Masterarbeit

Model-based support of Virtual Organizations in an RFID middleware

Andreas Leitner

Institut für Technische Informatik
Technische Universität Graz
Vorstand: O. Univ.-Prof. Dipl.-Ing. Dr. techn. Reinhold Weiß

Begutachter:	Dipl.-Ing. Dr.techn. Christian Kreiner
Betreuer:	Dipl.-Ing. Michael Thonhauser

Graz, im Mai 2010

Abstract

Modern distributed computer systems, with mobile and embedded devices as first class citizens, are formed from heterogeneous platforms. Due to their distributed nature dynamic reconfiguration and adaptation seems to be a strong requirement as well as the support of organizational structures enabling shared usage of resources. Model-based techniques have been recognized to provide a level of abstraction for software development and to support dynamically (re-)configuration of the software at runtime.

A portable runtime architecture is presented that explicitly honors ownership and realm of control of hardware devices, resources and application components specified by multiple models. This thesis is based on a previously realized Model-based Component Container (MCC), which defines and implements a concrete model paradigm covering one domain-specific aspect. Several MCCs are used for the execution of the specific models of an application component (a.k.a Model-Based Software Component (MBSC)) and are managed by specific runtime nodes contained in the portable runtime node architecture.

The architecture has been prototypically implemented and has been evaluated in a scenario, demonstrating the feasibility of the proposed architecture to support the dynamic formation of virtual organizations out of several independent organizations to realize a dynamic reconfigurable building access system.

Kurzfassung

Moderne verteilte Systeme, bestehend aus mobilen und eingebetteten Geräten, kennzeichnen sich durch ihre lose Kopplung, und der Heterogenität der zugrunde liegenden Plattform. Aufgrund der Verteiltheit dieser Systeme spielen dynamische Anpassung und Rekonfiguration sowie auch die gemeinsame Nutzung von Ressourcen durch Virtuelle Organisationen eine zentrale Rolle. Modellgetriebene Softwareentwicklung bewährte sich als eine Technik, die durch Abstraktion dynamische Konfiguration und Rekonfiguration zur Laufzeit ermöglicht.

Diese Masterarbeit präsentiert einen auf Modellen basierenden Ansatz für eine Middleware, die es ermöglicht mehrere unabhängige Softwarekomponenten, welche potentiell in den Verantwortungsbereich von verschiedenen Organisationen fallen, zu verwalten und Ressourcen dynamisch zuzuteilen.

Die vorgestellte Architektur wurde in einem Prototyp realisiert, und anhand eines dynamisch konfigurierbaren Gebäudezutritts-Szenarios evaluiert. Diese Evaluierung zeigt die Möglichkeiten dises Ansatzes zur Unterstützung der dynamischen Aspekte von virtuellen Organisationen, welche sich aus mehreren unabhängigen Organisationen formen, auf.

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz,am ....

(Unterschrift)

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

....

date (signature)

Contents

1. Introduction **1**

2. Related work **3**

2.1. Distributed systems . . . 3
2.2. Grid Computing . . . 4
2.2.1. Architecture . . . 5
2.2.2. Classification . . . 8
2.2.3. Virtual Organization . . . 18
2.3. Cloud Computing . . . 20
2.3.1. Architecture . . . 21
2.3.2. Resource Management . . . 22
2.4. You-R® Open . . . 24
2.4.1. EPC Global . . . 24
2.4.2. Tube . . . 25
2.5. Model Driven Software Development . . . 26
2.5.1. Metamodeling . . . 28
2.5.2. Entity Container . . . 30
2.5.3. Model-based Component Container . . . 33
2.5.4. Model-Based Software Component . . . 38

3. Design of a model-based middleware for virtual organizations **41**

3.1. Requirements . . . 42
3.2. Architecture . . . 44
3.3. Virtual Organization Node Architecture . . . 46
3.3.1. MCCs in the controlling unit . . . 47
3.3.2. Event queues . . . 49
3.3.3. Node identification . . . 50
3.4. Virtual Organization Node Design . . . 50
3.4.1. Class diagram . . . 50
3.4.2. Startup . . . 51
3.4.3. Event handling . . . 53
3.4.4. Component location . . . 56
3.5. Resource Node . . . 57
3.5.1. Resource management . . . 59
3.5.2. Lookup service component . . . 60
3.5.3. Value Added Reseller principle . . . 62

4. Implementation of a model-based middleware for virtual organizations 65
4.1. .Net Framework 65
4.1.1. .Net Compact Framework 66
4.1.2. .Net Micro Framework 67
4.1.3. Comparison 69
4.2. Hardware 70
4.3. Scenario 70
4.3.1. RN RFID reader 72
4.3.2. RN PDA 75
4.4. Results 78

5. Conclusion 81

A. Event Mechanism 83

B. Development tools 87

Literaturverzeichnis 88

List of Figures

1.1. Spanning multiple Virtual Organizations 1

2.1. The evolution of Grid technologies 5
2.2. The layered Grid Architecture 6
2.3. Combination of Collective and Resource layer protocols 8
2.4. A classification of traditional and emerging Grids 8
2.5. Ad hoc grid architecture 11
2.6. Wireless access grid architecture 13
2.7. Classification of Wireless Grids 13
2.8. Mobile Grid Architecture 15
2.9. Composition of virtual organizations 20
2.10. Grids and Clouds overview 21
2.11. Cloud- and Grid- Architecture compared 22
2.12. Triangle model 24
2.13. Structure of the You-R® Open middleware 24
2.14. Layers of a You-R® Open Tube 26
2.15. Model Driven Architecture by OMG 27
2.16. The principle of MDA 28
2.17. PIM, PSM and transformation 28
2.18. The four metalevels 29
2.19. Metamodel and inheritance 31
2.20. Architecture of an Entity Container 32
2.21. Enitity Container desing 32
2.22. Statemachine MCC 34
2.23. Visual description of the M2 layer of the data-MCC 34
2.24. Visual description of a sample M1 layer of a data-MCC 35
2.25. Visual description of a sample M0 layer of a data-MCC 35
2.26. Visual description of the M2 layer of the statemachine-MCC 36
2.27. Sample statemachine of a door opener 36
2.28. Visual description of the M1 layer of a door opener statemachine-MCC . . . 37
2.29. Visual description of the M0 layer of a door opener statemachine-MCC . . . 37
2.30. Model-Based Runtime Node 39

3.1. High-level Architecture 42
3.2. Architecture with one operator and two tenants 45
3.3. Component Diagram 45
3.4. Structure of a standard VON 46
3.5. State diagram of the VON controller's statemachine-MCC 48
3.6. M1 model of a data-MCC used to specify MCCs in the application area . . 48

3.7. M1 model of the data-MCC to realize the publisher subscriber mechanism . 49
3.8. M1 model of a data-MCC used to specify the VON structure 49
3.9. VON component 52
3.10. Detailed startup sequence 54
3.11. Half-Sync/Half-Async pattern 55
3.12. Overview of the event mechanism 56
3.13. Broker pattern 57
3.14. Layered Broker architecture 58
3.15. RN class diagram 59
3.16. Resource management component 61
3.17. Lookup service component 61
3.18. Performing internal and external lookup 62
3.19. Value added reseller principle 63
3.20. Sample setup of the VAR principle 64

4.1. Common Language Runtime diagram 65
4.2. Microsoft Embedded Products 67
4.3. Micro Framework Architecture 68
4.4. Overall configuration and VON deployment 71
4.5. VON1 of the facility management organization 72
4.6. VON2 of the facility management organization 73
4.7. VON of departmentA 74
4.8. VON of departmentB 74
4.9. VON1 of the security service organization 75
4.10. VON2 of the security service organization 76
4.11. VON3 of the security service organization 76
4.12. Code size of the different implementation components 78
4.13. Memory fragmentation at runtime 80

A.1. Events forwarded to specific MCCs 83
A.2. Handling events with specified destination address 84
A.3. Handling events with no destination address specified 85

List of Tables

2.1. Ad hoc Grid projects 12
2.2. Wireless Grid Usage Pattern 14
2.3. Mobile Grid projects 16
2.4. Grid projects which address interactivity 17
2.5. Grid project which address personalization 17
2.6. Grid project which address manageability 18

4.1. Key features of the .Net MF boards 70
4.2. VONs and their associated URNs 72
4.3. VONs and their assigned native functions 77
4.4. Event sources and their subscribers 77
4.5. JSON models of the MCCs in the controlling unit 79
4.6. JSON models of all VONs involved in the scenario 79
4.7. Allocated memory without models 79
4.8. Period of time for transferring models to a VON 80

1. Introduction

Today we are confronted with an increasing number of distributed systems made up of mobile and embedded devices, which are integrated as part of their surroundings. The performance and storage capacities of these mobile and embedded devices are increasing, thus enabling the usage of these devices as first class members in distributed systems. For using these enhanced device features, applications for distributed pervasive systems have to deal with platform heterogeneity and still existing resource constraints of these devices.

Being a first class member in a distributed system, devices resources (e.g. CPU, disk space, connectors) should be available for usage by other members of these distributed systems. To enable this shared usage, support for authentication, authorization and billing is required, which is provided by the concept of Virtual Organizations (VO). The idea of a VO has been developed in the context of Grid Computing systems [FK04], which have evolved out of the requirements for managed distributed systems consisting of servers and special resources (e.g. telescopes) provided by different administrative authorities.

This master's thesis presents a model-based approach for a middleware enabling mobile and embedded devices within an actual organization (AO) to form VOs whereas it is possible for an AO to belong to multiple VOs (Figure 1.1). While usage of models primarily allows to deal with problems of platform heterogeneity, this approach,through a modular runtime architecture, also enables the management of shared resources on these devices.

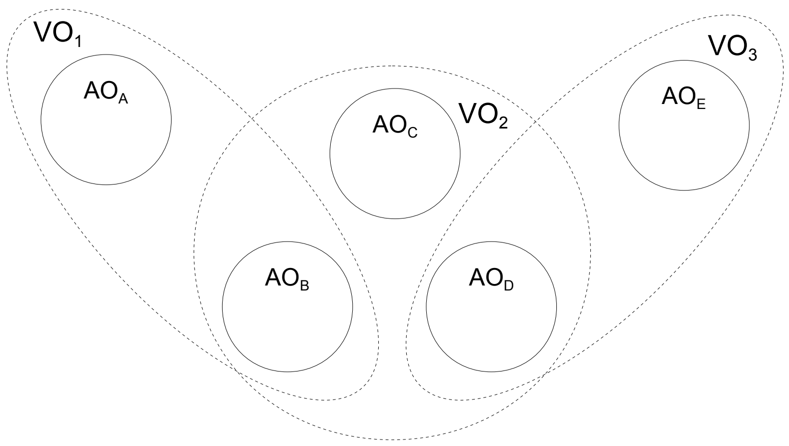

Figure 1.1.: Spanning multiple Virtual Organizations (based on [ANG04])

Based on the Entity Container (EC) which was presented by Schmölzer et al. in

[SMK+05] a Model-based Component Container (MCC) was designed and implemented on the Institute of Technical Informatics before this thesis started. The MCC can be seen as a part of a model-based software component which implements functionality by a set of models that will be executed at runtime.

On the basis of this MCC a infrastructure component called virtual organization runtime node (VON) is designed, satisfying the requirements for model-based configuration and reconfiguration of resources, application components and VO memberships. Due to the fact that a VON is a self-contained application component with a uniform resource name (URN) associated to it, it is furthermore possible to migrate a VON from one device to another as long as it does either not use local resources, or resources can be accessed in the same way. Another important feature of the presented approach is that all relevant runtime data are stored in MCCs which makes the reconfiguration or migration of whole application components possible at each point in system runtime.

Chapter 2 is dedicated to the theoretical work and literature research done for this master's thesis. An introduction to distributed systems is given in Section 2.1. Section 2.2 addresses the Grid computing concept, giving an overview about the architectural aspect and trying to find a classification of the various types of Grid systems. Additional the approach of VOs is covered in this section. Section 2.3 treats the term Cloud computing. The differences in the architecture compared to Grid computing are discussed and a description about resource management in Cloud computing is given. Section 2.5 introduces the basic concepts of Model Driven Software Development (MDSD). A special emphasis is put upon the MCC which is an integral component of this thesis.

Chapter 3 deals with the architecture and actual design of the model-based middleware supporting VOs. Based on the requirements given in Section 3.1 the architecture and the design of a VON is given in Section 3.2 to 3.4. Further the communication strategies between VONs are described and resource management issues are covered in Section 3.5.

Chapter 4 analyses the issues connected with the implementation of the middleware and the proposed use case of a building access system. The different .Net runtime platforms are discussed and a short overview of two .Net Micro Framework evaluation boards used for this work is given. Furthermore the outcomes of this thesis are evaluated in a sample scenario, with respect to memory consumption and runtime behavior.

2. Related work

2.1. Distributed systems

Tannenbaum and Van Steen give the following quite loose definition of a distributed system:

> *A distributed system is a collection of independent computers that appears to its users as a single coherent system.* [TS07, p.2]

The advantages of using distributed systems are that it is easier to integrate different applications, running on different devices into a single coherent system. Another important advantage can be an increased scalability with respect to the underlying network when the system is probably designed. Tannenbaum and Van Steen distinguish between distributed information systems, distributed embedded systems and distributed computing systems.

Distributed Information Systems

Distributed information systems deal with the integration of applications into an enterprise-wide information system. One can distinguish two levels of integration. The first level of integration deals with the field of *distributed transactions* where it is possible for clients to wrap a number of requests into one larger request and send it to the server. The second level deals with *enterprise application integration*. At this level applications are communicating directly with each other. This interapplication communication led to different communication models such as *remote procedure calls* (RPC), *remote method invocation* (RMI) and to *message-oriented middlewares* (MOM), also known as the publisher/subscriber paradigm. [TS07]

Distributed Embedded Systems

Distributed information systems are mainly characterized by their stability. Nodes have permanent network connection and they are mostly fixed regarding their location. In contrast, mobile and embedded computing devices are often characterized by being small, battery powered, not fixed regarding their location and communicating only via a wireless connection. Pervasive applications are part of our surrounding, can be configured by their owners, but self-configuration is also feasible by automatically discovering their environments. Do meet all this various conditions, Grimm et al. [GDL$^+$04] defined three requirements for distributed pervasive systems:

The first requirement, **embracing contextual changes**, means that a device must be always aware of a possible change in the environment. If a change happens the device should be able to react appropriate. **Encouraging ad hoc composition** is the second one and means that it should be possible to compose applications, services and devices at runtime. Furthermore interposition must be simple to dynamically changing behavior of the application. The last requirement, **recognize sharing as the default**, refer to the fact that systems need to make it easy to access information and to share this information between different applications.

Distributed Computing Systems

This class of computing systems describes systems which are used for high-performance computing tasks. A distinction between two subgroups can be made. Cluster computing, which is characterized by homogeneous hardware and closely connected workstations or PCs on one side, and grid computing where the hardware is mostly heterogeneous and workstations are widely distributed on the other side.

Cluster computing system are characterized by a collection of similar workstations with homogeneous hardware, on which the same operating system is running and all nodes are closely connected with a high speed local-area network. Primary computer clusters are used for computational purpose, where a single program is run in parallel on multiple machines. A typical compute job which run on a cluster my require frequent communications among nodes, which implies that the workstations are densely located and share a dedicated network.

Grid computing systems on the other hand are characterized by a high degree of heterogeneity. They can differ concerning hardware, operating systems, networks, security policy, etc. No assumptions are made regarding these properties. In Grid computing different resources of different organizations are brought together to form a Grid. The people belonging to this Grid have access rights to all the resources available within the Grid. Typical resources are compute servers, data storage facilities and databases but in some cases it could be also special devices like networked telescopes, sensors, etc. Foster et al. defines following key elements for a Grid system [FK04]:

1. **Coordinates distributed resources:** Coordination between resources that are generally not under the control of some centralized device is provided.
2. **Using standard, open, general-purpose protocols and interfaces:** To be able to address issues like authentication, authorization and resource discovery the Grid is built from multipurpose protocols and interfaces.
3. **Deliver nontrivial qualities of service:** In a Grid it is possible to combine different units of resources to deliver various qualities of service. The utility of the combined system is higher than that of the sum of its parts.

