

Service Discovery, Access and Cooperation in Virtual Communities

Authors

Shudong Chen (TU/e)
Johan J. Lukkien (TU/e)
Igor Radovanović (TU/e)

I-share document: Contribution Deliverable 1.5
I-share status: final

Abstract

Service Oriented Architecture (SOA) is emerging as an enabling technology for sharing distributed heterogeneous resources that may be under the control of different owners in a network such as Internet. However, if the number of the available services over a network is huge, publishing the available services, discovering the desired services while maintaining a transparent access control is not an easy task. It includes research challenges such as how a service provider protects its ownership, privacy and how a service user finds a desired service securely. In this report, we present a SOA with a virtual community approach to address these challenges. The approach includes definition and formation of a virtual community, mechanisms of service discovery, access and cooperation inside a virtual community. We also introduce the Ambient Open Service Architecture (AMOSA) software platform which is the test bed of our approach. Using AMOSA, heterogeneous services which are using different types of technologies, for instance, UPnP and Web Services can be rewrapped into AMOSA services and then be used within a single framework. AMOSA services can be discovered using uniform mechanism and communicate with each other without protocol adaptation at the service side. Moreover, services can be organized into virtual communities to achieve secure service discovery and cooperation.

Table of contents

Abstract.....	2
Table of contents	3
1. Introduction.....	4
2. SOA with Virtual Community.....	5
2.1 Value and limitations of SOA.....	5
2.2 What is a virtual community?.....	7
2.3 Research issues in virtual community.....	9
2.3.1 Overlay concept of a virtual community	9
2.3.2 Virtual community formation and maintenance	13
2.3.3 Secure service discovery, access and cooperation.....	24
3. Prototype: AMOSA Platform	26
3.1 Incentives	26
3.2. Functionality Overview	26
3.2.1 A_ITF specifications.....	26
3.2.2 A_MAP overview	29
3.2.3 Other functional parts	30
3.3 Outlook	30
4. Conclusions.....	32
References.....	32

1. Introduction

Sharing of resources, data and functionality via digital networks becomes increasingly important. Through sharing, integration and cooperation of heterogeneous hardware and software components, distributed applications can be achieved with lower cost and better overall system utilization and performance. Individual users can obtain shared data, resources, e.g. storage space, network bandwidth, processing memory, and functionalities which are not available locally. Though conceptually obvious, a lot of challenges derive from this sharing concept. On the one hand, from users' perspective, how can a user locate the required data, resources and functionalities easily, accurately and efficiently in the scope of a digital network as the Internet? How to insure the integrity of the response to user's request for sharing items and how to develop trust in sharing parties? On the other hand, from providers' perspective, how can a provider protect his ownership and privacy? How to control the access to these shared items? How to secure the communication with users? How to assure the performance, such as usability, response latency and reliability, provided by shared items?

Several approaches [1-6] have been proposed to solve some these issues. Peer-to-Peer (P2P) technology [1-4] focuses on data and resource sharing; the essence of this technology is to solve the network bandwidth restriction problem in downloading. Service Oriented Architecture (SOA) approach [6] addresses functionality sharing; in SOA, software components are wrapped into network-exposed services with explicitly described interfaces to provide functionalities; applications can then be built by interconnecting services, leaving the binding until runtime. Successful and famous standards at the moment are Web service [7] and Universal Plug and Play (UPnP) [8]. Both P2P and SOA are approaches with benefits as well as downsides. For instance, privacy has always been a problem for traditional P2P file-sharing systems because there is no systematic support for implicit end-user control. In SOA, potential performance problems need further discussion. Moreover, security is a common issue that exists in both P2P and SOA research fields.

In our work we are interested in the SOA approach for composing applications from services. We want to investigate this as the middleware that enables Ambient Intelligence. Part of the work comprises the mentioned issues of discovery scoping and of privacy and security. To address these, we extend the notion of SOA with the concept of "virtual community" as follows. Shared functionalities are first wrapped into services. These services are then organized in virtual communities by adding additional virtual community functionalities to them. In this way, a service in a virtual community has the properties of a plain service: its external network interface still uses the SOA service interface; SOA rules such as Simple Object Access Protocol (SOAP) [9] and HTTP protocol are still suitable to the service cooperation. At the same time, enhanced service functionalities are provided to the end users: services are only exposed to authenticated community members, an access control policy is applied to the service access process and all exchanged messages inside the virtual community are encrypted. Using this SOA with virtual community approach (SOAVC), privacy of service providers can be better protected because only members can discover and access these services and communication between a service user and a service provider is protected as well. In addition, reliable service discovery results can be achieved efficiently through virtual community maintenance policies and trustable recommendation based on past interactions.

The remainder of this report is organized as follows. In Section 2 we describe the concept, functionality and the maintenance mechanism of a virtual community as well as the service discovery and access protocols that operate within the scope of a virtual community. The design and implementation of our prototype, the Ambient Open Service Architecture (AMOS) software platform, is presented in Section 3. Finally some conclusions are drawn in Section 4.

2. SOA with Virtual Community

The vision of Ambient Intelligent (AmI) [10] suggests a future in which mostly invisible devices support people in their daily life. This support is unobtrusive, adaptive and is experienced as intelligent. Although this vision has triggered a massive amount of research, a clear view of what AmI implies for hardware and software architecture and technology is lacking. In many realizations of ambient intelligent scenarios, the intelligent response is a built-in concept, particularly for the scenario. Instead we need to develop technology that admits cooperation scenario's not envisaged at design time. Many terminals (sensors, actuators, storage devices, but also advanced processing) do not have the information about their operating context and should therefore expose their functionality for coordination by third parties. A Service Oriented Architecture addresses just that.

In its simplest form a SOA consists of network-exposed services, discoverable and usable by clients. One step further the services may be connected to form distributed applications. The client that does this does not necessarily take part in this; it is referred to as an orchestrator. A downside of the SOA approach is a lack of control by service owners. In this work we address this through the introduction of a virtual community.

In this section, we first briefly introduce how SOA enables the functionality sharing and what limitations still underline the current approaches. Then we present essentials of our approach where we extend the SOA with the concept of a virtual community through describing functionalities of a virtual community, virtual community formation and maintenance mechanisms, protocols for secure service discovery, access and cooperation in a virtual community.

2.1 Value and limitations of SOA

SOA is based on the principle that functionality is encapsulated and executed within self-contained software components; this functionality is exposed on the network as so-called services. These services are subsequently used to be composed into an application. This approach calls for providing standardized service interfaces in order to be integrated. This approach can also be used to integrate existing applications, by wrapping them into a shell that exposes these standardized services. To subsequently describe how SOA enables the functionality sharing we first depict what a service is.

A service is a contractually specified overall functionality that is offered on a network. The functionality is provided at a service access point and made available by a service provider to a service requester / user through a service interface. A service access point is the address where a service is available through its interface. A service interface is the means to access a service; it consists of a set of actions and responses that make the service available. In addition to a service interface, there is a service contract specification, or service description that describes the functional and non-functional aspects of a service. Functional aspects of a service are the effect of the actions on service state variables and on output parameters of these actions. In addition, they indicate the access protocols that are the rules as how and in what sequence actions must be invoked and how any response and events will be generated. Non-functional aspects typically comprise quality properties. An example is a 3D video generator service whose provider is the one of the SAN groups' PCs. Service users can access to its 3D video generation functionality by invoking an action in the "3DGen" interface access point through the following contracted execution rules. The invocation is sent to the particular access point "<http://nbwin450.campus.tue.nl:8081/3DGen>" which, for this example, is known by service users in advance. Many services offering the same 3D video generation functionality may coexist in a network but they may have different properties that make them

different. Examples are static properties, like supported video standards, multiple viewpoints needed, as well as dynamic ones, for instance, current resource usage, predicted latency, etc.

The advantage of exposing a service like this 3D video generator service is that it can be used now without reference to an Operating System platform, an implementation language or machine architecture. The 3D video generator service itself does not 'know' in which context it is being used. A service user may access the service directly; alternatively, a composition of a service by externally binding its interface to that of another service may yield a networked application. This external service composition yields a powerful way to realize ambient applications. Essentially this is what we are interested in and are investigating.

Currently, established technologies for SOA include for example the Web Service in business environments, as well as UPnP and JINI in the networked CE world. These technologies naturally implement the philosophy of SOA by using protocols based on widely accepted standards, such as XML-based SOAP Protocol. An example is an e-business application that uses an email service, a payment service and a web-interface. In the CE domain examples comprise the cooperation of a multimedia-service and a player-service to show multimedia. Service composition in these examples entails coordination of the information flow between services.

SOA has some fundamental advantages over traditional distributed application systems:

- The functionality in a SOA can be reused to a high degree;
- Relying on standards it provides a highly flexible and adaptable implementation for services and applications;
- It supports an extreme form of separation of concerns;
- Eventually, it becomes possible to switch from a particular service to a different one without adaptations.

Basically, at a conceptual level, SOA is composed of three core roles [11]:

- *Service registry*: It acts as an intermediary between providers and requesters. Most of the service registries sort registered services in different categories. In the remainder of this report, the registry is also called a service repository or just a repository.
- *Service provider*: The Service Provider defines a service description and publishes it to the service registry.
- *Service requester*: The service requester, which is called as a service user as well in this report, can use the service registry's search capabilities to find service descriptions and their respective providers.

The three activities the service requester and provider in a SOA can perform are as depicted in Figure 1 [11]:

- *Publish*: The service provider has to publish the service description in order to allow the requester to find it. Where it is published depends on the architecture.
- *Discover*: In the discovery process the service requester retrieves a service description directly or queries the service registry for the type of service required.
- *Invoke*: In this step the service requester initiates an interaction with the discovered service at runtime using the binding details provided in the service description to call or contact the service.

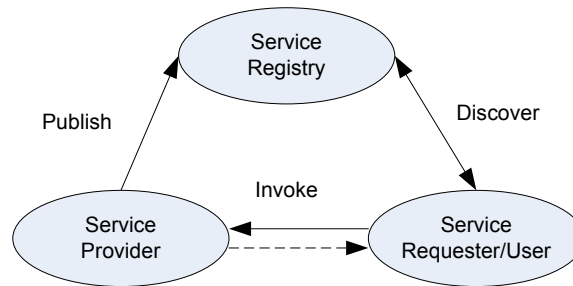


Figure.1. Activities in SOA

According to the concepts, SOA may have a central point of service registry, often seen as the bottleneck in the UDDI solution adopted by web services. Services published in this service registry are supposed to be discoverable by all service requesters. This leads to a privacy and security problem when a service provider only wants to reveal his services to a trusted service requester / user.

In addition, service providers and service requesters should have some guarantee that the agreements in the contract, for example, promised functionalities and QoS in the service description, are supported. That is difficult when a SOA lacks monitoring and enforcement authority. Invocation happens between service providers and requesters in a fully distributed point-to-point way. Point-to-point cooperation is on the one hand increasing the flexibility, and on the other hand is subject to reducing a possibility for a service provider to provide contracted QoS since there is no enforcement mechanism, for instance, behaviour being monitored by a third party. An example here is that a service provider may cease its offered service without informing its users about it and a service user may try to use the service without paying (in whatever form).

Figure 1 shows a simplest form of a SOA. We are interested in investigating the SOA approach for composing applications from services. Services can be connected by a fourth party called an *orchestrator*. This orchestrator connects and binds discovered services externally to form distributed applications.

In this report we present a SOA extended with the concept of a virtual community that is focused on secure service discovery and better QoS guarantee in service cooperation and relies on maintenance mechanisms. This SOA with virtual communities approach helps protect the privacy of a service provider and includes incentive for high QoS provision by using monitoring and reputation.

2.2 What is a virtual community?

The term virtual community has been interpreted in many diverse ways. For example, people use computers to communicate and form friendships that form the basis of virtual communities [12]. An interest based virtual community can be characterized on the basis of a shared (intellectual) interest, for instance, members of a political organization, or a Lords of the Rings fan club. Meanwhile a functional virtual community can be defined as a group of users participating on a single application platform, for example, an online game such as Ultima Online [13].

