Abstract

The last three decades proved Moore’s Law. We witnessed an exponential increase in processing power, memory capacity and communication bandwidth and we expect this increase to continue for at least another decade. The effect is a growing complexity of computer systems and the need for highly qualified administrators. The question must be posed how computer systems can be managed in future if we project the actual progression in systems intricacy.

This paper focuses on autonomic computing as a potential solution. We describe how a distributed system can be built to satisfy the demands for self-configuration, self-healing, context awareness and anticipatory. Furthermore we describe our application example – Smart Doorplates – and the appliance of the previously discussed demands to that system.

1. Introduction

Recently the paradigm of autonomic computing was raised by IBM [4, 8] which proposes systems that are self-configuring, self-healing, self-protecting, and self-optimizing. Additional desirable properties are context-awareness and anticipatory.

The paradigms of autonomic computing span all system levels from hardware up to software [5]. System design varies depending on where the autonomic computing requirements are demanded: for all elements of the system or only for the overall system. Our application is composed of distributed elements (the doorplates) which are not autonomous. But the system comprised of the doorplates meets some demands of autonomic computing.

The next section discusses our approach to an autonomic application in a general way. Section 3 describes the application scenario and the system architecture of the Smart Doorplate system. Section 4 shows how we meet the autonomic computing requirements in our prototype system.

2. Design considerations

The Smart Doorplate application that we specify in the next section should be self-configuring to minimize the administration effort, self-healing to automatically adapt to new and maybe unexpected environmental changes, context aware to find a person’s actual location, and anticipatory to make predictions about a person’s future location.

Self-protection and self-optimization is regarded only shortly due to our implementation status. We focus on the afore mentioned four aspects.

2.1. Self-configuration

An essential requirement of autonomic computing is self-configuration. Only the absolutely necessary administration tasks should be required to configure or customize a system. Therefore it is important to figure out the minimal information required for each component to work as a part of an autonomic system. The configuration should be restricted to those parts which obviously build the system to keep it simple and clear. Most of the falsely configured parts of a system are in fact not yet configured parts, because the administrator was not aware that these parts had to be configured.

It might be hardly possible to completely configure a system by only specifying the obvious parts of an application. Often the impact of parameters of an application is neither easy to understand nor is it obvious which value could be the correct one. There are two solutions to this challenge.

The first and in our opinion preferable way is to build a system which is able to gather the missing information during runtime. The application must be able to start with predefined parameters or without any information. This kind of self-configuration is based on the assumption that the configuration can be completed based on information extracted from application messages or from specialized configuration messages.
The second way is to make the parts of the application, that need configuration by hand, visible to the administrator. To configure the parameters correctly it is important to know the dependencies between the parameters and their impact on the application.

2.2. Self-healing

The ultimate vision of self-healing is a system that replaces its malfunctioning components like the human body with cells which are constantly replaced with new cells. An autonomic computing system should continue working by compensation of failing components similar to a human body that adopts to the new circumstances during the healing process. Unlike the human body an autonomic computing system needs also self-healing if the system structure changes – not only missing parts, but also newly added parts must be recognized and smoothly integrated into the system. Similar requirements exist for self-healing as for self-configuration. The autonomic computing system should be able to gather information needed to compensate missing parts or to integrate newly added parts during runtime. No action should be required by the administrator except removing or adding components.

The task of an autonomic computing system may consist of several services. Each service can be replicated within the system and at least one instance must be available. This availability can be checked by a supervisor service which inspects the services regularly.

If a needed service is missed, the supervisor service starts a new instance of the required service. Thus we guarantee that at least the minimum number of services is running. To implement that kind of monitoring each service should have an extra interface for the communication with the self-healing-service.

In a deeper vision self-healing can be extended toward self-optimization. If the monitoring service not only launches new services by the time it detects a missing service, but also detects the need for services, it would be able to adjust the amount of services in ratio to the current workload. For that mechanism to work reasonable it must be able not only to start new instances of a service, but it must also be possible to stop unused services.

2.3. Context awareness

Context awareness is one of the most important features in autonomic computing, as already known from ubiquitous computing [1]. If an application has no information about its environment it might not behave the way it is expected to. The accumulation, fusion, representation and storage of sensor signals and information gathered from user interaction need special care. A central storage of information is neither scalable nor fault tolerant. It is preferable if the system is built up of smaller services which only hold a part of the context information.