2.2. Grid Computing

This section provides an overview of Grid systems. In the first part the evolution of Grid systems is described and in the second one the architecture of Grids is discussed. The term "'Grid"' is known since the mid 1990 and is by Foster and Kesselman in the following definition:

Grid technologies provide mechanisms for sharing and coordinating the use of diverse resources ad thus enable the creation, from geographically and organizationally distributed components, of virtual computing systems that are sufficiently integrated to deliver desired qualities of service.[FK04, p.44]

Foster et al. distinguish four different phases in this evolution of Grids as illustrated in Figure 2.1:

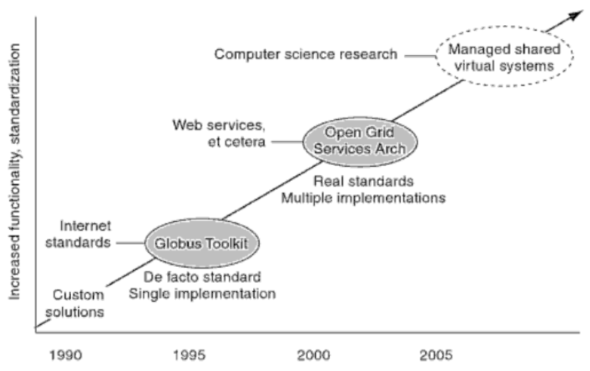

Figure 2.1.: The evolution of Grid technologies (from [FK04], p.44)

- **Custom solution.** This phase started in the early 1990 and the focus was to explore what was possible and how things are working. There were limited functionality in security and scalability and the application were mostly built directly on the Internet protocols.
- **Globus Toolkit.** The Globus Toolkit version 2(GT2)[FK96] emerged and was from then on de facto standard for Grid computing. The Globus Toolkit is a software toolkit used for building grids. GT2 provided solutions for common problems like authentication, resource discovery and resource access. With the help of GT2 it was from now on possible to construct Grid application with defined APIs and protocols.
- **Open Grid Services Architecture.** In 2002 the Open Grid Service Architecture (OGSA), a community standard with multiple implementations, emerged [FKNT02]. In addition to standard interfaces and behaviors which were already defined in GT2, OGSA provides a framework were one can define interoperable and portable services.
- **Managed, Shared Virtual Systems.** The initial technical specification from OGSA was an important step, but much more has to be done to realize the full Grid vision. [FK04] predicts an expanding set of services which address a higher number of entities and smaller device footprints. Furthermore they predict an increasing degree of virtualization, increased quality of service and a richer form of sharing.

2.2.1. Architecture

The Grid architecture proposed by Foster and Kesselman in [FK04] is based on the principles of the *hourglass model* which is depicted in Figure 2.2. The narrow part of the model defines a small set of abstractions and protocols. In the proposed architecture the narrow neck consists of *Resource* and *Connectivity* protocols. They are designed in such a way that they can be implemented on top of various resource types which are defined at the *Fabric* layer. In turn they can be used as a base to construct a wide range of services at the *Collective* layer.

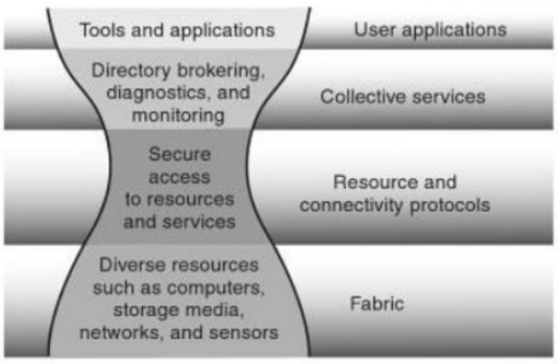
Tools and applications
User applications
Directory brokering, diagnostics, and monitoring
Collective services
Secure access to resources and services
Resource and connectivity protocols
Diverse resources such as computers, storage media, networks, and sensors
Fabric

Figure 2.2.: The layered Grid architecture (from [FK04], p.47)

2.2.1.1. Fabric Layer

The Fabric layer provide access to low level resources such as compute resources, storage resources, network resources and sensors. But a resource could also be a logical entity like a file system or a distributed computer pool. It has to be considered that the implemented functionality on the Fabric layer has an influence on the sharing possibilities. Richer functionality on the Fabric enables more sharing operations, whereas deployment is simplified when there are only a few demands for the Fabric layer. Foster et al. suggest in [FKT01] that resources should implement a minimum of both: *Enquiry mechanisms* that allows discovery of the resource structure and *resource management mechanisms* that provide control of delivered quality of services.
As an example a **computational resource** should implement following mechanisms:

- Mechanisms for starting, monitoring and controlling programs and mechanisms for executing processes.
- Management mechanisms which control the resources allocated to processes.
- Enquiry functions for specifying hardware and software characteristics and functions for gathering informations concerning current load and queue state.

2.2.1.2. Connectivity Layer

Foster et al. summarizes the features of the Connectivity layer as follows:

> *The Connectivity layer defines core communication and authentication protocols required for Grid-specific network transactions. Communication protocols enable the exchange of data between Fabric layer resources.*[FKT01, p.9]

Other responsibilities of this layer are authentication mechanisms and security aspects. Because of the complexity of security problems it is important for any solution to use existing standards. Regarding to authentication, Butler et al. defines in [BEF+00] the following characteristics:

- **Single sign on.** Grid resources defined in the Fabric layer have to be usable after the first authentification.
- **Delegation.** A program should be able to delegate rights to another program.
- **Integration with security solutions.** Interoperations of Grid security solutions with various local security solutions, like Kerberos and Unix must be guaranteed.
- **User-based trust relationships.** If a user has the right to use resources A and B, the user should be able to use resource A together with resource B without interaction of the security administrators of A an B.

2.2.1.3. Resource Layer

The focus of the Resource layer is the interaction with a single resource. Typical responsibilities of this layer are secure negotiation, initiation, monitoring and accounting of individual resources. It is using the communication functions of the Connectivity layer to call directly the functions provided by the Fabric layer to access and control resources [TS07].

The Resource layer is made up of two main protocols. The first one is the *information protocol* which gathers formation about state and structure of a resource. The second one is the *management protocol* having the task to negotiate access to shared resources and perform operations like data access or process creation. The protocols specified in the Connectivity layer and Resource layer form the neck in the hourglass model and thus should be limited to a small set.

2.2.1.4. Collective Layer

Rather than focusing on a single resource like the Resource layer, the Collective layer handles access to a collection of resources. This layer contains protocols and services for resource discovery, resource allocation and scheduling of tasks. The Collective layer is built on the top of the narrow neck in the hourglass model and can consist of many different protocols implementing a variety of behaviors, which in turn may be offered in form of a service to a virtual organization [TS07]. According to Foster and Kesselman [FK04] are directory services, co-allocation, scheduling and brokering services, monitoring and diagnostic services, data replication services, software discovery services and community accounting and payment services, typical services implemented at the Collection layer.

In contrast to Resource layer protocols which have to be general in purpose and are widely deployed, the protocols of the Collective layer have a wide spectrum from general purpose to highly application or domain specific. Figure 2.3 shows one possibility how APIs and SDKs of the Collective and Resource layer can be combined to deliver functionality to the application. In the given example the co-allocation API and SDK uses the resource management API and SDK to manipulate underlaying resources. On top of the co-allocation API and SDK, a co-reservation service protocol is defined and implemented. This co-reservation service again uses the underlaying co-allocation API to provide co-allocation operation and additional functionality. The application than can use the co-reservation protocol to request for example a end-to-end network reservation.

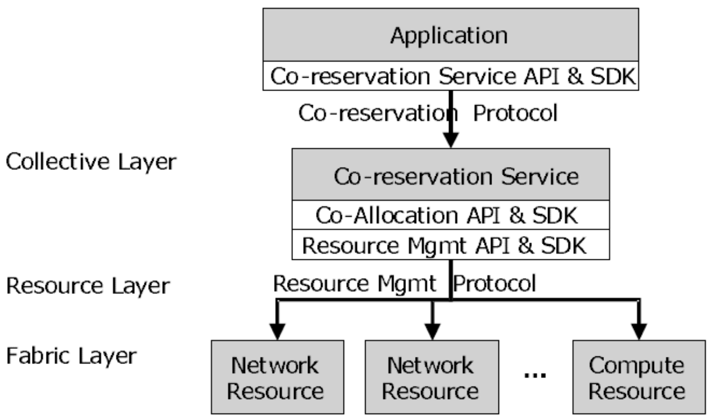

Figure 2.3.: Combination of Collective and Resource layer protocols (from [FKT01])

2.2.2. Classification

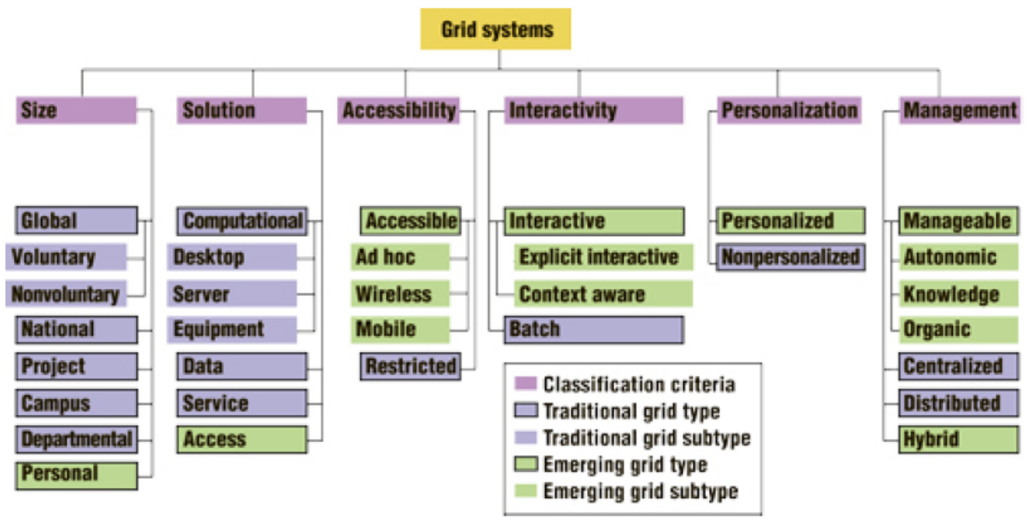

Figure 2.4.: A classification of traditional and emerging Grids (from [KLAR08])

This section give an overview about traditional and emerging Grid systems. It describes the evolution of Grids and categorize upcoming Grids.

The evolution of Grid systems, illustrated in Figure 2.1 and already described took place in three phases. The first generation started in the early 1990s and was marked by supercomputers which sharing resources. The objective of this model of metacomputing was to provide resources to a range of high performance applications. The main issues in the second generation were heterogeneity, scalability and adaptability. To achieve large scale computation it was essential to provide interoperability. This vision of combining different Grid technologies with the help of a Grid middleware was presented in [FK99]. The third generation Grids have evolved out of the necessity for a more service oriented approach. With this new approach it is possible to reuse existing components and information resources to built grid application in a flexible manner.

However, Kurdi et al. claim in [KLAR08] that the architecture of third generation Grids doesn't meet the requirements of next generation Grids (NGG) which is referred to as *Managed, Shard Virtual Systems* by Foster and Kesselman in Figure 2.1. There is a fundamental gab between current technologies and the NGG vision. To fulfill this vision, *pervasiveness* and the *ability to self-manage* are the top two research priorities of the NGG2 Group [Gro04]. Furthermore pervasiveness have been sub-categorized into the four primitive features *accessibility*, *user-centricity*, *interactivity* and *manageability.* Grids which explicitly address this four design features are referred as emergency Grids whereas traditional Grid systems lack these features.

The next sections describe classification categories for traditional Grid systems as well as for emerging Grid systems as depicted in Figure 2.4.

2.2.2.1. Classification by Solution

Grids classified by solution can be split down in following subgroups:

- **Computational Grids.** Computational grids mainly offers CPU cycles. Foster and Kesselman defined a computational Grid as follows:

  > *A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities.*[FK99, p.18]

- **Data Grids.** Data Grids provide an infrastructure to access and manage large amounts of data which are geographically distributed over many repositories. These type of grid systems are often combined with computational grid computing systems.

- **Service Grids.** Service Grids provide services like disk storage or CPU cycles which are purchased on demand by interested parties in the area of scientific computing or enterprise computing.

- **Access Grids.** The technology of Access Grids was developed at Argonne National Laboratory in Chicago and is defined as follows:

  > *The Access Grid is an ensemble of resources and technology which provides a virtual collaboration tool.*[Acc10]

An Access Grid can be seen as an advanced videoconferencing system which allows people from different locations to interact in real time over the Internet. Furthermore it enables participants to collaborate by using a variety of shared application. Additionally to sharing presentation material, large-format displays can be used for multiple video sources to allow room to room conferencing.[Acc10]

2.2.2.2. Classification by Size

According to [KLAR08], Grids classified by size can be split down in following subgroups:

- **Global Grids.** Global Grids provide grid power to organizations or individuals and are established over the Internet. These type of Grids offer an efficient solution for distributed computing by letting users contribute their unused computer power for complex scientific tasks.
- **National Grids.** National Grids are usually government funded and restricted to computer resources within a country.
- **Project Grids.** Project Grids are similar to National Grids but can be geographically distributed over multiple administrative domains. They are only available to cooperating organizations.
- **Campus Grids.** Also called Intra-Grids are restricted to a single organization.
- **Departmental Grids.** Are more restricted than Project Grids and are only available to people within one department.
- **Personal Grids.** Personal Grids are at a very early stage. They have the most limited scope and are only available at a personal level.

2.2.2.3. Classification by Accessibility

Accessible Grids have a highly dynamic nature. The structure of the underlaying organization and VOs change frequently, due to nodes entering and living the network or nodes switching on and off. Accessible Grids make resources available to devices regardless of the physical capabilities and geographical location. The traditional, restricted access Grids don't provide such accessibility because nodes are mostly stationary within a predefined infrastructure. Kurdi et al. define in [KLAR08] three Grid types which support accessibility:

Ad hoc Grids

Amin et al. give following definition of an ad hoc Grid:

> *An ad hoc Grid is a computing architecture offering structure-, technology-, and control-independent Grid solutions that support sporadic and adhoc use modalities.*[ALM04, p.16]

Unlike traditional Grids which have well known Grid entry points like a Web page to get a Grid account, an ad hoc Grid doesn't have any formal entry point. Instead every member of a Grid represents an entry point and nodes can join a Grid as long as they can discover a member in that Grid. Another definition is given by Smith et al.:

An ad hoc grid is a spontaneous formation of cooperating heterogeneous computing nodes into a logical community without a preconfigured fixed infrastructure and with only minimal administrative requirements.[SFF04, p.202]

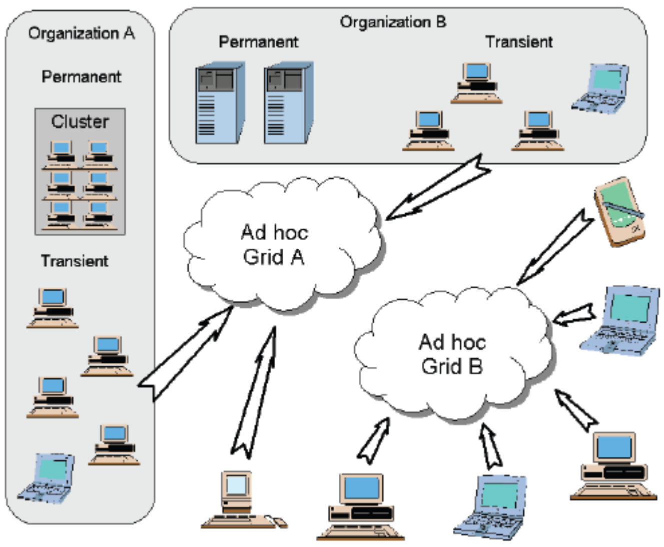

Figure 2.5.: Ad hoc Grid Architecture (from [SFF04])

Figure 2.5 shows two different possibilities to compose an ad hoc Grid. While Grid A includes high performance computers as well as transient nodes, Grid B is composed solely of transient nodes. Ad hoc Grid A bear resemblance to traditional Grids while ad hoc Grid B shows the shift to a more personal Grid system without resources of large organizations. Smith et al. highlight four main challenges when building ad hoc Grids in a heterogeneous environment:

- **Node Discovery**
  Due to the fact that the network topology is of dynamic nature in ad hoc Grids, the appearance and leaving of a new node should be detected as fast as possible. However, a balance between keeping the network structure up to date and flooding the ad hoc network with discovery messages has to be found.

- **Node Property Assessment**
  To be able to deploy services to nodes in a heterogeneous environment meta information of node wanting to participate in the Grid must be available. These are information like operating system type, hardware resources and required libraries.

- **Service Deployment**
  Because of the fluctuating availability of the nodes in an ad hoc Grid, manual deployment of services is time-consuming, difficult to manage and therefore not feasible. Service deployment has to take place on demand and has to become part of the Grid application itself. The application has to have the ability to perform the deployment of a service to a new discovered node and to use the node afterwards for its application flow.

Project name	**Website**
The Marburg Ad-hoc Grid Environment	http://mage.uni-marburg.de/
OurGrid	http://www.ourgrid.org/
MyGrid	http://www.mygrid.org.uk/
Java CoG Kit Ad hoc Grid framework	http://www.globus.org/cog/java/
ASG Ad hoc Service Grid	http://www.kbs.tu-berlin.de/menue/forschung/projekte/asg_-_ad_hoc_service_grid/

Table 2.1.: Ad hoc Grid projects

- **Service Security**
  In traditional systems only a few people are able to install services on nodes, whereas in ad hoc Grid systems potential lot of users, everyone unknown to each other, can operate on the same node. Therefore in ad hoc Grids several new security aspects must be considered.

Wireless Grids

> *Wireless grids, a new type of resource-sharing network, connect sensors, mobile phones, and other edge devices with each other and with wired grids. Ad hoc distributed resource sharing allows these devices to offer new resources and locations of use for grid computing.*[MHB04, p.24]

Usually in a wireless Grid devices are acting as real nodes in the Grid system and provide computing power and data storage to the Grid. A special type of a wireless Grid, a so called Wireless access Grid is a Grid system where all wireless devices are pure access points to the Grid, that means they don't contribute any computing power or storage capacities (Figure 2.6). If resources are required by these nodes, they acquire them from the backbone grid.

