Security Architecture in *Gaia* *

Prashant Viswanathan, Binny Gill and Roy Campbell

University of Illinois, Urbana-Champaign IL, USA

Abstract. Ubiquitous computing promotes physical spaces with hundreds of computational devices extending the user's view of the computational environment beyond the physical limitations of a traditional distributed system. Gaia OS is a middleware operating system that makes the realization of such spaces a possibility. In any such environment Security and Privacy are primary concerns and mechanisms to solve them have to be built into the very core of the system so that their deployment is acceptable and feasible. In this paper we describe a security architecture for Gaia OS and Active Spaces. Our architecture provides dynamism and flexibility to manage the security concerns in such a computing systems.

1 Introduction

Security has always been a primary concern in any computing model. This is particularly true of ubiquitous computing environments where resources and information are shared on a very large scale. Sharing of resources by users in a distributed system makes sense only when it is done in a controlled manner. Security issues are even more important in a ubiquitous computing environment as contextual information such as a user's location constitute an integral part of the computing model.

The proliferation of ubiquitous computing makes the construction of Active Spaces a reality. Active Spaces will enable users to leverage the computing power of a large number of computing devices to enhance user productivity. Active Spaces are a distributed system in which a user exhibits "mobility". With the number of Ubiquitous Computing applications steadily increasing we need an application framework to facilitate their construction and deployment. For any successful deployment of an Active Space in the real world we believe that the underlying framework should have the security mechanisms built into its core. In Gaia OS we have built in a set of powerful security mechanisms into the core of the system.

A computing environment for an Active Space needs several security measures to ensure privacy of users and for protecting users and resources from the malicious intent of other users and resources. There are various security challenges in such a system that include Authentication, Access Control, Privacy of Location and Key Distribution. In an Active Space users can authenticate themselves in multiple ways. Access Control is required for files and services, and the

* This research is supported by the National Science Foundation grant NSF 98-70736

system has to provide mechanisms by which users can secure their privacy. Scalability is another major issue in such a system. Users and programs also need mechanisms for controlled amplification of rights and delegation of authority.

In Gaia OS we use Credentials as a basis for Authentication and Access Control to support scalability and multiple administrative domains. We use authentication mechanisms, involving public/private keys and symmetric keys. Roles and Attributes in combination with a rich set of Policies allow dynamic and flexible Access Control. Our Access Control mechanisms involve the use of both *Credentials* (similar to Capabilities) and *Policies* (extension of Access Control Lists). A "Set-User Credential" service allows Amplification of Rights and the Credentials allow users to delegate authority in a controlled manner to other programs.

The next section describes Active Spaces and Gaia, a framework that enables such an Active Space. Section 3 discusses the problems in greater detail. Section 4 discusses Roles, Policies and Credentials around which the Gaia Security Architecture is built. In Sections 5,6 we discuss Authentication and Access Control. We then discuss Secure Loading of Components, Bootstrapping and "Set-User Credential" service in Sections 7,8,9. On going work and future research is discussed in Sections 10 and we conclude in Section 11.

2 Gaia and Active Spaces

Gaia supports an Active Space by facilitating the interaction with physical space. It does this by making physical spaces programmable. Active Space is a generic computing system and like any other consists of hardware, software, middleware operating system, application programs and users. The hardware consists of all computing devices in an active space. The operating system in an Active Space provides a uniform view of both logical and physical devices. All the entities in an active space, which comprise the set of users and the application programs can use the power of this generic computing model to enable Active Spaces. Active Spaces and Gaia are discussed further in much greater detail in [2, 3].

Gaia OS is a system software infrastructure that enables Active Spaces. It is a component-based distributed meta-operating system that runs on existing operating system and is implemented on top of existing middleware platforms. Fig 1 show the architecture of Gaia.

The Unified Object Bus defines the distributed object model for the OS and allows dynamic creation and reconfiguration of components. Gaia Services are implemented as CORBA objects[9], run in the context of an active space and can be accessed by any entity with the functionality to interact with the unified object bus. These services include Security, QoS, Resource Manager, Environment Service and a Data Object Service. The Gaia Application model is based on on the Model-View-Controller and provides facilities to create and register different components of the application (model, view and controller) and manipulate such components.

