An Enhanced Context Aware Applications Architecture Using Collaborative Model

S.Srinivas Reddy, S.Siva Rama Krishna, B.Gantaiah

1,2,3Lakireddy Bali Reddy College of Engineering, Mylavaram, Krishna, AP, India

Abstract
Asynchronous Event Handling and Synchronous Exception Handling mechanisms are devised as part of a forward recovery model to program robust context-aware applications which enable designing recovery actions to handle different kinds of failures such as service discovery failures, service binding failures, exceptions raised by a service, and context invalidations. Previously these context aware applications run on trusted servers which are assumed to be resilient to crash failures also. In this paper, we present a collaborative model for context-aware applications along with server deployment scenario. The proposed approach is developed for building collaborative activities in which users and environment services cooperate towards some shared objectives and tasks. We propose to use new server deployment scenarios such as replications and back up procedures to build more robust application architecture. The specification model allows expression of policies related to context-based discovery and secure access of resources, and role-based interactions among users and environmental services.

Keywords
Context aware application, Collaborative Model, Synchronous & Asynchronous Event Handling

I. Introduction
Context awareness is a central aspect of pervasive computing applications, characterizing their ability to adapt and perform tasks based on the ambient context conditions. Context notion is defined 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 applications themselves”. Context may also include system capabilities; services offered and sought the activities and tasks in which people and computing entities are engaged, and their situational roles, beliefs, and intentions. The computing environment used to deploy context aware applications is called an active space. Such a space may span multiple physical areas and autonomous administrative domains. An active space provides the infrastructure services that are required for deploying and running context-aware applications. A typical user is generally involved in many activities such as office workflow tasks, distributed meetings, collaborative tasks, and personal activities such as shopping or entertainment. Many of these activities may involve multiple users collaborating on some shared tasks. Applications built for such environments are characterized by dynamic integration of large number of components based on the user context and ambient conditions. Due to this kind of applications different failure conditions and situations are raised:

1. Context-based dynamic resource discovery and binding may fail because of unavailability of required type of resources in the environment.
2. Failures could arise while accessing the resources/services because of incompatible resource access protocols or insufficient security privileges. Also a resource may encounter internal failures.
3. Some users working towards a common goal in a collaborative application may fail to perform certain obligated tasks.

In this paper, the model presented here provides a composition framework for building context-based collaborative activities by dynamically integrating users, applications, and environmental resources along with server deployment scenario also. An important aspect of our approach is to build context-based collaborative applications in pervasive computing environments from their high level specifications. This model provides a variety of primitives for context-based binding of resources in an activity. This model primarily specifies various context-based requirements for resource access, security and robustness. They are: specification of resources and services required in an activity, dependencies of these requirements on context information, directives for context-based resource discovery, and specification of dynamic security policies. In proposed system, we use techniques such as replicating the trusted servers and running the various managers in a primary backup mode to build a more robust application architecture for handling the crashed trusted servers or managers.

II. Related Work
During the past few years, the research focus in context aware computing has shifted to scalable and reconfigurable architectures. It can be used to automatically recognize user requirements from the application situation and accordingly adapt functionality and interaction patterns of the system. One difficulty faced by context-aware application designers is the lack of generic infrastructure for developing context-aware applications. In addition, existing applications have focused mainly on location information. Several researchers have proposed and prototyped general architectures to support context-aware application; Schilit’s infrastructure for ubiquitous computing is probably the earliest attempt in this direction. It is agent-based and supports the gathering of context about devices and users. Applications build on the architecture can request for these context information from the different agents such as device agents, user agents and active maps to provide proactive services to users.

Dey’s Context Toolkit aims at facilitating the development of context-aware applications by offering a small set of generic ‘base’ classes from which an application developer can derive application specific classes. These classes are organized around high level context-aware application functions, namely collecting sensor data, combining data from multiple sensors and translating sensor data into alternate formats. Context Toolkit delivers a standardized way of implementing the syntactic part of context-aware systems. However, the reasoning about contextual information must be implemented for each domain and application. This makes it flexible only in the design and implementation phase, but and not during run-time.

The Service-oriented Context-Aware Middleware (SOCAM) project is an architecture for the building and rapid prototyping of context-aware mobile services. It supports context acquisition, interpretation, discovery and dissemination and its main feature is its support for context reasoning through which high-level implicit
contexts can be derived from low-level explicit contexts. Contexts are represented as predicates written in OWL (Web Ontology Language) which makes it flexible. Interpreter acquires context data from distributed context providers and offers it in mostly processed form to clients.

Gaia aims at making physical spaces intelligent by providing services to manage the spaces and their associated states. Context providers are data sources of context information. Other agents can query them or listen on their event channel where they keep sending context events. They can also advertise the context they provide with the context provider lookup service that allows other agents to discover them. Context history service stores past contexts and may be queried.