Agarwal et al. proposes in [ANG04] to classify wireless Grids according to the architecture or according to the function of the Grid. The two main characteristics to classify the architecture are the degree of heterogeneity of the devices and the level of control. When doing so the following three classes of wireless Grids arise as depicted in Figure 2.7.

- **Local Cluster or Homogeneous Wireless Grid**
  Represents the simplest form of a wireless Grid. All devices are homogeneous, have the same hardware architecture, and the same operating system installed. Due to the homogeneity of all nodes the integration of the devices into the Grid and the administration of the resources will be an easier task. This systems are likely to be found in organizations with a single administrative domain. An example could be a Grid system in a hospital, where many mobile devices (handhelds) are used to coordinate medical personnel.

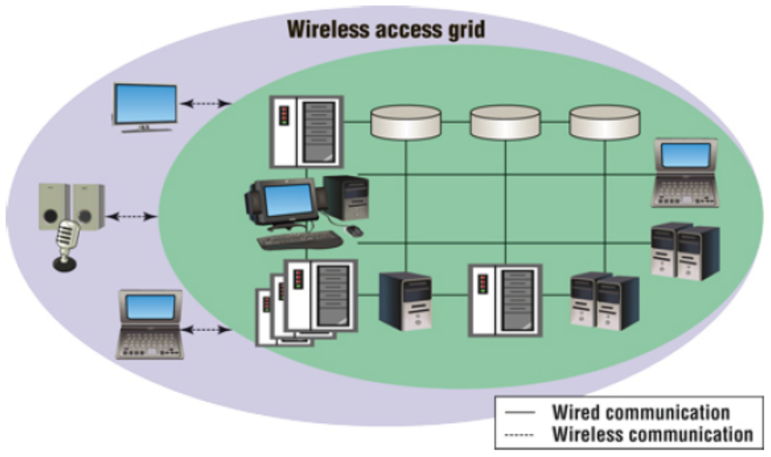

Figure 2.6.: Wireless Access Grid Architecture (from [KLAR08])

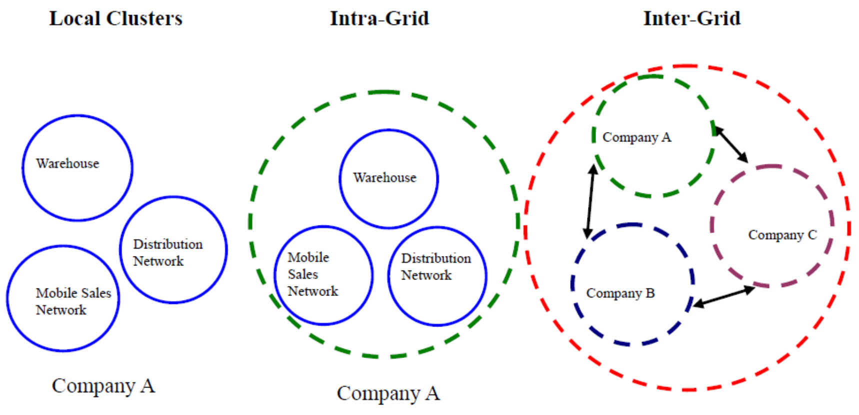

Figure 2.7.: Classification of Wireless Grids (from [ANG04])

Grid Type	Possible Architecture	Mainly Provides
Computitional	Cluster, Intra, Inter	computational power
Data	Cluster, Intra, Inter	data access and storage
Utility	Intra, Inter	On-demand access to all kinds of resources

Table 2.2.: Wireless Grid Usage Pattern [ANG04]

- **Wireless Intra-Grids**
  An Intra-Grid includes wireless devices which belong to different local clusters. This local clusters or divisions may be located in different geographical areas and in different administrative domains. An example could be a wireless Grid in a company that supports two different divisions. One could be the sales devision and the second one the manufacturing division where wireless sensors, controlled by the Grid, are used for tracking inventory.

- **Inter-Grid**
  An Inter-Grid includes multiple actual organizations (AO) and span a larger geographical or organizational area. When multiple AOs come together a Virtual Organization (VO) is formed and different AOs can collaborate to share resources and knowledge. Inter-Grids gain a higher complexity due to the scalability requirement. New security arrangements and policies have to be introduced because the system operates across multiple organizations. An example could be a scenario where an Austrian tourist visits China and tries to use a local e-commerce service with his PDA. A possible transaction would involve multiple organizations like the service provider, traveler's credit card company, the Chinese wireless service provider and the e-commerce provider.

As shown in Table 2.2, wireless Grids can also be classified by usage patterns. *Computational wireless Grids* are used when the power constraints limit the computational resources of a device and more devices are needed to fulfill a given purpose. An example would be a wireless sensor network to monitor conditions for a earthquake prediction system. *Data wireless Grids* are needed to provide secure and shared access to distributed data. An example would be an urgent search of a donor with a rare blood type. The hospital would make an request to the medical database in the vicinity to find a potential donor. Potential donors then will get an alert message by the local service provider. *Utility wireless Grids* can be used to provide special pieces of software or hardware in an on-demand manner. Consumer can request resources when needed and get charged on a consumption basis. An example would be Grid system where users can request information about traffic conditions and routing or for using e-commerce products and services.

Mobile Grids

In the past mobile devices where just marginal relevant to Grid computing because of their limitations in computing power, persistent storage, battery lifetime, screen size and

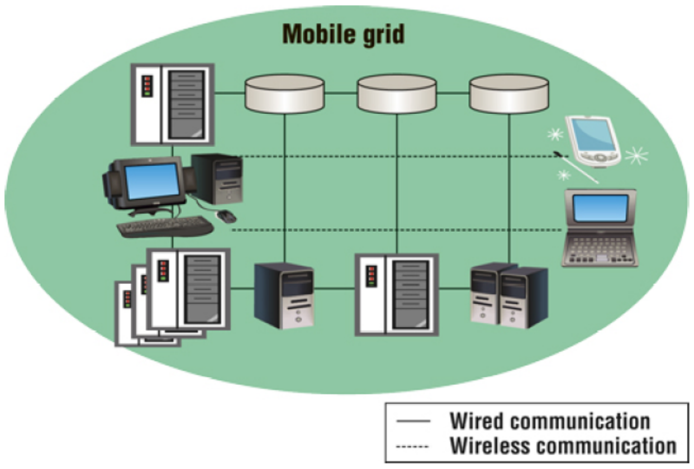

Figure 2.8.: Mobile Grid Architecture (from [KLAR08])

connectivity [LSV04]. But of the huge amount of devices sold each year, mobile devices became attractive for Grid systems. As shown in Figure 2.8 these devices are participants of the Grid and provide computational or data services to the Grid. Another aspect underlining the importance of wireless devices is their usage as communication and computation devices in natural disaster and on battlefield scenarios, where no other devices might be feasible.

The general Grid concept will change and new functionalities of the Grid will be needed to meet the capabilities of the mobile Grid. Solutions are needed for interoperability issues between diverse technologies as well as for new Quality of Service (QoS) concepts and security concepts. Litke et al. define a mobile Grid as follows:

> *Mobile Grid is a platform that should address mobility issues by means of enabling both fixed and mobile users to have access to both fixed and mobile Grid resources utilizing transparently and efficiently the underlying technologies.*[LSV04, p.1]

Furthermore Litke et al. present in [LSV04] four challenges to meet the requirements of the new computing paradigm.

- **Resource Discovery and Selection**
  Due to the dynamic environment of mobile Grids new mechanisms for resource discovery and selection are needed. Metadata such as time constraints and resource constraints (resource accessibility, system workload, network performance) have to be taken in account. Also the financial criterion has to be considered for a proper resource selection to enable customers to use the Grid to solve everyday life problems.

- **Job Scheduling**
  Job scheduling in a mobile Grid environment is much more complicated due to the higher number of constraints (QoS, fault tolerance, security etc). To optimize the

job scheduling mechanism not only the performance of a resource is crucial, but also the availability of the resource and the reliability of the resource to provide the QoS constraints over the full job execution time.

- **Resource Discovery and Selection**
  Due to the dynamic environment of mobile Grids unpredictable changes like network failures or system performance degradation can occur. In case of such an unpredictable behavior it is still important to meet the requirements of the user. Job monitoring, job migration, job rescheduling and job replication are measures which help to fulfill this requirements.

- **Replica Management for large Data Sets**
  Large data sets may be distributed over many physical locations with different QoS constraints. Replica management is important because there is a need to keep track of the different portions of data. Information about data replication and the replicas storage location are of significant importance because storage devices can change their connection status constantly and even their physical location can change.

Project name	**Website**
akogrimo	http://www.mobilegrids.org/
MGS Project	http://appsrv.cse.cuhk.edu.hk/ kwng/mgs/mgs.htm
InviNet	[MMCA02]

Table 2.3.: Mobile Grid projects

2.2.2.4. Classification by Interactivity

Video gaming and real-time embedded control systems are new application areas in NGGs. The traditional communication model is not meant for online activity and fast response times. Interactivity in Grids can be realized at two layers [KLAR08].

In the **web portal layer** approach jobs are submitted to a secure shell process [HHX$^+$05], whereas in the **middleware layer** approach jobs are submitted directly to the middleware which has an extension to support interactivity. But this is not the only possible sort of interactivity in Grids. Another possible form of a Grid would be a context-aware Grid where the Grid is interacting with the surrounding by sensors and actuators. Table 2.4 shows some projects dealing explicitly with the interactivity in Grid systems.

2.2.2.5. Classification by User-centricity

The majority of traditional Grid systems has been designed for people involved in research or for specific goals in large industry domains. For individuals outside this domain it is not possible to construct or even use Grid system for their own problems. Hence most

Project name	**Website**
CrossGrid	http://www.eu-crossgrid.org/
eduatain@grid	http://www.edutaingrid.eu/
RUNES	http://www.ist-runes.org/
SENSE	http://www.sense-ist.org/
MORE	http://www.ist-more.org/

Table 2.4.: Grid projects which address interactivity

traditional Grids are nopersonal Grids. Personalized Grids on the other hand are emerging Grid systems which focus on the needs of a systems's user. Kurdi et al. [KLAR08] uses the term *user-centric grids* to refer to **personalized Grids** and **personal Grids**.

Personalized Grid systems offer their users a highly customizable Web portal. The users, belonging to different domains, can adapt the whole system to their needs. A personal Grid is a system where individuals not only make their own resources available to the Grid, but also take advantage of the resources in the Grid for their personal purpose. Han and Park specify in [HP03] four design requirements for personal Grids:

1. A personal Grid must be self-organized without complicated configuration and infrastructural support.
2. A personal Grid must be lightweight to admit even low-powered computers.
3. A personal Grid must provide convenient interface that enables users to easily access the system.
4. A personal Grid must provide a reusable service framework for promoting the participation of users who are not familiar with computer programming.

Project name	**Website**
akogrimo	http://www.mobilegrids.org/
myGrid	http://www.mygrid.org.uk/

Table 2.5.: Grid project which address personalization

2.2.2.6. Classification by Manageability

Because todays Grids are very complex, managing and organizing this system became a real challenge. A manageable Grid is a new type of Grid focusing on a simplified configuration and administration of the whole system. A simplified management of the Grid enhances the scalability and management costs get reduced. Kurdi et al. [KLAR08] subcategorize manageable Grids into autonomic Grids, knowledge Grids and organic Grids.

Autonomic Grids came from the term *Autonomic Computing* which is defined by IBM as:

> *computing systems that can manage themselves given high-level objectives from administrators.*[KC03, p.41]

Furthermore there are four aspects of self-management within autonomic computing [KC03]:

1. *Self-configuration.* On the basis of high-level policies the configuration of the system happens automatically.
2. *Self-optimization.* The system strive continually for opportunities to improve their own performance and efficiency.
3. *Self-healing.* The system automatically detects and repairs local software and hardware problems.
4. *Self-protection.* The detection of and recovery from attacks is automatic. The system uses early warning to foresee and prevent system wide failures.

The aim of a **knowledge Grid** is to move a Grid away from the traditional computation and data management services towards an infrastructure where resources and services have well-defined meanings to both, machines and humans. Examples of knowledge Grids can be found in Table 2.6.

Organic Grids refer to a new concept for desktop Grids where a decentralized P2P approache, mobile agents and a distributed scheduling scheme plays an important role. Research on organic Grids is still in the initial stage.

Project name	**Website**
OptimalGrid	http://www.almaden.ibm.com/cs/projects/optimalgrid/
OntoGrid	http://www.ontogrid.net/ontogrid/index.html
InteliGrid	http://inteligrid.eu-project.info/
K-Wf Grid	http://www.kwfgrid.eu/
Semantic Grid Research Group	http://www.semanticgrid.org/OGF/

Table 2.6.: Grid project which address manageability

2.2.3. Virtual Organization

Because Foster and Kesselman define a Grid

> *as a system that coordinates distributed resources using standard, open, general-purpose protocols and interfaces to deliver nontrivial quality of service,* [FK04, p.46]

one key aspect of Grid Computing is to couple heterogeneous computing systems across different administrative domains to increase the efficiency of computing tasks. To achieve this increased efficiency it is important to enable the usage of shared resources which in turn lead to the approach of Virtual Organizations (VO). Bird et al. define the original idea of a VO as:

> *a dynamic group of users with a common goal coming together for a specific, short-lived collaborative venture and then dissolving.*[BJK09, p.41]

They note that this idea has never been realized because due to the complexity of deployment and authorization of such a dynamic structure. A typical VO tends to be a comparatively long-lived organization that brings together a group of researchers with a common purpose. Because of the complexity associated with the management and organization of Grid infrastructures through VOs a lot of problems might arise and whole projects can fail. There are other definition to help to understand why it is difficult to manage VOs. Desanctis and Mong gave the following definition for a VO:

> *A virtual organization is a collection of geographically distributed, functionally and/or culturally diverse entities that are linked by electronic forms of communication and relay on lateral, dynamic relationships for coordination.*[DM99, p.674]

A few years later Foster et al. defined a VO as

> *flexible, secure, coordinated resource sharing among dynamic collections of individuals institutions and resources.*[FKT01, p.1]

In June 2008 the US National Science Foundation's office of Cyberinfrastructure made a study on VOs as sociotechnical systems and defined a VO as follows:

> *A virtual organization is a group of individuals whose members and resources may be dispersed geographically, but who function as a coherent unit through the use of cyberinfrastructure.*[Fou08]

Virtual organizations may be known by a range of different names but all have common characteristics, e.g. distributed across space, distributed across time, dynamic structures and processes, computationally enabled and computationally enhanced. [Fou08]

Virtual organizations have a high potential to change the way how computers are used to solve problems. Foster et al. highlights in [FKT01] four different scenarios where virtual organizations are formed to obtain a common goal, by the cooperation of multiple organizations. Figure 2.9 depicts a sample scenario with three actual organizations and two VOs. The first VO, P, links participants in an aerospace design consortium and the second VO, Q, links people together who have agreed to sharing computing power. The organization on the left side is part of P, the one on the right side is part of Q and the third is a member of both virtual organizations. Furthermore there are policies controlling the access to resources within an organization.

Resource sharing is conditional. Each organization owning resources makes them available depending on their location, the required functionality and the expected timeframe.

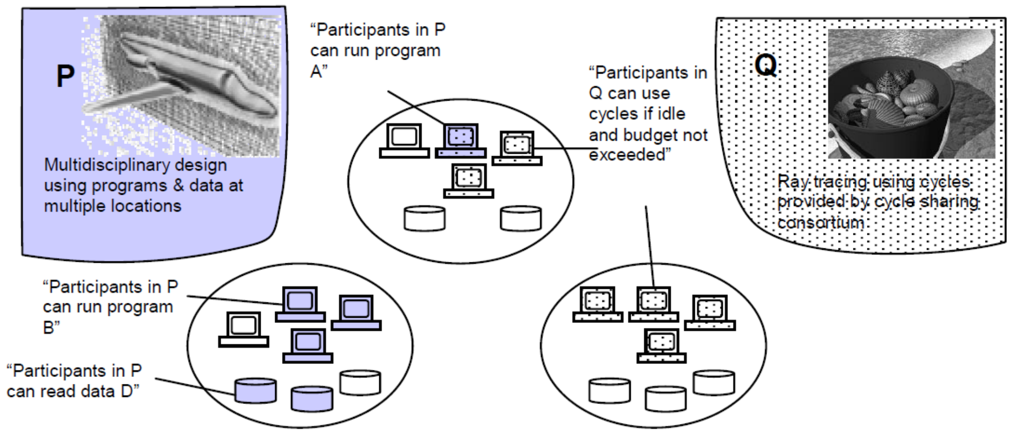
P
Multidisciplinary design using programs & data at multiple locations
"Participants in P can run program A"
"Participants in Q can use cycles if idle and budget not exceeded"
Q
Ray tracing using cycles provided by cycle sharing consortium
"Participants in P can run program B"
"Participants in P can read data D"

Figure 2.9.: Composition of virtual organizations (from [FKT01])

In Figure 2.9 for example the organization in the middle can restrict its resource *computer cycle* for participants in P, to resolve only *simple* problems. But also resource consumers can place constraints on resources they a supposed to work with. A participant in Q might for example only accept resources which are certified as *secure*.