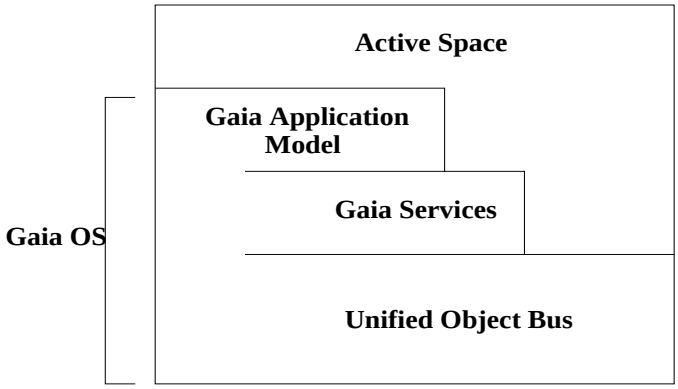

Fig. 1. Architecture of Gaia

3 Security Problems and Issues

In this section we discuss the security issues which manifest themselves in an Active Space and consequently in any framework which is a building block for such Active Spaces.

Security concerns in Ubiquitous Computing can be classified among the following categories, which are described in greater detail in the later sections.

- Authentication.
- Access Control.
- Secure Boot-Strapping.
- Privacy.
- Confidentiality.

Apart from Authentication and Access Control, Ubiquitous Computing environments have to deal with the confidentiality of information and privacy of users. Privacy concerns are a direct consequence of the system being location and context aware. As these computing environments are built on a huge scale Key Distribution and Scalability are major concerns. The scalability problem also introduces new challenges in the administration of such systems. Thus any framework which supports Ubiquitous Computing has to solve the above problems.

As Gaia is a component based framework we also have to provide mechanisms for securely loading such components. The system also has to be bootstrapped in a manner such that the integrity of the system is not compromised. Location information has to be protected and disbursed in a manner such that a user's privacy is not compromised. Gaia being a distributed system the security mechanisms should also be able to solve the following problems.

- **Delegation of authority to trusted program** : A user has to provide a program/service a mechanism by which the program/service can execute on the users behalf.
- **Delegation of authority to untrusted program** : If the user does not trust a program he might not want to give it complete authority to execute on its behalf. For example, Service S might not be trusted and instead of giving it complete authority a user might want to give it only the authority to user resources in `/Active_Space/Room_3234` on his behalf.
  The Confused Deputy problem also manifests itself. This problem arises when a program/service runs with authority stemming from two sources. It is sometimes necessary in such cases to limit authority stemming from a particular source only to some resources.
- **Simple authentication** In some cases a user might just want to authenticate himself to a service. The service doesn't need to execute using the user's authority. In such cases the user should be able to issue a Credential that the service cannot misuse to execute on the user's behalf.

4 Credentials, Policies and Roles

To solve the various security problems which manifest themselves in such a system we use *Roles*, *Policies* and *Credentials*. *Policies* are an extension of *Access Control Lists* that allows us to specify a richer and more powerful set of constraints and conditions. Jini uses a similar mechanism with the help of *Access Control Lists*[11] but is not as powerful as what we provide in Gaia. *Credentials* are similar to *Capabilities* and these are used in combination with Policies to do Access Control in Gaia OS.[7]
We provide both Role-Based and Discretionary Access Control in Gaia OS. The Authentication Service in Gaia OS provides different kinds of Credentials to achieve the various objectives.

