Towards an Organic Middleware for the Smart Doorplate Project

Wolfgang Trumler, Faruk Bagci, Jan Petzold, Theo Ungerer
University of Augsburg
Institute of Computer Science
Eichleitnerstr. 30, 86159 Augsburg, Germany
{trumler,petzold,bagci,ungerer}@informatik.uni-augsbrug.de

Abstract: We envision future office buildings that partly or fully implement a flexible office organization, i.e. office space is assigned dynamically on request. These organizational principles save office space, but require a sophisticated software system that is highly dynamic, scalable, context-aware and self-organizing by means of self-configuration, self-optimization and self-healing. We propose an organic middleware approach to support self-organization in ubiquitous in-door environments as exemplified by the Smart Doorplate Project.

1 Introduction

Smart buildings represent an important application area for ubiquitous computing that includes context-aware and networked computers and appliances. We envision future office buildings that partly or fully implement a flexible office organization where office rooms are dynamically assigned to currently present employees. Our aim is to design a middleware that supports the required dynamic office assignment and automatically supplies the personal environment e.g. telephone calls are routed to the current location either at the company or at the home office.

The regarded computing environment within an office building consists of servers, PCs, laptops and PDAs connected by wired or wireless networks. That allows to base the middleware on Java enabled devices. To enhance robustness and reduce configuration overhead we adopt the organic computing principles of self-configuration, self-healing and self-organization in conjunction with context-awareness and anticipation. Our concrete application example is chosen as part of the flexible office scenario. We focus on the organization of Smart Doorplates and its associated services.

The Smart Doorplates [TBPU03a, TBPU03b, TBPU04] 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. An arrow to the direction of the sought office is displayed if a visitor reaches the vicinity of a doorplate. If the office owner is present but busy the Smart Doorplate can act as a receptionist and show some status information about the office owner (e.g. on the phone) to the visitor.

The underlying infrastructure is hidden by the middleware which is based on a peer-to-peer network. The middleware supports transparent messaging and monitoring. The middleware is intended to self-organize the services that build the application. The self-organization is based on the three major requirements of self-configuration, self-healing and self-optimization. These self-x mechanism are demanded by the organic [VD03] and the autonomic computing principles [Ho01].

To build applications that can react to environmental changes and thus self-organize, a sophisticated monitoring is needed to collect and provide information about the services and resources of the nodes and the complete system itself.

Monitoring should be implemented locally for each node to minimize the messaging overhead of a centralized monitor. The information exchange for a global view of the system should induce as little communication overhead as possible. The collected information must be analyzed by metrics which calculate a value of excellence. So the metrics may calculate how good a node can offer a special resource or service. In terms of self-organization, the distributed monitoring is the driving force. The values calculated by the metrics depend not only on monitoring data, but also on the application itself.

The middleware must support the configuration and reconfiguration of nodes to host new services and to transfer running services to another node. The self-configuration for the initial configuration of the system, self-healing in case of failure and self-optimization during runtime are based on the reconfiguration capability and should be fostered by the middleware.

2 Organic Ubiquitous Middleware Architecture

The middleware is designed with the goal in mind to support the device-independent application of organic computing demands in ubiquitous environments where we expect a heterogeneous collection of devices with diverse capabilities of computing power, memory space, and energy supply. We choose an approach where the middleware decouples the application services from the messaging, monitoring and organic manager functionalities. The middleware consists of four main parts (see fig. 1): the Transport Interface, the Event Dispatcher, the Service Interface and Service Proxy, and the Organic Manager. Monitoring is done at the message transport level and the level of the Event Dispatcher and delivers the monitoring data to a Monitoring Info Space within the Organic Manager.

Transport Interface: The Transport Interface decouples the message delivery from the used transport system. It transforms or encapsulates the EventMessage-Objects to the format of the used transport system. We use JXTA for the current implementation, however, the Transport Interface can be replaced depending on the given communication infrastructure which is transparent to the rest of middleware.

Event Dispatcher: The Event Dispatcher is responsible for the delivery of incoming and outgoing messages. It offers services the functionality to send messages and to register themselves as listeners to specified types of messages. A service can register for different types of messages and is informed in case of an incoming message with one of the types

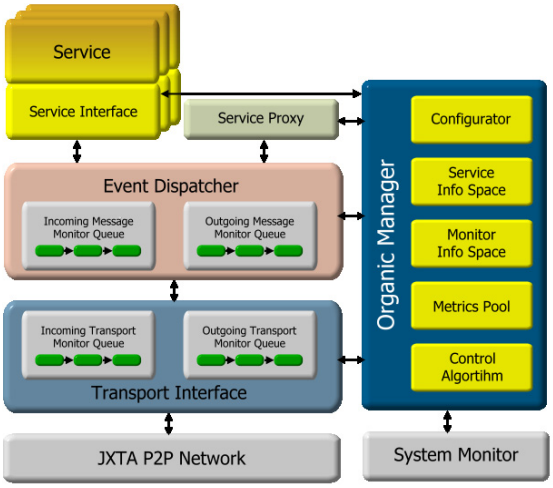

Figure 1: Organic Middleware Architecture

it registered for. The Event Dispatcher handles the delivery of broadcast and unicast messages. It knows whether a message can be delivered locally or if the message must be sent to a remote node.

Service Interface and Service Proxy: The Service Interface is the connector between the middleware and a service. A service that wants to participate must implement the Service Interface such that it can receive messages delivered by the Event Dispatcher. The interface also offers the functionality for sending messages.