2.3. Cloud Computing

Cloud computing describes a concept where applications are running in the Cloud rather than running on the local computer. The applications are provided and maintained on centralized facilities. Foster et al. gives the following definition for Cloud computing:

> *A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are deliver on demand to external customers over the Internet.*[FZRL08, p.1]

According to this definition the main differences between Cloud Computing and traditional distributed systems are the massive scalability, different levels of services deliverable to customers outside the Cloud, the stimulus by economies of scale and dynamic reconfigurability of services.

Foster et. al further wrote that Cloud computing educed out of Grid computing and overlaps with many already existing technologies like utility computing, service computing and of course distributed computing in general. There has been a shift in focus. While Grids mainly provide a infrastructure that delivers computing power and storage resources, the focus of Clouds is economy based and delivers more abstract resources and services. Figure 2.10 shows the relationship between Clouds, Grids, Supercomputers, Clusters and distributed systems in general. While Supercomputers and Clusters focus on traditional non-service oriented applications and lie on the left side of the scale, Clouds focusing clear on service oriented applications. Grids lie in the middle between non-service applications and service oriented applications.

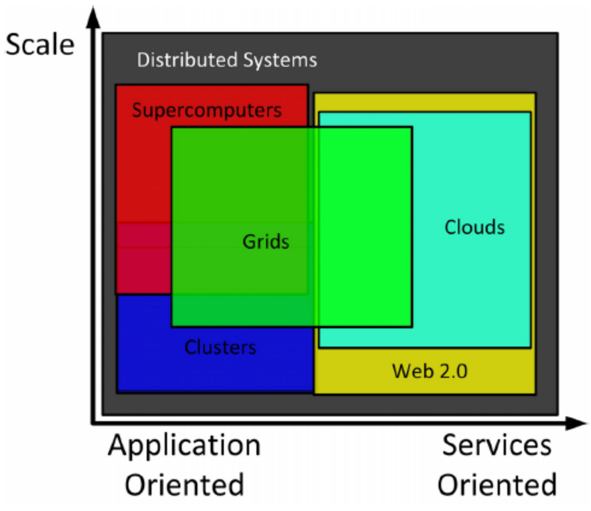

Figure 2.10.: Grids and Clouds Overview (from [FZRL08])

2002 Ian Foster wrote in [Fos02] a checklist to indicate what a Grid is an what not:

- *A Grid coordinates resources that are not subject to centralized controls*
- *A Grid uses standard, open, general-purpose protocols and interfaces*
- *A Grid delivers nontrivial qualities of service.*

Point three holds for Cloud computing because also clouds delivers qualities of service which satisfy complex user demands. But either point one nor point two holds true for Cloud computing.

The **business model** for traditionally software has been a one-time payment for unlimited use. In the business model for cloud computing the customer will pay on a consumption basis. He will pay the provider for his service. If the provided service is for instance a Compute Cloud the customer will get charged per instance-hour consumed for each instance type. And if the provided service is a Data Cloud the customer will pay for the data transfer. With this business model it is possible to get on demand access to 100.000 processors and data centers which are distributed throughout the world with just a credit card [FZRL08].

2.3.1. Architecture

The large pool of different resources provide by a Cloud can be accessed via standard protocols and standard interfaces. Web Services and some web 2.0 technologies like RSS and AJAX can be used as underlying protocol layer for a Cloud architecture. It is also

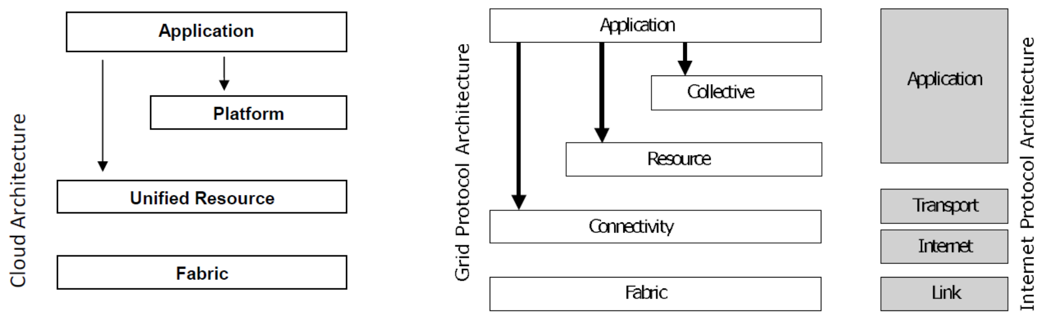

Figure 2.11.: Cloud- and Grid- Architecture compared (based on [FZRL08] and [FK04])

possible to built a Cloud on top of existing Grid technologies.

A four layer architecture for Cloud Computing is defined by [FZRL08], which is compared with the five layer Grid architecture in Figure 2.11. The **fabric layer** is the bottommost layer in this architecture and contains hardware level resources like compute resource and storage resource. The **resource layer** contains encapsulated resources. The encapsulation/abstraction takes place usually by virtualization. This resources can be provided to upper layers as integrated resources. The **platform layer** adds special middleware, tools and services on top of the resources. This layer provides a development/deployment environment for the overlaying application layer. This platform could be a hosting environment or a scheduling service. Finally the **application layer** contains the application which run in the Cloud.

Three different categories of services are delivered by Clouds according to [Wha10]. In the **Infrastructure-as-a-Service (IaaS)** category, an enterprise provides hardware at a resource usage-based pricing model. Customers use a application programming interface (API) which is provided by the enterprise to access and configure their hardware resources like virtual servers and storage. A company pays only as much capacity is needed and it is possible to dynamically scale up and down the resources. Because of his dynamically nature the pricing model is also called the ”’pay-for-what-you-use-model”’. The **Platform-as-a-Service (PaaS)** category defines development tools and software which is hosted by the provider. Customers create applications over the Internet directly on the provider's platform. Google's App Engine is an example of PaaS. The Google App Engine lets customers run their own web applications on Google's infrastructure. In the **Software-as-a-Service (SaaS)** category the company provides the hardware infrastructure as well as software which is remotely accessible by the customers through the Internet. The end user can use the service from anywhere because both the application and the data are hosted by the provider. In this category the pricing model is also a usage-based model.

2.3.2. Resource Management

Resource management is an important topic in order to understand the challenges which Clouds face today. Following topics are mentioned in [FZRL08]:

Compute Model

The compute model of a Cloud looks very different than that of a Grid. In most Grids a local resource manager (LRM) is responsible of the compute resources in a Grid. For instance, if a job needs 60 minutes do be done on 100 processors, the job will stay in the queue of the LRM until the 100 processors are available for 60 minutes. Because of this potentially long queuing times, Grid are usually not supporting interactive applications. The computing model of a Cloud in contrast has to support interactivity. Hence it is possibly for latency sensitive applications to operate on the Cloud. But ensuring a high level of QoS to the end user is not trivial and is one of the major challenges for Cloud computing.

Data Model

While other people predicate that Cloud computing is the future of Internet computing and all kinds of resources like storage, compute, etc. will be provisioned by the cloud, Foster et al. predict in [FZRL08] a **triangle model** which is depicted in figure 2.12. They are convinced that both Client computing and Grid computing coexist and develop hand in hand. It is mentioned that client computing can not be overlooked. People might have security concerns and therefore they don't want to run mission critical applications on the Cloud. Another reason might be that users want get their things done even when the Internet is down.

Data Location

The location of data is a major challenge regarding the scalability of an application. It is becoming a bottleneck if data is moved further away from the computing CPUs. Data provision on a local disk compared to data provision over a wide area network is indicated by a large speed difference. In order do not drastically affect the application performance it is of immense importance that data is distributed over many computers and the application fetches the data from the best place. In this way it is possible to minimize the communication costs.

Virtualization

Due to the fact that thousands or even millions of user applications can run on the Cloud and each applications appear to the user as if they were running concurrent, virtualization is absolutely essential for almost every Cloud. Virtualization provides abstraction to the underlaying hardware layer (fabric) and also provides encapsulation to applications so that they can be configured and controlled independently. Additional indications for virtualization within Clouds are server and application consolidation, configurability which can be not archived at the hardware level, increasing application availability and improving responsiveness.

In the past, virtualization had significant performance drawbacks for some applications. But processor manufacturers worked hard the past few years to minimize the performance gap between traditional compute resources and virtualized ones.

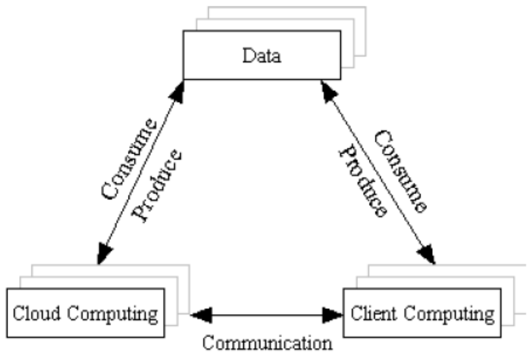

Figure 2.12.: Triangle model (from [FZRL08])

2.4. You-R® Open

You-R® Open is an RFID middleware developed by RF-iT Solutions GmbH. With the You-R® Open middleware it is possible to handle data, device and configuration tasks within the RFID operation environment. Furthermore it considers integration aspects into an exiting ERP and IT landscape. Because of the high number of different low level devices such as RFID readers (UHF, HF, LF), barcode scanners, label printers etc. and the fact that all interfaces are unstandardized it is useful to implement a middleware. These aspects are addressed by You-R® Open which furthermore provides tools to develop, deploy, operate and maintain the whole RFID infrastructure.

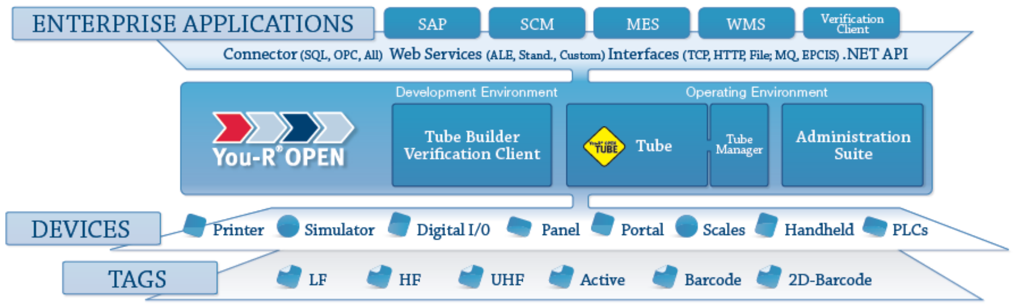

Figure 2.13.: Structure of the You-R® Open middleware [Ri09]

2.4.1. EPC Global

EPCglobal is leading the development for the standardized Electronic Product Code (EPC) network. The EPC network supports the Radio Frequency Identification (RFID) in today's trading networks. This global standards based technology combines radiofrequency tags, existing communication networks and the EPC to provide cost efficient, real-time based information about the current location of products. EPCglobal cooperates with customers and numerous technology providers to standardize all components within the EPCglobal network [EPC04]. The EPCglobal network is comprised of following components.

- **Electronic Product Code (EPC).** The Electronic Product Code is a unique number that identifies a specific object in the supply chain.
- **ID System.** There are two parts which form the ID System, namely EPC tags and EPC readers. An EPC tag is a microchip with an antenna. The EPC is stored on the tag and the tag is applied to items, cases and/or pallets. The EPC reader is an communication device which reads the information from the EPC tag with the help of radio waves. The information from the tag are delivered to a business information system using EPC middleware.
- **EPC Middleware.** The EPC Middleware enables data exchange between the EPC readers and a business information system. Furthermore it manages real-time read events, provides alerts and manages the communication to the EPC Information Service.
- **Discovery Services.** Is a service which supports the users of the system to gather information on a specific EPC. The access to this information is managed by the EPC IS and only authorized users are able to access these information. The Object Naming Service (ONS) is one component of the Discovery Service.
- **EPC Information Services(EPC IS).** The EPCglobal network is a cooperation of numerous servers which maintain information of products labeled with an EPC tag . These servers are combined to a network which provide the EPC IS. Each member of the network stores information to a certain EPC on his server and other members can access this information at a later point in time.

2.4.2. Tube

Tubes are the integral component of the You-R® Open RFID middleware and can be seen as devices which communicate on one side with the RFID reader and on the other side with enterprise IT-systems. The Tube component supports LF, HF, UHF RFID devices, barcode devices, simulation devices, built-in connectors to databases and it can be executed on servers, local workstations and hand-held devices. Figure 2.14 shows the architecture of a Tube with the different layers of abstraction. Functionalities like filtermechanisms for data, EPC pattern and health status services are implemented on these layers. The configuration of the different services of the layers and the implementation of the control logic can be done by using the You-R® Open Tube Builder.[Gmb07]

Beside the Tube the You-R® Open RFID middleware is made up of the following components:

- **You-R® Open Administration Suite.** The Administration Suite is an application to monitor and configure the software and hardware in a You-R® Open project.

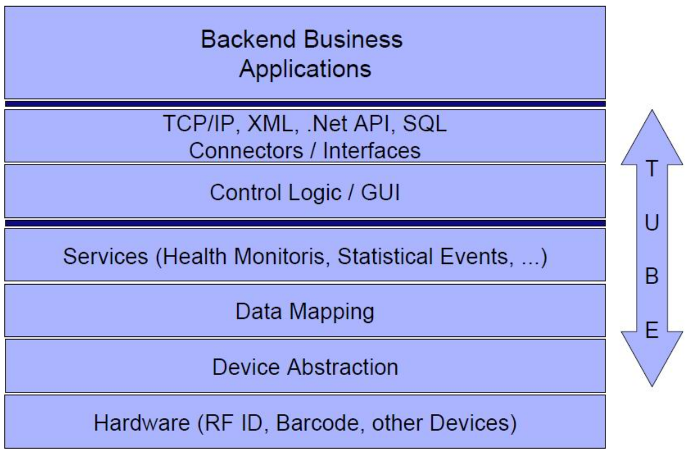
Backend Business Applications
TCP/IP, XML, .Net API, SQL Connectors / Interfaces
Control Logic / GUI
Services (Health Monitoris, Statistical Events, ...)
Data Mapping
Device Abstraction
Hardware (RF ID, Barcode, other Devices)
T U B E

Figure 2.14.: Layers of a You-R® Open Tube [Gmb07]

- **You-R® Verification Client.** The Verification Client is an application which checks the functionality of Tubes. The Verification Client is connected to Tubes via standard web services and has the possibility to switch between different data models and to manipulate data on the tags.
- **You-R® Open Tube Manager.** In combination with the Administration Suite, the Tube Manager is responsible to manage the Tubes. The Tube Manager hosts all Tube specific programs and configuration, is responsible for starting and stopping Tubes on the host and is in charge for data transfers between a mobile device and the Administration Suite
- **You-R® Open Tube Builder.** The Tube Builder is an graphical Application to built and configure Tubes. Either it is possible to use one of the numerous predefined Tubes which can be customized for a particular task or it can be built a Tube from scratch.

2.5. Model Driven Software Development

The concept of models is not new in software development and became even more popular with the existence of the Unified Modeling Language (UML). At the beginning the relationship between models and the resulting code was never formal. As a consequence models were just used for documentation purposes. This formal inexistent relationship has the disadvantage that the documentation with the models has always be kept up to date with the current implementation of the software. Many programmers consider therefore models as an overhead. Model driven software development (MDSD) has a different approach. Models are not only used for documentation purpose, but are considered equal to code. The implementation of the code is automated and happens out of the formally defined model.
Some important goals of MDSD are:

- With MDSD the development speed can be increased through automation. Code can be generated when transforming formal models.

- Software quality can be increased by automated transformations of formally-defined modeling languages.
- A higher level of reusabilty can be achieved.
- Through abstraction the complexity of software get reduced and manageability get improved.

A more comprehensive list can be found in [SV06].

The Object Management Group (OMG) introduced 2000 in [Gro00] the first time the Model Driven Architecture (MDA) as an MDSD approach. Figure 2.15 shows the concept of MDA. The center of the figure depicts the core of the architecture which is based on different modeling standards by OMG. These standards are UML, MOF (MetaObject Facility) and CWM (Common Warehouse Metamodel). When constructing an MDA-based application the first step will be the creation of a platform independent application model. This application model will be expressed with an appropriate core model. After an general application model is defined, platform specialists will convert the general model into a platform specific one. Figure 2.15 shows the target platforms in the thin ring around the core.

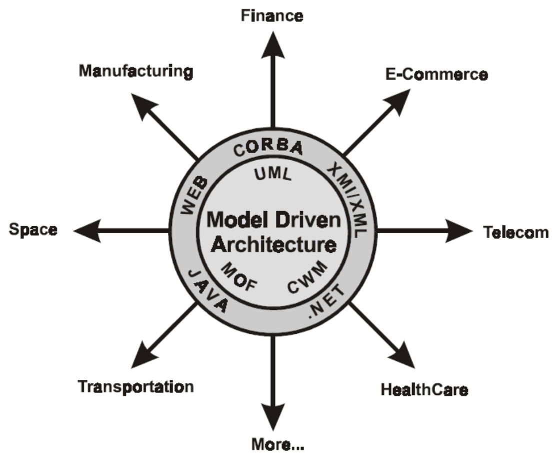

Figure 2.15.: Model Driven Architecture by OMG (from [Gro00], p.4)

Figure 2.16 shows the basic principle of MDA. At the top of the picture there are domain-related specifications which are defined with the help of Platform-Independent Models (PIMs). To model such a domain, a modeling language is needed, which is in most of the cases UML (adapted via profiles). A domain specific description has the advantage that it is completely independent of the following implementation on the target platform. Platform-Specific Models (PSMs) are created through a model-to-model transformation out of the platform independent model, which usually takes place automated and tool-supported. Code for a specific platform can be created, first, through model-to-model transformation based on different PSMs, and then through a model-to-code transformation which is depicted in Figure 2.17.