4.1 Credentials

A Credential in Gaia is a certificate which is issued to users/programs. Credentials are similar to capabilities as they control the degree to which authority can be delegated and In Gaia we have three different kinds of Credentials:

- *Generic Credential:* These Credentials solve the problem of delegating authority to a trusted program mentioned in Sec 3. A typical Credential looks as shown in Fig 2.
  These Credentials give the holder of the Credential all privileges associated with the Credential's owner. As a user might be a part of many roles and have many attributes, a list of roles and attributes for which the Credential is valid is also sent. A key that is shared by the Access Control Service and the Authentication Service signs the Credential. These services are described in Sec 5 and Sec 6 The Credential also has a time field that stores the time when the Credential was issued. This field is used to decide upon the expiration of the Credentials.

```
{
USERID = (bsgill)
ROLES = (student) (ACM member)
ATTRIBUTES = (Age = 23) (Field of Study = Computer Science)
}signed by AC
```

Fig. 2. A Credential

- *Restricted Credential:* When the *DELEGATION RESTRICTED TO* field is present, the Credential is termed a restricted Credential.
  The restricted Credential solves the problem of delegating authority to an untrusted program mentioned in Sec 3. In this case the holder of the Credential only has privileges to access resources enumerated in the *DELEGATION RESTRICTED TO* field.
- *Non Delegatable Credential:* These are Credentials that can be used by the client to prove its identity to the service without giving any of its own rights to the service. This is achieved by a Non-Delegatable Credential issued by the *Authentication Service*. This Credential has the ID of the *TARGET SERVICE* that the client wants to authenticate to. Thus, the service cannot use this Credential to contact other services pretending to be the client.

4.2 Policies

Policies are associated with all resources in the system. Policies in Gaia are expressed as boolean expressions in disjunctive normal form. These policies are extremely powerful, as they cannot only be expressed in terms of roles but also attributes.

A sample policy for a Directory Object is shown in Fig 3. It is also known that `admin` is higher in the role hierarchy than `student` and that `bsgill` is a `student`. Thus, any user with the `student` role or higher can enter the directory. Only the admin can create a file in the directory, and further that file should be named "mail.conf". The `args` array contains the arguments to the method in question (in this case, `create`). Only the admin can delete files from the directory. Access to any other operation is controlled according to the `default` method policy. One such operation is `list`. Thus, all students can list the directory during office hours, and `bsgill` can list files in the directory anytime until the sixth of January 2001 and of-course the admin can list the directory all the time.

Policies are also associated with other resources. For example consider a service, which allows users to display video on a High Definition TV. Then the CORBA object implementing this could be associated with a policy as shown in Fig 4. This policy says that the showVideoOnHDTV method can only by accessed by Junior_Admin roles whose age is greater than 50.

Gaia OS allows composition of Policies. We associate a default system policy with every type of component. Users can override this with their own default

```
enter
    student
;
create
    admin & args[1] = "mail.conf"
;
delete
    admin
;
default
    student & time > 8:00 & time < 17:00
    bsgill & date < 01/06/2001
    admin
;
```

Fig. 3. A Policy file for a Directory

```
showVideoOnHDTV
Junior_Admin & age > 50
;
```

Fig. 4. Example of a simple policy file for an object

policy. Further each running instance of a component can be given a distinct policy and this can be changed dynamically at run-time. Gaia OS checks if the per-instance policies are consistent with the user-level or system level policies. In case of inconsistency, the per-instance policy prevails. A Browser allows the user to look at the various instances of objects and components that have been created and allows the policies to be modified with the help of a *Policy Editor*. The representation of these policies is shown in Fig 5. When a component/object of type HDTV controller is started a default policy is applied unless the user starting the component has specified his own default policy (as is in this case). Further he can specify a different policy for each instance of the component he starts as shown in the figure.

4.3 Roles

Roles are represented in the Gaia File System. The hierarchy of roles is represented as a directory hierarchy in the file system. A typical hierarchy is shown in Fig 6.