Context information is distributed with redundancy over the different units of the system. The information of all units together build up the complete context. If one unit fails and there is another unit with the same information, the system will continue working without any notice to the user and the self-healing mechanism starts a new instance of the unit to replace the missing one. In case that a unit crashes and there is no other unit with the same information, the system might be able to continue work with reduced context awareness running in a reduced mode and enabling a self-healing mechanism to bring the system back to full operation.

2.4. Anticipation

Anticipation is the capability to have a projection of user needs and actions into the future. To anticipate the future behavior of a person, a system must analyze the past. The easiest way of anticipation is defined by static rules, which means that there is an unchangeable correlation between two events. The more interesting one is if there are changes in the resulting behavior. Context prediction in ubiquitous computing faces a similar problem [9].

To predict or anticipate a future situation learning techniques as e.g. Markov Chains [2], Bayesian Networks [7], Neural Networks [3] can be applied. The challenge is to transfer these algorithms to work with context information.

A common idea to record and analyze the past behavior is to build patterns. These patterns can be classified and weighted with probabilities of their occurrences. In that case the anticipation or prediction is reduced to find the most appropriate pattern for the actual situation described within the context. But the system will hardly reach a probability of 100% situations. So an autonomic computing system can only advise some activity and must be prepared that the user will not adopt it, but rather do something completely different. In this case the speculation fails and proactively performed actions must be reversed.

3. Toward the Smart Doorplate application

There are strong interconnections between the different self-mechanisms in autonomic computing. Our approach is to build up a system starting with rather simple implementations of self-mechanisms and constantly increase their functionality.
3.1. The aim of the application

Our sample application stems from the area of ubiquitous computing. We transfer simple doorplates into the vision of Smart Doorplates within an office building [12]. The doorplates are able to display current situational information about the office owner, to act instead of the office owner in case of absence, and to direct visitors to the current location of the office owner based on a location-tracking system. A short description of a scenario that is well known to all of us helps to understand the vision.

Sometimes you enter an office building without knowledge of the way to the office of a sought person. In large office buildings you register yourself at the reception and ask the way to the office, which may be difficult to follow. Moreover, often in particular in smaller offices there is no receptionist and nobody to lead you. Imagine a terminal near the entrance where you could type in your name and the person you are looking for (as you would tell the receptionist) and you receive automatically (or by the receptionist) a tag that identifies yourself and your target office throughout the building. The terminal or the receptionist tells you the initial direction to the office you are looking for. On your way to the office the doorplates recognize you by reading your RFID-tag and guide you to the office by displaying an arrow as shown in Figure 1.

If you arrive at the office the doorplate can tell you if the sought person is present by displaying a small icon beneath the person’s name. Moreover, a list of all present persons in the room may be displayed. If the person is not present you can ask the doorplate for her current location. The smart doorplate can tell you where to find the person and directs you to that room or anticipates that the person is soon coming back and advises you to wait.

Our vision of the Smart Doorplate leads much further than the scenario described here [12]. But as we want to show the current state of development we focused on the already implemented parts. At the moment only the registration terminal at the entrance is missing which we do not need in our office because every inhabitant has her own tag.

The vision leads to several application requirements with respect to autonomic computing: Doorplates should be easily attached or removed without disturbing the system (self-configuration, self-healing). Specific information should only be managed and retrieved by the owner or administrator. Information should be kept locally to provide privacy of the office owner or the visitor (self-protection). In a distributed system without centralized information, searching is sometimes arduous. Here self-optimization could be applied.

3.2. Architecture

The architecture of our application is based on a peer-to-peer communication layer. Thus we gain the flexibility to build a non centralized application structure. The peer-to-peer middleware is JXTA [10] implemented in Java. The smallest unit is a Doorplate on a Compaq iPAQ running Windows CE 3.0 and JVM J9 from IBM [6]. Figure 2 shows a simplified sample application setup of the peer-to-peer network consisting of Doorplates, a RFID-reader and various services as described later.
Messages are sent over so-called unidirectional, bidirectional or propagate pipes. Unidirectional pipes can send a message in only one direction from the sender to the receiver. Over a bidirectional pipe both peers can send and receive messages. Propagate pipes are unidirectional pipes where the message is sent to all peers that have an input pipe for that propagate pipe. This means that a message send by one peer can be received by many peers.

The tracking system is made up by RFID-readers [11] at special locations to become aware of persons entering and leaving a room which is reflected by a change in the location context.

