Int. J. Communications, Network and System Sciences, **2012, 5, 50-65**
http://dx.doi.org/10.4236/ijcns.2012.51007 Published Online January 2012 (http://www.SciRP.org/journal/ijcns)

A Generic Service Architecture for Secure Ubiquitous Computing Systems

Shudong Chen, Johan Lukkien, Richard Verhoeven
Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
Email: {shudong.chen, j.j.lukkien, p.h.f.m.verhoeven}@tue.nl

Received May 26, 2011; revised July 15, 2011; accepted August 1, 2011

ABSTRACT

The development of ubiquitous computing systems benefits tremendously from the service-oriented computing concept in seamless interoperation of heterogeneous devices. However, architectures, services interfaces and network implementation of the existing service-oriented systems differ case by case. Furthermore, many systems lack the capability of being applied to resource constrained devices, for example, sensors. Therefore, we propose a standardized approach to present a service to the network and to access a networked service, which can be adopted by arbitrary types of devices. In this approach, services are specified and exposed through a set of standardized interfaces. Moreover, a virtual community concept is introduced to determine a secure boundary within which services can be freely discovered, accessed and composed into applications; a hierarchical management scheme is presented which enables the third party management of services and their underlying resources. In this way, application control logic goes into the network and environment context is dealt with intelligently by the system. A prototype system is developed to validate our ideas. Results show the feasibility of this open distributed system software architecture.

Keywords: Generic Programming; Service Orientation; Ubiquitous Computing; Virtual Community; Context-Aware Resource Management

1. Introduction

Applications based on the concept of Ubiquitous Computing [1] rely on the seamless interoperation of heterogeneous devices including not only powerful computer equipment, but also resource constrained devices, like Personal Digital Assistants (PDA), Consumer Electronic (CE) equipment and small sensors. A successful ubiquitous computing system should enable users to focus on their requirements rather than on computing devices and technical issues. The development of ubiquitous computing systems is tremendously benefiting from the concept of service-oriented computing [2,3]. In service-oriented computing, capabilities of devices are exposed as networked services; applications are achieved through the composition of these services without further dependence on machine architecture, Operating System or language. Typical examples of service oriented applications include media streaming and data sharing and synchronization, but new applications are proposed continuously. Standards like Universal Plug and Play (UPnP) [4] and WSDL [5] are built around the service oriented concepts. The Cloud Computing [6] and Smart Planet [7], and, to a lesser extent, Grid technology [8], address a similar idea at a higher abstraction level and with different focuses.

Lots of research has been done spread over different applied areas of ubiquitous computing systems; no standard approach has resulted however. The architecture, the interfaces of services and the network implementation of these systems are different. Each system uses its own format to describe services and its own communication mechanisms. Additionally, application logic is typically encapsulated in the services and this makes the reusability of these systems rather difficult. Furthermore, many systems totally lack the capability of being applied to resource constrained devices like, for example, sensors.

We believe that a standardized approach to present a service to the network and to access a networked service, which can be adopted by arbitrary types of devices, is of high importance to provide a better interoperability and collaboration of heterogeneous devices and services. Security and privacy issues should be equally important as genericity and flexibility in this approach, since service providers do not want to share their services with just anybody. Secure service discovery and access must therefore be guaranteed. In addition, communication needs protection from being overheard. Moreover, application control logic should not be in the services but in the network. Environment context, like heterogeneity of net-

works, capabilities of devices, error-prone wireless channels and device mobility, should be dealt with intelligently by the system rather than by the users. In our research we have found the following to be the groundwork of the system architecture of ubiquitous computing systems.

- Control is separated from service functionality through *third party binding*. Services are passive and do not know their composition context.
- Services are specified and exposed in a generic way through a set of standardized interfaces, with parts for regular service functionality, external binding and external control.
- This interface definition should be more abstract than the standards (like UPnP and WSDL). We designed such an interface and separate (automated) mappings to UPnP and WSDL.
- A grouping and scoping concept is needed to limit service discovery and access. For this, we introduce the concept of virtual community. In this context a virtual community represents a security boundary that determines a secure service discovery, access and collaboration environment: within this environment, services can be freely discovered, accessed and composed into applications.
- Services and resources must be managed to allow robust applications and to adapt service quality. We designed a hierarchical management scheme that admits third party management. This management admits application oriented end-to-end QoS (Quality of Service) optimization through context-awareness.

In this paper, we propose an open distributed system software architecture which meets these mentioned requirements. The corresponding prototype system is called VICSDA (VIrtual Community-based Secure service Discovery and Access). In VICSDA, services hosted by devices are exposed in a generic way to enable the collaboration over a heterogeneous networking environment. Following component technology [9,10], capabilities of a service are expressed as *provided interfaces*; capabilities a service needs for its work are specified as *required interfaces*. Service compositions are created by connecting (binding) these required and provided interfaces. Special interfaces are used to program the extra-functional aspects including security, management and discovery, but also the mentioned binding. Separate *orchestrators* are capable of searching services and connecting them to form applications; the orchestrators can also expose themselves again as services. In this way, applications can be achieved by service collaboration [2,3] and application control logic goes to the network.

In order to protect services privacy, services can be organized into virtual communities (VCs) and become *community services* by implementing security-related interfaces. Consequently, only authenticated users can access community services. Service discovery, access, and collaboration all happen in the scope of a VC and are managed through a set of community services. Aiming at context-aware ubiquitous computing systems, management-related services and interfaces are implemented, based on a hierarchical resource management schema.

We present the approach in the following way. Section 2 presents the generic way of exposing a service to the network. How the prototype system VICSDA provides a reliable, flexible and scalable service collaboration environment through external orchestration is detailed in Section 3. Section 4 describes a 3D video streaming prototype to validate the feasibility of VICSDA. Finally, we discuss some related work and draw conclusions in Sections 5 and 6, respectively.

2. A Generic Approach to Program a Service

2.1. Lifecycle of a Ubiquitous Computing Application

With respect to the mentioned 5 points in Section 1, the lifecycle of a ubiquitous computing application, depicted in **Figure 1**, consists of the following phases:

2.1.1. Service Development and Deployment

A service provider exposes a set of functions in the form of a generic service to the network.

2.1.2. Service Advertisement

When a service enters the network, it makes its presence known to other services on the network. This can be done using two mechanisms: mediated or immediate. If there is a repository service appears, it sends out the advertisement through e.g. broadcast or multicast. Services who are interested in this repository record this information and register themselves at this repository next. If there is no repository available, services broadcast themselves.

2.1.3. Realize an Application

When there is a requirement of an application, an orchestrator is going to achieve it through service combination following the below steps. Security is an important concern in this step although not shown in the figure.

1) *Service discovery*: based on the situation whether there is a repository service, the orchestrator discovers required services through mediated or immediate way. It sends out service queries and then receives a list of available services with required functionalities.

2) *Service binding*: the orchestrator informs specific services to bind with each other in an order according to the application logic. During this process, the orchestra-