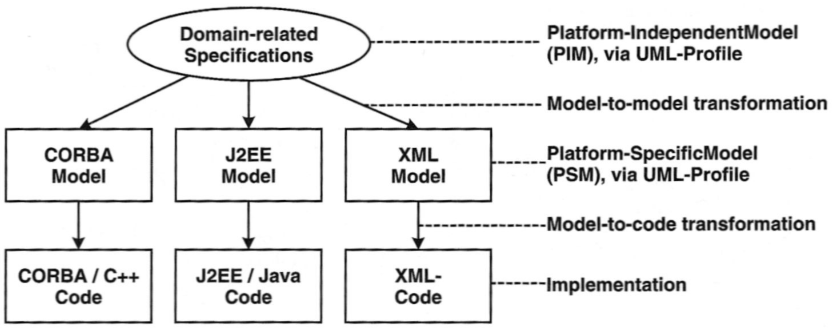
Domain-related Specifications
Platform-IndependentModel (PIM), via UML-Profile
Model-to-model transformation
CORBA Model
J2EE Model
XML Model
Platform-SpecificModel (PSM), via UML-Profile
Model-to-code transformation
CORBA / C++ Code
J2EE / Java Code
XML-Code
Implementation

Figure 2.16.: The principle of MDA (from [SV06], p.16)

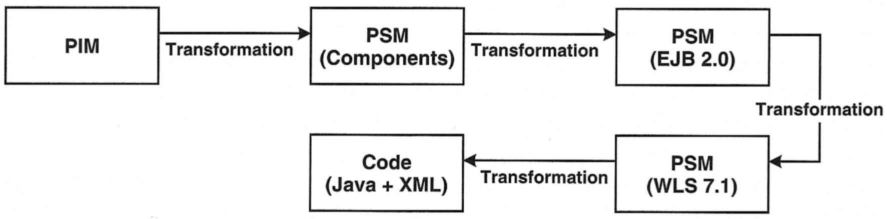

Figure 2.17.: PIM, PSM and transformation (from [SV06], p.17)

2.5.1. Metamodeling

Metamodels are an important aspect in MDSD. They describe in an abstract way the structure of models. A model on the one hand is an abstraction of a concrete thing in the real world, a metamodel on the other hand is a model which can be used for modeling a model. A metamodel defines the syntax for the modeling language as well as modeling rules and constraints. To define a model language formally, a metamodel is required which in turn needs a metamodeling language. Again to be able to define a metamodeling language formally, a meta meta model would be necessary. In order not to have to continue this abstraction steps to infinitum, OMG defined a hierarchy of four levels as depicted in Figure 2.18.

Stahl and Völter describe the four challenges for metamodeling [SV06]. **Domain-specific modeling languages** (DSL) are needed to operate within the context of a specific domain. The syntax of a DSL is described via a metamodel. **Model validation** has to be done against the constraints which are defined in the metamodel. With the help of mapping rules **transformations** are possible between two metamodels (model-to-model transformation). And finally **integrated modeling tools** are needed to support the process of transformation and code generation.

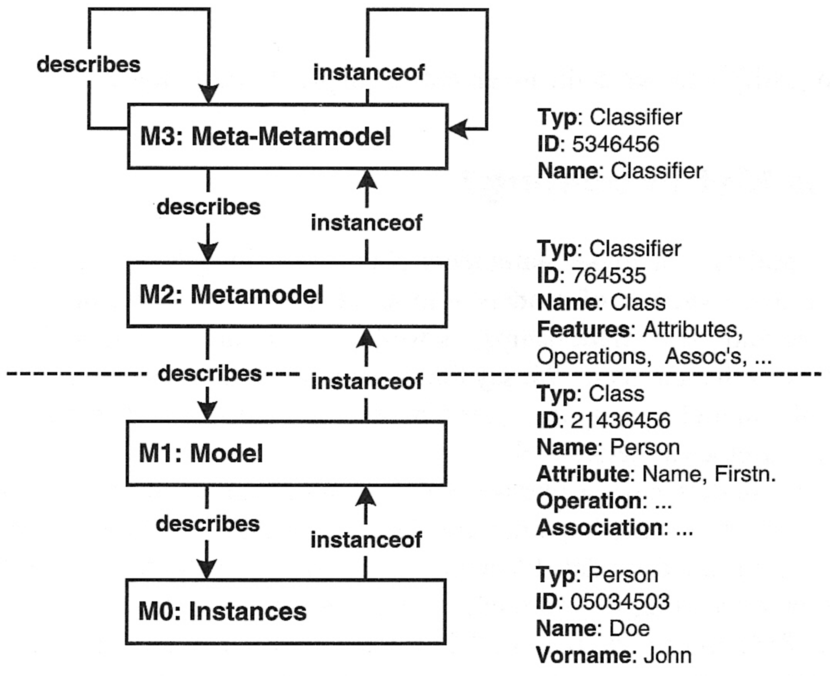

Figure 2.18.: The four metalevels (from [SV06], p.86)

Although models can be described in any modeling language, UML plays an important role because it is used for modeling purposes in many cases today. Hence it is important to consider UML when talking about metamodeling.

The Object Management Group (OMG) defines four metalevels in context with MDA (Figure 2.18). The first two levels, M0 and M1 are well known within object oriented software development. M1 could define for example a class diagram. In the class diagram there might be a particular class with the name *Person* and attributes like *forename* and *surname*. Then at runtime a concrete instance of a class could be created. In the example shown in Figure 2.18, the instance is of the type *Person* and has the forename *John* and the surname *Doe*.

When we consider the levels above the dashed line the things become more abstract. Because the model (class diagram) at M1 is defined via the UML language, M2 must be the UML metamodel. In the metamodel, the elements and constructs which are used at M1 must be defined. Hence all elements in the M1 model are instances of the M2 metamodel. When we consider the example, the type *Class* which is used at the metalevel M1 must get defined at the metalevel M2, which is in that case the UML metamodel. The construct *Class* which is defined in the UML metamodel is of the type *Classifier*. The type *Classifier* again is defined a level above in the meta-metamodel.

The meta-metamodel is defined by the OMG's MetaObject Facility (MOF). The purpose of MOF is to have the power to define different modeling languages at the M2 level. UML is an example of an modeling language at the metamodel level. With the help of MOF it is possible to define additional, standardized modeling languages which address for example only a specific domain. Furthermore it is also feasible to define non object oriented modeling languages. Above the MOF, no further levels exists. The MOF defines basically itself. The full MOF standard can be found under [Gro06].

As mentioned before, UML is an instance of the MOF, but actually UML existed before the MOF was defined. UML was primary only defined verbally. MOF was then later constructed for the purpose to define UML formally. As noted, based on MOF it is possible to define new modeling languages on M2. But rather do define a completely new language on M2 it is much more easier to take the UML model and extend it to specific needs.

Stahl and Völter define three possibilities to extend the UML metamodel [SV06]:

- Extension based on the UML's formal metamodel.
- Extension using stereotypes/profiles (UML 1.x)
- Extension using stereotypes/profiles (UML 2)

Figure 2.19 depicts the case where a new metamodel is defined through inheritance of the UML-metamodel. The extension of metamodels through inheritance is not only limited to UML, it is also possible with other modeling languages which are based on the MOF. The extension based on stereotypes and profiles in contrast is only possible with the UML metamodel because these are mechanisms which are specific to UML. Another important point which is shown in Figure 2.19 is that there is no difference in the metalevel when the metamodel is extended via inheritance.

2.5.2. Entity Container

Data persistency plays a fundamental role in a modern software development process. To improve the quality of systems and to reduce development costs some kind of support of data persistency should be provided to the object oriented programming paradigm and respectively to component based development.

Because today many enterprise applications are using Relational Database Management Systems (RDBMSs) to handle persistent data and programming languages like C++, Java, C# etc. relay on the object-oriented programming paradigm, the automatic management of the persistent data leads to the problem field of object-relational mapping. The object-relational mapping is a programming technique for transforming data between the object-oriented programming languages and relational databases.

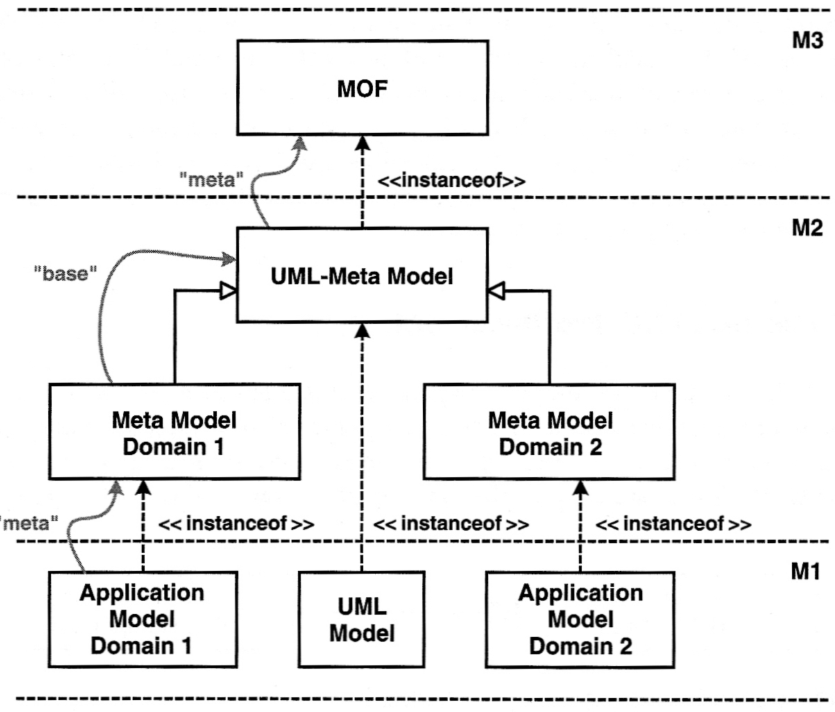

Figure 2.19.: Metamodel and inheritance (from [SV06], p.91)

Due to the fact that the mapping in practice is done by application developers and there are tight time and cost constraints in software projects the business logic is often mixed with persistency code. This mixture and the tight coupling to RDBM systems makes the reuse of code problematic.

To overcome these problems, Schmölzer et al. presented a new approach of an object-oriented persistency cache in [SMK+05] named the Entity Container (EC). With the help of the EC it is possible to keep transient data, and to access persistent data storage during application runtime.

The EC presented by Schmölzer et al. (depicted in Figure 2.20) acts as an access layer to different data storage systems like file systems, ODBMS and RDBMS. The picture shows that the EC is based on a persistency model and the application is only interacting with the EC to get access to persistent data. One key concept of the EC is that the persistency model is modeled with well known modeling tools like UML and that the description of the model is stored in an interchangeable format (XML Metadata Interchange). Due to the higher abstraction level of UML the model is independent of any specific programming language or persistency technique. Another feature of the EC is that the data in the EC is always checked against the data model, which ensures that the data structure and constraints are consistent with the high level design.

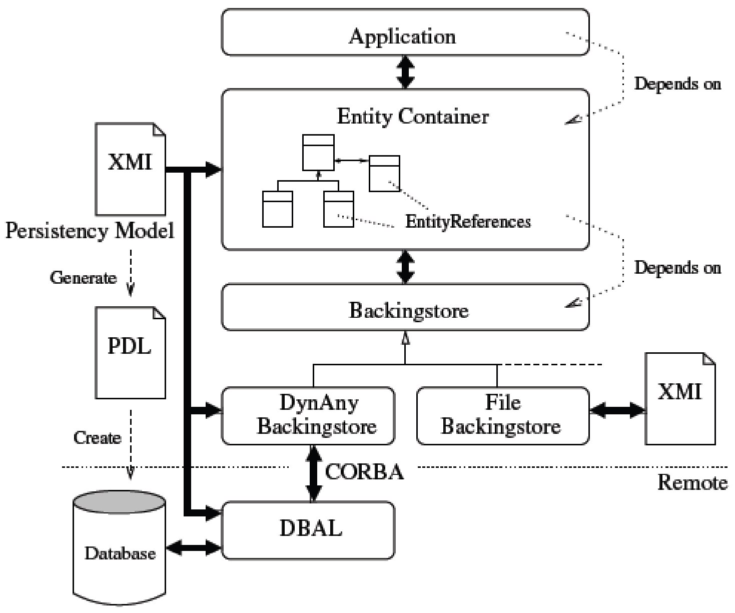

Figure 2.20.: Architecture of an Entity Container (from [SMK+05])

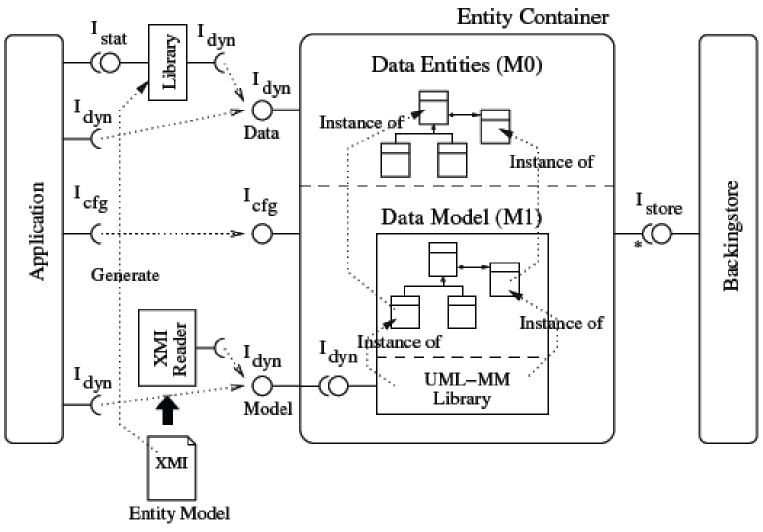

Figure 2.21.: Entity Container design (from [SMK+05])

The design of the EC itself is shown in Figure 2.21 and is based on two principles [SMK+05]:

- Data entities (objects) in M0 are always based on model entities in M1.
- The same interface is used to create a model entity on M1 as well as to create an instance of a model entity (object) on M0.

The main artifact in the EC concept is the data model which is the basis for concrete objects on M0 which are instances of data model entities in M1. The EC provides a dynamic interface *Idyn* in order to be able to create model entities as well as data entities from outside the EC. This can be done, as depicted in Figure 2.21 by the application itself or by an XMI reader.

2.5.3. Model-based Component Container

A Model-based Component Container (MCC) is constructed of several ECs as shown in Figure 2.22. One EC is used for each metalevel as defined by the OMG. Based on this approach it is possible to store data as objects in one layer and connect them in several ways as described in the layer above. The presented structure is divided in four layers:

- **Layer M0** (data layer). This layer holds the runtime data which correspond to the model in M1.
- **Layer M1** (model layer). This layer describes a data model and hence holds model information of the concrete objects in M0.
- **Layer M2** (metamodel). This layer provides basic components to create the model in M1.
- **Layer M3** (meta-metamodel layer). Consists of elements defined in the MOF.

For the reason that M2 specifies the capabilities of M1 a controller is needed which realizes the behavior defined in M1. This controlling unit interprets the platform independent data defined in M1 and respectively in M0 in a platform specific way. Several types of MCCs have been defined which are distinguished by different metamodels(M2) and different controlling units.

2.5.3.1. Data-MCC

The data MCC is used as a pure data cache. Hence it needs no controlling unit to interpret data. It has a fixed M2 layer and a variable M1 and M0 layers. The following section describes the layers M2 to M0 of the data MCC.

M2 description

This layer represents the metamodel layer and specifies the basic elements which can be used to describe a model in M1. This layer is fixed for all instances of a data MCC.

