An architecture for self-organization in pervasive systems

Aly. A. Syed\(^1\), Johan Lukkien\(^1\), Roxana Frunza\(^2\)

\(^1\)NXP Corporate Innovation and Technology/Research High Tech. Campus 32, 5656 AE Eindhoven, The Netherlands
\(^2\)Department of Mathematics and Computer Science, Den Dolech 2, 5612 AZ Eindhoven, The Netherlands

Abstract- Pervasive computing environments consist of many independent collaborating electronic devices, including sensors and actuators. Ad-Hoc extensibility of such systems is desirable but the current network technologies use the concept of a central coordinator device in the network or define application profiles which are not easy to extend and maintain. The distributed architecture proposed in this paper allows these devices to organize themselves automatically to execute some pervasive system application without the intervention of a central controlling device. The knowledge that defines interactions between these devices is derived from an ontological model of a particular domain. This knowledge is distributed over the devices such that every device only has information about its own interactions and operations. A simple demonstration of this architecture is presented.

I. INTRODUCTION

Pervasive computing is an emerging trend that makes computers physically available but effectively invisible to users\([1]\). Other names given to this trend are Ubiquitous computing and Ambient Intelligence. The concept was introduced by Weiser in 1991. When primarily concerning the objects involved, it is also called physical computing, the Internet of Things, or haptic computing\([2][3][4][5]\).

Pervasive computing environments consist of many independent devices capable of sensing, actuating, computing and communicating. It is difficult to maintain a pervasive system as a fixed environment. Common practice is that old devices are replaced in a system only when they break down, which is an ad-hoc process. Also mobile devices can enter and leave a particular environment at any time. If the installation and removal effort is large then the old devices may even be left in the environment if the energy usage is not significant. Pervasive computing environments must make these scattered abilities available in a coherent manner in order to realize applications through self-organization of the pervasive network. In existing technologies we find already some approaches to this. In Bluetooth \([13]\) and Zigbee \([9]\) certain cooperation scenarios are standardized as application profiles. Other standards, operating at a higher abstraction level include CORBA \([10]\), OSGi \([14]\), and UPnP \([6]\). Common to these systems are notions of service discovery, service mediation and service coordination. For example, CORBA targets large distributed computer systems and requires an object between a server and a client for mediation of services. UPnP\([6]\) requires a so-called control point that connects a media server and a media renderer. All of these models either fix an application type and thus hinder extensibility or require a third mediating party between a server and a client. In addition these models come with varying overhead that may be prohibitive for low-resource devices. For example, UPnP requires TCP/IP networking and XML parsing.

The mentioned systems work if they remain closed, meaning that the services in the system are known to a control entity. However, if a new device is brought into the network the control entity needs an update. The amount of knowledge that has to be stored into the control entity becomes prohibitively large. This knowledge management problem in combination with the need for explicit updates hinders the deployment of pervasive systems. In practice, connecting the right devices is delegated to the user (e.g. in the current day DLNA standardised AV architecture \([6]\)).

In this paper we present an architectural concept for pervasive systems that enables evolutionary development of a pervasive computing environment as well as flexible deployment of devices and services.

II. PROPOSED ARCHITECTURE

Our proposed architectural model draws on experiences in the human society as we will explain by the analogy of building a house. The person building the house knows that he needs bricks and knows that there is a brick company “ABC bricks”, perhaps seen advertised on a bill board. “ABC bricks” in turn
knows that it needs to have a certain type of sand and also knows that such sand is supplied by a certain other company and so forth. In order to bring the required activities together, links are built based on very simple rules such as who is the cheapest, who delivers better quality and who is on time. This knowledge is stored in many peoples’ brains and not necessarily in a single persons’ brain. Furthermore, the language used for communication is defined by domain experts, in this example the domain of house-building. Thus many people and agents with their specialisations come together to perform the task of building a house and the control of this process is not located in a single entity. Drawing on this example, the architecture as described below is designed such that the knowledge of the system is distributed among the devices who know what they have to do to perform a certain function.

Recently, Ontology based semantic knowledge is being used in world-wide-web applications for dynamic service discovery and invocation e.g [16]. In our architecture as described below, knowledge about device types needed to perform a certain task is extracted from an Ontology and stored in every device. This enables all devices to make their decisions autonomously based on this knowledge. As explained later in section III of this paper, this allows us to make a non-centralized and completely distributed architecture.