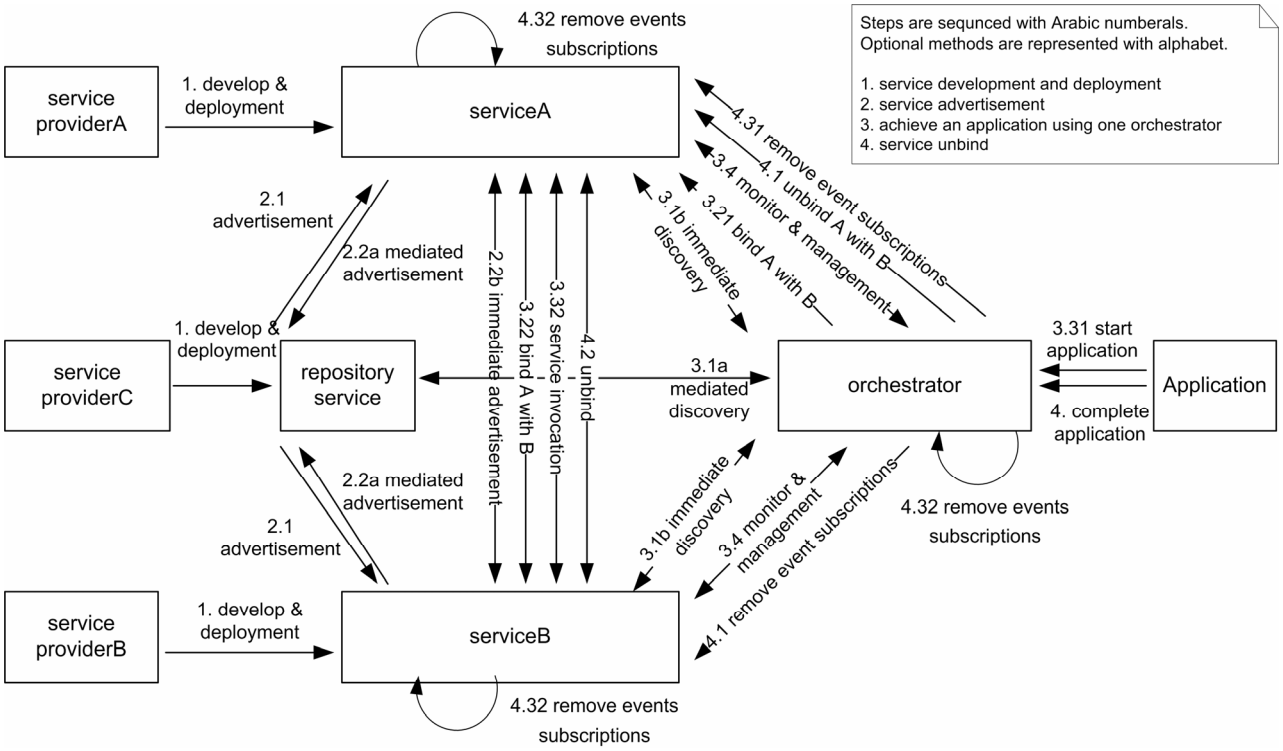

Figure 1. Lifecycle of a ubiquitous computing application.

tor checks the match between services, e.g., the interfaces and the communication protocols.

3) *Service invocation*: this is the stage where a service is connected and invoked after the application starts. A service is called directly by another service. Although control is still done by the orchestrator, as shown in step 4, the orchestrator is no longer involved in service function invocation.

4) *Run time service management*: during the application execution, in order to guarantee the overall performance, e.g., the throughput of a service and robustness of the whole system, services are monitored and managed with respect to their availability and resource usage.

2.1.4. Service Unbind

After the application finishes, the orchestrator orders services to disconnect from their bound services, clean up whatever state needs to be removed and relinquish the control which is implicitly transferred upon binding; this could also include to remove events subscriptions of services that were created for the purpose of the application.

2.2. V_ITF: Generic Service Interfacing

As stated in the lifecycle of a ubiquitous computing application, the first step is to expose a service to the network. In order to cope with the heterogeneous networking environment of ubiquitous computing systems, we propose a generic approach to present a service to the network to allow more easily collaboration between services and devices. In this approach, a service, no matter what type of its host device would be, is composed of a set of interfaces, namely V_ITF, shown in **Figure 2** (left).

Different from the existing Web Services [5] and UPnP specifications, where services only contain interfaces which express services capabilities, this generic approach defines that interfaces of a service can be divided into two types: *provided interface* and *required interface*.

A provided interface shares the same concept with an interface of a Web Service or an UPnP service which is that a provided interface expresses the capabilities of a service and can be accessed via an access point. An example here is a “display” function interface of a multimedia service. Through calling the ‘display’ interface at its access point, this multimedia service can render a specified multimedia stream.

A required interface describes functionality that a service needs to perform; it must be provided at a so-called *port*. Before a service can deliver capabilities described in its provided interfaces, its required interfaces (ports) must be connected to provided interfaces (access points) of other services which provide matching functionality. This is done through a *binding* procedure.

Inspired by the “port” concept defined in WSDL, in V_ITF, a required interface is composed of a *portName*

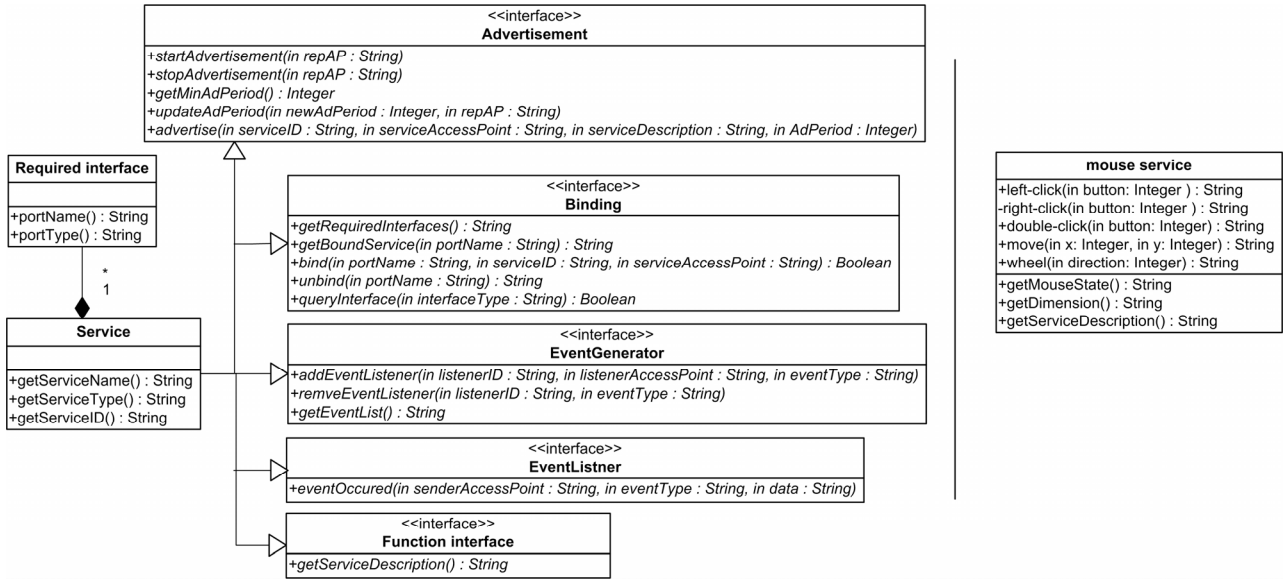

Figure 2. Static view of V_ITF (left) and example function interface of a mouse service (right).

and a *portType*. The portName describes what type of functionality is required. The portType describes abstract information of the potential functionalities of a service with a set of abstract methods and abstract messages involved. For example, the multimedia service has a required interface "control". The portType of this required interface is "mouse" and the portType lists a set of methods should be provided by a provided interface of an other service including *left-click*, *right-click*, etc. A required interface can only be bound to a provided interface with a matching type through the specified PortType. Assume that this multimedia service needs to provide a "control" functionality which enables the multimedia content to be controlled by remote mouse-click events, it can bind to a *mouse* service which provides a matching "mouse" interface with required click functionalities. A required interface cannot be invoked in the same way as invoking a provided interface because it doesn't associate with any access point. A required interface is only able to deliver functionalities, specified by portName, after it has been instantiated. This instantiation is done through the binding mechanism. Through binding, a required interface will be assigned an individual access point (see **Figure 3**).

2.2.1. Binding Interface

The purpose of the binding interface is to bind a required interface of a service to another service that provides that matching function interface. Using the example mouse controllable video streaming scenario, the binding mechanism is shown in **Figure 3**. An orchestrator is used to execute the binding. It firstly queries required interfaces of the multimedia service. When that required interface, *control*, is not bound to any service, it tries to search a mouse service on the network, for example, by querying a repository service. Once it knows the access point of an available mouse service, *mouse*, it instructs the multimedia service to bind to the mouse service through subscribing to *left-click* and *right-click* mouse events. After being bound to a mouse service, the multimedia service is configured to respond to the generated mouse-click events. Subsequently, the multimedia service can provide the video content being controlled by remote mouse events. When the binding is no longer needed, the orchestrator can instruct the multimedia service to unbind.

2.2.2. Function Interface

The *function* interface is created based on the functionality provided by a service. All published methods of a service are added to the function interface. In addition, the function interface allows other services to query the description of a service, which contains the methods that can be invoked on the service, as well as the parameters and return types of these methods. An example functional interface of a mouse service is shown in **Figure 2** (right) where other V_ITF interfaces are omitted.

2.2.3. Advertisement Interface

V_ITF is designed to support two service advertisement mechanisms: the mediated way which is achieved through a repository and the immediate way when there is no repository available on the network.