We distinguish two kinds of Service Interfaces. The simple Service Interface has already the full functionality needed to communicate within the system. Furthermore a Relocatable Service can be transferred to another node whereas a normal service is bound to the node it was started on. The binding on a special node is important for services that need a special hard- or software environment (e.g. fixed sensors or databases). In our Smart Doorplate application the Doorplate Service is bound to a specific doorplate computer and the Location Service to the respective location detection hardware, other services as e.g. the Direction Service and the Prediction Service are relocatable.

A Service Proxy is used to forward incoming messages to the new location of a service that has moved to another node. It has a limited lifetime and automatically dies after that time. The lifetime depends on the validity duration of a service advertisement.

Organic Manager: The Organic Manager is the major control instance of a node. All relevant information from the local monitors and services are collected here. Organic Managers from different nodes exchange information about their state and trigger a reconfiguration if necessary.

The Organic Manager contains the Configurator which is responsible for the configuration

of the services (init, start, stop, resume, end) on that node. It includes a Control Algorithm which evaluates the "actual state" of the node with the metrics from the Metrics Pool and decides whether a reconfiguration (relocation of a service) should be initiated. The Service Info Space and the Monitor Info Space represent the state repositories for the services respectively for the monitors. The Control Algorithm and the Metrics are just interfaces that must be implemented to adopt the middleware to different special application areas.

3 Self-organization Example

The self-organization of the middleware can be divided into the three tasks configuration, optimization and fault recovery, covered by self-configuration, self-optimization and self-healing respectively.

If a node of the middleware is started it announces itself to the other nodes. Initially no configuration is given and the system is in an equilibrium state. The first time the system self-organizes is when a configuration is given that should be satisfied by the middleware. The configuration is flooded to every node. If a node wants to host a service it informs the other nodes which mark the service as assigned. With the service reservation the node sends a "quality of service provision" value that tells how good the service can be served. If another node wants to host the same service it can override the reservation by sending a response with a higher value.

A typical self-optimization in our application concerns an unbalanced load of services on the different computing nodes. After the initial self-organization the system is running and monitored to collect information about its behavior and to detect potential cases for optimization. The system must be prevented from spontaneous reconfigurations due to workload peaks. If the control algorithm detects the need for a reconfiguration of a local service it sends a reconfiguration request to the Organic Managers of the other nodes. If a node is able to host the service it sends a response accordingly. A response is created if and only if the service can be hosted. If the node receives more than one response to the configuration request it selects the one with the best "quality of service provision" value.

Self-organization due to self-healing in the Smart Doorplate application is used to guarantee that all given services from the configuration are available any time. An Alive Monitor recognizes if a service is unavailable on a specific node. Therefore it assesses the availability of services in the Event Dispatcher by monitoring the messages which are exchanged between the services. Alternatively, if the service doesn't communicate for a given period of time, the Alive Monitor pings the service.

In case of node-bound services the only opportunity for self-healing mechanism is to restart the service hoping that the service will resume work. Relocatable services on the other hand could be started on another node.
The middleware would first try to restart the service on the same node to minimize the reconfiguration overhead. If this fails the other Organic Managers are asked to host the needed service with an "urgent"-flag notifying the need for a self-healing reconfiguration. In this case also nodes that would exceed their workload level will send an answer to the

request. The idea behind this behavior is to guarantee the availability of services prior to performance.

4 Conclusion and Future Work

Our middleware approach enables distributed and service-based ubiquitous applications to self-organize by means of self-configuration, self-optimization and self-healing. The monitoring across all layers of the middleware allows to disburden the administrator from the often complex configuration and optimization of distributed applications. The middleware is already running on a four doorplate prototype. We currently enhance the middleware by the Organic Manager component as described in section 2.

The next step will be to evaluate the quality and speed of the reconfiguration process. This is vital as it directly affects the self-optimization of the middleware. Another task is to measure the communication overhead of the middleware. This will be done at an extended Smart Doorplate installation which covers the floor of our institute. It will give us results from a real long-running application beside the purely calculated values out of the test bed in the lab.

References

[Ho01] Horn, P. Autonomic Computing: IBM's Perspective on the State of Information Technology. http://www.research.ibm.com/autonomic/. October 2001.

[TBPU03a] Trumler, W., Bagci, F., Petzold, J., und Ungerer, T.: Smart Doorplate. In: *First International Conference on Appliance Design (1AD)*. Bristol, GB. May 2003. Reprinted in Personal Ubiquitous Computing (2003) 7: 221-226.

[TBPU03b] Trumler, W., Bagci, F., Petzold, J., und Ungerer, T.: Smart Doorplate - Toward an Autonomic Computing System. In: *The Fifth Annual International Workshop on Active Middleware Services (AMS2003)*. S. 42–47. Seattle USA. 25. June 2003.

[TBPU04] Trumler, W., Bagci, F., Petzold, J., und Ungerer, T.: AMUN - Autonomic Middleware for Ubiquitious eNvironments applied to the Smart Doorplate Project. In: *International Conference on Autonomic Computing (ICAC-04)*. S. 274–275. New York, NY. May 17-18 2004.

[VD03] VDE/ITG/GI. Organic Computing: Computer- und Systemarchitektur im Jahr 2010. http://www.gi-ev.de/download/VDE-ITG-GI-Positionspapier Organic Computing.pdf. October 2003.