Pervasive system architecture
A pervasive system is viewed as a pool of available devices that are scattered physically in an environment. Every device can perform one or more sub-functions in response to a request; these sub-functions are also referred to as services. For this paper, the boundary of such a network is defined by physical reachability. In response to a user request for a particular function, the devices in the system arrange themselves such that they can complement each other if needed to perform the required overall function. We define a set of devices that have arranged themselves as described above to be a virtual device, which can be regarded as an application (see Figure 1). A virtual device can consist of any number of devices providing sub-functions as needed. It is possible for a device to be part of more than one virtual device; the device tracks if it has enough resources to fulfill a request. A virtual device stays in existence for as long as the required function is being performed. When the required function is completed, the virtual device falls apart and the disconnected sub-functions can be reused.

A first request for a particular function can be done by devices that contain the special functionality of a Request Center (RC). Request center functionality is to create a request for a certain function and send it into the network. We envisage human users as well as some intelligent devices in a network to contact such a request center to ask for some functionality.

Device architecture
A device can perform one or more sub-functions for some virtual device. As described in the above analogy, every device only has knowledge about what functions it can perform and which other sub-functions it needs to fulfill its responsibility in a virtual device. We propose to extract the knowledge needed for a device from an ontology, e.g. as defined in [8,11] and to store it in the form of recipes in a knowledgebase in a device at the time of its manufacturing or deployment (see Figure 2).

When devices receive a request for a sub-function, each device uses its knowledgebase to see if it understands that function and knows how to perform it. If so it confirms this to the requester who makes a selection from among the devices respond affirmatively and allows them to join the virtual device. Upon being selected a device sends out requests for performing the remaining sub-functions. In this way requests cascade in a tree-like fashion after which the services implementing the sub-functions are activated. The knowledgebase of a device contains different recipes corresponding to the different requests that it can handle. These recipes identify the sub-function(s) that this device can perform and also what sub-function(s) this device needs to find and connect to.

A device with RC functionality may offer sub-functions to a virtual device as well. RC is also responsible for assigning a unique identification to a virtual device. An RC remains a part of the virtual device throughout its life to receive and translate any subsequent commands from the user for device control functions.

In accord with the above, a device can be in one of the states identified in Figure 3. Let us consider a point in time when no virtual device exists in the network. At that time, all devices are in Initial state and form a pool of devices that can constitute a virtual device. A device changes to the Virtual device setup (VDS) state when it receives a request to perform a sub-function. In this state, a device checks if it knows the sub-function and how to decompose it. If the device can
provide the sub-function it informs the requesting device that it is available.

E&C (see Figure 3). In Initial state, devices do not send any messages.

Figure 5: There are three categories of messages defined, namely Search Message, Control Message, and Destroy Message.

When in VDS state a device can send a search message to the network looking for a particular sub-function that its Knowledgebase identifies. Once a Virtual Device has been constructed and starts execution, all the constituent devices of the Virtual Device are in the E&C state. In this state, devices can broadcast a control message to alter the functionality of a Virtual Device by changing some of the variables of a service (see below). This message contains the identification of the issuing device, the Virtual device ID, the message type i.e Control and a number of control modalities and their corresponding values.

Knowledgebase

Recipes
Recipe a: If request is play and type is MP3 then execute sequence [search sub-function ‘x’, connect, negotiate with next device, perform sub-function ‘x’, send data]

Capabilities
I can collaborate with
- Function Type A: to perform request play music
- Function Type C: to perform request browse music

Rules
On request to join virtual device:
- Choose the first one that answers

Properties
Properties of this device such as UI present or not etc.

Figure 6: An example knowledgebase schematic design.

At a certain point all devices in a virtual device are in the E&C state, and the user request is fulfilled. At that moment, the device that has the RC function in the Virtual Device

The requesting device decides whether it wants this device to participate in the virtual device in formation. It does so by consulting its own knowledgebase which may contain some rules for accepting an offer for participation in a virtual device. If this service offering device receives an acknowledgement it (recursively) requests support for any other sub-functions it needs to provide its own sub-functions. Upon completion it marks itself as part of the virtual device and goes into the Execution and control (E&C) state. Once a device has become a part of a Virtual Device, it is usually not available for any new virtual device requests unless it has the capability to participate in multiple Virtual Devices. In the case that no acknowledgement is received by the service offering device a device will automatically return to the Initial state after some predefined time.

This process of Virtual Device Setup is depicted in Figure 4.

Figure 4: devices setup initiated by a request from the user.

Messages
For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.

For communication between the devices, we assume that the network has broadcast capabilities. The devices communicate through broadcast messages when they are in state VDS or Initial state, as depicted in Figure 4. In Initial state, devices do not send any messages.

Figure 3: State diagram of a device

Message Payload

Message Header

Message Format

Message Type

Message ID

Message Data

Message Action

Message Status

Message Encryption

Figure 2: Message format and data structure.
broadcasts a destroy message with the Virtual Device ID in the network. On receiving this message, the Virtual Device is destroyed and all its constituent devices change to initial state.