In an immediate discovery protocol, a service broadcasts advertisements and queries while it listens to messages of broadcasted advertisements and queries. In case a service requires some interfaces from other services, it

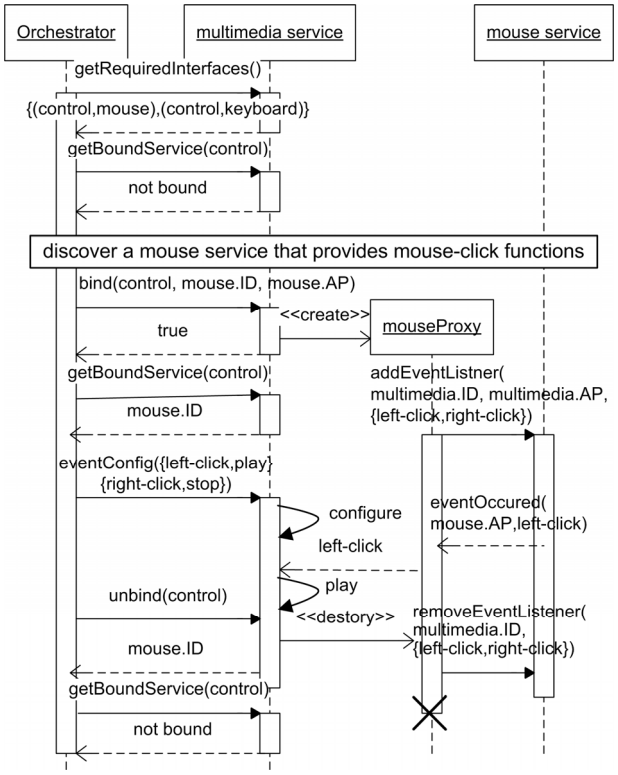

Figure 3. Binding mechanism between a required interface and a provided interface.

keeps a list of these services when it receives the advertisement messages from them. An example here is that a Repository starts periodically sending advertisement messages to all services in the network when it enters the network. Services who receive this advertisement will add it to their lists. Services' *startAdvertisement* method will be called internally to negotiate an advertisement period for service periodic registration at the Repository. With the knowledge of the access point of this Repository, services can un/register themselves to the Repository and can query the Repository for other services providing specific interfaces. This Repository service will be detailed in Section III. The service advertisement protocol is depicted in **Figure 4**. V_ITF will support the transition between these two service advertisement mechanisms.

2.2.4. Event Interface: Event Generator and Event Listener

Furthermore, in order to be aware of the context change of the environment, a service can subscribe or unsubscribe to the events generated by another service. This functionality is provided by the *EventGenerator* and *EventListener* interfaces. EventGenerator interface and EventListener interface interact with each other to achieve eventing. The EventGenerator provides methods to register and unregister event listeners, as well as a method to inform all registered event listeners of the occurrence of an event. The EventListener interface provides a callback method that can be called by an event generator once subscribed events occur.

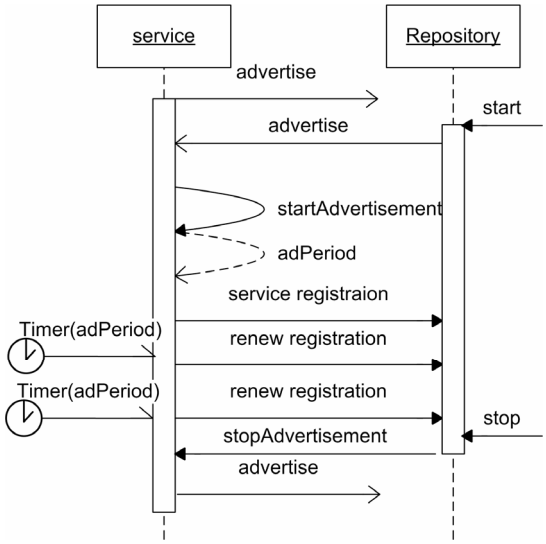

Figure 4. Service advertisement protocol.

From service developers' perspective, this generic V_ITF approach offers the following advantages.

- V_ITF is an abstract specification that allows mapping to concrete deployment platforms. For example, access points of services are composed of protocol, host, port, and service name, e.g., protocol: HTTP; host: 131.155.68.172; port: 1107; service name: display. From this generic access point description, deployment platform specific access points can be created.
- V_ITF is an architecture concept to generalize service expressiveness to the network. Interface implementation of a specific service may or may not align with the boundary of its host device. For example, a service provided by a laptop is capable to implement all V_ITF interfaces. However, a service hosted by a resource constrained sensor can only implement the function interfaces, leaving the storage of its service description to other powerful node on the network. On the other hand, extensions to the V_ITF functionalities shown in **Figure 2** are possible. As foreseen, if a service wants to have access control functionalities security related interfaces can be implemented and integrated.
- V_ITF provides a required interface definition which allows late binding through using advertisement, discovery, and run-time binding. This enables to use external orchestrators to implement applications. They determine at run-time what services are available, how to compose them and what kind of protocols should be used during the communication, in order to make certain functionality available. Therefore the application logic goes to the network through these required inter-

faces and the behavior of services and the underlying resources can be managed via the network.

2.3. V_MAP: Service Deployment Tool

V_ITF is designed deployment platform independent. It is possible to wrap a V_ITF service into a deployment platform specific service without requiring any platform specifics. We provide this wrap tooling support by designing the V_MAP, which is a tool that wraps a deployment platform independent V_ITF service into a deployment platform specific service. UPnP and Web Service are two supported deployment platforms currently. However, more deployment platforms can be supported by extending V_MAP. The structure of V_MAP is depicted in **Figure 5**.

On a high level, V_MAP takes a mapping description (*i.e*. an XML file containing a description of a V_ITF service) as input and it can produce UPnP and Web Service services as output. These services provide the functionality that is described in the mapping description, as well as the functionalities defined in V_ITF. The wrapping is done by the *Deployer*. Because this wrapping is inherently deployment platform (and stack) specific, each deployment platform needs its own Deployer. As a consequence, two Deployers have been created, one for UPnP and one for WebService. Each Deployer reads a V_ITF service and modifies the source code in order to transform it into a deployable service. If new deployment platform support is needed, a new Deployer will be created.

From service developers' point of view, V_MAP tool is simple to use. Given an existing application and a mapping description file that describes what functionality of the application should be exposed to the network, V_MAP automatically generates deployable services. Compared to the way making application functionality available on the network manually, V_MAP can dramatically save developers' time.

3. VICSDA: The Prototyping System of V_ITF

Next to genericity and flexibility of V_ITF, security and maintaining control of shared services and resources over the network should be of high importance in ubiquitous computing applications. From a service user's perspective, how can he locate and access a service securely? During the application execution, how to prevent the logic chain from being interrupted? From a service provider's perspective, how to protect his ownership and privacy of his services? For instance, in a video streaming scenario, it is needed to assure that an orchestrator only binds a video source service to trusted video sink services instead of to malicious ones.

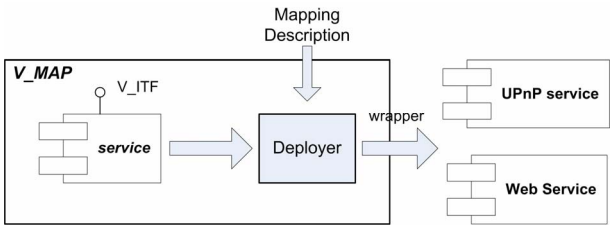

Figure 5. V_MAP overview.

For the sake of security and controllability, we present VICSDA [12,13], the prototyping system of V_ITF with security and management extensions. VICSDA forms a service-oriented virtual community (VC) overlay where services are wrapped as community services, service activities are done based on community membership, and behaviors of all the community members and services are monitored and managed. In this section the design of VICSDA including how to form and maintain a VC, and how to achieve applications with guaranteed QoS using external orchestration will be presented.

3.1. Virtual Community Extensions to V_ITF

A VC is a dynamic contract-based aggregation whose members have commonalities and interact via shared services by means of a digital network like the Internet. It has rules that each member has to follow. It provides services to members and it has the potential to develop applications through external service orchestration. It monitors the environment of a running application and aims at providing optimized QoS to users. In the rest of this paper, when we discuss VCs we are not referring to any aggregation of people, but to the communication among them which is done through the collaboration of services hosted by digital devices.

A user can apply to join a VC and become a member. Members should obey the contracted management policy of that VC, for instance, they should trust each other and provide promised QoS. Their behavior associated to their reputation is monitored to guarantee a contracted QoS provision. In case of misbehavior, they pay some form of penalty. Each member is autonomous which means he has the right to determine what services that he owns can be shared and which member can access his shared services. And he is free to decide to deregister from a VC at anytime.