Roles are defined by creating appropriate directories in the resource hierarchy. Roles are administered by enforcing access policies in the role hierarchy representation in the file system. Thus, if an administrator wants to add a role or a user to a role, he should have the permissions to create the corresponding file or

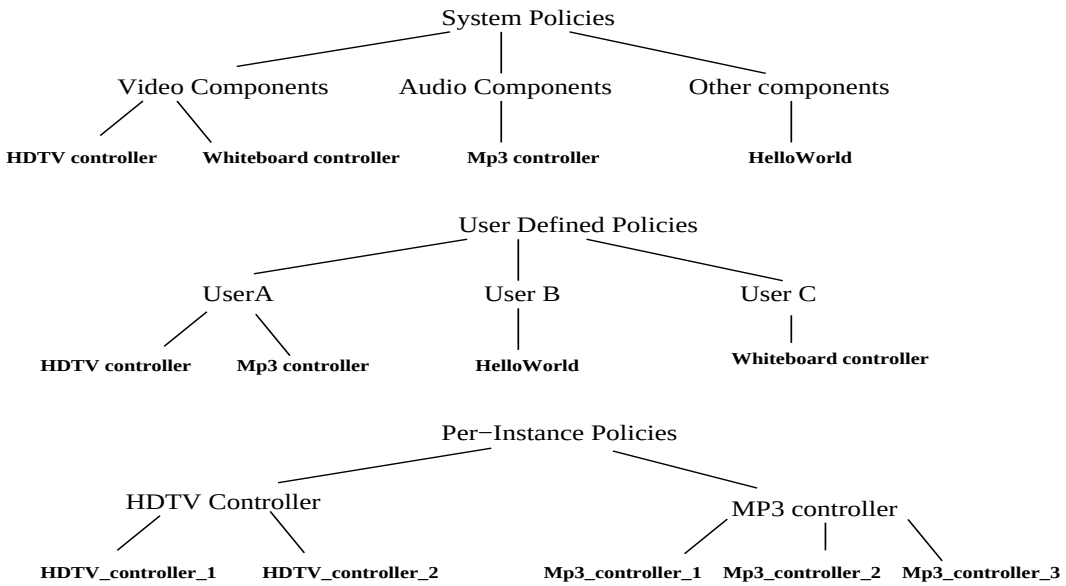

Fig. 5. Representation of Policies

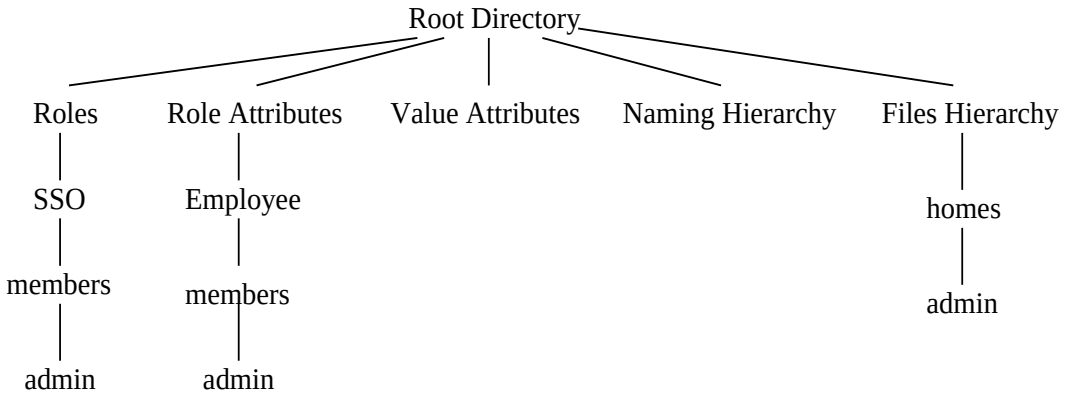

Fig. 6. Hierarchy in the file system

directory in the role hierarchy. The access policies associated with the files and directories in the role hierarchy determine the policies for role administration in our access control model. The Access Control service examines the policies associated with the files and directories and makes decisions which determine whether roles/users can be added or deleted. The representation of roles in such a manner greatly simplifies the administration of the system. Similarly access control is provided for services in Gaia.

4.4 Credentials and Roles

Credentials and Roles are closely tied together. If a user authenticates himself and chooses not to activate any role he gets only the Credential as a user ($\{\text{user}\}_{signedAS}$). However, if he authenticates himself and decides to activate Roles A and B, then his Credential also indicates that he has privileges belonging to Roles A and B ($\{\text{user,Role A, RoleB}\}_{signedAS}$).