Since these messages are broadcast, not the receipt but the interpretation of the message header tells a device if it needs to interpret the payload or not. By interpreting the payload and its own knowledgebase, a device can identify if it can perform the action(s) stored in the payload.

Knowledgebase
The knowledgebase of a device is extracted from an ontology and is augmented by device-specific information stored in the form of “Recipes”, “Capabilities”, “Rules”, and “Properties” as shown in Figure 6. This information is necessary to participate and collaborate with other devices in a network and is the discriminating element in our architecture. The information in the knowledgebase is organized in a way that facilitates its interpretation. Recipes prescribe what a device should do to perform a role in a network of devices. Rules allow a device to make decisions and Capabilities and Properties let it check whether it matches the requested function.

Virtual device control
When a virtual device is setup, all devices that make up this virtual device interpret control messages. When a control message is received by devices that are part of a Virtual Device, each of these devices checks its knowledgebase to see if it is responsible for regulating one of the control modalities in the message and, if so, it exercises this control. For example, the device that is providing audio functionality knows that it controls the audio volume. Control commands of a virtual device can be issued by a request-center upon receipt of a control request at the UI or any other device participating in a Virtual Device.

III. ANALYSIS OF THE ARCHITECTURE

Distributed vs. Centralized
Many present day networked systems require a device in a network to play the role of orchestrating the devices in the network to perform some task. If this orchestrating device breaks down, then the whole network does not function. Also, since this orchestrating device has to bring the devices in the network together, it must be updated with information about all the devices present in the network, about any new devices that are added and about how to join these devices. The expectation is that there will be many different types of devices in pervasive computing systems. The number of devices will be variable and the number of functions that such system could perform would also not be easy to predict. Therefore, the amount of information stored in the orchestration device can become too large to maintain easily and the process of making it up-to-date will be error-prone and frustrating for users.

In the distributed model as proposed in this paper, there is no central device that has to know about all devices in the network and their possible connections. Every device has enough knowledge to understand with which device it needs to collaborate. If a new type of device is added to the network, it can update its knowledgebase with information about how it could collaborate with the existing devices. Therefore, this architecture is less susceptible to failure of any single device. Even when the RC where a user interacts with a Virtual Device or some other device fails, the rest of the network can still perform useful functions for a user that are still possible without the use of the failed device. Hence, we expect better scalability and fault resilience.

Implementation scalability
The devices in a system only have to know their own interactions. This implies that devices with more functionality, and therefore, more interactions need a more elaborate knowledgebase as compared to low functionality devices. The implementation of the knowledgebase interpretation engine can therefore be scaled to match the expected knowledgebase complexity. E.g., a sensor will need very limited knowledgebase capabilities as compared to say a mobile phone.

Communication technology dependence
Communication is based on messages that have to be understood by a receiving device. The receiving device takes action by interpreting its own knowledgebase. The simple format allows the message format to be bound to a simple MAC protocol, but also to a UDP or TCP multicast overlay. The only condition is that the communication technology should have broadcast capability.

Context dependent behavior
Devices built using the architecture proposed in this paper use their own knowledgebase to interpret messages and take actions. The knowledgebase is extracted from an ontology. Therefore it is possible to extract and store context related information in a device at the time of its deployment, e.g. as presented in [11]. It is also possible to define more contexts and the corresponding knowledgebase data and store it in the device during its life. The RC capable device, in this case, must implement functionality to interpret users settings into knowledgebase information and forward it to some devices. Devices that allow updating of context information, in this case must implement a knowledgebase update functionality locally.

IV. RESULTS
As a proof of concept, we have implemented the system as shown in Figure 7[12]. The user scenario is as follows: John wants to play music. He interacts with the Music Rendering device which also has a user interface. John also has some music decoding devices and music servers in his house. On Johns request to play MP3 encoded music, the server and the MP3 decoder form a network with the rendering device to play music while on Johns’ request to play WMA music, the music...
server and the WMA decoder device form a network with the music rendering device.

![Diagram showing the system components](image)

Figure 7: System implemented as a proof of concept

We have designed an ontology for playback of digital audio in which the Music Server can serve MP3 and WMA encoded music, and has a relationship with the different format decoders namely the MP3 Decoder and WMA Decoder. These decoders can decode the corresponding formats to the LPCM format of digital music and have a relationship with the music rendering device. The music rendering device is capable of playing the LPCM music data format.