If a service provider becomes a VC member, he can publish his services into this VC. Plain services, which implement the V_ITF specifications, will consequently become VC services and be shared among members. This VC service registration is preceded by adding VC related functionalities to the plain services. **Figure 6** shows these extended VC related interfaces added to the original V_ITF design in order to achieve service access

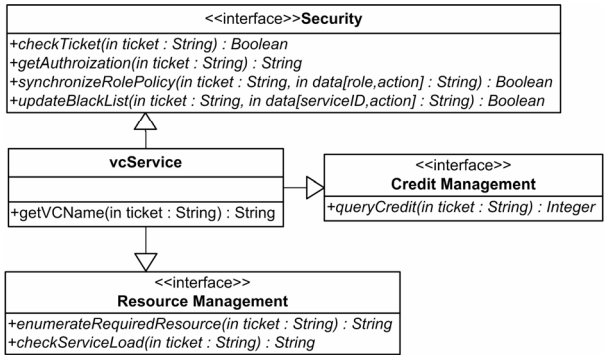

Figure 6. V_ITF with virtual community related functionalities.

control and management functionalities.

3.1.1. Security Interface

The *security* interface is used for checking the identification and capability of a service user, which can be fetched from a *ticket*, to access a VC service. All activities including advertisement, discovery, access and collaboration are executed within the scope of a VC. Only authenticated users can access a VC service and communication messages are encrypted. Due to the fact that services are autonomous, VC service providers can define their services' local access control policy through defining capabilities of different member roles and blocking malicious service users by editing a black list.

3.1.2. Resource Management Interface

The *resource management* interface is designed to deal with the dynamic virtual community environment, where availability of a service, processing capability of a service host device, and network bandwidth can change anytime. This interface provides methods for checking the load of a service and monitoring the real time resource usage of a service, like memory, CPU, or battery power. With this information, VICSDA can be aware of the change of an application's running environment and execute dynamic resource management.

3.1.3. Credit Management Interface

In order to govern VC members' behavior and facilitate the maintenance of a VC, for example maintain the member list, the service list and a better QoS guarantee for service cooperation, each VC service is evaluated with a *credit* value. This value will be periodically checked by the special services that make up the VC. When a service's credit is below a predefined threshold value, that service will be deregistered from this VC compulsively.

To summarize, a VC can be envisaged as an overlay network for the existing services. In this overlay, a service has enhanced functionalities:

- It can filter access requests using its access control policy and all the exchanged messages are encrypted.
- Service activities including advertisement, discovery, access and collaboration are executed based on VC membership.
- It also supports the use of fine-grained VC control policies while leaving ultimate control of the local access to services at service providers.
- Meanwhile, a community service still has the properties of a plain service: all network interfaces still use the Service Oriented Architecture (SOA) [2,3,14] service interface; SOA standards such as Simple Object Access Protocol (SOAP) [15] and Hypertext Transfer Protocol (HTTP) [16] protocols are still suitable for service collaboration.

3.2. Virtual Community Based Secure Service Discovery and Access Control

Basic functionalities of a VC, for example, member de/registration, service de/registration, authentication and encryption, service collaboration and fault recovery are provided by specialized services depicted in **Figure 7**.

A VC is composed of several fundamental services: a *VCEntry* service with a *JoinPolicy* and a *vcBlackList*; a *CertificateMan* service with a *RoleInfo* listing designed roles of this VC; a *Repository* service which is assigned as this VC's repository; an *Orchestrator* service to discover and compose services; a *DevMan* service running on each device with storing a device's services in a *serviceList* and monitoring underlying resource usage using a *soft-stateTable*; a *ResourceMan* service serving as a resource scheduling and decision making engine; and a *FaultRec* service designed for the robustness of a VC. *VCEntry* and *CertificateMan* are accessible to any parties including requesters on the network and internal VC members, while the other services are only available for VC members. For example, *Repository*, *DevMan* and *ResourceMan* can only be invoked by authenticated users.

3.2.1. Member Registration

A service provider needs to become a member before it registers services into a Repository. When a new VC is formed, a VCEntry service is created which is an entry service for users on the network to access this VC. The first action is the member registration. With the knowledge of this new VCEntry service's access point (for example, obtained from broadcast messages over the network), users who intend to join this VC can access this service to become a member. *JoinPolicy* is a description of the agreed community joining policy. An example for a joining policy is that only those who supply personal information including IP address and e-mail ad-

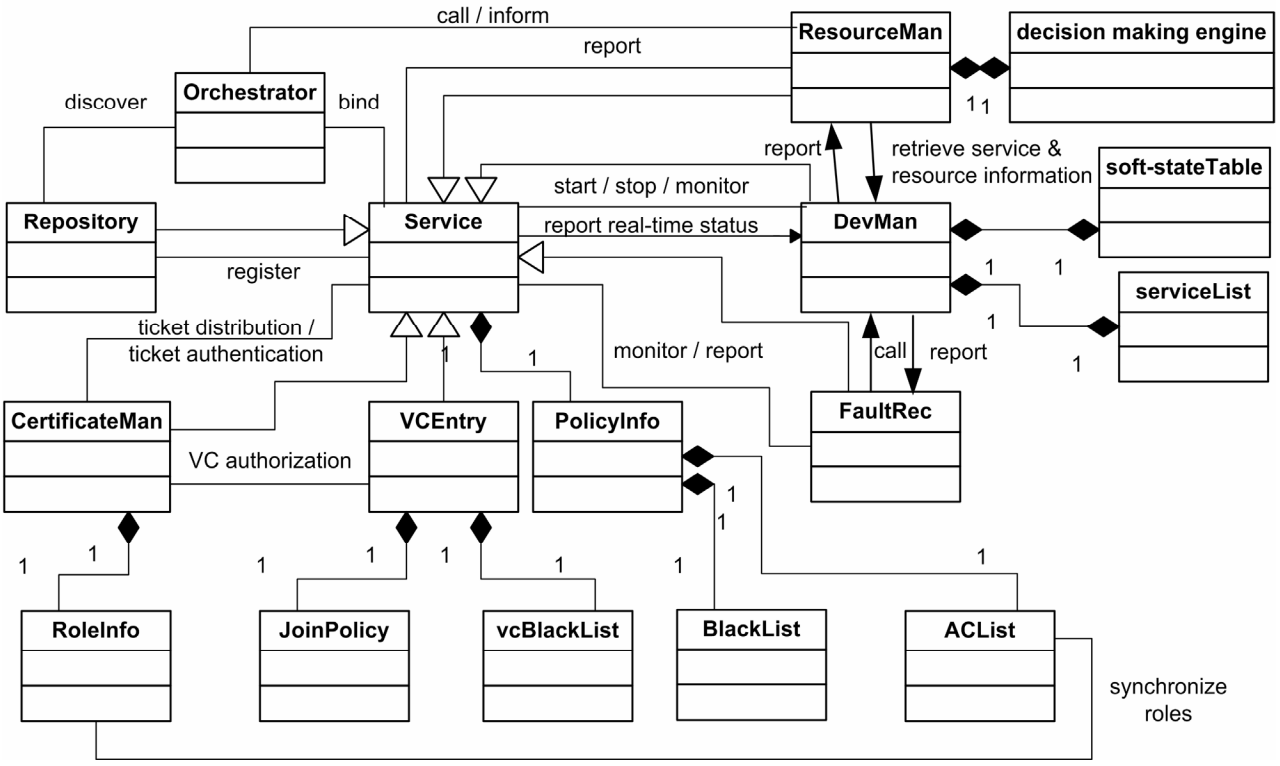

Figure 7. Logical architecture of VICSDA.

dress can be approved to register as a member. When one meets the community joining policy, one will be authorized to be a community member.

A member is granted with specific roles and a certificate (ticket) signed by a CertificateMan. The new member could be a service user to access other services, or a service provider to register/deregister a service, or a combination of these example roles. A member can execute actions corresponding to his role while carrying a valid ticket. **Figure 8** depicts this member registration process.

3.2.2. Service Registration

In VICSDA, we use a Repository service to cache service registrations, receive service discovery queries, and perform matching between queries and registrations. **Figure 9** illustrates the service registration design.

As a member, a service provider can register services into the Repository. His ticket to access the Repository will first be checked. With a valid ticket, services can be added to the Repository. When a service becomes inactive, its registration should be accordingly changed. Soft state registration is used to keep the consistency between a service and its registration at the Repository. Registrations have a specific validity period after which they need to be renewed [17,18,38]. This period is negotiated by a service and the Repository based on what they are capable of handling. When registering, a service specifies the minimum period, *serviceAdPeriod*, it is willing to handle and the Repository responds with the actual period, *adPeriod*, to be used. This allows a trade-off to be made between registrations freshness versus renewal overhead. Typically both services and Repository will increase the period when under heavy load and decrease the period when there are enough spare resources. The registration update process is triggered by a timer with *adPeriod* as the period. A service sends the renewal message to the Repository and the Repository updates it registrations.

