual ACIS International Conference on*, pages 251 – 256, 2005.

[89] Abhishek Singh and Michael Conway. Survey of context aware frameworks - analysis and criticism, 2006.

[90] Kostas Stefanidis, Evaggelia Pitoura, and Panos Vassiliadis. Adding context to preferences. In *In Proc. ICDE*, pages 846–855, 2007.

[91] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.

[92] H. van Kranenburg, M.S. Bargh, S. Iacob, and A. Peddemors. A context management framework for supporting context-aware distributed applications. *Communications Magazine, IEEE*, 44(8):67 –74, 2006.

[93] William Van Woensel, Sven Casteleyn, and Olga De Troyer. A framework for decentralized, context-aware mobile applications using semantic web technology. In Robert Meersman, Pilar Herrero, and Tharam Dillon, editors, *On the Move to Meaningful Internet Systems: OTM 2009 Workshops*, volume 5872 of *Lecture Notes in Computer Science*, pages 88–97. Springer Berlin / Heidelberg, 2009.

[94] Lode Vanacken, Joan De Boeck, Chris Raymaekers, and Karin Coninx. An event-condition-action approach for contextual interaction in virtual environments. In Peter Forbrig and Fabio Paternò, editors, *Engineering Interactive Systems*, volume 5247 of *Lecture Notes in Computer Science*, pages 126–133. Springer Berlin / Heidelberg, 2008.

[95] J. Viterbo, V. Sacramento, R. Rocha, G. Baptista, M. Malcher, and M. Endler. A middleware architecture for context-aware and location-based mobile applications. In *Software Engineering Workshop, 2008. SEW '08. 32nd Annual IEEE*, pages 52 –61, 2008.

[96] Jinling Wang, Beihong Jin, and Jing Li. An ontology-based publish/subscribe system. In *In Middleware*, pages 232–253, 2004.

[97] M.H. Williams, I. Roussaki, M. Strimpakou, Y. Yang, L. MacKinnon, R. Dewar, N. Milyaev, C. Pils, and M. Anagnostou. Context-awareness and personalisation in the daidalos pervasive environment. In *Pervasive Services, 2005. ICPS '05. Proceedings. International Conference on*, pages 98 – 107, 7 2005.

[98] Di Zheng, Yan Jia, Peng Zhou, and Wei-Hong Han. Context-aware middleware support for component based applications in pervasive computing. In *Proceedings of the 7th international conference on Advanced parallel processing technologies*, APPT'07, pages 161–171, Berlin, Heidelberg, 2007. Springer-Verlag.