If subsequently another device that can also decode MP3 enters the system and the user make a request to play a MP3 music, the Media server device gets a response from the new and the old MP3 devices. The Media server must therefore make a choice as to which device it will accept to join the virtual device based on rules stored in its knowledgebase. The simplest rule that we used is to select the device that manages to offers its services first. More complicated rules can be added to the knowledgebase such that decision is made on the basis of power consumption, quality of service provided etc. When a device, e.g. MP3 decoder leaves the system and there is no way to offer the required function in the network then the RC device times out without completion of the virtual device creation. In this case the user is notified by the RC that this function cannot be performed.

We have also shown that the knowledgebase can contain different recipes for different user contexts. E.g. if a user “A” wants to listen to music, then his preference for audio settings are used by the media renderer device, while for a user “B” the corresponding settings are used. When then user “A” notifies the renderer device that the context is “party”, then the renderer uses different settings from its knowledgebase.

Validation using UPnP devices

We have implemented the defined user scenario by mapping the architecture concept onto the UPnP AV architecture [6]. Although by itself UPnP is not the implementation technology of choice for small devices we can use the service-oriented approach it provides which matches the proposed architecture nicely. While in UPnP a control point is typically controlled by a user, in our system there is actually a control part in each device for the KB interpretation and cascaded build-up of the virtual device. We call devices equipped with this functionality smarties. The proposed sub-functions map onto UPnP services in a device, the KB interpretation function maps onto a Smarty Control, and a Smarty Service as described below. The Device Interface (Figure 2) is split between the Smarty Control and Smarty Service. In the UPnP AV architecture, two devices work together after a control point establishes the connection between them. The control point invokes actions on the services that a device exposes. We have introduced a new type of service in UPnP devices called the Smarty Service, see Figure 8. We have also embedded in every device an internal control function called Smart control. The Smarty service provides interfaces that allow an external control point to invoke behavior. Upon receipt of a message (as described in Figure 5), the Smarty Service analyzes the knowledgebase and invokes the execution of the specified actions. In UPnP technology only a control point can invoke services; hence the Smarty service must pass this responsibility to the Smarty control function. This way, we have been able to implement the new architecture concept on top of the existing UPnP architecture without changing it. An advantage of this is that UPnP devices can behave as regular UPnP devices when another UPnP device wants to work with them. For mapping the architecture on UPnP, we have constructed a knowledgebase in XML format.

![Diagram showing the proposed architecture](image)

Figure 8: The proposed architecture mapped on to UPnP

In terms of UPnP concepts, Smart control is a modified UPnP control point that is a part of every device. Smarty control is able to interact with UPnP services in the same device and also in other devices in the network. Smarty service is a UPnP service that has the responsibility to communicate with other devices, to read and understand the knowledgebase and to tell to the Smarty control what UPnP services to invoke. A Smarty control can access a Smarty service that resides in a different Smarty device using the UPnP invocation mechanisms.

The system interactions for virtual device set up and playback of music as described in the scenario presented earlier in this section are shown in the interaction diagram below of Figure 9.
The demonstrator was implemented using three separate devices based on an ARM processor: the media server, an MP3 decoder and an LPCM renderer. A fourth device, WMA decoder, was implemented on a Microsoft Windows based desktop PC. While the code size of the required components is roughly 100KB, the total code size of a device in this implementation is approximately 1.5 MB mainly contributed by the UPnP stack overhead and the media processing functionality. Smaller code sizes are definitely possible using specific programming environment for sensors, e.g. as in [17].

![Diagram](image_url)

**Figure 9**: Interaction diagram for Virtual Device set up and Execution and Control phase for Music playback application

V. FUTURE WORK

In the present implementations we have used a wired network between the devices. An implementation of this concept using a wireless networking technology such a IEEE 802.15.4 [15] is needed.

The knowledgebase is a key element of this architecture. Extraction of the knowledgebase for a device from an ontology needs to be formalized. Since the architecture concept works with a knowledgebase and since by changing the knowledgebase the behavior of a device can be changed, it is possible to make devices learn from other devices to update their own knowledgebase. This feature needs to be investigated.

An important issue for such distributed systems will be the quality of service for the user. Depending on the devices that form a virtual device, it is possible that the user experiences different quality of service. Investigation into this effect and on ways of coping with it is needed.

Finally, the architecture has to be extended with mechanisms for resource management, sharing and access control for practical applications.

VI. CONCLUSION

In this paper we have proposed a distributed architecture model for pervasive computing systems. The key aspect of this architecture is that the devices forming a pervasive computing network can be autonomous because the knowledge drawn from an Ontology that they need for their operation. This knowledge is embedded in each device. We have implemented a proof of concept for such a system and demonstrated the ability of such architecture to perform a complex task. The results are promising for pervasive systems in terms of scalability and resilience against (partial) failure.

REFERENCES