Services are autonomous. They are free to define their own access control policy. With respect to this feature, VICSDA is designed to support the use of fine-grained VC control policies while leaving ultimate control of the local access to services at service providers. A community service is added an *ACList* and a *BlackList* as community properties. Service providers can specify each service's local access control policy by defining capabilities of different roles and blocking malicious service users by editing the BlackList.

3.2.3. Secure Service Access

Access to a community service is restricted to authorized members. Different access actions are granted to different members according to their registered roles. A service user is required to provide a valid ticket to call functions at a service. This ticket shows which VC this user

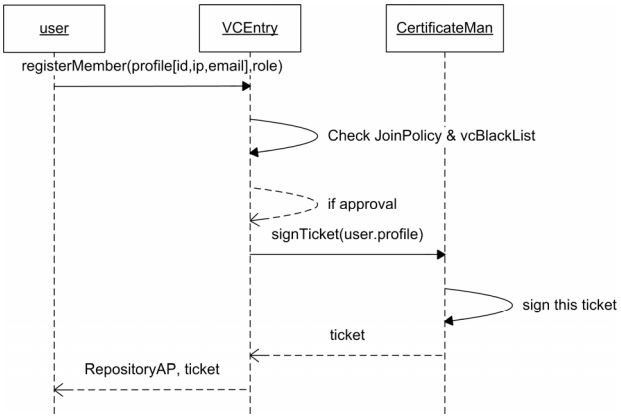

Figure 8. VC member registration.

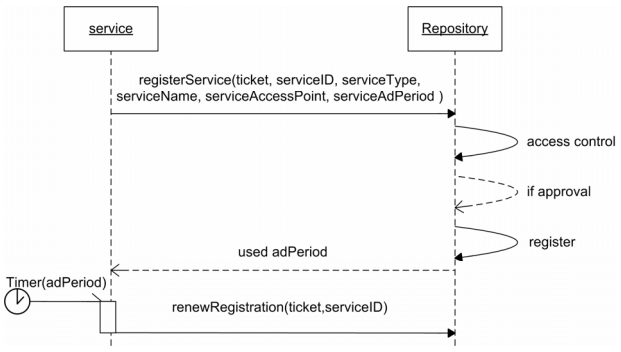

Figure 9. VC service registration.

belongs to and what kind of roles he can play there. A service first validates this ticket and then grants capabilities according to that user's roles. Using the authentication and authorization of a VC, un-trusted or malicious access requests to services can be filtered. **Figure 10** shows the design of this access control approach, which applies to all community activities.

One of the main goals of forming a VC is securing the interactions and keeping privacy of service providers. We design that all exchanged messages in a VC are encrypted and transferred in a secure channel set up between senders and receivers. Message receivers have to decrypt received messages before they can use them. Asymmetric cryptography [19] is used for communication secrecy. The cryptographic key pair for encryption and decryption is distributed by the CertificateMan.

3.2.4. External Orchestration

In VICSDA, service combination is done by external orchestration, an *Orchestrator*, which binds required services at runtime. The focus of forming a VC is to enlarge the notion of external orchestration with a secure enriched service collaboration and composition. The current VICSDA platform can provide the ability to compose services into applications within virtual community boundaries.

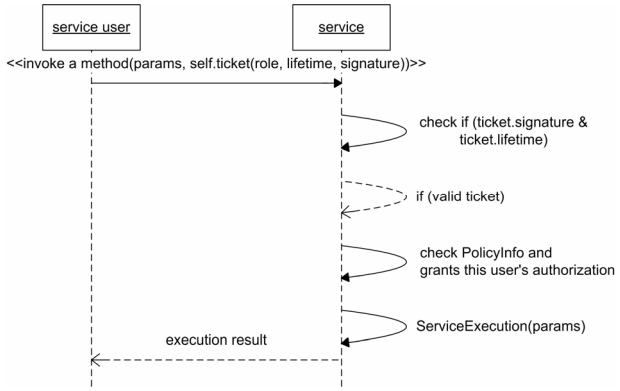

Figure 10. Secure service access in VICSDA.

3.3. Context-Aware Resource Management Based on Hierarchical Monitoring

We have accomplished building applications through external service orchestration with open source code available at [20]. Most of these examples are multimedia applications, for instance video streaming over heterogeneous networks where the environment is inherently dynamic. The availability of required services, resources of underlying devices, or the network bandwidth can change at any time. Without proper management the QoS of this type of application is unacceptable. This requires VICSDA to provide adaptive resource management functionality, which can adapt the application QoS to the actual state of the environment. Aiming at this, VICSDA takes sophisticated QoS monitoring and prediction actions and provides dynamic adaptation in order to satisfy the expected end-to-end QoS specified by end-users. Additionally, VICSDA can dynamically redistribute reserved resources within application activities to meet applications' requirements when unexpected perturbations lead to resource scarcity.

3.3.1. Obtaining Resources Controllability

In principle, in order to execute resource management a manner to manage services and resources which belong to different providers is required. Therefore, a device management service, *DevMan*, is implemented [21] on each device. It can activate and deactivate services hosted by that device, register these services into VCs, and manage the amount of resources used by that service in its registered VC. Only with this enabled full control of services and underlying resources, can the QoS of distributed services be precisely estimated. Also, via the DevMan, service performance adaptations can be executed.

Two components, a *serviceList* and a *soft-stateTable* are designed to maintain the service information of a device, where static information of all active and inactive services are kept in the serviceList while the softstate Table caches the dynamic resources usage by services

invocations coming from VCs in order to deal with simultaneous invocations. The structure of the service- List is shown below.

```
[{serviceName, serviceType, serviceID,
  vcs[{vcName, serviceAccessPoint,
       processID, adPeriod,
    resources[{resourceID,maxAmount}]}]}]
```

A service can be registered into multiple VCs by the DevMan service of its hosting device. Using this serviceList, DevMan has the knowledge about which VC a service has registered (*vcName*), which process is running as a service instance for a VC (*processID*), how much resources can be used maximally by that process. If a service is not deployed or is deactivated, values of the vcs entry will be set to null. Details about the update mechanism will be addressed later in this section.

The structure of the *soft-stateTable* is represented as below.

```
[{resourceID, vcs[{vcName, maxAmount}],
  tasks [{taskID, taskType, serviceID,
          vcName, currAmount, flag}]}]
```

resourceID represents different types of resources, e.g. CPU, physical memory, virtual memory; *vcs* describes the maximum amount of resources a device can use for each VC. Each tasks entry expresses that a type of resource is allocated to which task (*taskID*), the type of this task (*taskType*), serving for which service (*serviceID*) in which VC (*vcName*) and current usage amount (*currAmount*). Tasks, viz. processes in multitasking operating systems, for service invocations, are independent and compete for resources. Possible status of a task can be *running*, *ready*, *blocked*, *completed*, *failed* and *cancelled*. Correspondingly, the resources they are competing for have different status (represented by *flag*) including *allocated*, *reserved*, *released*.

With the information of dynamic resources utilization by service in VCs logged in the soft-stateTable, DevMan can manage the resource reservation, scheduling, and even dynamic reallocation. For instance, when a device is heavily loaded, the DevMan can suspend a task which is handling a resource consuming service to release some amount of resources, or deactivate a service to withdraw all reserved resources when resources are excessively used by it. This helps the DevMan to prevent excessive resource use by one service which would result in a low overall performance of a device. Also important is that DevMan is a service can thus be accessed from outside the device, admitting a global management schema.

Service registrations and resource usage amount are soft-state data and have limited validity periods. In order to acquire a high accuracy of the observation and management, these data should be kept as fresh as possible. In VICSDA, the update process is triggered by a timer with *adPeriod*, which is used for service registration, as the period. Upon update, the DevMan observes the process where a VC service instance is running and then updates the registrations of the serviceList and that of the soft-stateTable. Besides the periodic update, the soft-stateTable can also be updated by events, for in stance, when the DevMan updates a *tasks* entry's *flag* based on a scheduling decision, the entire soft-stateTable will be updated as well.

3.3.2. Hierarchical Monitoring Architecture

When we are discussing resource management, we assume that the Orchestrator has discovered all required services from the Repository and it needs to know the best composition and collaboration between services in order to provide a high application QoS.

In service oriented applications, the end-to-end QoS delivery from the service provider to the service user may span over different types of networks and can be therefore divided into the node level, network level and application level. Devices that host services belong to the node level. At the network level, the capability of the delivery channel is one of the crucial factors which affect the applications performance. End users specify their application-oriented QoS requirements at the application level.