When a program gets a Credential it executes with the full rights of the user, roles that the Credential identifies unless it is a Restricted Credential. Also at any time it should be possible for a user to change his Credential so that he can deactivate existing roles or activate additional roles.

When a user starts a program and wants to give the program his Credential he should be able to specify that the Credential be given only for specific roles.

5 Authentication

Authentication consists of validating a user's claim regarding his identity. The Authentication Service (AS) provides this functionality. The authentication service provides different kinds of Credentials (see Sec 4.1)to solve the problems discussed in Sec 3. These Credentials enable the Access Control Service to provide Discretionary, Mandatory and Role Based Access Control.

The authentication service issues a Credential when a user proves his identity. A user could prove his identity in many different ways. He could use a traditional login/password mechanism. He might also use a smart card that proves his identity. Other devices such as I-buttons[16] can be used to store the private key of the user and a challenge/response method could be used to prove his identity. The use of private and public keys ensures that the system scales as required. Further a user can add and delete roles in his Credential using the services of the Authentication Service. He can also change the type of his Credential using the API provided by the Authentication Service.

6 Access Control

Access Control in GAIA is discussed in details in [1]. Here we provide a very short description of the same. GAIA provides an extensive mechanism to enforce different kinds of access control. It provides:

1. Role-Based Access Control[4][5][6].
2. Discretionary Access Control.
3. Mandatory Access Control.

Role-Based Access Control is applied to all the System Files and Services. ARBAC[4] also ensures that administration of the system is easy and scalable. **Discretionary Access Control** is provided to users so that they can secure their private files. **Mandatary Access Control** provides military grade security in the system and is essential if the system is deployed in a Military environment. We need a combination of all three, as a single access control mechanism is not powerful and expressive enough to satisfy all access control requirements of such a system.

The Access Control Service on obtaining the name of the resource can fetch the corresponding policy file and decide whether or not access is allowed. The two methods available in the interface for the Access Control Service are **canAccessGivenPolicy** and **canAccessGivenPath** both of which take a Credential as input. They also take Arguments as a parameter as some policy might be specified in terms of the arguments.

For example, when a user wants to access a file, he uses the File Service. The File Service contacts the Access Control Service with the path for the file and the Credentials of the user. The Access Control Service then tells the File Service whether the user is authorized or not to access the file. The Credential is generated by the Authentication Service and is encrypted. The Authentication Service and Access Control Service share the same key.

The Access Control Service makes its decisions based on the policy file for the resource/object/service. These policy files can be edited by the owner of the resource to give permission to others users. There are also meta-data files which determine who has access to the policy files.

6.1 Security Interceptors

We use interceptors[8] to add additional security contexts to the requests that are propagated from the client to the server. These interceptors insert the Credential of the caller and this mechanism is transparent to the application. Interceptors also give access to the method name and the parameters that are being passed to access the method. Fig 7 shows this mechanism in the case of a File Server.

The interceptor is responsible for contacting the Access Control service. In case access is denied it raises an exception. Otherwise it lets the normal path of

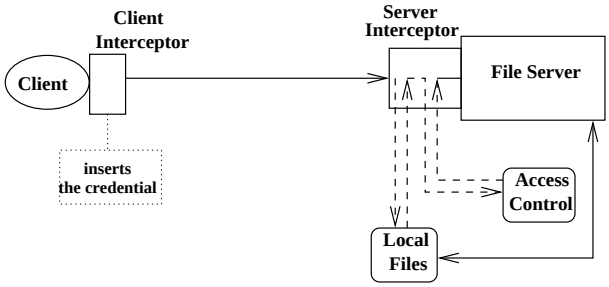

Fig. 7. File Server with Interceptor.

invocation continue as if nothing had happened. In Gaia OS we have interceptors which work with TAO[14] and ORBACUS[15]. These are request-level Portable Interceptors defined as part of the CORBA specification[10].