In our opinion, users' initiatives to create virtual communities are always benefit driven. Users with some commonalities, such as having a common interest in some subjects, or intending to get benefits from sharing information, hardware and software with others, can create virtual communities. To obtain benefit is the basic commonality they have. This benefit

driven virtual community formation initiative can be divided into several categories: e.g. interest driven, application driven, relationship driven, etc. A simple example of an interest driven virtual community formation is when a group of people with common interest in music form a virtual community; they can then exchange classical music, the latest news and recommendations about the new music from each other. An instance of an application driven virtual community formation is when the several parties form a drug discovery virtual community. Through accessing the shared drug screening services, they can finish a drug research application without purchasing all the commercial databases and computing resources. Another example is a relationship driven virtual community formation instance; all employees in one company can form a virtual community to share documents or exchange opinions.

These example communities rely on different mechanisms for their representation. The (electronic part of the) community formed by company employees has usually a physical basis through the network that is physically located inside the company. This is sometimes extended to employees that are outside by extending this network virtually (using a VPN). The same method is used to join two distinct company locations. The drug-discovery community relies on access control at the various services that are needed, and even physical travelling to facilities. The music and data sharing can also have a form of registration and access control; besides, it can also reside entirely in the protocols that actually perform the sharing.

In this report, we describe a virtual community as follows: it is a dynamic contract-based aggregation whose members have commonalities and interact via shared services by means of a digital network or the Internet. A virtual community is built on the concept of digital network connection. It is designed to attract and retain members who become more than superficially involved in community events. It has rules that each member has to follow. It provides services to members and it has the potential to develop different applications through service cooperation.

In a virtual community, we restrict attention to a group of service providers / services who have trusted each other and are expected to communicate honestly with each other within a membership boundary. Their reputation relates to their behaviour. A virtual community is then characterized by the following vital elements: a distinctive focus, membership identification, trust and reputation, and an application orientation. Particularly, the distinctive focus of forming a virtual community in this report is to enlarge the notion of SOA with a secure enriched service sharing and collaboration.

A user can apply to join a virtual community and become a member of that virtual community. Each member should obey the contracted management policy of that virtual community, for instance, they should trust each other and provide the promised quality of service. A member's behaviour is monitored and relate to his reputation. In case of misbehaviour, he will pay some form of penalty. Each member is autonomous. That means that he has the right to determine what services that he owns can be shared with others and which member can access to his shared services. Furthermore, the virtual community is a dynamic aggregation so that a member is free to decide to leave the virtual community at any time.

Services can be shared among virtual community members. Presence of a service can only be known by members of that community. That means that only community members can discover and access to these shared services. Recommendation for service selection in the service discovery process can be gained from a recommendation service of a virtual community. Communication between service provider and service users can be protected using the virtual community security mechanism.

Existing virtual community research focuses on different issues. We describe those different types of virtual communities by discussing an example in each type: BitTorrent, Orkut and Fednet.

BitTorrent [14] is an essential one of the many P2P file-sharing systems [14-20]. Its virtual community focuses mainly on the P2P overlay network concept. Users with similar interests will be grouped into a virtual community automatically to speedup the download and solve the network bandwidth restriction problem. Members in the same virtual community will share resources and files. Here, resources include CPU, memory, hard disk storage, and network bandwidth, etc. The virtual community formation objective of BitTorrent is that through sharing to improve the performance of the whole community. However, the anonymity in its community forming limits the scope of applications.

Orkut [21] is a Web site that aims at bringing social relationships online, providing user-to-user messaging, user-definable news groups, and on-line photo albums, in addition to its social-network management functions. Users are invited into Orkut by existing users, directly establishing a link to an existing social network spanning the whole user community. Users can then invite new friends and navigate the social structure to find friends and acquaintances to establish more direct links. A powerful feature of Orkut is that it can show the path of friends that link you to any other person, for example, enabling you to discover mutual friends and interests (visible via the public newsgroup memberships). These explicit Orkut communities attach the notion of communities to activities which have the similar concerns of services in our approach. But Orkut is less service oriented than our virtual community and there is no composition of activities into new applications.

Fednet [22] is a temporary ad-hoc federation of networks with the objective of achieving a specific common goal. It focuses on virtual communities formed by personal networks (PN), which are person-centric networks connecting personal devices and artifacts regardless of geographical location. Fednets are driven by purpose and opportunity. Fednets are constituted by a set of independent networks under mutually agreed rules for cooperation. The rules for cooperation are policies to control under what circumstances (context) with what networks and for what purposes cooperation is allowed. Different from our approach which has a service concept, Fednet focus on forming virtual communities that can be independent of location. So far Fednet is still a concept and there is no clear technology concept.

2.3 Research issues in virtual community

In this report, we focus on virtual community whose members interact through service invocation via digital networks or Internet. When we discuss virtual communities we are not referring to any aggregate of people, but to the communication among them which is done by the service sharing and cooperation.

In the following we briefly discuss some research directions that are specified in the context of virtual communities.

- Functionalities and architectural components of a virtual community
- Formation and maintenance of a virtual community
- Service discovery, service access and service cooperation in a virtual community

2.3.1 Overlay concept of a virtual community

Essential to organizing services into a virtual community are secure, trustable service discovery and guaranteed service cooperation. Towards these goals, we define the functionalities constituting a virtual community, depicted in Figure 2.

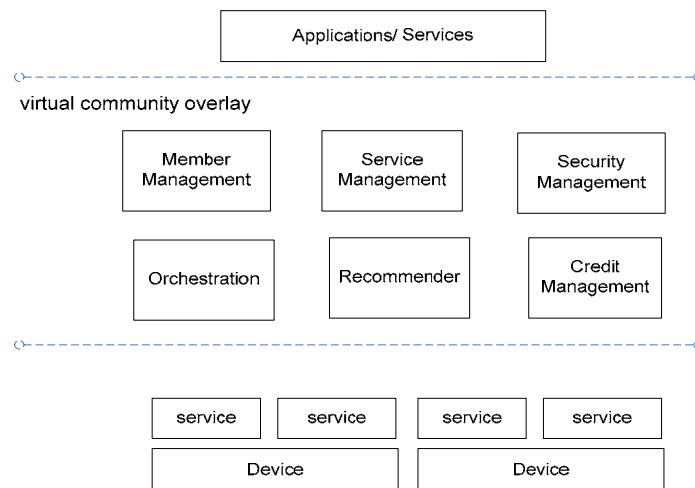


Figure.2 Overlay Functionalities of a Virtual Community

In the SOAVC approach shared functionalities are first wrapped into services. These services are then organized in virtual communities by adding additional functionalities to them. From this point of view, a virtual community can be treated as an overlay network for the existing services. In this overlay, a service owns enhanced functionalities: it can only be exposed to the authenticated virtual community members; it can filter access requests using its access control policy to and all the exchanged messages are encrypted. In addition, a reliable service discovery can be achieved through trustable recommendation based on the past interactions. All activities including publication, discovery, invocation and cooperation are executed within the scope of a virtual community. Meanwhile, a virtual community service has the properties of a plain service: its external network interface still uses the SOA service interface; SOA rules such as the SOAP and the HTTP protocols are still suitable to service cooperation.

As shown in Figure 2, there are six functionalities in a virtual community overlay. Detailed descriptions of these functionalities are shown in Table.1.

Table.1 Virtual Community Functionalities

Functionality	Task
Member Management	To deal with member's activities inside a virtual community including registering a user as a member and deregistering a member.
Service Management	To deal with service related activities including service publication, discovery and access. This is a generic functionality that all services have; this allows them to use the virtual community.
Security Management	To protect the privacy of service providers and to provide secure service access. An example is to verify authentication of a service requester.
Credit Management	To deal with activities related to virtual community maintenance. For example, monitor and evaluate services' behaviour, help to maintain registered service and member lists, etc.

Recommender	To make trustable recommendations for a service user in the service selection stage. For example, to select a service with the best quality.
Orchestration	To offer a generic service with special virtual community roles that can be extended for achieving particular applications through external service orchestration.

Specially, the *Credit Management* is designed to facilitate maintenance of a virtual community. Using this independent component, a contract-based policy for service cooperation and virtual community maintenance can be defined; members and services' behaviour can be monitored and evaluated. For example, the creator of a new virtual community can establish a credit threshold. A member who provides reliable and high QoS could get his credit increased gradually while another one who behaves unsatisfactorily will get his credit decreased and may even be forced to deregister from the virtual community when his credit has dropped below the threshold.

Moreover, trustable recommendations given by the *Recommender* can help a service requester / user to select the service with the best perspective quality.

Figure 2 presents the overlay context of a virtual community and the basic functionalities it has. Using UML notation [23], Figure 3 depicts the essential ingredients that together compose a virtual community. The design of the VC follows the principles of SOA. The basic functionalities that a virtual community has are implemented by multiple services, which are *VCEntry*, *ServiceMgt*, *CertificateMgt*, *CreditMgt*, *Recommender*, *Orchestrator* and *Repository* as well as several data elements, which are *JoinPolicy*, *PolicyInfo*, *RoleInfo*, *BlackList*, *ServiceCredit*, etc. *VCEntry* and *CertificateMgt* are accessible to any parties including requesters on the network and internal community members, while some other services are only available for community members. For example, *Repository*, *ServiceMgt*, *CreditMgt* and *Recommender* can only be invoked by authenticated community members.

VCEntry is a service to deal with members' actions inside a community including member registration and deregistration. *JoinPolicy* is a description of the agreed community joining policy. An example for a joining policy is that only those who supply actual personal information including IP address and e-mail address can be approved to register as a member. When one satisfied the *JohnPolicy* has been authorized to become a member, specific roles are assigned to it by the *VCEntry*. It could be a service requester to access other services, or a service provider to register / deregister a service, or an administrator to change the threshold which is stored in the *CreditMgt*, or a combination of these example roles. As a member, it can execute corresponding actions according to its role while carrying a valid ticket signed by the *CertificateMgt*.

When a member publishes services into a virtual community, those services will be registered at the *Repository*. The *Repository* is a service repository provider that is dedicated to resolve queries. It caches service advertisements, receives queries, and performs matching between queries with available services. To execute this publication, *ServiceMgt* will be called by the *Repository*. *ServiceMgt* can label a plain service as a community service by adding for example the *Credit*, the *ACLlist* and the *BlackList* properties to it, then advertises it to the *Repository*. *Credit* is an attribute to represent the reputation of a service related to its past interactions. A service can specify its local access control policy by editing its *BlackList* and assigning allowed actions to different requesters. These virtual community labels will be removed by the *ServiceMgt* when a service is deregistered.

The *CreditMgt* is designed to facilitate the virtual community maintenance and a contracted QoS guarantee for service cooperation. It monitors service providers' behaviour by

periodically comparing their credit values with the threshold which is set by the virtual community administrator. A service providing reliable and high QoS will get its credit increased gradually while when its credit is smaller than the threshold, the *CreditMgt* will expel it from the virtual community. This service's information will be removed from the *Repository* and then the *ServiceMgt* is informed to remove its virtual community labels.