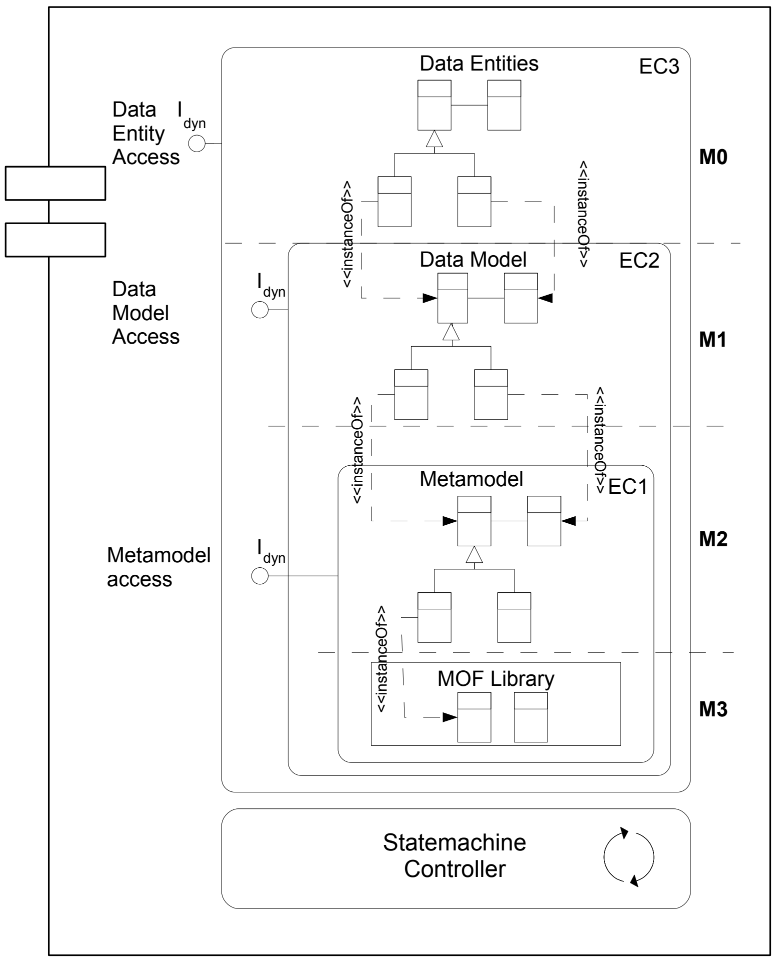

Figure 2.22.: Statemachine MCC (based on [SMK+05])

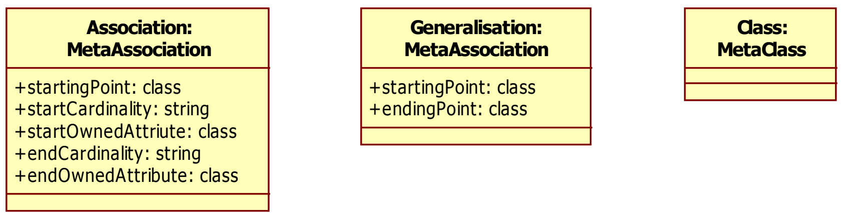
Association: MetaAssociation
+startingPoint: class
+startCardinality: string
+startOwnedAttriute: class
+endCardinality: string
+endOwnedAttribute: class
Generalisation: MetaAssociation
+startingPoint: class
+endingPoint: class
Class: MetaClass

Figure 2.23.: Visual description of the M2 layer of the data-MCC

M1 description

This layer describes the datamodel itself. Figure 2.24 shows a sample class model which uses the defined elements of M2. The depicted classes are instances of M2 elements as well as the associations. This layer can be dynamically configured and changed during runtime.

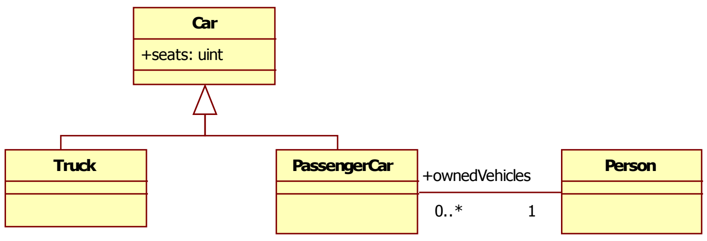

Figure 2.24.: Visual description of a sample M1 layer of the data-MCC

M0 description

On this layer the real data objects are stored based on the model described in M1. Again all elements which are shown in Figure 2.25 are instances of M1 elements including the associations between the objects. This layer can be dynamically configured and changed during runtime.

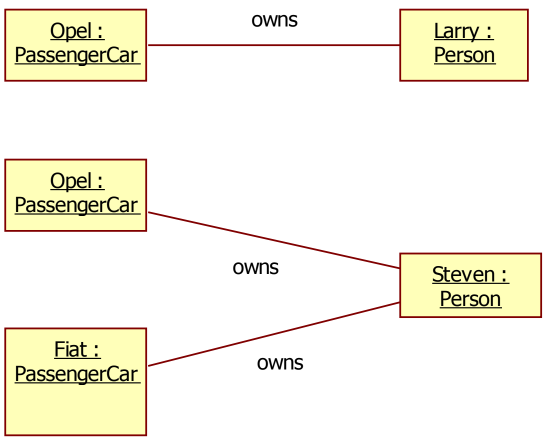

Figure 2.25.: Visual description of a sample M0 layer of the data-MCC

2.5.3.2. Statemachine-MCC

The statemachine-MCC is also based on ECs but unlike the data-MCC it has a controlling unit which realizes the behavior defined in the M1 layer. The following section describes the layers M2 to M0 using the example of a simple door opener.

M2 description

This layer represents the metamodel layer and specifies the basic elements which can be used to describe a model on M1. This layer is fixed for all instances of a statemachine-MCC.

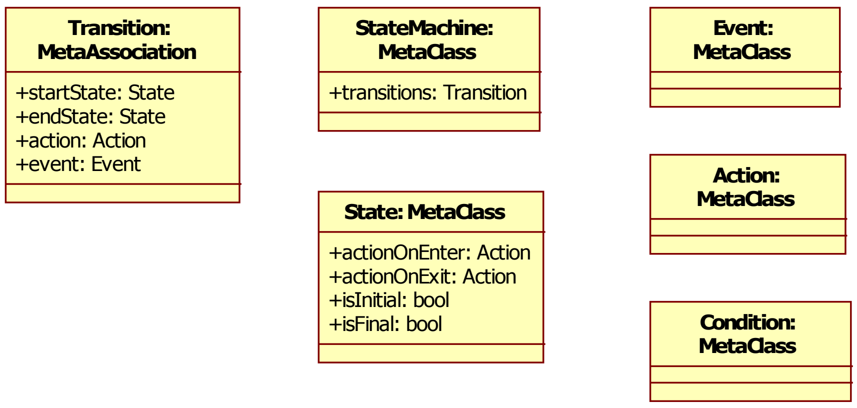

Figure 2.26.: Visual description of the M2 layer of the statemachine-MCC

M1 description

This layer describes the statemachine model itself. It uses the basic elements of M2 to construct a whole statemachine with transitions, conditions, events and actions. All elements presented in the statechart 2.27 are instances of M2 elements. Conditions and actions can be formulated with the help of the Entity Container Query Language (ECQL) [KTK10]. The controller unit which is an integral component of the MCC executes the model at runtime.

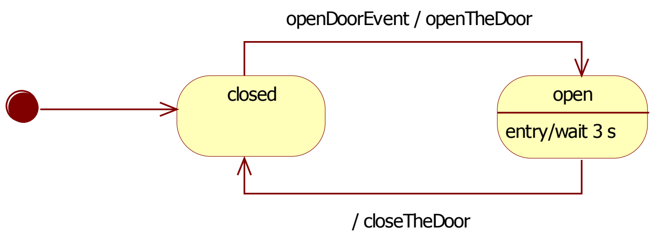

Figure 2.27.: Sample statemachine of a door opener

M0 description

This layer contains an instance of a statemachine defined on M1. Additionally event objects can be inserted at runtime, which in turn are used by the controlling unit as input parameters in order to keep the statemachine running.

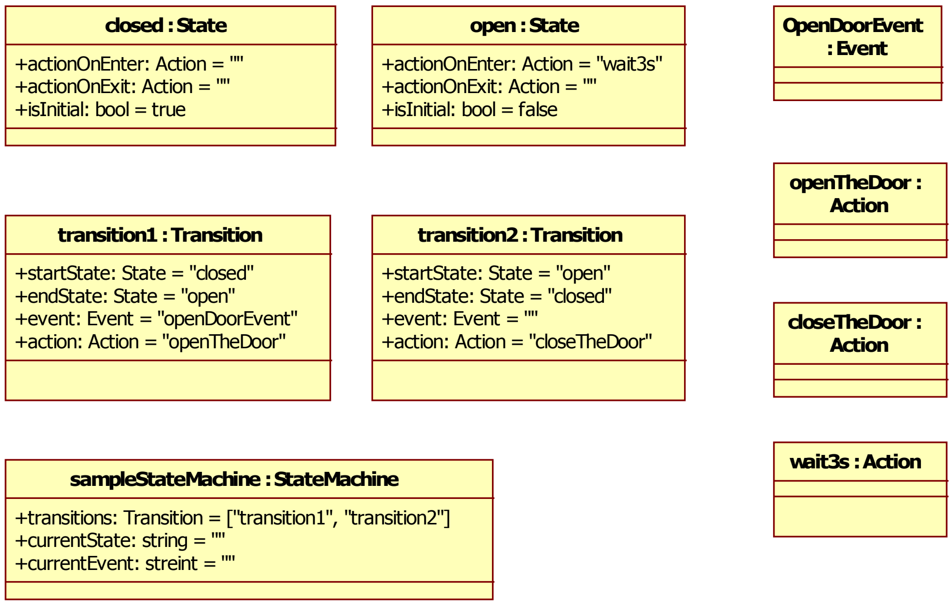
closed : State
+actionOnEnter: Action = ""
+actionOnExit: Action = ""
+isInitial: bool = true
open : State
+actionOnEnter: Action = "wait3s"
+actionOnExit: Action = ""
+isInitial: bool = false
OpenDoorEvent : Event
openTheDoor : Action
transition1 : Transition
+startState: State = "closed"
+endState: State = "open"
+event: Event = "openDoorEvent"
+action: Action = "openTheDoor"
transition2 : Transition
+startState: State = "open"
+endState: State = "closed"
+event: Event = ""
+action: Action = "closeTheDoor"
closeTheDoor : Action
wait3s : Action
sampleStateMachine : StateMachine
+transitions: Transition = ["transition1", "transition2"]
+currentState: string = ""
+currentEvent: streint = ""

Figure 2.28.: Visual description of the M1 layer of a door opener statemachine-MCC

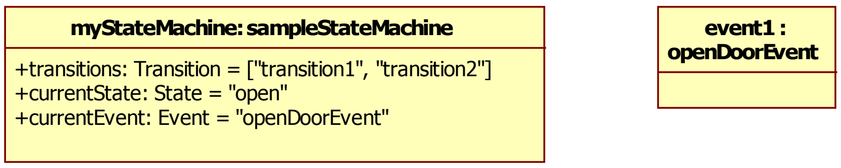
myStateMachine: sampleStateMachine
+transitions: Transition = ["transition1", "transition2"]
+currentState: State = "open"
+currentEvent: Event = "openDoorEvent"
event1 : openDoorEvent

Figure 2.29.: Visual description of the M0 layer of a door opener statemachine-MCC

2.5.3.3. Proxy-MCC

This MCC type provides a local interface to the data-MCC and the statemachine-MCC which are executed in another VON. A proxy-MCC gets connected to the target MCC by configuring it with the target's URN. A proxy can either target the M0 or the M1 container of the source MCC. Hence two proxys for M1 respectively M0 give access to all variable runtime model-layers in a MCC.

2.5.4. Model-Based Software Component

To support the reuse of software components different approaches like Component Based Software Engineering and Model-Driven Software Development have been proposed. Due of the highly heterogeneous platforms in todays mobile devices, MDSD offers a more flexible approach and hence a reduction of platform dependencies. There are two different techniques in MDSD according to Stahl and Völter [SV06]:

- Generate code out of the models at development or built time.
- Interpretation of models at runtime.

The problem with the first approach is that the code has to be regenerated after each change to the models. The second approach by contrast allows a late binding and a dynamic reconfiguration of the modeled application component

Based on the second approach and on the proposed Entity Container [SMK$^+$05], Thonhauser et al. introduce in [TKS09] a framework which supports the execution of model-based software components (MBSC). MBSCs are made up of a functional part and a technical part. The functional part consist of models for behavior, data and user interface, whereas the technical part makes use of already existing component models such as EJB, .NET or CCM. An example for a behavioral model would be a statemachine and user interface models could define the different GUI elements on various platforms. Figure 2.30 shows the different components within an exemplary runtime node.

- **Component metamodel.** The component metamodel corresponds to the M2 level in the four level metamodel hierarchy. Using the elements of MOF (M3), the metamodel defines all elements which are available in the application models (statemachine, data, user interface).
- **Application model.** The application model corresponds to the M1 level in the four level metamodel hierarchy. All elements in the application model are instances of elements in the metamodel.
- **Runtime node.** The runtime node consist of different MCCs managing data objects which are created through the interaction of the single controllers. The controllers are responsible to interpret the platform independent data in the ECs, in a platform specific way. An example could be a user interface which is provided in a platform independent representation inside an MCC and must be rendered by a controller using a Java Swing library.

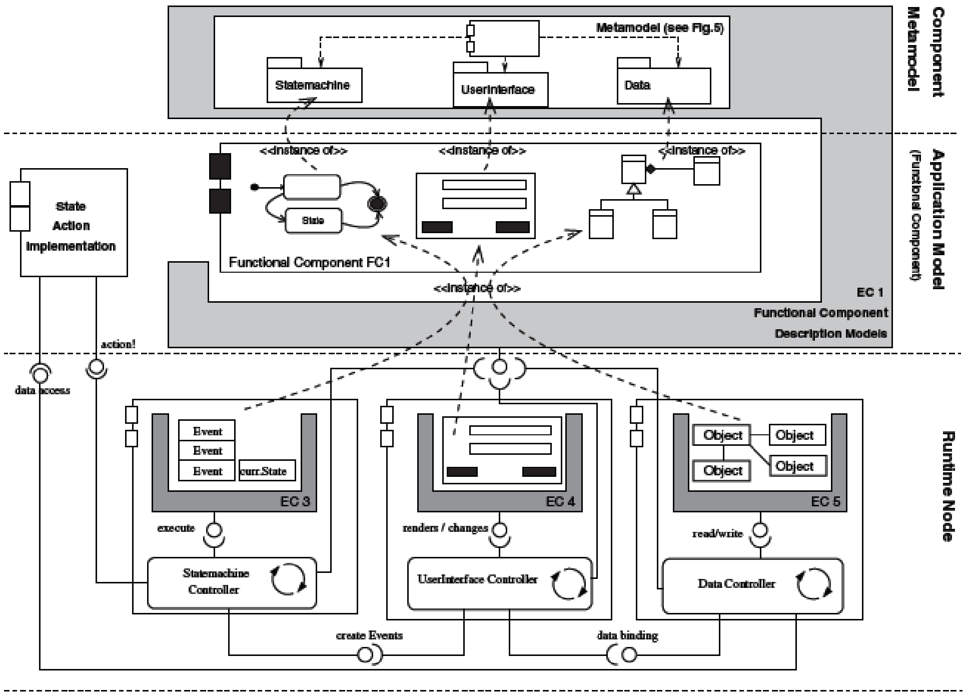

Figure 2.30.: Model-Based Runtime Node (from [TKS09])

3. Design of a model-based middleware for virtual organizations

This chapter describes the architecture and annotates the design consideration which had to be taken. Based on the Entity Container (EC), a model-based approach is presented for supporting virtual organizations (VOs). The usage of models allows to deal with the problem of platform heterogeneity, enables a dynamic configuration of resources and mapping to different application components, and supports a modular runtime architecture.

The following high-level requirements were defined at the beginning:

- The resources of a device must be assignable to different independent application components, in a model based way.
- Different organizations must be able to run and maintain application components on the same device.
- The behavior of the application must be changeable during runtime. In order to meet this requirement, a model-based approach is mandatory.

Based on these requirements a high level model which is depicted in Figure 3.1 was designed. The following components are presented in the picture:

Model based component container(MCC): Is based on the ideas of the EC in [SMK$^+$05] and represents the smallest model based component in the architecture. A MCC is constructed of several ECs, with each EC being responsible for one layer of the MOF hierarchy. A detailed description of the different MCC types and their structure is given in Sec. 2.5.3.

Model based software component (MBSC): The MBSC has been proposed by Thonhauser et al. in [TKS09] and is used as a theoretical basis for this thesis. Here the MBSC has been reduced to a component which provide a frame to host several MCCs. A MBSC is a full functional application component which handles a specific task. The behavior and the data needed for a proper execution are provided in a model based way by different MCCs. The behavior of a MBSC can be modified by changing the content of the MCCs or by adding additional MCCs. The MBSC presented in this thesis has limited functionality and therefore it is only possible to create one MBSC per Virtual Organization Node.

Virtual organization node (VON): A VON is a container with a controlling unit and can hold several MCCs. The controlling unit is responsible to startup and intialize the MCCs and their components. Furthermore the controller gets access rights to resources from the *resource node* in order to enable MCCs to use these resources during application execution.

Resource node (RN): On each device one resource node is running. The RN has a device controller which attaches all local resources to the architecture. Resources can be either natively implemented driver functions to control the hardware, or events which are offered by the hardware. Furthermore the device controller has the ability to dynamically grant access rights to VONs running within the RN.