III. Context-Based Collaborative Activities

In our collaboration model, an activity defines how a group of users cooperates toward some common objectives by performing tasks involving shared resources and infrastructure services. Shared resources and services are represented as objects in the collaboration space defined by an activity. In an activity, users are represented by their roles, and roles are assigned privileges to perform certain tasks. An activity defines a scope for user roles and shared objects, and specifies policies for their interactions. A role defines a set of operations. A role operation may involve execution of some actions on objects defined within the activity. A role operation can only be invoked by a member in the role. A role operation can have a precondition that must be satisfied when the operation is executed. Context-based access control policies and coordination constraints are specified as operation preconditions. Such operations are termed reaction.

A. Object Namespace Within an Activity

The object declaration in an activity defines a namespace for the objects which represent resources and services that are required in the activity. There are two kinds of namespaces defined within an activity. The first kind defines names for objects that are shared by all participants in the activity. This is called activity namespace. These activity-wide names are visible in all the role operations inside an activity. The other kind of namespace defines names for objects that are private to each member in a given role. This is called rolemember namespace.

B. Context-Based Conditions

Preconditions are specified for role operations, admission, and activation to enforce required coordination and security constraints. Such conditions are expressed in terms of predicates based on events occurring within the activity, role memberships of participants, and query methods of the environmental resources representing external context information. There are two categories of events, internal and external, that are utilized in our specification model. Internal events are generated by execution of operations and reactions within an activity. External events are generated by the objects in the environments. Such events are used in a specification to capture policies that depend on the external context.

C. Role

A role is an abstraction as well as a mechanism for specifying and enforcing users’ privileges in an activity. A role defines a set of operations that can be executed by its members. The admission of users as members in a role and invocation of other operations by the members is required to be controlled in order to enforce required security and coordination policies. These policies are specified in the form of preconditions associated with role operations and admission. Associated with each role, there are two types of constraints that are imposed on all role members:

1. Role admission constraints must be satisfied when a user is to be admitted to a role. These constraints can be based on the history of the operations previously executed, the context of the user, such as the user’s membership in other roles, or state of an object.

2. Role activation constraints must be satisfied for performing any role operations and reactions. Similar to role admission constraints, activation constraints can be based on event history as well as user and object context.

D. Reaction Specification

Construction of smart environments requires the implementation of services that automatically execute some actions when certain events occur. Such system services are modeled as activities having no roles, and the automatic operation invocation is modeled as a reaction.

In contrast to an operation, a reaction is not invoked by a user but is automatically executed when certain events occur. Similar to an operation, a reaction is executed only when its precondition is true. The reaction is invoked when any of the events specified in the When clause occurs. If the precondition is true, then the corresponding action is executed. Reactions can be specified in the context of an activity or a role.

E. Binding Specification

Binding of a name in an activity’s namespace to a resource is specified at different levels in our model. The resource binding policies can be specified in four different ways:

1. Binding to a new object: The binding primitive with new specifies that the resource should be created and should be bound to the name.

2. Binding to an existing resource through URL: This form of binding primitive with direct specifies that the resource identified by the given URL should be used in binding. This binding primitive is used when the resource already exists and its location is known.

3. Binding to an existing resource through URL: This form of binding primitive specifies that the name should be bound to the resource or service that is currently being referenced by another name.

4. Binding through discovery: This form of the binding primitive is useful when a resource with a particular set of attributes is needed to be discovered in the environment. In the binding directive, some of the attributes of a required resource may need to be determined at the time of discovery, based on certain context information.

IV. Execution Model With Components

To support the context-based ubiquitous collaboration model presented here, to support dynamic resource discovery and binding, and context-based security and coordination policies. To manage a collaboration activity, from its specification we derive policies related to different management aspects. These aspects are related to management of roles and objects. The policies for a role pertain to role membership management, enforcement of context-based security and coordination requirements, and subscription and notification of context-related events. To construct the runtime environment to manage an activity and enforce the policies specified for the activity, the middleware provides two types of generic managers: role manager and objectmanager.
A. Role Manager

When an activity is instantiated, a role manager is created for each of the roles and an object manager for each of the objects defined within the activity as shown above. A function of a role manager is to manage the user memberships in the role and enforce role admission policies. Moreover, it maintains an object-space for each member in the role. For each object in an object-space, the role manager maintains a references to the corresponding object manager.

Role managers provide two remote interfaces: one is for role specific operations, and the other is for event subscription and notification. Event subscription policy specifies the subscribers for the operation-related events generated by this role. Event notification policy specifies the events that must be subscribed to by other managers to enforce preconditions for operations and role admission.