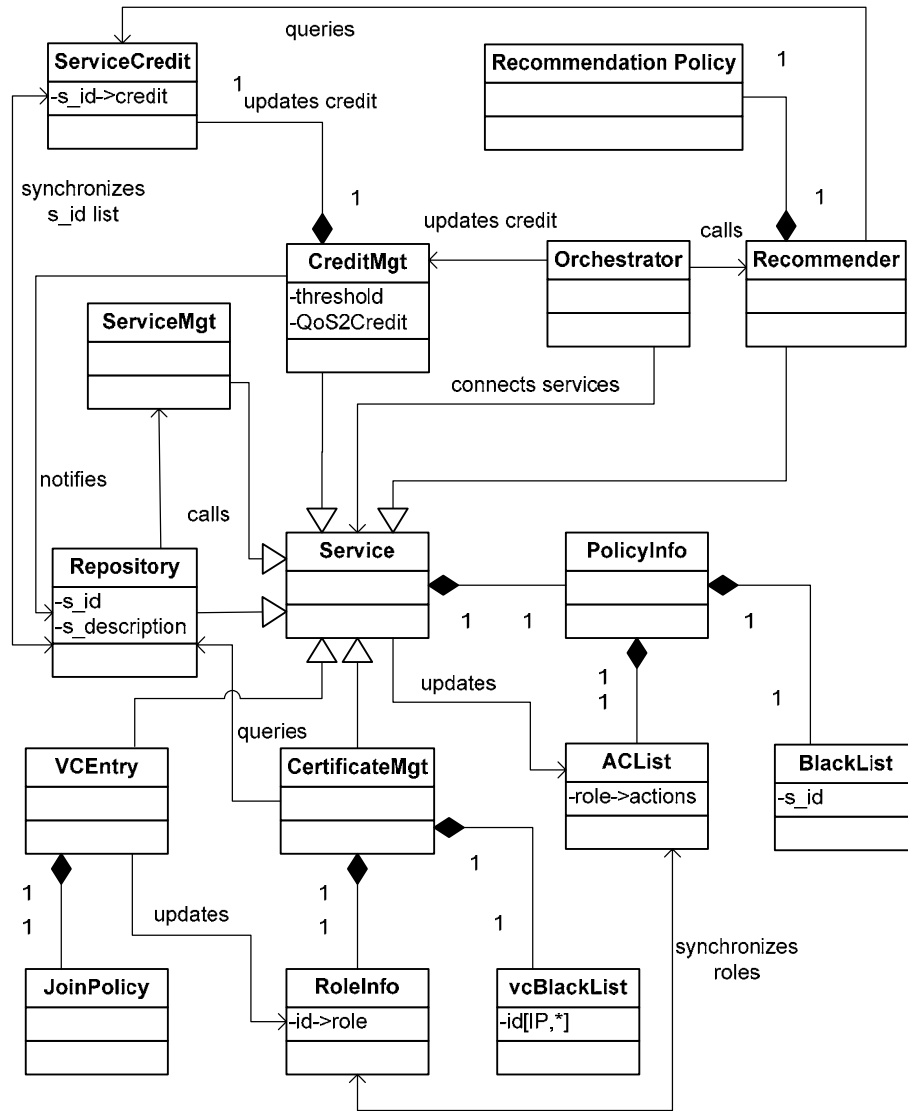


Figure.3. Ingredients of a Virtual Community

Members with particular requirements can establish an *Orchestrator* which connects services and makes them cooperate together to accomplish an application. During the application execution, the *Orchestrator* can call the *Recommender* for recommendations about service selection. The *Recommender* will query those candidates' credit values and make recommendations using a specific *Recommendation Policy*. After the service utilization process has finished, a service user, for example, the *Orchestrator* can give feedback on the provided QoS of those services. In this case, the *CreditMgt* will firstly translate these QoS feedback into credits using *QoS2Credit* policy and update the corresponding credit states in *ServiceCredit*. Alternatively, a service provider can also evaluate the reputation of his service users. For example, if he treats a service user as a malicious user, he can invoke the *ServiceMgt* to add that malicious user's identity into the *BlackList* of services owned by him.

As a result, requests coming from those who are in the *BlackList* can be rejected by a service even if they have a valid ticket signed by the *CertificateMgt*. We name this as the *autonomous access control policy* of a service, whose working principle will be described in detail in the service access section.

2.3.2 Virtual community formation and maintenance

The virtual community formation process can be carried out when an initiative arises. The process of establishing a new virtual community is shown in Figure 4.

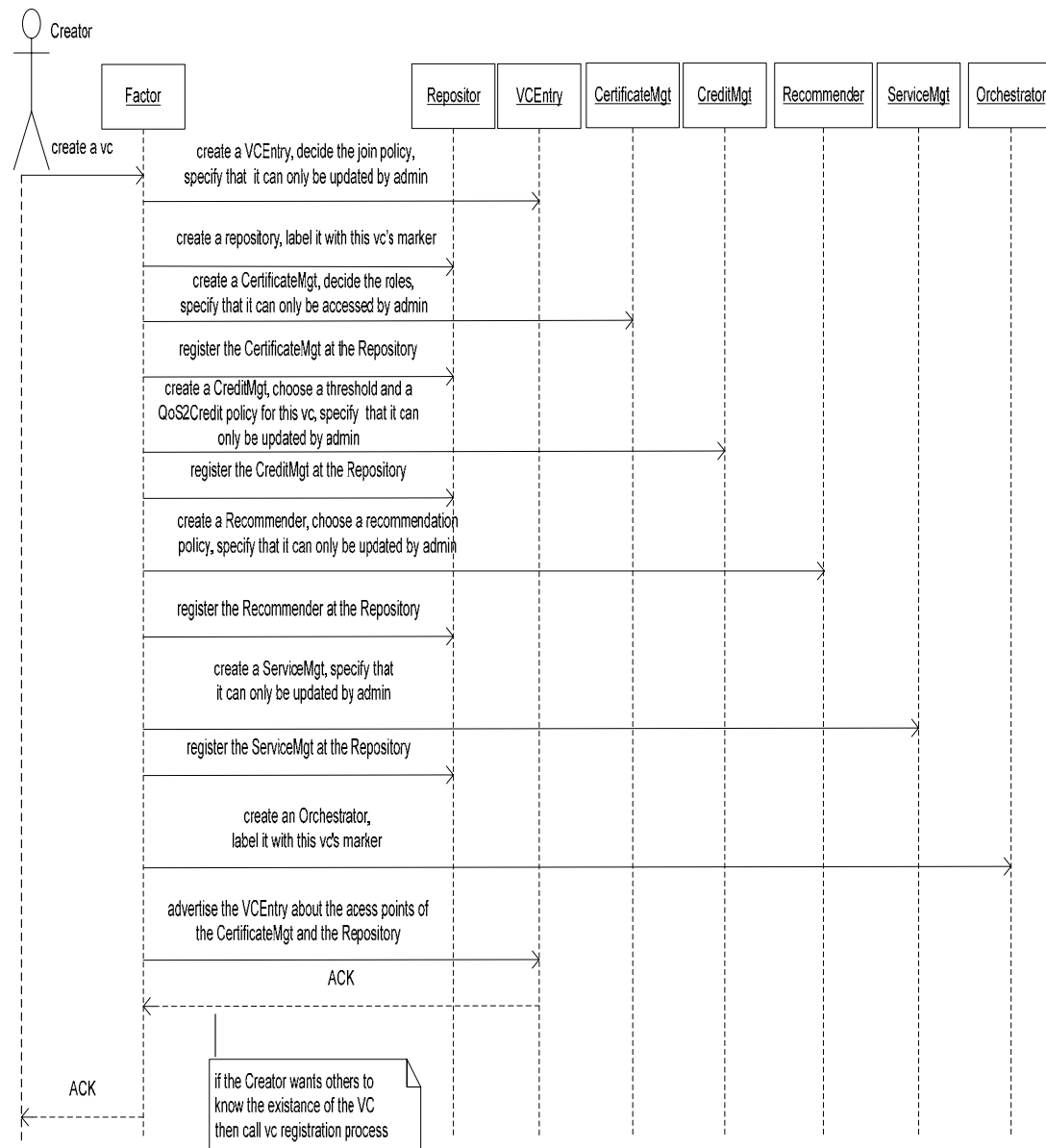


Figure.4 Virtual Community Formation

The creator of the virtual community has the right to define the management policy. As presented, the virtual community is created using the concept of a SOA as the support. All the software components are present as network exposed services. Aiming at this, the creator, represented by a software entity called a *Factory*, has to create all the needed services and

data structures. It will create a *VCEntity* service including the *JoinPolicy* as well as a *Repository* service which is assigned as this virtual community's repository. Then a *CertificateMgt* service with designed roles, a *CreditMgt* service with a *threshold* and a *QoS2Credit* policy, a *Recommender* service with a chosen *Recommendation Policy*, and a *ServiceMgt* service will be created. Furthermore, for all these services it will be specified that they can only be updated by a member who has the administrator's authority. Later on, the *CertificateMgt*, *CreditMgt*, *Recommender* and *ServiceMgt* services will be registered at the *Repository*. An *Orchestrator* will also be created which at this stage is only a generic service that has special roles, for example, a service user who can access to the *Repository* and can bind all registered services. When an application request comes, a user can extend this *Orchestrator* for particular requirements. The access points of the *CertificateMgt* and the *Repository* should be advertised to the *VCEntity*. If the creator wants to advertise the existence of this new virtual community to the network, the next step will be the virtual community registration.

In order to attract more users to join this new established virtual community, a service description of this community including its access point, type and other attributes should be advertised on the network. The establishment of a virtual community should preferable be a process that does not require manual involvement. However, as a bootstrap step, manual involvement maybe desirable in certain cases, for example, in order to get to know where to advertise virtual community descriptions a virtual community creator has to check a yellow page or to acquire knowledge from trusted friends. In this case, to discover a virtual community repository has become our starting point to register a new virtual community. Existing protocols to this service discovery zero configuration problem include the Microsoft and HP's uPNP Simple Service Discovery Protocol (SSDP) [24], Apple's Multicast DNS (mDNS) and DNS based Service Discovery (DNS-SD) [25-27] and the Service Location Protocol (SLP) [28]. In this report, we use the methods shown in Figure 5 to process the virtual community repository discovery.

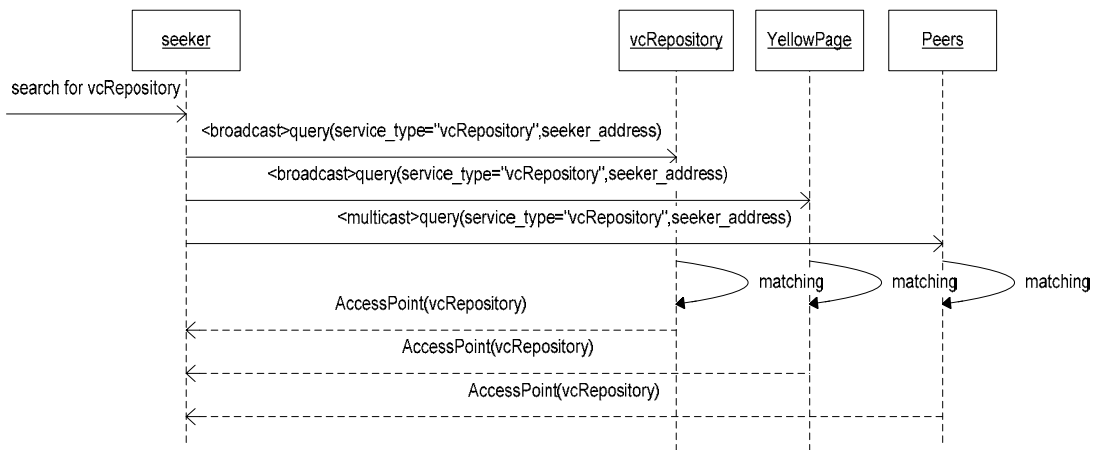


Figure. 5a. Virtual Community Repository Discovery in the Immediate Way

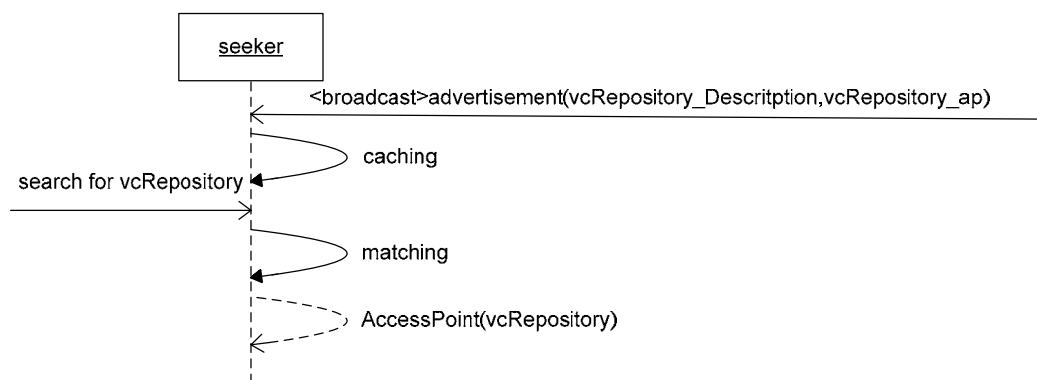


Figure. 5b. Virtual Community Repository Discovery in the Immediate Way

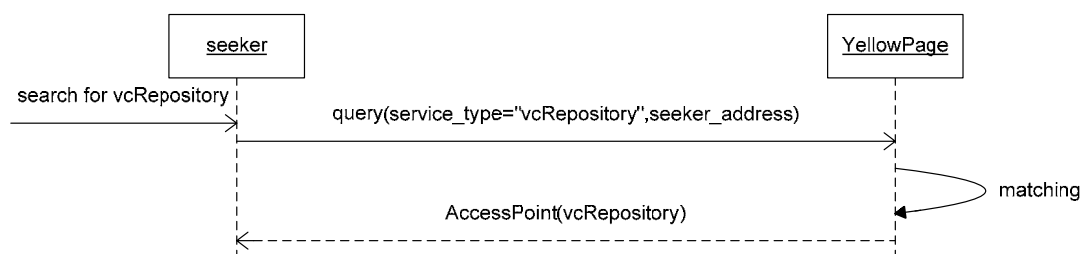


Figure.5c. Virtual Community Repository Discovery in the Mediated Way

Three kinds of intermediate have been listed in Figure 5a.