With respect to meeting the expected end-to-end QoS, VICSDA takes hierarchical monitoring and prediction actions [39] as depicted in **Figure 11**. This monitoring is achieved by two distinct services, the *ResourceMan* and the *DevMan*. They function at the node level and the network level respectively. There is one central ResourceMan service running while each device runs its individual DevMan service.

At the node level, devices are monitored to manage their local resources for handling invocations to services, which in turn can provide expected QoS. The DevMan monitors each hosted service by retrieving statistics from the serviceList and the soft-stateTable. With the current resource utilization information, the DevMan can manage the resource allocation to services in order to provide a high overall performance of the device. For instance, it can deactivate some services which are not serving any applications to release resources for another resource consuming service invocation.

Network capacity is crucial to service oriented applications. Especially in time-sensitive multimedia applications, packet loss and jitter will severely impact the video arrival rate at the destination service and the perceived QoS by end users. Therefore, at the network level, the capacity of the underlying networks is probed as another input parameter provided to the ResourceMan service to make resource management decisions and adaptations. We use existing work on network performance probing [22,23] to measure the bandwidth of the delivery channel

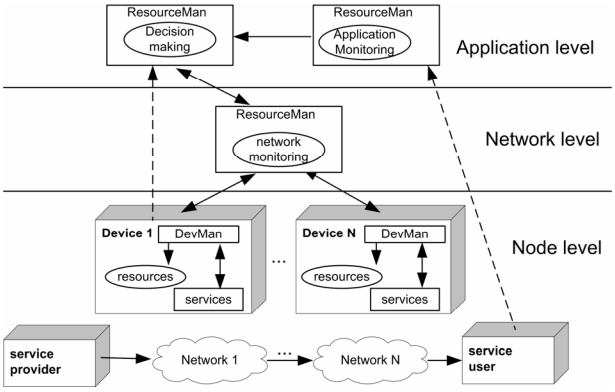

Figure 11. Cross-layer QoS monitoring architecture.

and anticipate network latency.

The functionality of the ResourceMan service at the application level is to process the gathered statistics from lower levels and to make service coordination decisions according to the required application QoS specified by end users. Additionally, it would be useful when service users report the delivered QoS, such that the service coordination decisions can be more efficient. Therefore, the ResourceMan is designed to also be able to receive feedback information at the application level. The mapping between the application-oriented QoS requirements into the performance-oriented resource metrics is provided by VICSDA [24].

The decision making engine of the ResourceMan service will make service coordination decisions which include the compatibility of a pair of services and the capability of a service to execute a specific task. Service compatibility means the matching between supported protocols of different services. Using the video streaming application as an example, the ResourceMan needs to check the compatibility between the encoding protocol of a video streaming service and the decoding protocol of a video display service. These protocols are part of the service properties and can be fetched from the service description by the ResourceMan. We have defined the schema of a VC service description to store this static information of a service [25], such as, required memory to run this service, its supported protocols, etc.

With the estimated network condition and observed resource usage of the required services through the hierarchical monitoring architecture, the ResourceMan can then perform a schedulability test to estimate the capability of a device to execute a specific task of a service, for example, in the video streaming application, whether a display service can decode received bit streams within a given time constraint. In order to avoid the high kernel load caused by launching this video decoding task to that display service, the test is executed at a soft state level instead of directly at an operating system kernel. For this, the ResourceMan uses the current scheduling algorithm of the underlying node to test the schedulability of a task. For example, the host node of the display service is using the Earliest-Deadline-First (EDF) [26] as the scheduling algorithm to manage the principle resource CPU. With the knowledge of the frame rate of a video, the CPU requirement to decode it, and the current CPU usage, the ResourceMan can perform the schedulability test of the display service. Operating system interrupts are ignored in the test.

Moreover, in order to make the ResourceMan be aware of changes within the running environment of an application and to make adaptations accordingly, the publish/subscribe scheme of V_ITF is used [27]. Services publish specific events according to the requirements of different applications. Other services can subscribe to their interested events. For instance, a video display service publishes a *resource scarcity* event which will be generated when it cannot process arrived video within a given time constraint. The ResourceMan can subscribe to this event and be subsequently asynchronously notified when this event occurs. Subsequently, the service coordination adaptation will be triggered.

4. Prototype

We developed a prototyping system VICSDA and on top of it we built an interactive free view point 3D video streaming demo to address the feasibility of VICSDA. This application demonstrates within a VC, a 3D video that is interpolated between four cameras and rendered either on a high performance PC or a PDA. During the rendering an end user can select an arbitrary view point by moving or clicking the mouse which controls the display. The displayed content will automatically adjust to that view point. Moreover, users' intention to change the display is tracked. The video can be redirected to another display when that display's controlling mouse is double clicked. The mouse-s, video streaming and display services are available as services and connected by an orchestrator. The encoding and delivery protocol of the video is adaptive to the capacity of the delivery channel and of the display device. For instance, when the video is redirected from the PC to the PDA, the resolution of the video will be decreased from 800 × 600 pixels to 320 × 240 pixels. The video streams will be truncated from a 3D video format to a 2D video format to assure the resource-constrained PDA can decode all received frames in time and provide the end user a desired perceived QoS.

Using this demo, ideas of V_ITF and VICSDA have been examined including: 1) services are developed using the generic service interfacing V_ITF; 2) services can be deployed using different standards on different platforms; 3) control is done separately by a third party; 4) service

discovery and access is limited within the scope of a VC for ownership protection; and 5) context-aware resource management is achieved through hierarchical monitoring.

Figure 12 is a snapshot of the virtual community monitoring tool. A video showing the prototype and the 3D video streaming demo is available from http://www.win.tue.nl/san/amosa/download.php.

For the realization, a 3D video streaming software [28], which encodes and compresses the 3D video, has been wrapped as a 3D video streaming service and is registered at the Repository. First, the Orchestrator discovers the required services from the Repository: the 3D video streaming service, the display services, and the controlling mouse services. Next, it binds them together based on the service coordination decision made by the ResourceMan. The ResourceMan checks the availability of required services (active states, capability of a service user), the compatibility between them, and provided performance (capability to render the 3D video). Later, during the streaming, the ResourceMan monitors the capability of the display service and the delivery channel and makes service coordination adaptations if necessary. Consequently, the 3D video streaming service will adapt its encoding and delivery protocol for the next frame.

The following digital devices are used: four cameras (to shoot a 5 minutes long video from different view point), two intermediate PCs (to generate the 3D video, and to host the 3D video streaming service), a PC (host of one display service), a PDA (host of another display service), and two control devices (one PC mouse and one PDA touch screen). The physical deployment and the system deployment of this application are given in **Figures 13** and **14** respectively.

With the satisfactory video rendering as a concrete result which were shown as that only an authenticated user can access VC services and the encoding and delivery protocol of the video can adapt to the capacities of services, we conclude that the V_ITF design is feasible, the concept of VC and corresponding prototyping system VICSDA is efficient, and the hierarchical monitoring architecture is doable.

5. Related Work

We compare our generic service programming approach

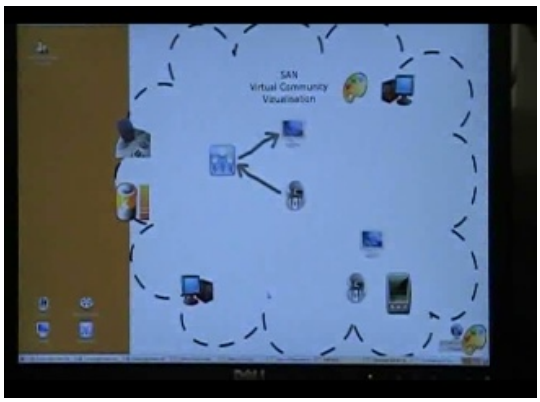

Figure 12. Snapshot of the virtual community monitoring tool.

with several universal service programming methods, which spread over Web Services and UPnP specifications for service oriented applications, and OSAS designed for sensor networks.

Web Services are application programming interfaces (API) that can be accessed over the Internet via the HTTP protocol, and executed on a remote system hosting the requested services. Web Service is designed to support interoperable machine-to-machine interaction over a network. However, it is sometimes criticized for not being loosely coupled, because it is often implemented by mapping services directly to language-specific functions and by coupling application logic into services themselves. Many system developers feel this decreases the reusability of services. Different from Web Services, V_ITF separates services binding from service functionality much more explicitly. Moreover, Web Service implementation is commonly heavy-weight. In contrast, V_ITF is capable to be adopted by arbitrary types of devices including resource constrained sensors.