7 Secure Dynamic Loading of Components

As Gaia OS is a component based framework; we have to ensure that there are appropriate mechanisms for securely loading such components. There are several aspects to this problem, which are discussed in the following subsections.

7.1 Component Repository as part of the file system

The component repository can be stored on the file system. This simplifies its security management and further ensures that the generic security mechanism for the system can be used for this too.

7.2 Access Control for Component Repository

There are two sides to access control for the component repository. First of all, a policy dictates who can place components into the component repository. For example, if components are stored in a directory called `/ComponentRepository` the policy for the directory can specify who can add their components to the Component Repository. A user can be given permissions to add components to the Component Repository by giving him write permissions for the directory.

Secondly, when a user adds a component to the Component Repository, being the owner of the component(file) he can create a policy for it. Using this policy he can limit access to the component to certain users or roles, or users satisfying certain attributes.

7.3 Access Control for Components

It is sometimes necessary to allow different users to instantiate components inside a single process space (component container). [1] This is done by associating security policies with each component. A Secure Component Manager manages the secure dynamic loading of such components and with the help of interceptors controls access to such components. Each component is started with a policy file. The creator of the component specifies access to certain users or roles using this policy file. This provides method level access control to objects as discussed in Sec 4.2.

Credentials in a Component Container This section explains how Credentials are associated with components and provides some insight into the implementation.

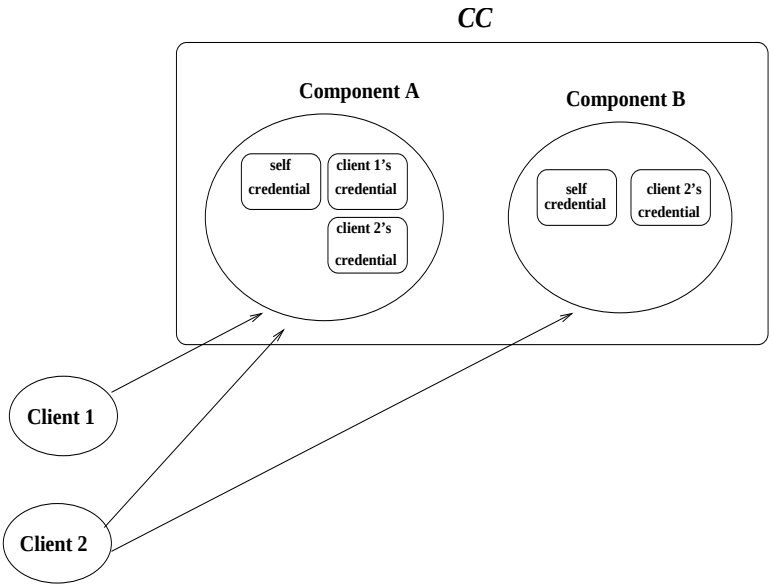

Fig. 8. Credentials in the ComponentContainer

With every component in the ComponentContainer we associate two Credentials: **"Self-Credential"** and **"Client-Credential"** (See Fig 8). Consider a component container CC. If user A creates and loads a component inside the component container user A's Credential is the component's **"Self-Credential"**. Now a client C may make request invocations on this component.

[1] A Component Container is a place holder for components in which components are instantiated and follow their life cycle.

In such a scenario C's Credentials are assigned to the **"Client-Credential"** of the component. Thus the component possesses both the creator's and the client's Credentials and can use them appropriately. For example, when the component wants to perform certain operations on behalf of the client it shall use **"Client-Credential"** and when it wants to use its own Credential it shall use **"Self-Credential"**.

On creation the **"Self-Credential"** of the component is set to that of its creator. When a component behaves as a client the interceptor on the client side inserts the Credential into the request. The interceptor on the server side transparently extracts the Credential and **"Client-Credential"** of the server side component is assigned this value. In case of more than one client making requests on the component concurrently a mapping of Client threads and Credentials is maintained. By default the client side interceptor inserts the **"Self-Credential"** into every request. If the client wishes to use any other Credential other than the **"Self-Credential"** it sets the **"Self-Credential"** appropriately. The Secure Component Manager's API has methods that can be called to obtain either the **"Self-Credential"** or the **"Client-Credential"** corresponding to a particular client. Its API also has methods by which the value of **"Self-Credential"** can be set.