- *vcRepository*. It is a virtual community repository service. As a special instance of a service repository, all kinds of virtual communities can register there and be discovered by virtual community seekers.
- *YellowPage*. It is an instance of service repository. It caches all kinds of service descriptions and service seekers can query expected services, for instance a virtual community repository service or a 3D video generator service. Examples of YellowPage are telephone books, Google [29] and other search engines. Compared to the *vcRepository*, no registration is needed.
- *Peers*. Peers are potential information providers such as trustable friends or family members.

In the case when a *vcRepository* and a *YellowPage* cannot be found, messages coming from *Peers* will be very useful. A peer can inform the seeker about the existence of a *vcRepository* which only provides service to trusted or recommended users. Furthermore, a seeker can get extra-functional information of a *vcRepository*, such as reliability, response time and prices which usually are not provided by service description and matching.

There are two different ways to discover a virtual community repository through these mediums: the immediate way and the mediated way. We use *broadcast* to indicate that the message propagation domain is a local network domain and we use *multicast* to indicate that the messages are propagated in a multicast group or an overlay network.

In the immediate way, interactions between a virtual community repository seeker with the three kinds of mediums are done in advertisements. The inputs to the advertisement process are the expected service description and the message sender's address. The former contains

the service type, while the latter specifies the address where the response should be sent back. A number of options are possible here.

- Figure 5a: A virtual community repository seeker can broadcast the service query to the network or multicast to his peers. When the query arrives at listening mediums, mediums will respond to matching queries.
- Figure 5b: Instead of sending out the virtual community repository query to the mediums, a seeker quietly listens to service advertisements coming from the network. These advertisements can come from any services. For instance, mediums regularly send out service advertisements to all nodes in the network by broadcasting or to some interested nodes by multicasting. The virtual community seeker caches the advertisements for later use.
- A hybrid form of figure 5a and figure 5b.

From this point on, in the immediate way, the matching is done by either the service providers or the service seekers.

In the mediated way, as shown in Figure 5c, the interaction happens between the virtual community seeker and the YelloPage. With the access point of the YellowPage as the seeker's preknowledge, queries for virtual community repositories are sent directly to the access point of the yellow page. Afterwards, this matching process takes place at the yellow page.

In both ways, the matching result is a set of access points of the virtual community repositories. It is sent directly to the service seeker's access point (extracted from the query). It is possible for a response to contain multiple matching vcRepository access points. It is then up to the seeker to select the most suitable one. This may lead to iterative queries with refined requirements.

With the knowledge how to access a virtual community repository, a creator can advertise its virtual community to the network including the description of this virtual community and the access point of the *VCEnt*ry. He can follow the methods depicted in Figure 6 to do the virtual community advertisement or registration.

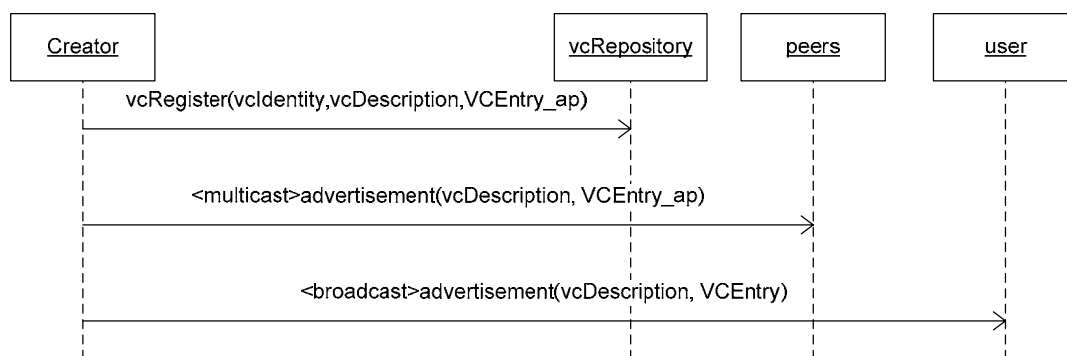


Figure.6. Virtual Community Registration

An option is register at a vcRepository directly by sending the virtual community identification, description and its access point to that vcRepository. Another option is propagate the virtual community description along with the access point of the *VCEnt*ry through the network. Peers or potential users with interest will cache this advertisement for later use.

If a user is interested in an existing virtual community and wants to join it driven by benefit, she needs pre-knowledge of where to access that virtual community. For example the description of that virtual community and the access point of a *VCEntry*. The means to discover a virtual community repository described in Figure 4 can also be applied to the *VCEntry* discovery with only changing the content of queries from a *vcRository* into a *VCEntry*.

One of the main goals of forming a virtual community is securing the interactions and keeping privacy of service providers. We design that all the exchanged messages in a virtual community are encrypted and transferred in a secure channel set up between senders and receivers. Message receivers have to firstly decrypt received messages before they can use them.

Once a user applies to become a community member, the member registration process is activated. As shown in Figure 7, after a user sends her identity to the *VCEntry*, the *VCEntry* will check with the *JoinPolicy* then come to the conclusion whether her application could be approved or not. If the member registration application has been approved, the *VCEntry* will invoke the *CertificateMgt* to assign her with an available role and add her information to the *RoleInfo*. Upon success of these processes, the *VCEntry* will inform her about the access points of the *Repository* and the *CertificateMgt* as a response message from the message registration process.

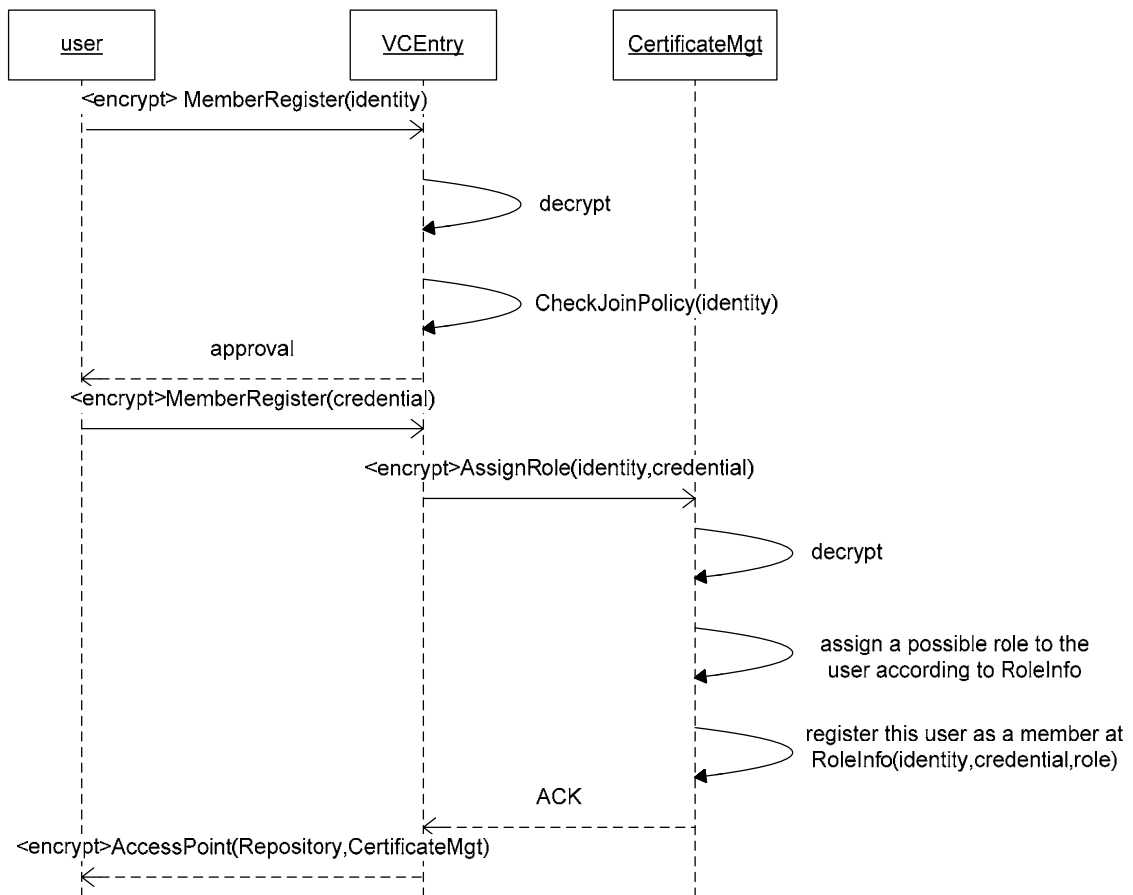


Figure.7 Member Registration

In order to increase the security of service discovery and of the access process in a virtual community, we require that only authorized members can register and access services. Considering the distributed nature of a virtual community and the autonomy of services, we add access control functionality to each service. Consequently, whenever a service is accessed, this access control process will be activated to check whether a service user has a validate ticket signed by the *CertificateMgt*. A ticket consists of several elements: the *role* of a service user which indicates what actions she can execute to that service; the *signature* of a recognized party, for instance the *CertificateMgt*; and a *lifetime*, each ticket will only validate in that lifetime. When a ticket expires, a service user has to apply another new ticket from the *CertificateMgt*. Detailed protocol design of this ticket resolution or validating, revocation and recovery can make use, for example of XML Key Management Specification (XKMS) [30] which includes XML Key Registration Service Specification (XKRS) and XML Key Information Service Specification (XKISS).

Carrying a valid ticket, a member can register services at the *Repository*. Figure 8 has described the detailed interactions happening in a service registration process.

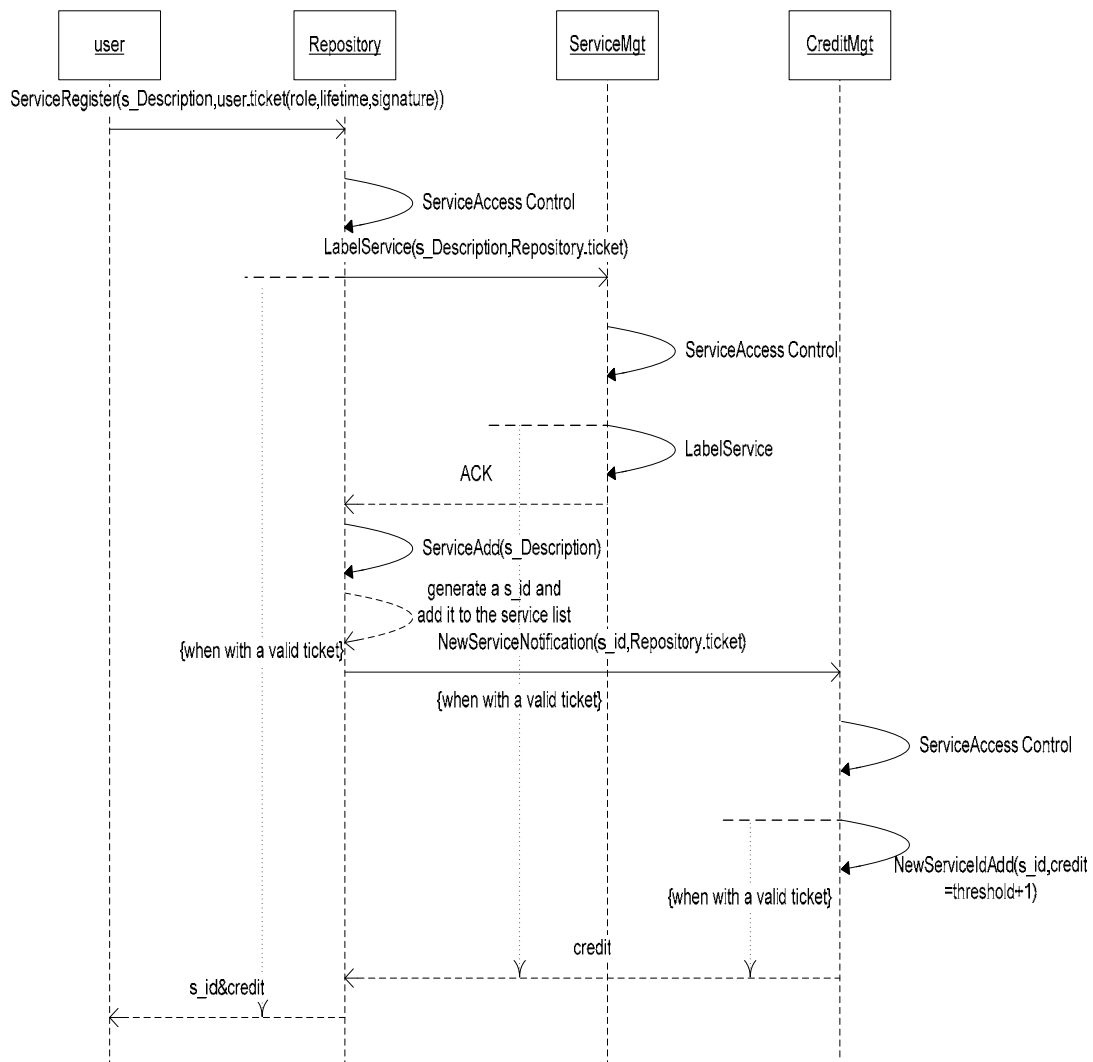


Figure.8. Service Registration

When the *Repository* receives the service registration request from a member, it will invoke the *ServiceMgt* to label that service with a virtual community marker. With an acknowledgement from the *ServiceMgt*, the *Repository* will firstly register this service as a new service of this virtual community and secondly inform the *CreditMgt* that “a new service has been registered”. Then the *CreditMgt* will set a credit for that service as “threshold + 1” for instance. Afterwards, the *Repository* will inform the member about the virtual community identity and the credit value of that service. Then the service registration will finish.