Figure 3 shows the architecture of the application. The figure illustrates the different layers used by the application. The lowest level shows the transport mechanisms used for the messages in the peer-to-peer network. We currently use TCP/IP based communications over Ethernet for the PCs and Bluetooth for the iPAQs.

The second and third level build up the JXTA peer-to-peer network. There is a noticeable difference concerning the JXTA implementations for Java 2 Standard Edition (J2SE) and Java 2 Micro Edition (J2ME). The JXTA J2ME implementation (JXME) consists of only three classes. To hide the implementation dependent parts of JXTA we introduced a Service Layer. The Service Layer contains all the services used by the application.

The design of the Service Layer is shown in figure 4. Both JXTA implementations use pipes for the communication but differ in the methods used to read elements out of the messages and write elements into them. The Pipe Abstraction Layer handles this part. Furthermore it transforms the message elements into Service Message Objects which are sent and received by services.

The Service Communication Layer is used to pass the service messages to the Pipe Abstraction and to inform registered services in case of arriving messages delivered from the Pipe Abstraction. Because of the event driven communication mechanism a service must register itself by the Service Communication Layer to receive messages.

The application is comprised of the Doorplates and four services. The Doorplates display situational information and are used for person interaction.

The Location Service collects the location change messages and creates appropriate location events. Location change messages can be sent by any sensor capable of recognizing a person’s location. At the moment we only use the RFID-readers and an extra peer for simulation purpose. In future there might be a camera which observes an office floor and produces location change messages accordingly. RFID-reader and camera information may be utilized by the Location Service to determine a person’s location. The location messages send by the Location Service are used by the Doorplates and by the Prediction Service.

The prediction or anticipation for a person’s next location is computed by the Prediction Service. The Prediction Service is used as a container for a replaceable Prediction Engine. The Prediction Engine tries to predict a person’s next location based on the already received location messages. This offers the possibility to run more Prediction Engines with different algorithms at the same time to evaluate their accuracy.

The Prediction Service listens for all location messages and build up its pattern of context changes. If someone is interested in a predicted location of an absent office owner the Prediction Service can be asked via the Doorplate.

The Doorplates and the Prediction Service use input propagate pipes to receive location messages. The tag-readers create location change messages used by the Location Service to create location messages. The communication with the Prediction Service and the Direction Service is realized with bidirectional pipes.
The Direction Service has a map of the building to determine the direction from one office to another. If a Doorplate wants to know the direction to an office it asks the Direction Service. The Direction Service answers with direction information like north, west etc.

The Monitoring Service is used for self-healing of the application. It monitors the running services and keeps track of the overall system availability. If a service is missed it tries to start a new instance of that service on another peer.

### 3.3. Implementing the vision

**Self-configuration:** The complete application is composed of single Doorplates which can act as stand alone units with restricted functionality. The customization of a Doorplate is done by configuring the room number and the names of the persons residing in that room. Each Doorplate stores only the information associated with the room it stands for. If the Doorplate is informed that a person enters or leaves the room it updates the local storage to reflect the actual situation.

The application enforces its complete functionality if there are more Doorplates and if the Location Service, currently based on RFID-tags only, is enabled. Every Doorplate is informed and updates its local storage as already explained. As a result the information building the complete application context is distributed across all Doorplates. If someone likes to locate a person, a request is sent over the propagate pipe and the Doorplate that finds the person’s ID in its local storage will send a response.

There is no centrally stored data. The application context is made up by the entirety of the distributed context information. The configuration is updated automatically by new messages, when there is a change detected by the Location Service. No need for extra configuration exists other than the required configuration done by the office owner to personalize the information she likes to be displayed at the Doorplate. Everything else like the location of a person and the direction leading toward there can be determined during runtime by using the mechanisms the peer-to-peer network offers.

If a new Doorplate is added to the system there is no information available other than its room number and the habitants. But as soon as there is a location change the new Doorplate is informed and can again update its local information. So the new Doorplate is smoothly integrated in the application.

Another situation arises if a Doorplate is reconfigured by changing the room number or the names. The reconfiguration is reflected immediately and any information related to the room number will be stored by the newly configured Doorplate from now on.

**Self-healing:** The self-healing has two independent mechanism. The first is responsible for the healing of context information and works in a similar manner as the self-configuration. If a Doorplate runs out of order or is unreachable only for a while, its information will be lost for the application, which means that the information concerning that room will be no longer stored or updated and that the already collected information, who else is in that room, is no longer available.