8 Secure Bootstrapping

The bootstrapping is performed across two levels. The first boot level consists of bootstrapping the Security Services, Naming Service and other system services like the Trader Service (top level). These domain services constitute the Trusted Computing Base for GAIA.

At the second level, an active space is bootstrapped. To ensure the integrity of the boot process, the Naming Service is "secured" at all times. By this we mean that Active Spaces can register in the appropriate context only upon proving their identity. For this purpose, we use Public Key Crypto System. The Authentication Service has the public key of all the Active Spaces. When the Acitve Space boots and wants to register itself in the naming service it proves its identity by encrypting a challenge string with its private key.

Consider for example an "Active Room" being switched on. Any Active Space in Gaia OS has its own event service and other such services. These have to be registered in the Naming Service. The Active Space, thus has to prove its identity to the Naming Service to register in the appropriate context and at the same time verify the integrity and authenticity of the Naming Service (which was booted in the first level). All this forms a part of the secure bootstrapping process.

9 "SetUser Credential" Service

In Gaia we require a mechanism by which a process running with lower privileges can create an activity with higher privileges. Thus, there can be an executable that runs with a specific Credential irrespective of the user who invokes it. This

feature is necessary to facilitate bootstrapping so that while any user can trigger bootstrapping the Active Space runs with its own Credential. For example, the earliest employee to work might be the one to "switch-on" an "Active Supermarket" but the "Active Supermarket" runs with its own Credential rather than that of the employee who started it. Further, in a system that supports persistent objects we should be able to resurrect persistent objects with their original permissions. These problems require a "Set-User-Credential" mechanism, similar to the "Set-User-ID" in Unix, using which we can specify the Credential that the executable will acquire once it is invoked.

The "Set-User-Credential" service is used by owners of resources to specify the Credentials with which the components they own will execute, irrespective of the user who loads them. Further, this service will also be used by users to load components for which the Credential has been set. This service loads the component in a Component Container with the set Credential.

The "Set-User-Credential' 'service looks at system policies that dictate what components can be started with promoted Credentials upon. This interaction is shown in Fig 9.

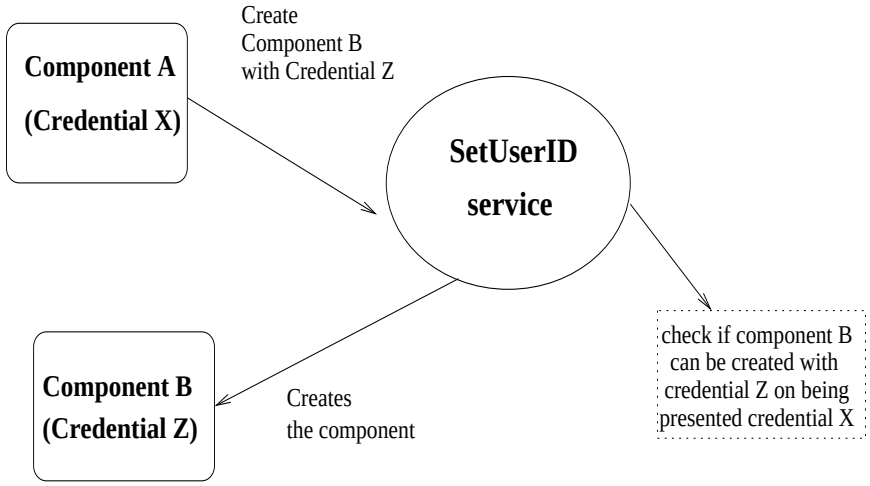

Fig. 9. SetUserID service

10 On Going Work and Future Research

Active Spaces are characterized by applications which are "locaton-aware". We are building a Location Management System which tracks and detects users in Gaia. It is important that a user's privacy requirements are taken care of in

such a system. We plan to integrate two different mechanisms into our Location Management System that will provide users privacy ranging from **absolute** to **limited**. In the **absolute** case the system learns a user location only when the user desires so and in the other case the system is always aware of the user's location but exercises discretion in distributing this sensitive information to others.