Detailed interaction of labelling a service with a virtual community marker is shown in Figure 9. This is an internal process happening between the *Repository* and the *ServiceMgt*. This process is transparent to a member. With the execution of this process, a service will be enriched with additional virtual community properties. To do this, the *ServiceMgt* adds virtual community tags into the original service description and creates a *PolicyInfo* item, which consists of the *ACLlist* and the *BlackList*, for that service. Available roles of members and corresponding actions are stored in the *ACLlist*. Malicious users’ information can be inserted into a service’s *BlackList*.

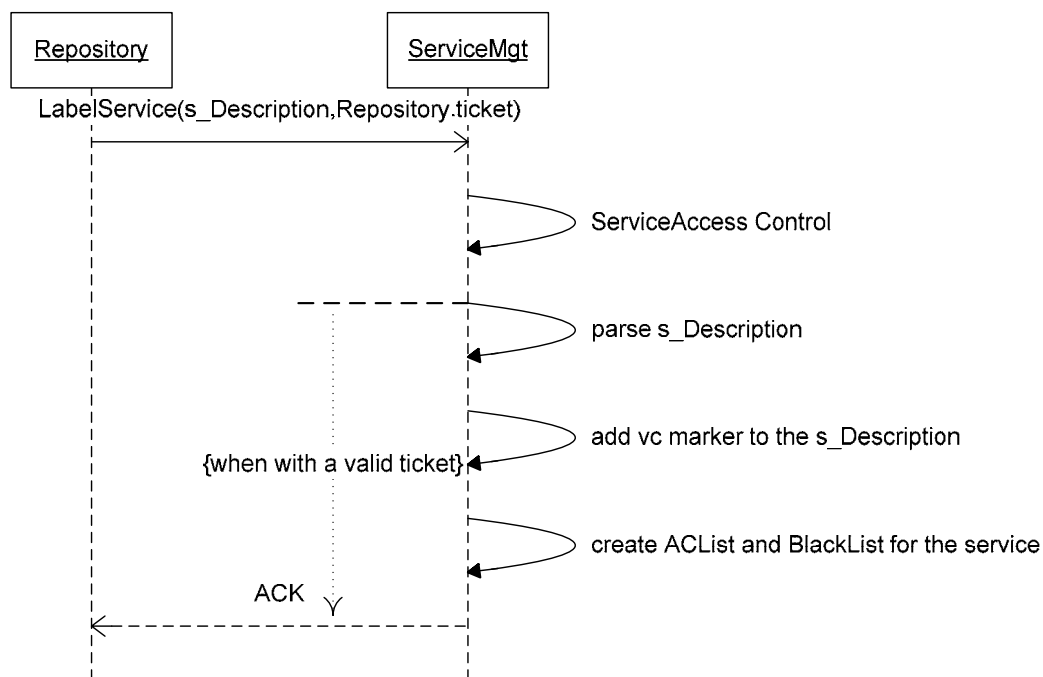


Figure.9. Virtual Community Service Labelling

A service provider can initiate the *PolicyInfo* update process to synchronize the available roles stored in its local *ACLlist* with the *RoleInfo*. Interactions of this process can be found in Figure 10.

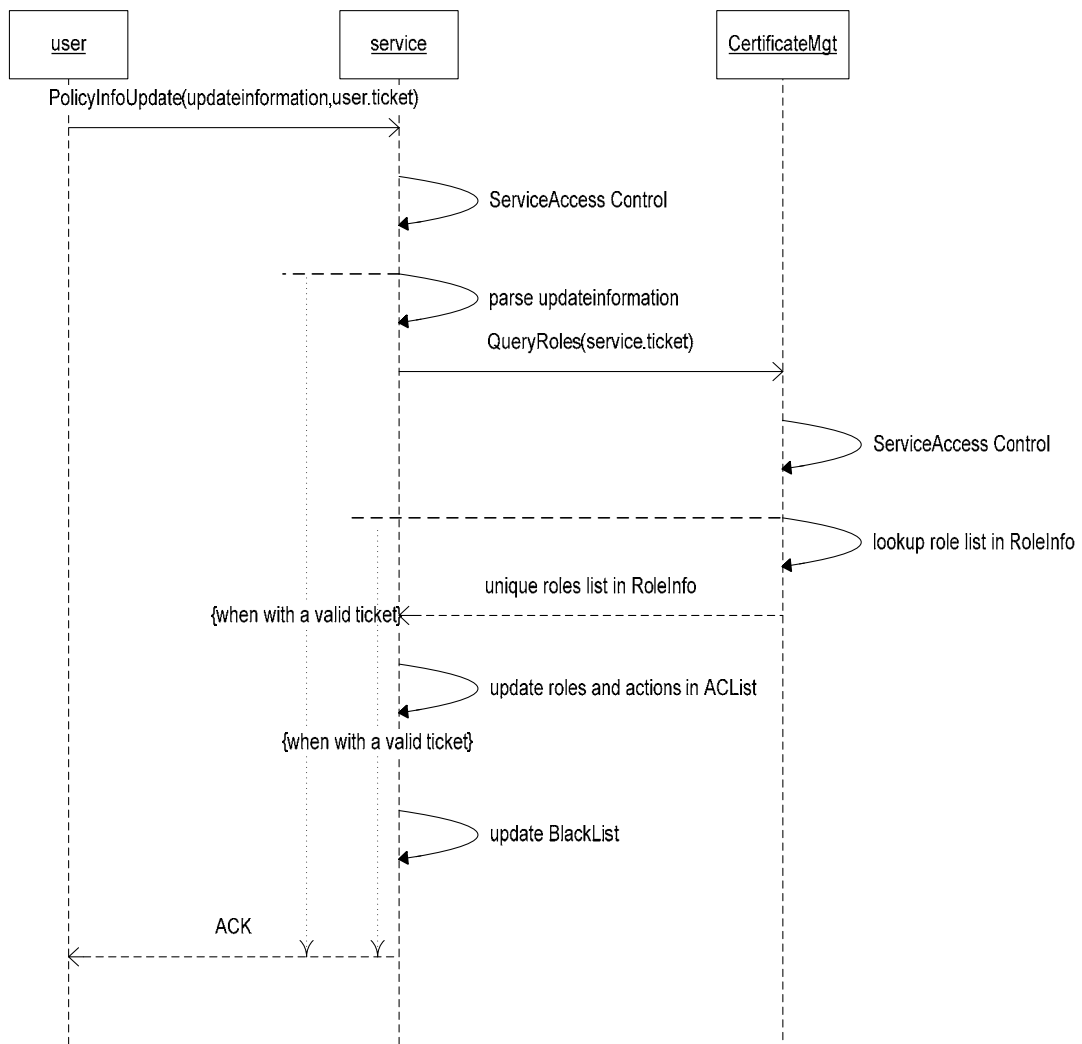


Figure.10. Service PolicyInfo Update

How a virtual community maintains these registered members and services and makes their cooperation running properly is addressed in Figure 11.

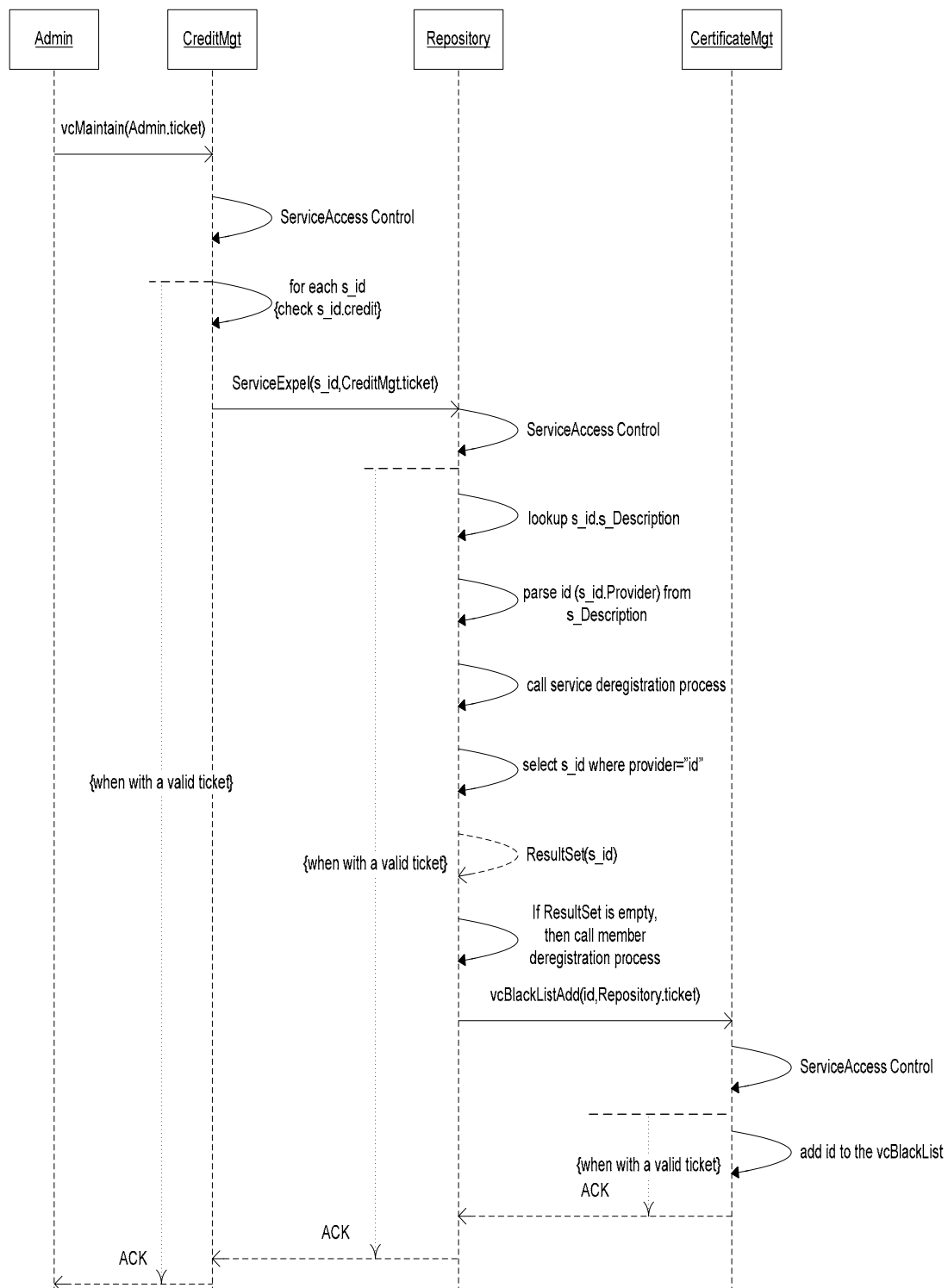


Figure.11. Virtual Community Maintenance

Services' credit will be checked by the *CreditMgt* periodically or at the time when a service's credit is updated. Once the *CreditMgt* detects that a service's credit has dropped below the threshold, it will notify the *Repository* to deregister that service. Firstly, the *Repository* will parse the description of that to-be-expelled service to check get its provider's id. After the

service has been deregistered (Figure 13), the *Repository* will check whether this service is the only registered service by its provider. If it is, the member deregistration process will be called and the identification of its service provider will be added to the *vcBlackList* through invoking the *CertificateMgt* service (Figure 12).