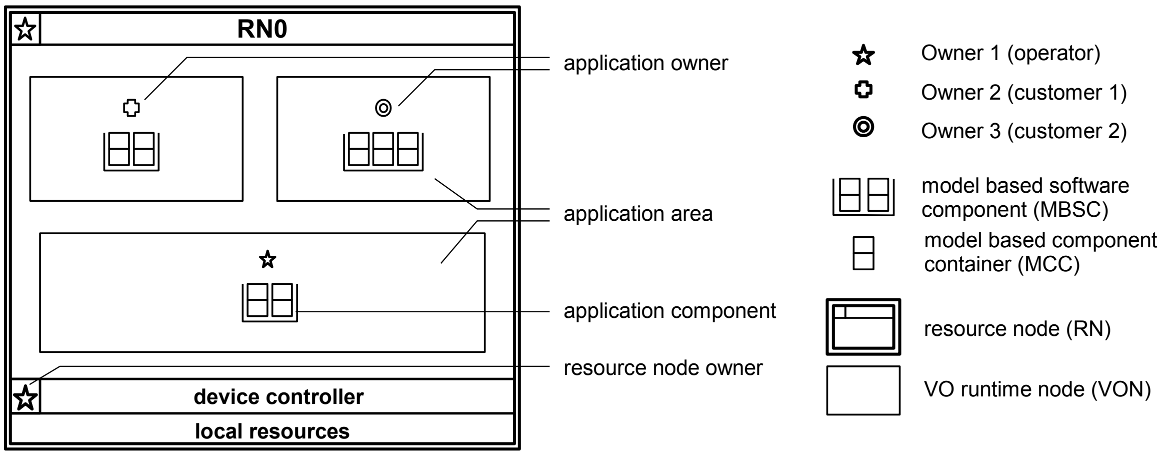

Figure 3.1.: High-level Architecture

3.1. Requirements

Based on the high level requirements and on the first architectural design a next iteration had been made to elaborate detailed requirements for the project.

R1. A VON should be able to hold several MBSCs.
In general it should be possible to maintain several MBSCs in one VON. Due to complexity reasons it is sufficient for the initial implementation to host only one MBSC per VON.

R2. The numbers and types of MCCs within a MBSC must be configurable.
A controlling unit which itself holds an MCC is needed to accomplish this configuration. The MCC within the controlling unit holds all the necessary data to determine which MCCs are needed for a MBSC in the application area of a VON. The number and also the type of MCCs must be dynamically reconfigurable during runtime.

R3. The local resources within a RN must be configurable.
Local resources (in our case natively implemented functions which access HW elements) must be assignable to different VONs in a dynamic way.

R4. The model and the data of an MCC must be remotely modifiable.
In order to change the model (M1 layer) and also the data (M0 layer) in the MCC, a mechanism is needed to access the MCCs remotely. This should be realized with the help of a proxy-MCC.

R5. Proxy-MCCs for data exchange between VONs
There should be only one way to exchange data to preserve flexibility. This should take place via proxy-MCCs and JSON.

R6. Location independency of proxy-MCCs
There is a difference in data exchange depending of the location of the proxy-MCC and the remote-MCC. For a proper solution the *Broker* pattern should be implemented.

R7. A RN must be able to hold several VONs.
In order to form virtual organizations it must be possible to run several VONs in one RN. Again, changing the numbers of VONs and owner of a VON at runtime must be possible.

R8. The VAR principle must be feasible.
It must be possible to form a hierarchy of VONs. It must be feasible for a VON to get data from a VON below in the hierarchy and make data available to VONs higher in the hierarchy.

R9. Separation between VON owner and the application owner.
The provider of a VON do not need to be the owner of the application in the VON. That means that one party only provides the empty skeleton of a VON and another party could transfer the application (MCCs) into the VON.

R10. There must be an event mechanism within a VON.
In order to realize event notification between VONs it should be possible for a VON to subscribe to events which are published by other VONs. A model based *Observer* pattern should be implemented to achieve this notification mechanism.

R11. A VON must be able to migrate to another RN.
If an adaption of the system is required at runtime, the owner of a VON can decide to move his VON to another RN. Therefore it must be possible to serialize the whole

content of a VON to JSON and reconstruct the VON on another RN.

R12. The VON must be runnable on several .NET runtime environments.
The application must be implemented in C# and must be able to run on following .NET runtime environments: Microsoft .NET Framework, Microsoft .NET Compact Framework, Microsoft .NET Microframework and Mono.

3.2. Architecture

Due to the fact that the here presented approach only supports one MBSC per VON, the VON takes over the functionality of a MBSC and provides therefore a runtime environment for several MCCs. The ownership of a VON is divided into two different parts. The first is the *creator* of the VON and the second one is the *content owner* of the VON. They can be the same but do not have to be. An example for separated ownership would be when *companyX* creates a Node on the device and provide this runtime environment to *companyY*. *CompanyY* in turn transfers its application into the node provided by *companyX* and thus has the ability to dynamically change runtime parameters.

After precisely elaborating the requirements from Sec. 3.1 a detailed design of the application framework which is depicted in Figure 3.2 was made. The picture shows a RN which is running on a device owned by the *operator*. The RN creates three VONs, one for the *operator* and two further ones for tenants. Hence the VONs are just runtime environments for MCCs each VON owner has to transmit his application models into the VON. This transmission is done in two steps. The first model is transmitted to the VON controller and specifies the number and the type of the MCCs which have to be created in the application area. Having created the empty MCCs in the application area the owner of the VON can transmit the models for each MCC. Furthermore the picture shows that data is exchanged between VONs with the help of proxy-MCCs. Proxy-MCCs are acting as local representatives of MCCs which reside on another VON.

Figure 3.3 provides a component oriented overview about the system. The whole system is divided into four packages. The *MCC* package on the top left was already fully implemented at the beginning and could be used for this project. It realizes a model based component container(MCC) based on the EC approach. The MCC package realizes the data-MCC as well as the statemachine-MCC. The *Remoting* package was partly implemented at the beginning of this project and had to be adopted to fulfill all requirements concerning the remoting aspect.

The main focus of this project however, lies on the *Virtual Organization* package and its components. The package is divided into the *Resource Mgmt.* component, *Lookup Service* component and the *VON* component. The *Resource Mgmt.* component is responsible to manage all local resources on the device. The *VON* component realizes the *VO runtime node* and the controlling unit. The last component, the *Lookup Service* component manages the container identification both local and remoting. In the case a MCC is accessed

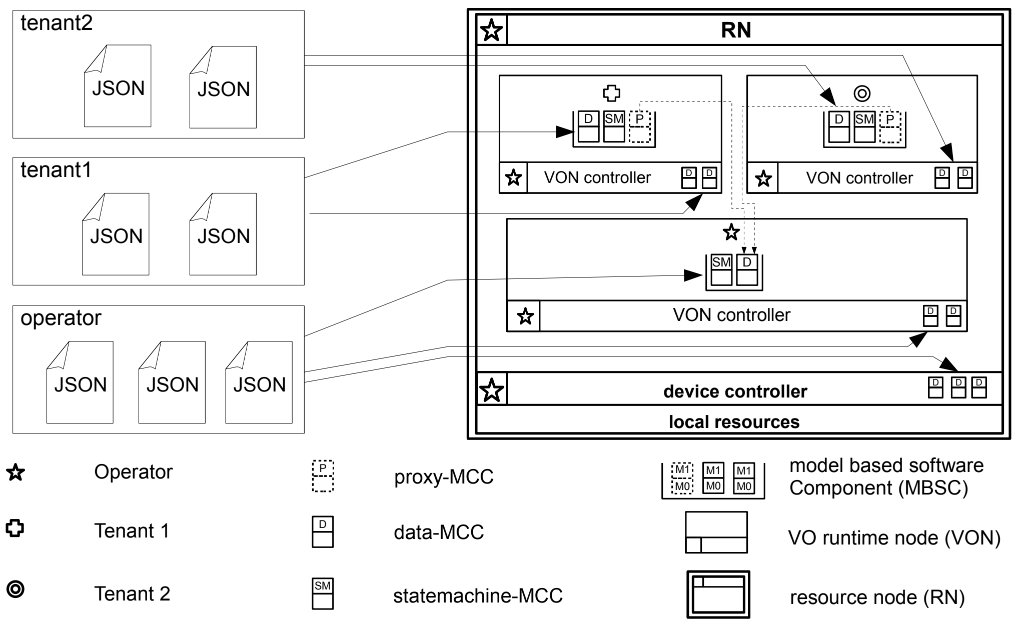

Figure 3.2.: Architecture with one operator and two tenants

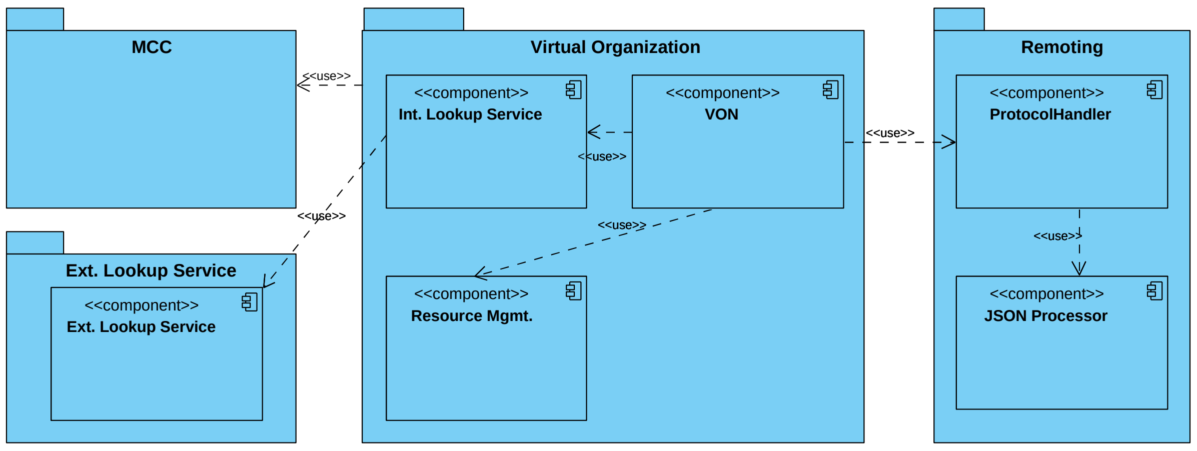

Figure 3.3.: Component Diagram

through a proxy-MCC and both resides on different devices, a global lookup service is needed, which is represented as the *Ext. Lookup Service* package at the bottom left in Figure 3.3.

3.3. Virtual Organization Node Architecture

This section describe the principle structure of the VON component. It explains the various elements in the controlling unit and outline how node identification takes place.

A VON features a controller unit which itself has several MCCs. This controlling unit as depicted in Figure 3.4 hosts one statemachine-MCC and two further data-MCCs. The statemachine-MCC is responsible for the lifecycle management of the VON and the two data-MCCs are responsible for the MCC structure in the application area and the publisher subscriber mechanism. Through these MCCs it is possible to control and manage the application components within the VON in a model based way. Additional to the MCCs, two separated event queues form an integral feature of the controller. One is responsible do deliver events into the application area and one is responsible for events addressing the VONs statemachine-MCC. If a statemachine can not handle an event which was sent by the queue, because it is in a state where it can not response to that event, the statemachine discards it and waits for the next event.

By default the controlling unit of a VON has:

- a statemachine-MCC for lifecycle management
- a data-MCC to specify the number and the type of MCCs in the application area
- a data-MCC to realize the publisher-subscriber mechanism

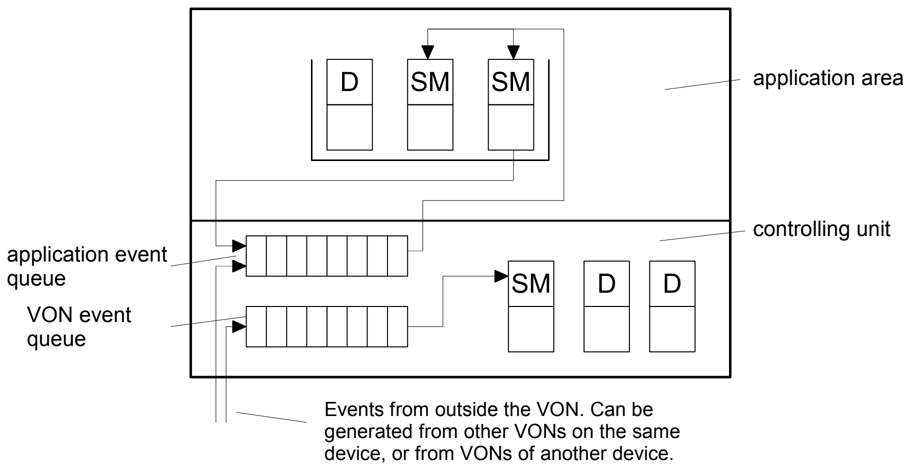
D
SM
SM
application area
controlling unit
application event queue
VON event queue
SM
D
D
Events from outside the VON. Can be generated from other VONs on the same device, or from VONs of another device.

Figure 3.4.: Structure of a standard VON

All standard configuration aspects are covered with this three MCCs. However, in the case a VON should be able to create new VONs one additional data-MCC is needed. By this MCC the owner of the VON has the capability to create new VONs and to pass access rights of local resources to particular VONs.

By closer consideration a VON with this additional MCC fulfill all requirements concerning the RN. So in fact a RN could be seen as a VON enhanced by the capability to manage other VONs. Through this enhanced capabilities there is a separation between VON owner and application owner feasible and hence, also the VAR principle can be realized.

3.3.1. MCCs in the controlling unit

This section is going to describe the three respectively four MCCs in the controlling unit in more detail.

3.3.1.1. Statemachine MCC of the VON

This statemachine-MCC resides in the controlling unit and manages the lifecycle of the VON. Figure 3.5 shows the state diagram of this MCC with six possible events to control the VON:

- **startNode**. This event activates the VON. When the VON is in the *Run* state, the event queue of the VON is processed and all statemachines in the application area getting events from that queue.

- **stopNode**. This event deactivates the VON. In the *Stopped* state the statemachines in the application area don't receive any events. The events however don't get lost because they are buffered in an event queue. As soon as the VON is in the *Run* state the event queue is processed again.

- **updateNode**. After the data-MCCs within the controlling unit VON getting new M1 or M0 data, the update command has to be called in order to process the newly transferred configuration data.

- **resetNode**. Destroys all MCCs in the application area and deletes the M0 layers of the VON configuration containers.

- **cloneNode**. Copies the whole application area of a VON to another VON.

- **migrateNode**. The whole VON content migrates to another VON. With this command it is possible to move a VON away from one device towards another.

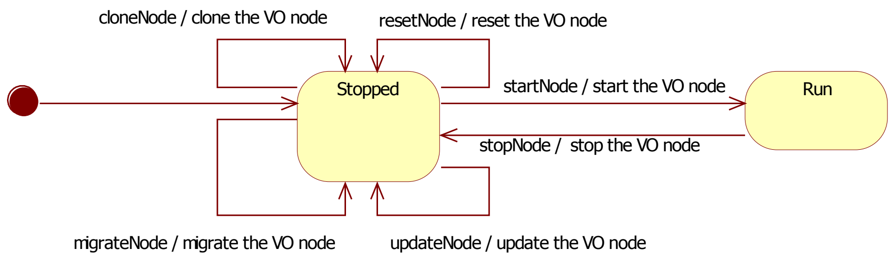
cloneNode / clone the VO node
resetNode / reset the VO node
Stopped
startNode / start the VO node
Run
stopNode / stop the VO node
migrateNode / migrate the VO node
updateNode / update the VO node

Figure 3.5.: State diagram of the VON controller's statemachine-MCC

3.3.1.2. MCC to specify MCCs in the application area

This data-MCC which also resides in the controlling unit is used to define the type and the numbers of MCCs in the application area. Only the VON owner can configure this MCC. Due to the fact that the M1 layer is fixed , only the M0 layer is modifiable. Regarding to the M1 model which is depicted in Figure 3.6 it is possible to define instances of three different MCC types on the M0 level. After a modification of M0 of this MCC a *updateNode* event must be send to the statemachine-MCC of the VON controller unit to apply the changes.

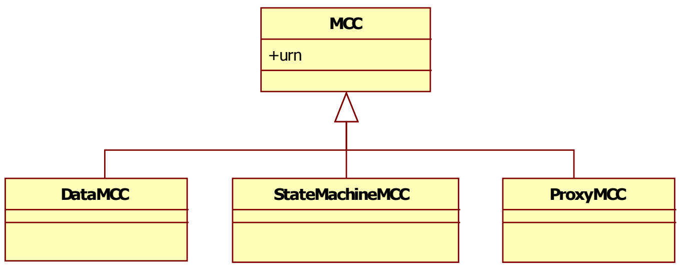

Figure 3.6.: M1 model of a data-MCC used to specify MCCs in the application area

3.3.1.3. MCC to realize the publisher subscriber mechanism

The presented approach allows two possible flows of events. The first possibility is to address the target of an event directly. That would be the case when *statemachine1* of a VON generates an event and sends it directly to *statemachine2* in the same VON, because the address is hardcoded in *statemachine1*. The second option is the publisher subscriber mechanism. This would be the case when *statemachine1* is generating an event but sends it not directly to *statemachine2*. The controller get noticed every time an event occurs and checks if any other statemachine is interested in that event. If so, it notifies all interested statemachines, by forwarding this event.

The MCC presented in Figure 3.7 has two tasks. On the one hand it contains elements with event-names to which other VONs can subscribe (PublishedEvent), on the other hand it contains elements with an event-name and an URN to which the VON itself want to subscribe to (EventToSubscribe). In the latter case it is necessary to send a *updateNode* event to the statemachine of the controlling unit in order to apply the changes.