But as the location of the persons will change sooner or later, the new location will be updated by the Doorplate controlling the room she entered and the context will again reflect the current state. If the Doorplate resumes work and its local context is out of date, it might nevertheless answer if it is asked about a person’s location. Additionally another correct answer is sent by another Doorplate if the person had changed location. To identify the correct information, we store the time when an information change occurred and provide the time stamp with the answer. So, if two or more answers to a request emerge, the different timestamps allow to select the newest one. The Doorplate will update its context information over time as it is informed by the Location Service.

The self-healing is mainly based on the exchange of the location change messages received by every Doorplate to update the local context. If there would be no more location messages sent, the application would be unable to rebuild the complete actual context. To face this situation, every Doorplate sends a location backup message to confirm the actual position of all its present persons and to give other Doorplates the possibility to update their information, if it hasn’t received any message for a predefined period of time.

The second self-healing mechanism is the Monitor Service which observes the running services. It checks the availability of the different services periodically by sending a request to the services. If the request isn’t answered within a given amount of time, the Monitor Service tries to start a new instance of the missing service on a different node. Therefore it asks the nodes which one is capable of running the desired service. It selects one of the answering nodes to start the new service.

**Context awareness:** An application that has some knowledge of its environment and recognize changes in the environment is called context aware. But awareness is just one part of it. The information must be stored and processed to react accordingly.

The Smart Doorplate application uses the location of persons identified by RFID-tags as its context. The context information is stored in a distributed manner. Each Doorplate stores the information relevant for it. All Doorplates are informed if a location change is detected. The location message contains the room a person entered, the identifica-
tion and name of the person, and the creation time of the message. If a Doorplate with the given room number receives the message, it updates its local context. If a person enters her office, the Doorplate will display an icon beneath the person’s name. If another person enters the room, the name may be added to a list of present persons.

If a Doorplate receives a message that a person has left the room, it will remove the person from the list of present persons and if necessary remove the icon on the screen. If neighboring Doorplates store at least some messages for a defined amount of time, self-healing in case of a faulty Doorplate is improved.

**Anticipation:** The anticipation demanded for autonomic computing might be very hard to realize since it needs some kind of intelligence to foresee the user’s need. In our application anticipation is consciously reduced to the prediction of a person’s future location. The actual implementation uses a Prediction Service as a peer group service which can be used by any Doorplate.

The Prediction Service applies a replaceable Prediction Engine to evaluate a specific prediction algorithm. The challenge is to transfer known prediction algorithms to handle context information. We actually use an algorithm based on Markov Chains [2]. The Prediction Engine records the previous locations of every person. With these patterns and the probabilities of the possible locations a person’s next location is predicted. The Prediction Service informs the Prediction Engine in case of location changes and the Prediction Engine will update its information.

A visitor who misses someone can ask for the actual location of a person by clicking on the person’s name at the Doorplate. She will also get the next location predicted by the Prediction Engine. At the moment it is not possible to give hints like ”will be back in five minutes”, because we do not treat time information yet. For now we anticipate the next location with a given probability, depending on recognized patterns.

### 4. Future work

The application was build with autonomic computing in mind. However, at the moment there is no security mechanism implemented. One of the next steps will be to extend the system with self-protection. Our approach is to use already known security mechanisms which can detect intrusion of a single unit to protect the unit itself. The self-protection mechanism of the system will be derived from the exchange of alarm messages sent by the peer detecting a misbehavior to the other peers. That information can be used by the other peers to strengthen their security shield depending upon the kind of intrusion.

### 5. Conclusion

The implementation chosen for the Smart Doorplates is a first step into autonomic computing for a completely distributed application. The configuration of the application is reduced to the absolute minimum. The application system configures and heals itself during runtime by exchanging messages over the peer-to-peer network using the propagate pipes from JXTA. The location changes are recognized by RFID-tag readers controlling the office floor to detect persons entering and leaving an office. If a tag-reader identifies a tag it sends a location message to the Doorplates which can update their local context information.

The application context is composed of the local contexts on each Doorplate which has the advantage that the context won’t be lost if one Doorplate denies service. Anticipation is implemented by the prediction of a person’s future location. The prediction is realized as a peer group service and can be replaced by different implementations of Prediction Engines for evaluation purpose.

### References