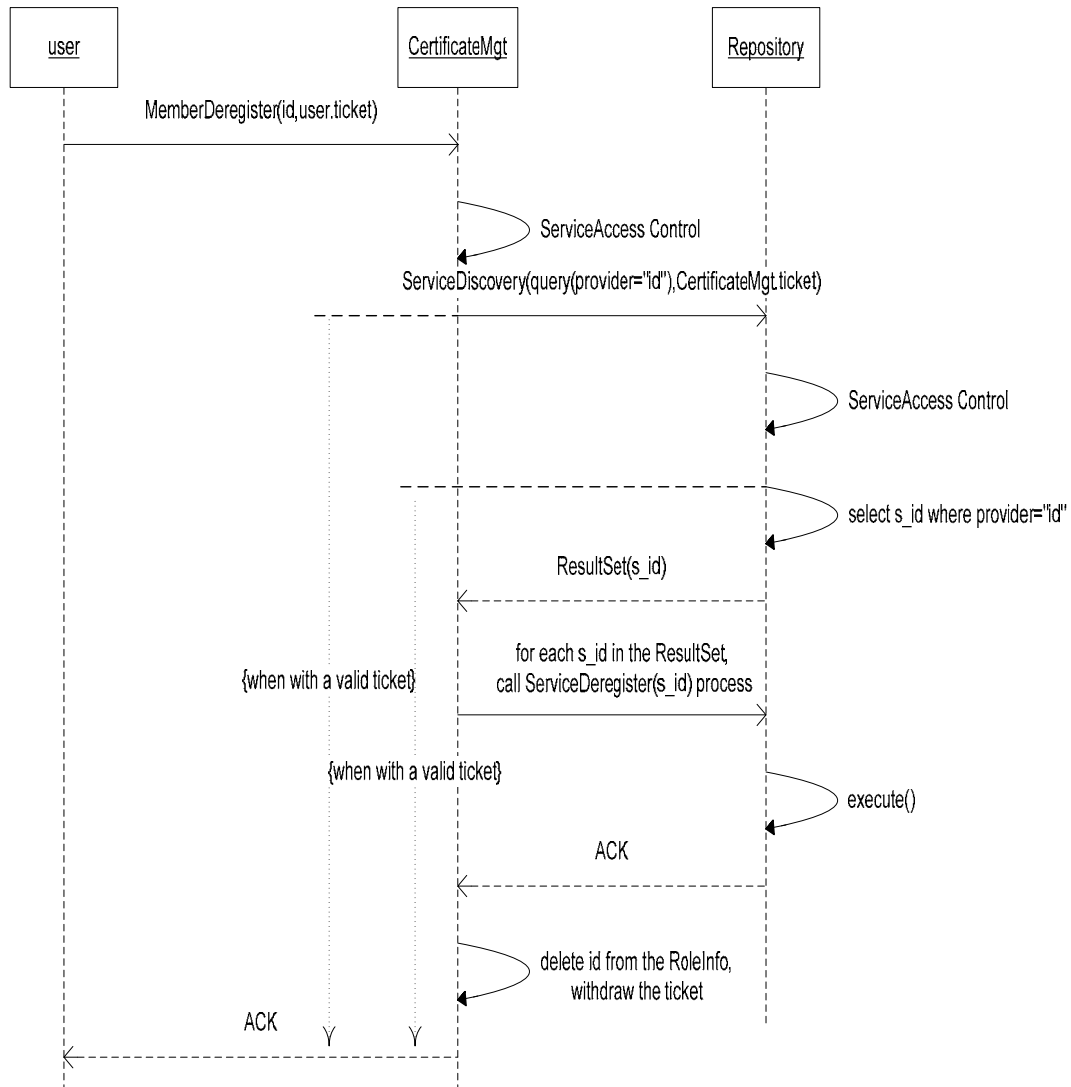


Figure.12. Member Deregistration

When a member deregisters from a virtual community, all his registered services in that virtual community are deregistered firstly. All information related to him will also be dropped, for instance, the unique member id, the roles, the ticket, etc.

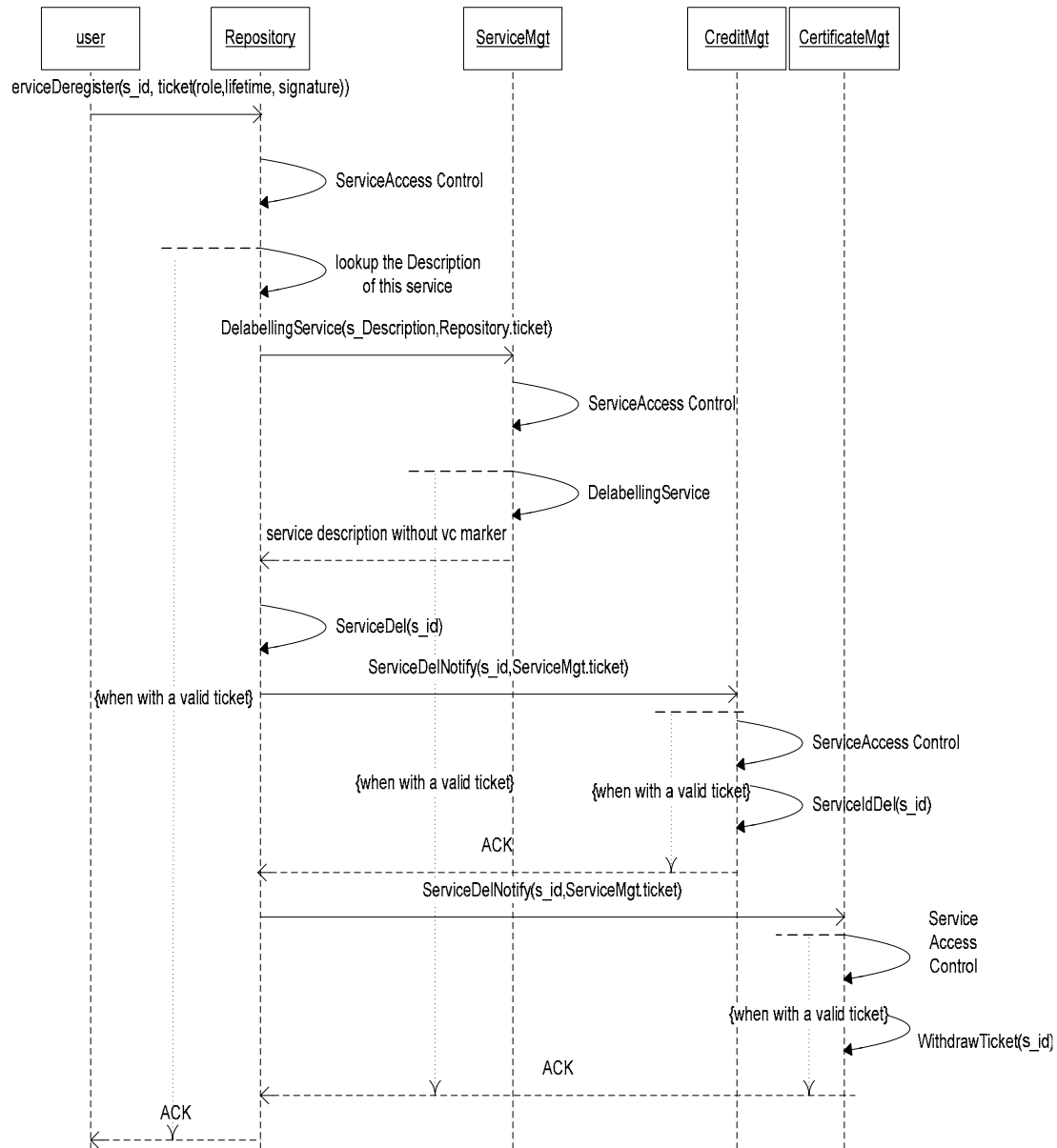


Figure.13. Service Deregistration

When a service is deregistered from the *Repository*, all the virtual community related functionalities and properties will be deleted. These are virtual community markers, the *ACL*ist, the *BlackList* and the unique service id. The removal of these items is done by the service delabelling process which is the reverse process of service labeling.

A virtual community can be terminated by a user with administrator’s authority, for instance, the creator. Referring to Figure 14, during the course of virtual community termination, there are two main stages. Firstly, stop network exposed virtual community functionalities from serving network nodes by disabling the access point of the *VCEntry*; deregistering the *CreditMgt*, the *CertificateMgt*, the *Recommender*, the *ServiceMgt* from the *Repository*; removing the virtual community marker of the *Repository*; and disable all Orchestrators. Secondly, because this virtual community isn’t available anymore, we’d better broadcast this information over through the network as well as deregister it from the *vcRepository*.

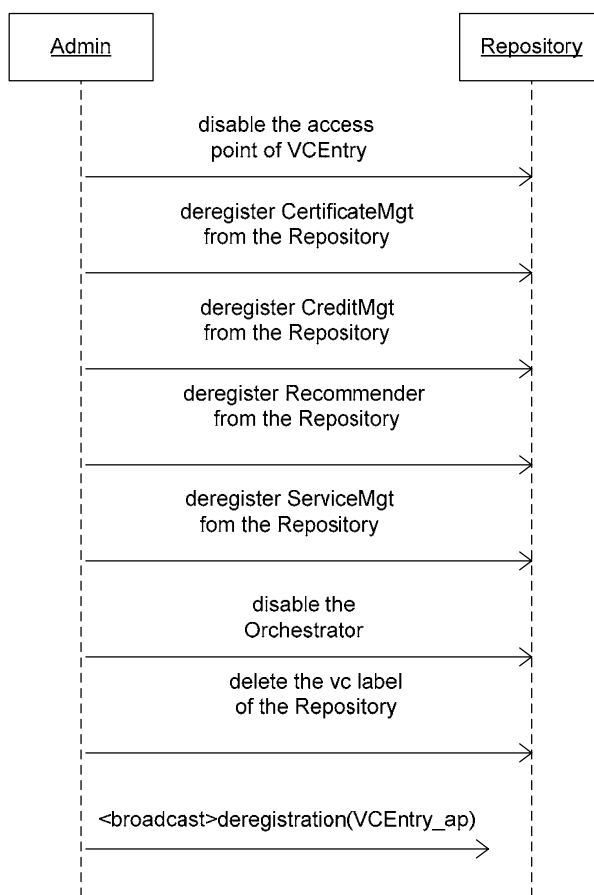


Figure.14. Virtual Community Termination

2.3.3 Secure service discovery, access and cooperation

Service access and service cooperation in virtual community have been involved in all the processes discussed in section 2.3.2. As an example, in the service registration process (Figure 8), when a member accesses the *Repository* service, he is invoking the *ServiceRegister* interface of that service. In order to accomplish this service registration process the *Repository*, the *ServiceMgt* and the *CreditMgt* have to cooperate. Figure 16 is a universal process of service access in virtual community which can be specialized into service discovery, service interface invocation, service static state update, service cooperation, etc.

From Figure 15, we can learn that virtual community adopts a double protected service access approach. With this approach, the virtual community management mechanism and the autonomy of a service itself as well are both considered. Overall, services can only be discovered and accessed by authorized members of a virtual community. Meanwhile, service providers have their autonomy to prohibit access coming from users they do not trust, even when they satisfy the security rules of the virtual community itself. The local service autonomous access control is done by checking the *PolicyInfo* and granting a service user's authorization.

The virtual community repository discovery (Figure 5) and the *VCEntry* discovery are both instances of a service discovery process. Similarly, service discovery in virtual community is also a specific case of service discovery. The difference among them is that service discovery *within* the community is always mediated, while both the virtual community repository and *VCEntry* discovery can be executed in two ways: the immediate way and mediated way. The

service discovery query can be directly sent to that *Repository* (a repository of a virtual community) rather than be advertised to the entire network. Access point of this *Repository* is only known by community members; in this way the scope of service discovery has been limited to the bounds of the virtual community. This makes the *Repository* service a single point of failure for the community; standard techniques can be used to migrate this but this is beyond the scope of our discussion.