B. Object Manager

An object manager maintains a reference to a resource or service and enforces context-dependent object binding and access policies. It provides a remote interface for role managers to invoke methods on the object as part of role operations and reactions. Similar to a role manager, an object manager also has an event subscription and notification interface. Events notified to an object manager are used to trigger context-based resource binding and enforce dynamic access control policies. An object manager may also generate certain events representing specific changes in the state of the object.

C. Interaction Model

In the interaction model supported in this environment, users can only interact with role managers. When a user invokes an operation to access an object, the role manager interacts with the corresponding object manager to initiate a session between the user and the object. Each user participates in the environment through a User Coordination Interface (UCI) executing on the user’s computing device. The UCI maintains the RMI stubs for the various roles in which the user can be a member. Also, during any interactions with a role manager, it provides any device-specific certificates and context-information to the role manager.

V. Robustness Issues

In this paper, major robustness issues are: Service binding failures, Context invalidations, Trusted Servers that are prone to crashes. In this section, we mainly discuss about handling the crashes in servers by maintaining Replica Management Services.

A. Service Binding Failure

Service binding fails when the service to which the object is bound becomes inaccessible. The recovery model needs to support detection of service binding failures. Recovery actions to handle certain kinds of failures can be programmed within an object manager. If the cause of binding failure is access revocation by the service, the object manager can try to discover and bind to another service of that type.

B. Context Invalidations

To monitor the context conditions associated with an operation, the design model provides the ContextGuard mechanism, which can be associated with a role operation. It consists of two parts. First is the specification of the required context condition. Second is the specification of all the context events that may potentially lead to the violation of the context guard condition. The role manager evaluates the guard condition whenever the trigger events occur. If this condition fails to hold, the role manager terminates the session, thus revoking the user’s access.

C. Crashes in Trusted Servers

For building context-aware applications, Trusted servers are immune to crash failures of various managers, one can use techniques such as replicating the trusted servers and running the various managers. To solve this issue, replication management service is provided. In our approach, replication is used to balance the load of data requests within the system both on the network and host levels, and to improve reliability.

A recovery model by [5] is implemented for context aware applications. It consists of mechanisms for asynchronous event handling and synchronous exception handling. Events represent occurrence of a particular state (normal or failure) related to a context-aware application. Exceptions represent a subclass of events that arise synchronously during the execution of an action within a role operation or a reaction. By using this model, major robustness issues are resolved.
Based on access cost and replication gains, the replica management system decides when to create a replica and where to place it. These decisions are made based on a cost model that evaluates the maintenance cost and access performance gains of creating each replica. The estimates of costs and gains are based on many factors, such as run-time accumulated read/write statistics, the chosen consistency algorithm, run-time measured response time, bandwidth, and replica size. These parameters are changing during the program execution, so they need to be measured at runtime and fed to an optimization procedure that minimizes data access costs by dynamically changing the replicas number and placement.

Each entity maintains a replica set, which is initially empty. Replicas are created at entity that receive high number of requests. Each entity in the replica set maintains an index list of all entities that it contains, plus a list of the locations of these entities that it is aware of. In the hierarchical topology, each entity maintains a list of its parents and children locations, while in the flat topology, each replica keeps a list of the locations of its neighbors. The index list entries contain context information such as size, name, type, and other attributes. Such index lists are called local replica indices. The replica set root(s) maintain(s) a list of all replica locations. This index list is called a global replica index. Used within a replica cataloging service, these indices identify the closest replica to a given site. A request is then transparently serviced by the nearest replica.

A replica is removed from a site when the user chooses to do so or when the system deletes it. The latter happens if the data is not used anymore locally, or if no requests have been made to access it remotely from other sites after a certain amount of time or when space is needed for more frequently used data. The runtime system is then responsible for updating the replica connection graph and making the necessary changes to replica indices. The connection graph should then be reorganized while maintaining its topology.

The replica management service includes a runtime component that dynamically evaluates the application and user’s needs and accordingly adapts the replicas distribution. The runtime system monitors the user’s behavior, collecting accumulated read/write statistics and measuring response time, bandwidth, and replica size.

VI. Conclusion

In this paper, the model presented here provides a composition framework for building context-based collaborative activities by dynamically integrating users, applications, and environmental resources along with server deployment scenario also. An important aspect of our approach is to build context-based collaborative applications in pervasive computing environments from their high level specifications. This model provides a variety of primitives for context-based binding of resources in an activity. This model primarily specifies various context-based requirements for resource access, security and robustness. They are: specification of resources and services required in an activity, dependencies of these requirements on context information, directives for context-based resource discovery, and specification of dynamic security policies. In proposed system, we use techniques such as replicating the trusted servers and running the various managers in a primary backup mode to build a more robust application architecture for handling the crashed trusted servers or managers.

References