UPnP is an emerging standard for consumer electronics promoted by the UPnP Forum. It is technology for dynamically attaching devices directly to a computer. UPnP devices are "plug-and-play" in that when connected to a network they automatically announce their network address and supported device and services types, enabling clients that recognize those types to immediately begin using the device. Unfortunately, many UPnP device implementations lack authentication mechanisms while the available security protocols are complex,

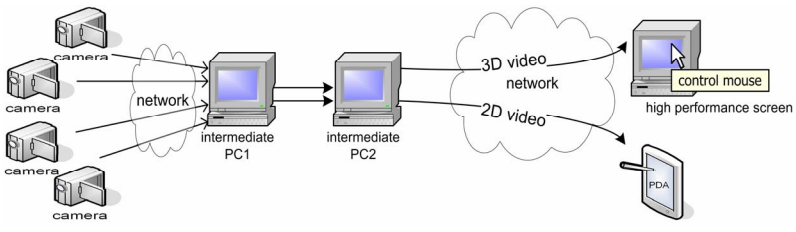

Figure 13. Physical deployment of the 3D video streaming prototype.

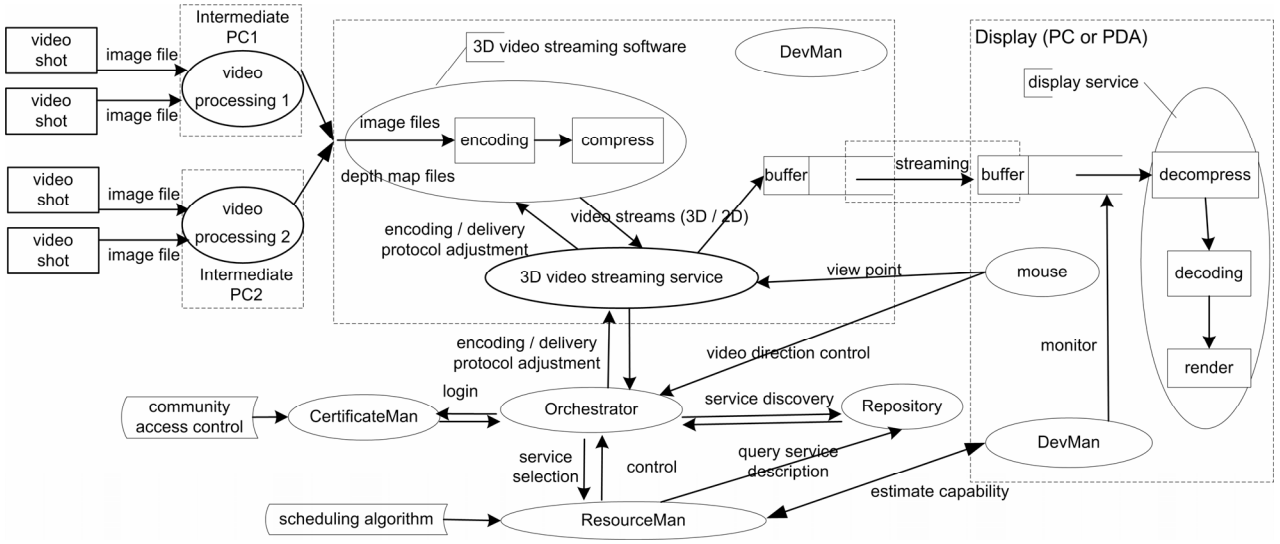

Figure 14. Physical deployment of the 3D video streaming prototype.

and by default assume local systems and their users are completely trustworthy. From this point of view, V_ITF provides services with better security characteristics by defining their lightweight access control policies.

Another distinct aspect between V_ITF and the above mentioned service oriented specifications is the third party binding. In most existing specifications, extra-functional aspects of an application, like service availability, performance and security, are tangled with functionalities of required services. Although web service orchestration and choreography [29] allows the reusability of services, the dynamic service discovery and the real time adaption are still a puzzle. V_ITF enables this separation through external orchestration which takes care of the service discovery, service binding and security concerns. Through the generic resource management interface, the dynamic adaption is solved. Moreover, an orchestrator can be represented as a service, just like a workflow service.

OSAS (Open Service Architecture for Sensors) [11] addresses a similar idea as V_ITF however focusing at sensor networks. It is an event-based programming system. The OSAS system provides the notion of services (consisting of actions and events) and subscription (that bind events to actions). Basic services provided by the OSAS subsystem include an bytecode interpreter, flooding and code upload. All subsequent functionality is built from these primitives. We can regard OSAS as a special version of V_ITF in the sensor networks domain. OSAS services only implement part of the interfaces defined in V_ITF, such as function and advertisement interfaces. Other functionality, e.g., service description could be built on top of these.

In the field of collaborative computing, there have already been some virtual community research activities. The virtual community concept in peer-to-peer (P2P) systems, for example BitTorrent [31] and Tribler [32] focus on the P2P overlay network. 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. The objective of BitTorrent is that through virtual community formation the performance of the whole community is improved. Tribler, based on social phenomena such as friendship and trust, can help to automatically build a robust semantic and social overlay on top of BitTorrent and can yield good cooperative downloading performance with respect to existing solutions. However, the anonymity in this P2P community formation limits the scope of applications. Although a potential use of P2P networks is to share video streams in real time [33], the main purpose of P2P networks is still off-line file sharing. Moreover, secure resource sharing is always an issue in P2P systems.

A Personal Network (PN) [34,35] aims to achieve seamless communication between electronic devices in an ad-hoc fashion. A PN enables a user remote access to any of his personal services and content as if they were physically present in his vicinity. A PN supports applications based on sharing resources and takes context and location information into account. Architectures of PNs, resource discovery, self-organization, routing, and security are addressed. However, PNs mainly address the connectivity issue while the architecture of application building and the particular way of service sharing is not within scope. Also functionalities like context awareness, service discovery and resource management need to be realized.

VICSDA provides secure service sharing within a VC and the denial of access to community services from outside a VC. This relates to the concept of Authorized Domains, as proposed by the Marlin [36] and Coral [37] Digital Rights Management (DRM) based platforms. These platforms are built for sharing multimedia content across multiple devices in an in-home network. Multimedia content is protected using a governance rule specified by users. Protected content issued for a specific domain can be consumed on any device that has joined this domain. These are, in fact, VC related works applied in different fields but without the service orientation concept and leaving the membership withdrawal issue open.

6. Conclusions

This paper presents a service interfacing approach V_ITF, which provides a generic way to expose a service to the network and to access a networked service. In the V_ITF specification, a service is composed of a set of standard interfaces besides the specific functions it can perform. Its interfaces can be divided into provided interfaces and required interfaces. The required interface definitions allow the availability of new functionalities through service binding. The specifically designed binding interface deals with the binding between two types of interfaces. Interface implementation of a specific service may not align with the boundary of its host device. Therefore, V_ITF can be adopted implemented on arbitrary types of devices including resource constrained sensors. V_ITF allows mapping to concrete deployment platforms. A mapping tool, V_MAP, is described, which currently can support wrapping a deployment platform independent V_ITF service into UPnP and Web Service deployment platforms.

In order to maintain access control to shared services over the network, we introduced virtual community concept to services. A VC defines a secure boundary for service activities. Within this boundary, service discovery, access and collaboration are done freely, while services are monitored and managed. To build such a virtual community overlay the flexible V_ITF is extended with security, resource management and credit management interfaces. Regarding the autonomous nature of a service the VC design supports the use of fine-grained VC control policies while allowing service providers defining their local access policy to services. Methods of how to form a VC, how to register as a member and later share services with other community members are introduced subsequently. Services are registered at the Repository through the DevMan service which also performs monitoring and management. Each device has one DevMan service which tracks the dynamic resource usage of hosted services on that device. With the performance statistics of services and of the network gathered by the ResourceMan service, performance of required services can be predicted. The publish/subscribe scheme is used to be aware of changes in the environment. In this way, inherent dynamics of the ubiquitous computing systems can be handled and application-oriented QoS can be guaranteed.

The corresponding prototype system VICSDA is developed and a 3D video streaming application is implemented. Concrete outcome proves the feasibility of the generic service programming and access approach. Secure service discovery and access control have been achieved. The application shows the successful separation of service control from service functionality through the external orchestration. The optimized the displayed video quality over adapting the 3D video's encoding and delivery protocol based on the capacity of the delivery channel and of the display devices showed the effectiveness of the hierarchical monitoring design.