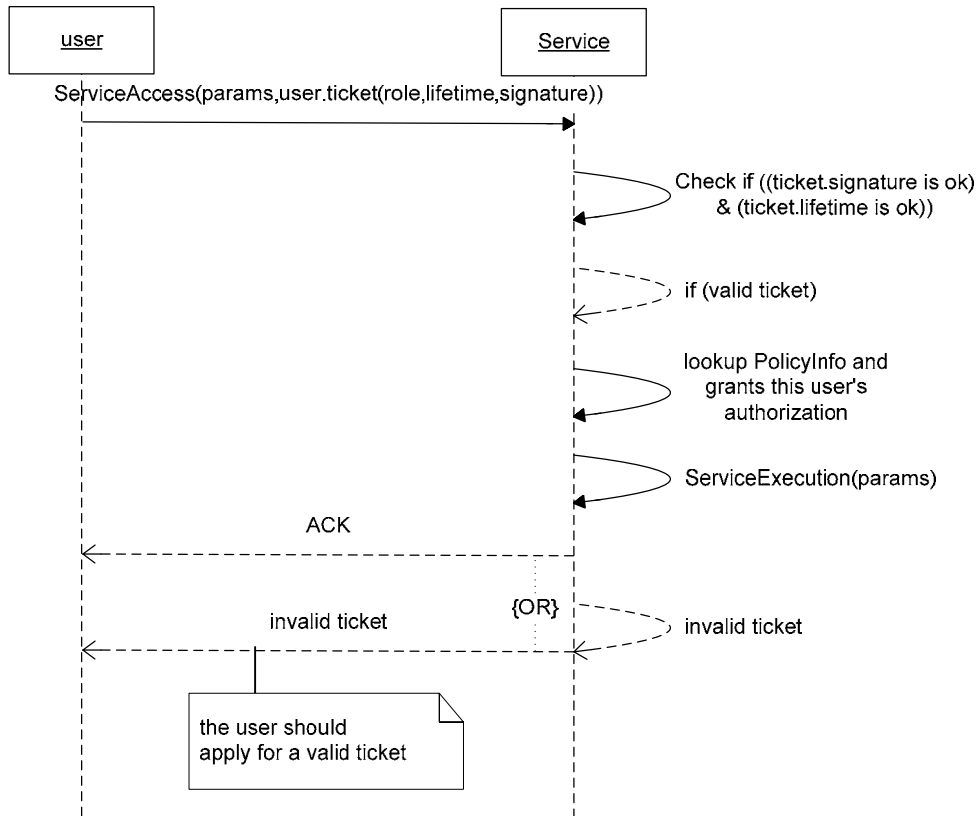


Figure.15. Service Access in Virtual Community

As mentioned, if a service discovery response contains multiple matching services, the *Recommender* can give trustable recommendation to help the service seeker to select the most suitable one with the best expected service quality.

Moreover, after finishing the service cooperation, an *Orchestrator* can evaluate the provided quality of utilized services. The comments can be rated with qualifying dimensions, for instance, excellent, good, normal, or bad; or with more granular, enumerated dimensions, for instance, throughput, response latency, reliability, etc. Using the *QoS2Credit* policy, these comments are translated into service credits then queried by the *CreditMgt* or the *Recommender*. The selection of this *QoS2Credit* policy will tightly coupled with the techniques to describe the QoS of a service. Some standards and methods to describe the QoS will be borrowed from [31-32].

Adopting the double protected service access approach, the virtual community restricts the presence and availability of a service only to its members. With the help of contracted management mechanisms, services are monitored. In this way, they are forced to be reliable, stable and provide a high QoS. In return, the virtual community approach makes the service discovery more secure, efficient and effective. Moreover, the trustable recommendation is another added value to this SOAVC approach.

3. Prototype: AMOSA Platform

3.1 Incentives

The Ambient Open Service Architecture (AMOSA) software platform is a project that aims at bringing better reusability of the existing SOA approaches and at improving the development of service oriented application through external coordination. The major focus of this section is to point out the potential that the current AMOSA technology offers for virtual community forming. This analysis includes the applicability of the AMOSA platform to fully enable virtual community formation, secure service discovery, dynamic service allocation and cooperation in virtual communities. We demonstrate the applicability of the secure service cooperation with external orchestration, as a first kind of the SOAVC approach in a high-level use-case to exemplify the activities within a virtual community concept.

In AMOSA a service is specified through defining provided and required interfaces, where an interface defines a collection of operations and parameters. There are standardized interfaces for specific tasks like binding and management. AMOSA defines bindings to Web services and UPnP and there is tooling support to perform this mapping. AMOSA provides discoverability and interoperability across these standards. Within AMOSA language support is also studied to define the services and service compositions.

AMOSA provides a good test bed for the virtual community approach; with better flexibility in building services with virtual community functionality and composing new applications in an agile manner by leveraging existing SOA infrastructure, virtual community research issues we have discussed in this report, e.g., service sharing, secure service discovery and cooperation in virtual community can be tested.

3.2. Functionality Overview

As shown in Figure 16, AMOSA is a layer below the application layer and it provides an abstract service platform, which can be mapped onto concrete deployment platforms, like UPnP, Web Services, and Jini.

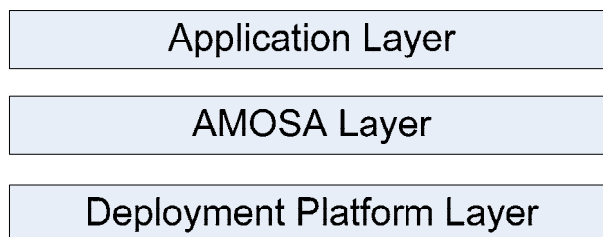


Figure.16. AMOSA Layered Architecture

In the remainder part of this section, major parts of the current version of AMOSA are discussed, viz. the interface definition and the mapping tool.

3.2.1 A_ITF specifications

A_ITF is an (abstract) component interface specification that defines the functionality each AMOSA service provides. This includes at least a functional part (for calling functions and passing parameters), a management part (for establishing connections to other components and manipulating components), a security part, and a fault detection part.

A_ITF is an abstract, high level specification that allows mappings to concrete SOAs, such as UPnP, Web Service, and Jini. For the first two, a mapping has been implemented.

The design of A_ITF allows for interoperability and composability by providing a clear interface and by allowing late binding by using advertisement, discovery, and run-time binding. Orchestrators determine at run-time what services are available and how to compose them, in order to make certain functionality available.

The A_ITF design is implementation language independent and uses AMOSA-level data types. As a consequence, an AMOSA service can be written using different programming languages and can be run on different operating systems, as long as the specified interfaces are provided and used.

Currently, an AMOSA service providing the A_ITF functionality is divided into five major interfaces: *General*, *Advertisement*, *Binding*, *Functional*, and *Eventing*, as shown in Figure 17.

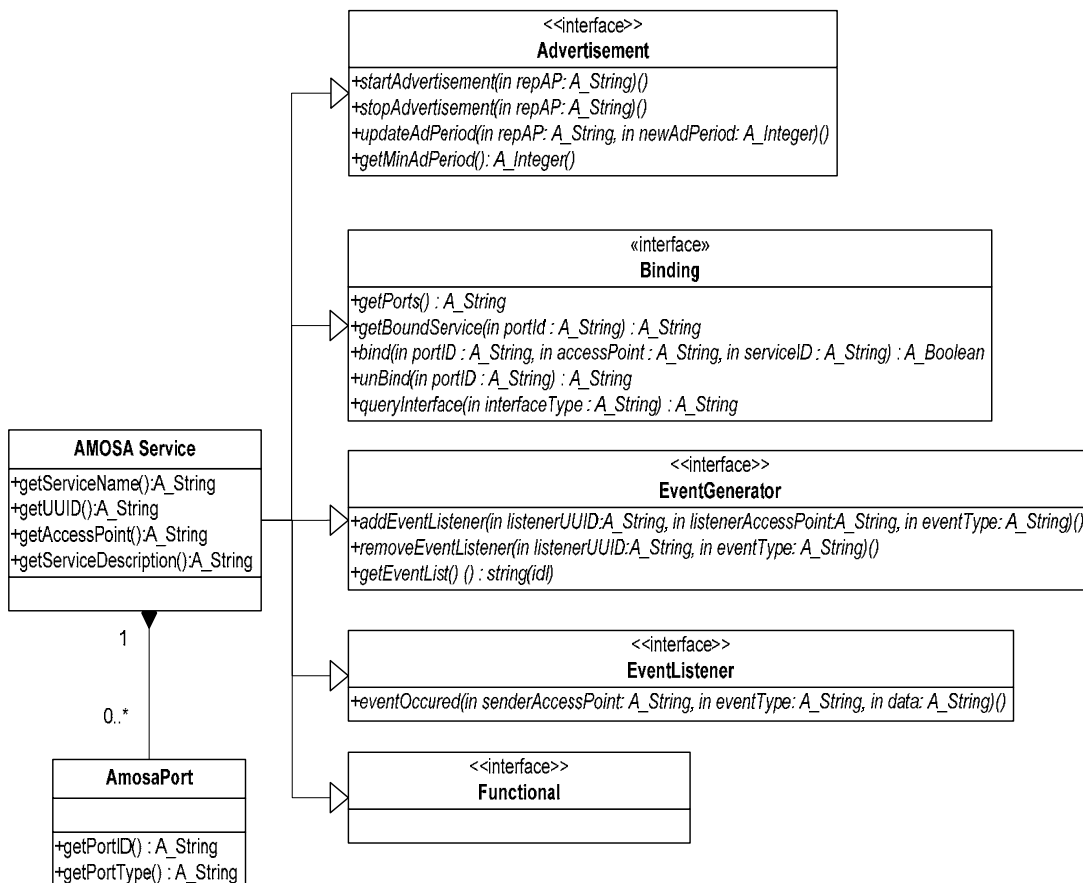


Figure.17. A_ITF Static View

- General port

An AMOSA Service can have one or more *ports*. A port is a *required interface*, which means that it needs to be bound to another AMOSA Service that provides this

interface before it can provide certain functionality. The provided interface can be accessed via an *Access Point*. Ports are characterized by a *Port ID* and a *Port Type*. A port can only be bound to a providing interface with a matching type. This binding functionality is provided by the *Binding* interface. Access points are composed of *Protocol*, *Host*, *Port*, and *Service Name* (for example: protocol: HTTP; Host: 131.155.68.172; Port: 1107; Service name: Calculator).

- Advertisement interface

The *Advertisement* interface is used by a special service, a Repository service. A Repository starts broadcasting advertisement messages in the network when it enters the network or when it restarts working. Using methods provided by the Repository, an AMOSA Service can register and deregister itself to the Repository and an orchestrator can query the repository for other services providing specific interfaces.

An advertisement message contains at least the following items:

- Service Identifier (SID)
- Service Access Point (SAP)
- Service Description (including the provided interface types and required interface types)

- Binding interface

The purpose of the Binding interface is to bind a port (i.e. required interface) of an AMOSA service to another AMOSA Service that provides that interface. In a typical scenario, an orchestrator will first ask a service A for its ports. Per port, it can then ask whether or not that port is already bound to a service. When a port is not bound to a service, it will try to find a service providing that interface (for example by querying the repository). Once it knows a service that provides that interface, it can instruct service A to bind to the service that provides the interface. When the binding is no longer needed, the orchestrator can instruct service A to unbind.

- Functional interface

The *Functional* interface is dynamically created, based on the functionality provided by the original application that implements the service. All published methods of this application (as defined in the service description file), are added to the functional interface.

- EventGenerator interface and EventListener interface

Furthermore, AMOSA Services can subscribe or unsubscribe to the events generated by another AMOSA Service. This functionality is provided by the *EventGenerator* and *EventListener* interfaces. Each event has a particular event type. Event listeners can register to receive only events of a specific type, in order to reduce overhead caused by sending, receiving, and filtering irrelevant messages.

The EventGenerator provides methods to register and deregister event listeners, as well as a method to inform all registered event listeners of the occurrence of an event.

The EventListener interface provides a call-back method that can be called by an event generator once the event listener is registered to occur.

The lifecycle of an AMOSA service consists of the following phases:

1. *Service Advertisement*: when a service enters the network, it has to make its presence known to other services on the network.
2. *Service Discovery*: when a user needs a particular functionality, it sends out a query and then receives a list of available services with required functionalities.
3. *Service Selection*: Select one service per interface that a user requires.
4. *Service Usage*: The actual usage of a service including invocation, binding or eventing.