10.1 System-driven location tracking

This is an approach where all location information is stored within the system. The system is responsible for securing this data. The perils associated with this approach is that if the system are is compromised, it compromises the privacy of all users. This also means that the user trusts the system with this information and there is no guarantee that another powerful user of the system such as an administrator cannot gain access to such information by changing the policies. A user defines policies which govern the manner in which his location information is distributed to other users/applications. The system can employ devices which include RF-Badges[13] and I-buttons[16] or a traditional login mechanism to detect a user's location.

10.2 User-driven location tracking

A second approach would be take the full support of wearable computing and store all such data on the user's wearable. Beacons would broadcast their location and the sensors on the user's wearable/handheld would know their location. The user can then tell the system about his location when he needs to use "location-aware" services. Even while specifying his location, he can specify it in different granularities depending on how much information he wants to divulge. Such systems are typically more expensive to build though they provide a higher degree of privacy. This is discussed in more details in [12].

11 Conclusion

Security is one of the most important aspects in a ubiquitous computing envrionment and the Gaia architecture provides the mechanisms to build powerful security features into such a system. We show how we can have different kinds of Access Control and Authentication mechanisms with the help of Roles, Credentials and Policies. The simplicity of our system and its dynamism help us extend traditional operating system security into this new computing environment.

References

1. Binny Sher Gill. "Dynamic Policy-Driven Role-Based Access Control for Active Spaces". M.S Thesis, University of Illinois, Urbana-Champaign, 2001.

2. Fabio Kon et al. "A Flexible Interoperable Framework for Active Spaces". OOPSLA'2000 Workshop on Pervasive Computing. Minneapolis, MN, October 16, 2000
3. Manuel Roman and Roy Campbell. "Gaia: Enabling Active Spaces". 9th ACM SIGOPS European Workshop. September 17th-20th, 2000. Kolding, Denmark
4. Ravi Sandhu and Qamar Munawer. "The ARBAC97 Model for Role-based Administration of Roles". Proceedings of the second ACM workshop on Role-based access control, pages 41-50, Fairfax, VA, USA. October 1997. ACM.
5. Ravi Sandhu. "Role Activation Hierarchies". Proceedings of the third ACM workshop on Role-based access control, pages 33-40, Fairfax, VA, USA. October 1998. ACM.
6. Ravi Sandhu and Qamar Munawer. "How to do Discretionary Access Control Using Roles". Proceedings of the third ACM workshop on Role-based access control, pages 47-54, Fairfax, VA, USA. October 1998. ACM.
7. John Barkley et al. "Comparing simple role based access control models and access control lists". Proceedings of the second ACM workshop on Role-based access control, pages 127-132, Fairfax, VA, USA. October 1997. ACM.
8. Priya Narasimhan et al. "Using Interceptors to Enhance CORBA". IEEE Computer 32[7], p. 62-68, 1999.
9. Object Management Group. "CORBA 3.0". CORBA standard specification.
10. Object Management Group. "Portable Interceptors". CORBA standard specification. http://www.omg.org
11. Sun MicroSystems. "Jini Architectural Overview". Technical White Paper.
12. Bradley Rhodes et al. "Wearable Computing Meets Ubiquitous Computing: Reaping the best of both worlds".
13. Want, R., Hopper, A., Falcao, V., Gibbons, J. (1992). "The Active Badge Location System." ACM Transactions on Information Systems 10(1) pp. 91-102.
14. "TAO: The ACE ORB" http://www.cs.wustl.edu/ schmidt/TAO.html
15. ORBACUS. http://www.ooc.com
16. iButtons. http://www.ibutton.com