Applying V_ITF into the low capability device domain, such as sensor networks is our current research focus while some of the described functionalities still need implementation (like the immediate service disco- very protocol). In addition we combine V_ITF with a component framework in order to move to a system that allows dynamic software updates while remaining pre- dictable and secure. Strengthened authorization and au- thentication mechanism will be researched in order to provide the secure access to services hosted by sensor nodes. Fault detection and recovery of the system will also be addressed in the future to facilitate enhanced reliability and robustness of VICSDA.

7. Acknowledgements

This work is supported by the research project of Freeband I-Share: Intelligent Middleware for Sharing Resources for Storage, Communication and Processing of Multimedia Data, supported by the Dutch government.

We would like to thank our anonymous reviewers. Their invaluable feedback helped substantially in improving the quality of the paper. Our thanks are also to Remi Bosman, Melissa Tjiong, Goran Petrovic, Peter de Width, Jurjen Middendorp, and Pim Vullers for their technical feedback at the implementation stage of this work.

REFERENCES

[1] M. Weiser, "Some Computer Science Problems in Ubiquitous Computing," *Communications of the ACM*, Vol. 36, No. 7, 1993 pp: 75-84. doi:10.1145/159544.159617

[2] B. Srivastava and J. Koehler, "Web Service Composition —Current Solutions and Open Problems," Online document. http://www.zurich.ibm.com/pdf/ebizz/icaps-ws.pdf

[3] W. Zhang, F. Liu, S. Chen and F. Ma, "Automatic Ser-

vices Composition in the Grid Environments," *Proceedings of 6th International Conference on Computational Science*, Reading, 28-31 May 2006, pp. 1004-1007.

[4] UPnP Forum, "UPnP Device Architecture. Version 1.0," 2000.

[5] E. Cerami, "Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL," O'Reilly Media, Sebastopol, 2002.

[6] P. Watson, P. Lord, F. Gibson, *et al*. "Cloud Computing for E-Science with CARMEN," *Proceedings of the 2nd Iberian Grid Infrastructure* (*IBERGRID*), Porto, 12-14 May 2008.

[7] IBM, "Smarter Planet". http://www.ibm.com/smarterplanet/

[8] S. Chen, W. Zhang and F. Ma, "A Cooperative Computing Platform for Drug Discovery and Design," *Proceedings of the IEEE International Conference on Computational Science*, Shanghai, 15-18 September 2004, pp. 523-526.

[9] Microsoft COM website. http://www.microsoft.com/com/default.mspx

[10] S. Vinoski, "CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments," *IEEE Communications Magazine*, Vol. 14, No. 2, 1997, pp. 46-55.

[11] R. Bosman, J. J. Lukkien and R. Verhoeven, "An Integral Approach to Programming Sensor Networks," *Proceedings of the 6th Annual IEEE Consumer Communications & Networking Conference*, Las Vegas, 10-13 January 2009, pp. 1-5.

[12] S. Chen, J. J. Lukkien and I. Radovanovic, "Freeband I-Share Deliverable 1.5. Service Discovery, Access and Cooperation in Virtual Communities," 2007. http://www.win.tue.nl/san/projects/ishare/D1.5.pdf

[13] S. Chen, J. J. Lukkien, I. Radovanovic, M. Tjiong, R. Bosman and R. Verhoeven, "VICSDA: Using Virtual Communities to Secure Service Discovery and Access," *Proceedings of the Fourth International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, Vancouver, August 2007, pp. 7-13.

[14] SOA. http://www.service-architecture.com/index.html

[15] W3C, "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)," W3C Recommendation, 27 April 2007. http://www.w3.org/TR/soap12-part1/

[16] R. Fielding, J. Gettys, *et al*., "Hypertext Transfer Protocol —HTTP/1.1." http://www.ietf.org/rfc/rfc2616.txt

[17] X. Zhang, M. A. Hiltunen, K. Marzullo and R. D. Schlichting, "Customizable Service State Durability for Service Oriented Architectures," *Sixth European Dependable Computing Conference*, Coimbra, 18-20 October 2006, pp. 119-128. doi:10.1109/EDCC.2006.8

[18] B. C. Ling, E. Kiciman and A. Fox, "Session State: Beyond Soft State," *Proceedings of the Symposium on Networked Systems Design and Implementation*, San Francisco, 29-31 March 2004, pp. 22-22.

[19] R. Housley, W. Ford, W. Polk and D. Solo, "Internet X.509 Public Key Infrastructure: Certificate and CRL Profile." http://www.ietf.org/rfc/rfc3280.txt

[20] "IShare: Sharing Resources in Virtual Communities for Storage, Communications and Processing of Multimedia Data," 2008. http://www.win.tue.nl/san/amosa/ishare/intro.php

[21] S. Chen and J. J. Lukkien, "Obtaining Resource Controllability in Service Cooperation Environments," *Proceedings of the 7th International ACM Conference on Mobile and Ubiquitous Multimedia*, Umea, 3-5 December 2008.

[22] D. Q. Liu and J. Baker, "Streaming Multimedia over Wireless Mesh Networks," *International Journal of Communications, Network and System Sciences*, Vol. 1, No. 2, 2008, pp. 105-206.

[23] C. F. van Antwerpen, "Interface Selection Layer Improving QoS Using Interface Pair Selection," Master's Thesis, Eindhoven University of Technology, Eindhoven, 2005.

[24] S. Chen, J. J. Lukkien, R. Verhoeven, P. Vullers and G. Petrovic, "Context-aware Resource Management for End-to-End QoS Provision in Service Oriented Applications," *Proceedings of Workshop on Service Discovery and Composition in Ubiquitous and Pervasive Environments*, New Orleans, 30 November-4 December 2008, pp. 1-6.

[25] S. Chen, J. J. Lukkien, R. Verhoeven, R. Bosman and M. Tjiong, "I-Share—VICSDA System Design and Prototype. Deliverable 1.16," December 2007. http://www.win.tue.nl/san/projects/ishare/D1.16.pdf

[26] J. A. Stankovic, M. Spuri, K. Ramamritham and G. C. Buttazzo, "Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms," *The Springer International Series in Engineering and Computer Science*, Vol. 460, 1998. doi:10.1007/978-1-4615-5535-3

[27] P. Eugster, P. Felber, *et al*., "The Many Faces of Publish/Subscribe," *ACM Computing Surveys*, Vol. 35, No. 2, 2003, pp. 114-131. doi:10.1145/857076.857078

[28] G. Petrovic and P. H. N. de With, "Near-Future Streaming Framework for 3D-TV Applications," *Proceedings of the IEEE International Conference on Multimedia & Expo*, Toronto, 9-12 July 2006, pp. 1881-1884

[29] C. Peltz, "Web Services Orchestration and Choreography," *Computer*, Vol. 36, No. 10, 2003, pp. 46-52. doi:10.1109/MC.2003.1236471

[30] OSGi Alliance, "OSGi Service Platform Core Specification," Release 4, Version 4.1, April 2007.

[31] B. Cohen, "Incentives to Build Robustness in BitTorrent," *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, Berkeley, 2003, pp. 68-72.

[32] J. A. Pouwelse, P. Garbacki, *et al*., "Tribler: A Social-Based Peer-to-Peer System," *Proceedings of the 5th International Workshop on Peer-to-Peer Systems*, Santa Barbara, February 2006, pp. 127-128.

[33] J. D. Mol, D. H. P. Epema and H. J. Sips, "The Orchard Algorithm: Building Multicast Trees for P2P Video Multicasting without Free-Riding," *IEEE Transactions on Multimedia*, Vol. 9, No. 8, 2007, pp. 1593-1604. doi:10.1109/TMM.2007.907450

[34] I. G. Niemegeers and S. M. Heemstra de Groot, "From Personal Area Networks to Personal Networks: A User Oriented Approach," *Wireless Personal Communications*,

Vol. 22, No. 2, 2002, pp. 175-186.
doi:10.1023/A:1019912421877

[35] F. T. H. den Hartog, M. A. Blom, C. R. Lageweg, *et al*., "First Experiences with Personal Networks as an Enabling Platform for Service Providers," *Proceedings of the Second International Workshop on Personalized Networks*, Philadelphia, 6-10 August 2007, pp. 1-8.

[36] "Marlin Developer Community". http://www.marlin-community.com/

[37] "Coral Consortium Cooperation". http://www.coral-interop.org/

[38] M. Tjiong and J. J. Lukkien, "An Investigation into Soft-State Protocol Parameters," *Proceedings of the* 2008 *Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, 14-17 July 2008.

[39] A. Korostelev, J. Lukkien, J. Nesvadba and Y. Qian, "QoS Management in Distributed Service Oriented Systems," *Proceedings of* 25*th International Multi-Conference Parallel and Distributed Computing and Networks*, Innsbruck, 13-15 February 2007.