Assuming a repository service is available on the network, step 1 is performed by all AMOSA services using the Advertisement interface. Step 2 is performed by an orchestrator. During this process, the repository will match the query with the registered service descriptions. Once the orchestrator has a list of available services, it can select (step 3) one of them that provides the required functionality. Finally, the service can be used (step 4) by means of the functional interface and the binding interface, when needed. The service can be called directly at a functional interface by another service or it can be bound to another service by the orchestrator through the binding interface.

3.2.2 A_MAP overview

The A_MAP tool provides automatic generation of AMOSA services adding the standardized parts of the A_ITF interface to the specific functionality defined in the application that implements the service. In addition, it provides the wrapping of these deployment platform independent services into deployable deployment platform specific services. The current version of A_MAP supports publishing functionality of existing Java programs on both UPnP and WebService networks.

The current implementation of the A_MAP tool consists of two separate tools: the *Generator* that performs the generation of AMOSA services, and the *Deployer* that maps an AMOSA service into a deployable service. The design of A_MAP supports binding to an existing application written in a different language or deploying on different deployment platforms. In either case, it is not required to rewrite the entire tool. E.g., when the source language for the AMOSA services is changed from Java to for example C++, the infrastructure of the A_MAP tool can still be used, only the parts that actually generate code have to be changed in order to generate C++ code instead of Java code. These code generating parts of the tool have been separated as much as possible (specific methods in the *Amosa Generator* and a separate code generation class in the *Deployer*).

An overview of the translation process is provided in Figure 18.

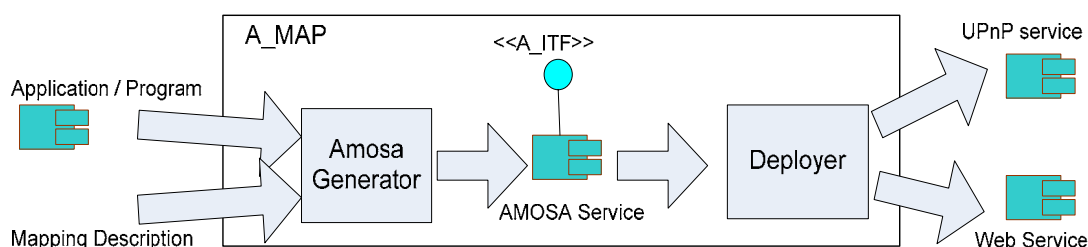


Figure.18. A_MAP Overview

3.2.3 Other functional parts

During deployment, support is needed for the ‘community’ of AMOSA services.

- A proxy / gateway to cast existing UPnP and Web Service interfaces into A_ITF. This is a wrapper to integrate existing UPnP-based services and Web Services into AMOSA service. This allows existing services to be integrated although it is not as simple as that.
- Particular AMOSA Orchestrators including a resource manager, an error monitor, and an orchestration platform. An orchestrator is the component that performs arrangement, coordination, and management of services that make up an application. AMOSA allows late binding by using advertisement, discovery, and run-time binding. Created orchestrators determine at run-time what services are available and how to compose them. The orchestration platform accepts an orchestration program and executes that. The resource manager performs allocation decisions across orchestrations.

3.3 Outlook

For the future, an extension of the A_ITF functionality is foreseen. Some of these extensions might be implemented by all AMOSA services; others might be implemented by specialized services to which services that do not implement this functionality themselves can bind via a port. For instance, virtual community related interfaces, security related interfaces, error detection and recovery related interfaces should be added into each of the AMOSA services. Meanwhile, we should design a resource management service and a recommendation service following the A_ITF specifications, and apply them to the AMOSA platform. The functionality introduction of these components is described briefly below.

Virtual community related interfaces

A virtual community will be formed on top of AMOSA services. This requires that additional virtual community related interfaces are added to services in order to enable automatic and developer-transparent operations. These interfaces should have the functionalities e.g. register services into and deregister services from a virtual community repository, namely *Publish* and *Unpublish* respectively, shown in Figure 19.

Security issue related interfaces

Furthermore, since that secure service discovery and cooperation is one of our major aims to form a virtual community, security related functionalities are needed to be included in the A_ITF. In Figure 19, the *iAccessControl* interface has been designed to implement service access control function for each service. In *iAccessControl*, four actions, which are *EnableAccess*, *DisableAccess*, *Apply4Ticket* and *TicketCheck*, can be bound by other AMOSA services.

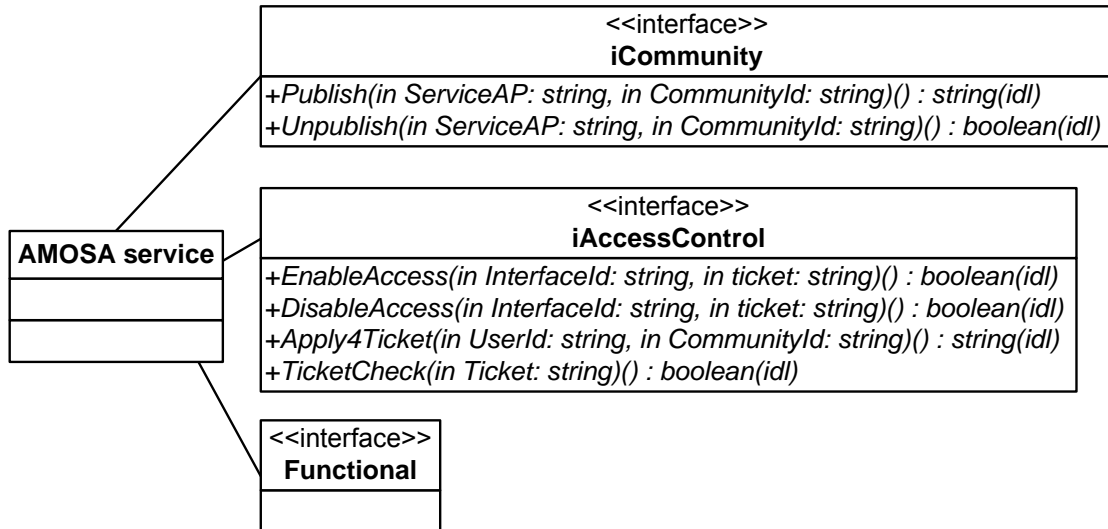


Figure.19. Current Extensions to A_ITF Specification

Resource management service

Since a single device can be a container of multiple services, resource competition and load balancing should be taken into account during service execution because the overall system performance can be affected by limited local resources or an unbalanced network load. Under these circumstances, a resource management service becomes necessary. Via the resource management service, each service on the same device can be reserved. These reservations can also be cancelled if the service has not yet been granted. A resource management service can limit the resource usage that a specific AMOSA service is using on its hosting device. A resource management service can query a service, for example a device manager service, for the resources (total or current amount) that are available on the device on which it is running.

Recommendation service

A good recommendation service should produce trustable and valuable instructions to its user. In our opinion, the recommendation service should explain how recommended programs are related to know programs, and how the recommended user can refine the instructions for more specific need. An AMOSA recommendation service should use refined recommendation policy to calculate the strong ties among virtual community services.

Error detection and recovery related functionalities

In order to enhance the robustness of an application and optimize the overall system performance, automatic error detection and recovery interfaces should be part of the AMOSA services' functionality. With the error detection interface, the availability of a device or a service and a service's run-time status will be monitored, for example by the orchestrator. Furthermore, using the error recovery functionality of a service, for example a device manager service can contain the errors, or restart failed services on one device. This will ensure there is no manually reconfiguration interference and function disruption when errors happen while applications are executing.

4. Conclusions

In this report, we have presented the SOA with virtual communities approach which is designed to solve problems existing in functionality sharing systems. With this approach we extend the notion of SOA with secure service sharing and service cooperation. A functional virtual community overlay has been added to existing SOA services. Services are organized in this abstract virtual community layer to enable secure and privacy sharing and achieve enhanced overall system performance. Services get additional virtual community functionalities with leveraging existing SOA infrastructure. Services' presence and availability is only exposed to authorized community members. Privacy of service providers can be ensured and communication between service users and service providers can be protected as well. Meanwhile, reliable service discovery results can be achieved efficiently through trustable recommendation based on past interactions. Contracted tit-for-tat management mechanism helps to inspire members' sharing initiation and to guarantee better QoS providing.

Research issues around virtual communities including virtual community formation, secure service discovery, access and cooperation, virtual community maintenance, trust and recommendation, etc have been discussed in detail. As a prototype of this SOAVC approach, the so-called AMOSA software platform has been designed and partly implemented. With AMOSA, services developed with different technologies can be discovered, accessed and composed using unique AMOSA-level language without translation or mapping (currently we concern UPnP protocol and web services). Furthermore, services can be combined into virtual communities to provide enhanced overall secure, reliable performance in service discovery and cooperation by using external choreography and orchestration.

References

- [1] Gnutella. <http://gnutella.wego.com/>
- [2] Document Object Model Level 2 Specification Napster. <http://www.napster.com/>
- [3] Kazaa. <http://www.kazaa.com/>
- [4] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In Proc. of SIGCOMM (Aug 2001), ACM, pp.149-160
- [5] Foster, I., Kesselman, C., and Tuecke, S.. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. Journal of Supercomputer Applications and High Performance Computing, 2001. <http://www.globus.org/research/papers/ogsa.pdf>
- [6] SOA. <http://www.service-architecture.com/index.html>
- [7] Ethan Cerami. Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. First Edition February 2002. ISBN: 0-596-00224-6, 304 pages. Publisher: O'Reilly
- [8] UPnP Device Architecture. Version 1.0. UPnP Forum. 2000
- [9] SOAP Version 1.2. Published Specification. <http://www.w3.org/TR/soap/>
- [10] Nigel Shadbolt. Ambient Intelligence. IEEE INTELLIGENT SYSTEMS. JULY/AUGUST 2003. Publisher: the IEEE Computer Society
- [11] H. Kreger. Web Services Conceptual Architecture (WSCA 1.0). 2001 <http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>
- [12] Howard Rheingold. The Virtual Community. [Http://www.rheingold.com/vc/book](http://www.rheingold.com/vc/book)
- [13] Ultima Online. <http://www.uo.com/>
- [14] Google Web site. <http://www.google.com>
- [15] Gnutella. <http://gnutella.wego.com/>
- [16] M. Stokes. Gnutella2 Specifications Part One: http://www.gnutella2.com/gnutella2_search.htm
- [17] B. Zhao, K. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, University of California at Berkeley, April 2001

- [18] A. Rowstron, P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems. Lecture Notes in Computer Science, 2001, Vol. 2218: pp. 329~350
- [19] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In: Proceedings of the IEEE/ACM SIGCOMM 2001, San Diego, Aug. 2001
- [20] Document Object Model Level 2 Specification Napster. <http://www.napster.com/>
- [21] Orkut. <http://www.orkut.com/>
- [22] I.G. Niemegeers and S.M. Heemstra de Root. FEDNETS: Context-Aware Ad-Hoc Network Federations. In Proc. Of Wireless Personal Communications (2005) 33: 305–318
- [23] Fowler, Martin. UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd ed., Addison-Wesley. ISBN 0-321-19368-7
- [24] Simple Service Discovery Protocol Version 1.0. Internet Draft.
http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt
- [25] Multicast DNS. Internet Draft 2006. <http://files.multicastdns.org/draft-cheshire-dnsex-multicastdns.txt>
- [26] DNS-Based Service Discovery. Internet Draft 2006. <http://files.multicastdns.org/draft-cheshire-dnsex-multicastdns.txt>
- [27] Daniel Steinberg and Stuart Cheshire. Zero Configuration Networking: The Definitive Guide. Published by O'Reilly Media
- [28] Service Location Protocol, Version 2. Standards Track document. <http://tools.ietf.org/html/rfc2608>
- [29] Cohen, B. Incentives to Build Robustness in BitTorrent. Proceedings 1st Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, 2003. <http://bittorrent.com/bittorrentecon.pdf>
- [30] XML Key Management Specification (XKMS). Technical Report. 2001.
<http://www.w3.org/TR/xkms/>
- [31] John Evans and Clarence Filstils. Deploying IP and MPLS QoS for Multiservice Networks: Theory and Practice. Published by Morgan Kaufmann, 2007, ISBN 0-12-370549-5
- [32] Quality of Service Networking. Chapter 49. White Book.
http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/qos.pdf