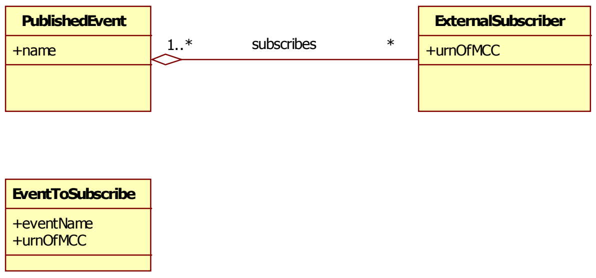

Figure 3.7.: M1 model of the data-MCC to realize the publisher subscriber mechanism

3.3.1.4. MCC to specify the VON structure in a RN

As mentioned above the RN can be seen as a VON with additional capabilities to create VONs and manage resources of that VONs. These abilities are obtained through an additional MCC in the controlling unit. Figure 3.8 shows the M1 layer of this data-MCC. The M1 layer is again fixed and only the M0 layer can be changed by the VON owner. After a modification of M0 of this MCC a *updateNode* event must be send to the statemachine-MCC of the VON controller unit to apply the changes. As shown in the picture the model can specify how many VONs have to be created, what resources they are allowed to use and to which VOs they are belonging to.

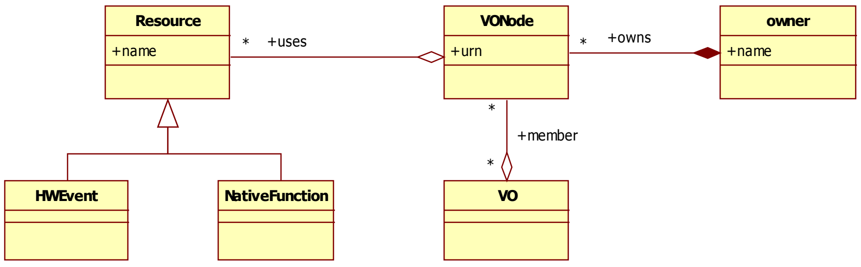

Figure 3.8.: M1 model of a data-MCC used to specify the VON structure

3.3.2. Event queues

The controlling unit of a VON includes two queues which contrive that events are delivered in a causal way and that events don't get lost. One queue manages the events which are intended for statemachine-MCCs in the application area, the other one delivers events to the statemachine-MCC of the controller unit.

Events can be generated from the underlaying hardware, from the statemachines theirselves or from a proxy instance. Events from the underlaying hardware are passed to the first RN which is automatically created at startup. The RN can use the event itself or can forward it to another VON according to the configuration in its data-MCC. Events from a proxy instance are always addressed directly to a statemachine-MCC. Events which are generated by statemachines can be either addressed directly to another statemachine or can be handled through the publisher subscriber mechanism as described above. Usually each event is addressed to a special statemachine instance, if not events are forwarded to all statemachine instances in a VON.

Whenever all statemachine-MCCs in the application area are in the idle state, the controlling unit takes the first element out of the queue and evaluates the target address. If there is a address specified, the controller unit forwards the event to the specified statemachine-MCC, if not, the event is forwarded to all statemachine-MCCs in the application area of the same VON. As soon as a statemachine-MCC gets an event from the controlling unit it evaluates all transitions which lead away from the current state. If there is any transition which can handle this event, the transition gets executed. If there is no transition which can handle the event the event gets discarded. After all statemachines treat the event, the next event from the queue gets forwarded. If the queue is empty, all statemachines stay idle as long as the next event occurs.

3.3.3. Node identification

To accomplishes node identification a uniform resource name (URN) is used. Both uniform resource names (URNs) and uniform resource locators (URLs) belong to uniform resource identifiers (URI). Whereas the former is used for identification and the latter for location or finding resources. The major difference between both can be described as ”’what”’ vs. ”’where”’. [Moa97] defines the following syntax for URNs:

<URN>::=”’urn:”’<NID>”’:”’<NSS>

<NID> is called the namespace identifier and <NSS> is the namespace specific string. Note that all phrases in quotes are mandatory. In the here presented approach it is just possible for a VON to hold one MBSC. In that case we need three hierarchies for the NIS to identify and address each component uniquely. For example to address a special MCC named *statemachine1* in the VON named *voNode1* which belong to the organization named *organization1* the URN would be *urn:vo:organization1:voNode1:statemachine1.*

3.4. Virtual Organization Node Design

3.4.1. Class diagram

The class diagram presented in Figure 3.9 realizes the VO runtime node and the corresponding controlling unit. Following classes are shown:

- **VONode.** Each VONode runs in its own execution thread and can host serveral MCCs (data-MCC, statemachine-MCC, proxy-MCC). Because the VONode represents a synchronous process the *Half-Sync/Half-Async* pattern is used to enable asynchronous event notification (see section 3.4.3.1).

- **MCC.** The *VONode* class as well as the *StandardController* class are using instances of the MCC component. The MCC component implements both the data-MCC and the statemachine-MCC.

- **Queue.** The *Queue* class is needed to create a separation between the asynchronous event occurrence and the synchronous event execution in the *VONode* class.

- **StandardController.** Realizes the controlling unit of a VON and is responsible for a proper configuration of the VON. The *StandardController* hosts two data-MCCs for configuration purposes and one statemachine-MCC. The statemachine-MCC manages the lifecycle of the VON. The first data-MCCs defines the number and the type of MCCs which are hosted by the *VONode* class. The second data-MCC enables statemachine-MCCs to subscribe to events which are generated by other statemachine-MCCs. See section 3.3.1 to get a better understanding about the purpose of the particular MCCs.

- **ProxyMCC.** Is used to access MCCs which reside on another VON. Implements the same interface as the data-MCC and the statemachine-MCC to read and write data on the M0 and M1 layer. The *proxy-MCC* forwards all requests to the *Broker* class.

- **Broker.** The *Broker* class gets request from the proxy-MCCs and forwards it depending on the location of the wanted MCC. It forwards the request either to a *ServerProxy* if the MCC resides on the same device or to the *Remoting* package which marshals and serializes the request and sends it over the Net to the appropriate device, if the MCC can not be located on the current device. The *Broker* uses the *LookupService* component in order to be able to decide whereto forward the incoming request.

3.4.2. Startup

This section is going to describe how the startup procedure is working in detail.

When starting a device, one RN is instantiated. All hardware resources are assigned to this RN at startup automatically. As mentioned in Sec. 3.3 a RN is an enhanced VON which hosts three data-MCCs for configuration purposes in the controlling unit:

- **Configuration MCC1.** MCC to specify the VON structure in a RN

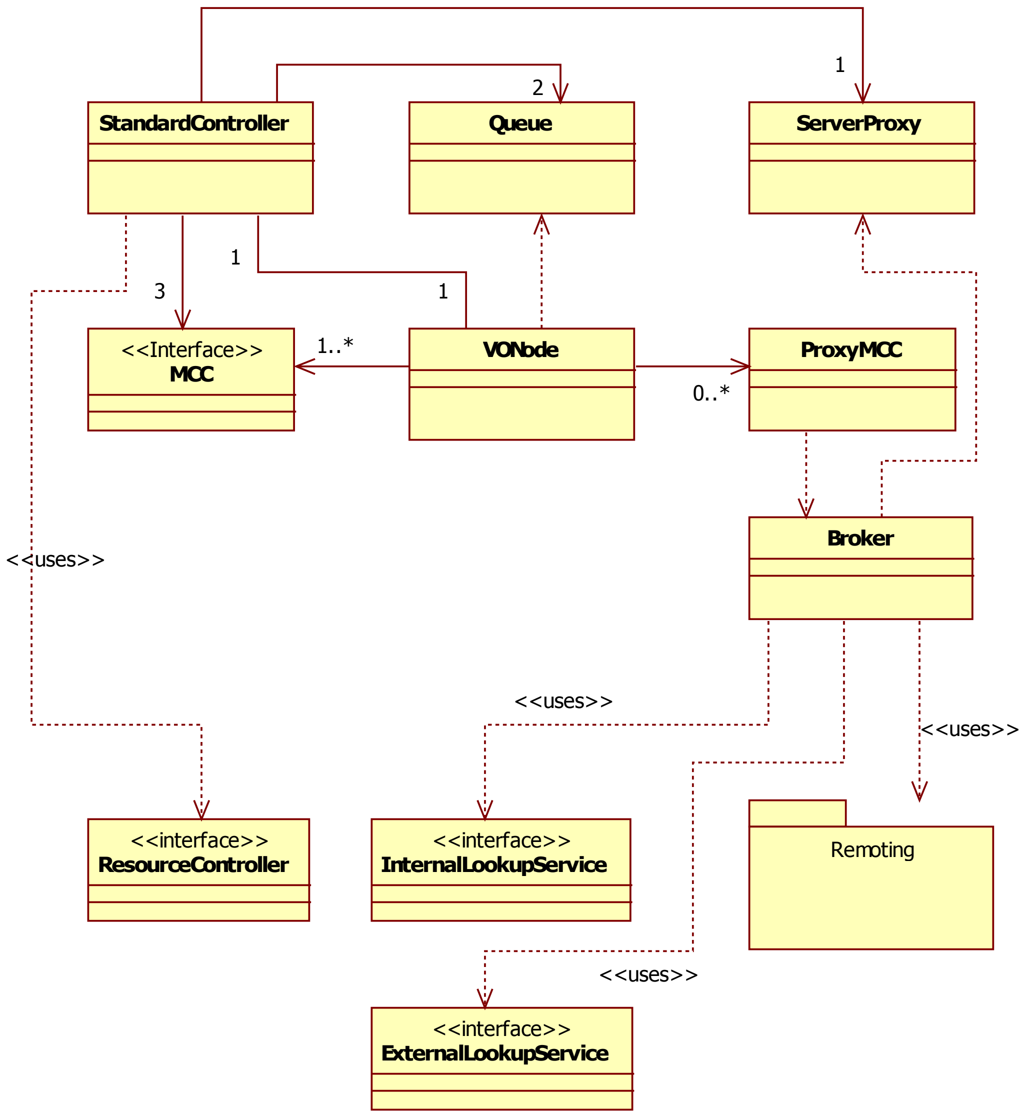

Figure 3.9.: VON component

- **Configuration MCC2.** MCC to specify MCCs in the application area
- **Configuration MCC3.** MCC to realize the publisher subscriber mechanism

After the initialization phase the owner of the RN has the possibility to transmit the M0 data to the configuration MCCs. The M0 data of *Configuration MCC1* defines further VONs in the RN and assigns them access rights to hardware resources. The M0 data of *Configuration MCC2* specifies the type and number of MCCs in the application area and with the help of *Configuration MCC3* statemachine-MCCs can subscribe to events which are published by other VONs.

After a successful transfer of the M0 data the RN-owner has to send an *update* event to the statemachine-MCC of the VON. The VON controller subsequently processes the M0 data of the configuration MCCs, generates further VONs in the RN and generates MCC instances in the application area. Once the MCCs in the application area are created the owner of the application area has the possibility to transfer his application to the previously created MCCs (M1 and M0 data). After transferring the model respectively data the *application owner* has to send a *start* event to the statemachine-MCC of the VON in order to enable event processing of the application component.

Figure 3.10 represents the startup sequence in detail. As mentioned before the configuration MCCs exists already after the startup of the VON. The owner of the VON, in this case called *operator* transfers M0 data to configuration MCC1 and configuration MCC2. After sending the *update* event to the statemachine-MCC of the VON controlling unit, the controller evaluates the M0 data, generates new VONs if specified and generates two MCCs in the application area. One MCC instance is of the type *statemachine-MCC* and the other of the type *data-MCC*. In the next step the operator transfers the application which is composed of M1 and M0 data to both application MCCs. When sending the *start* event to the statemachine-MCC of the VON controlling unit, the VONode gets enabled. As soon as the VONode is enabled the statemachine-MCCs can receive and process events coming from outside the VON.

3.4.3. Event handling

Events are an integral part of the whole architecture and are used for communication and notification. It has to be distinguished between three different sources of events:

- events which occur in the underlying hardware
- events which are send via proxys directly to statemachine-MCCs
- events which are generated by statemachine-MCCs themselves

Events which occur in the underlying hardware are device specific resources and have to be managed via the data-MCC described in Sec. 3.3.1.3. By means of this data-MCC it is possible to define VONs which get notified when the specific event occurs. If a VON is registered for this specific event and it arises, the event automatically get inserted to the queue and the controller forwards it afterwards to the VONode.

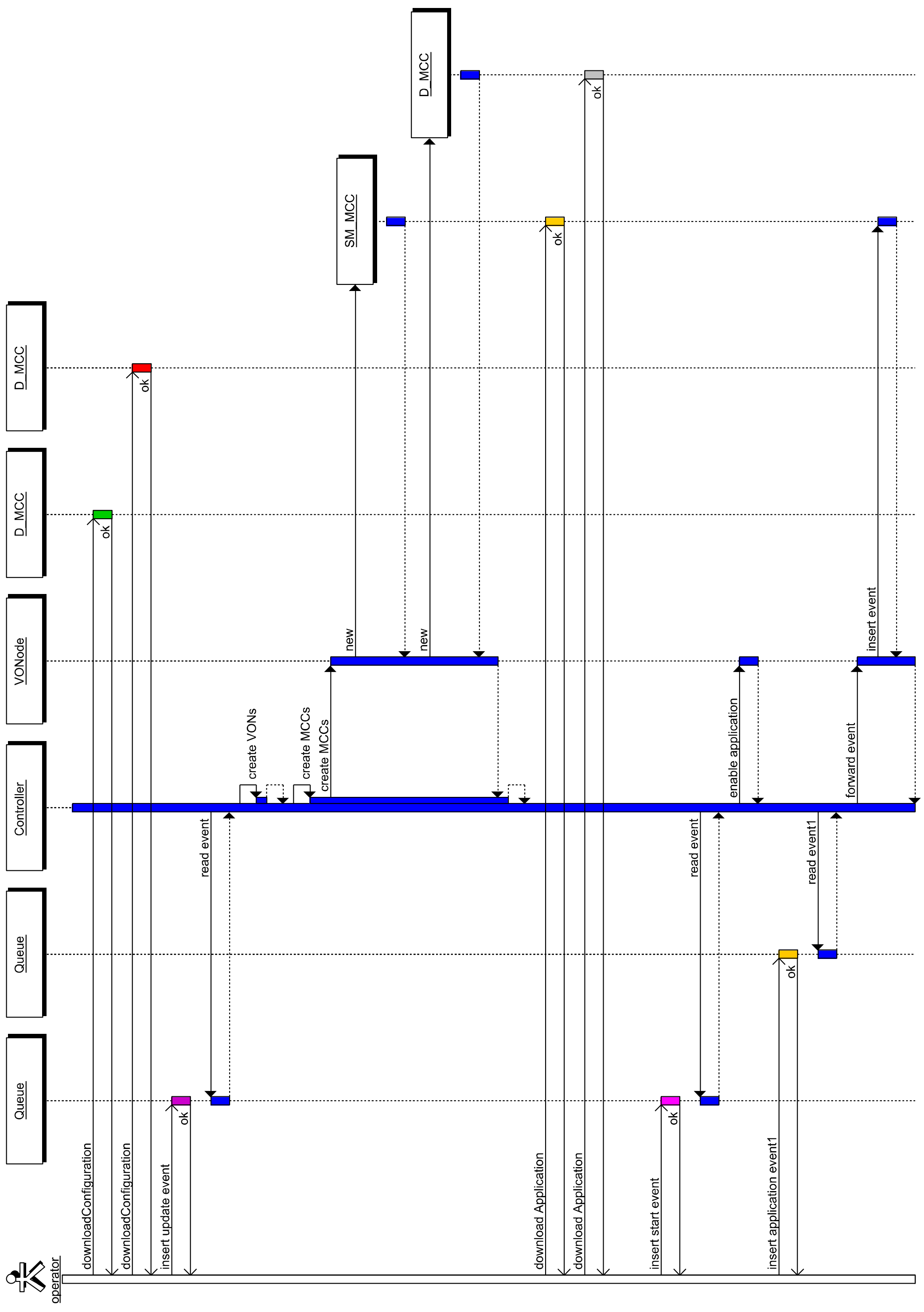

Figure 3.10.: Detailed startup sequence

The next possibility to create events is via proxy-MCCs. A proxy-MCC can be a local representative for a remote data-MCC as well as for a remote statemachine-MCC. Whereas the term remote-MCC refers to a MCC which resides on another VON. When an event gets send via a proxy-MCC to a statemachine-MCC the event gets also inserted into the queue to ensure causality. The controller again reads the event from the queue forwards it to the VONode which evaluates the destination address and sends the event to the appropriate statemachine-MCC.

The third possibility to create events is when a statemachine-MCC itself generates an event during an action execution. In this case the event gets send to the controller which checks if a destination address for the event is specified. The resulting actions by the controller are explained in Sec. A.

3.4.3.1. Half-Sync/Half-Async pattern

This pattern is used when software needs to perform synchronous as well as asynchronous service processing. The asynchronous part is mostly needed in conjunction with low-level system services whereas synchronicity is used to simplify application service processing. Buschmann et al. gives the following solution to this problem in [BHS07]:

> *Decompose the services of concurrent software into two separated layers, synchronous and asynchronous, and add a queueing layer to mediate communication between them.*

As shown in Figure 3.11 the Half-Sync/Half-Async pattern forces a strict separation between these layers. The upper layer processes higher-level services such as data queries or file transfers in separated threads synchronously. The bottom layer conversely process low-level services like protocol handlers asynchronously. Communication between asynchronous and synchronous services takes place via message exchange over a queueing layer.

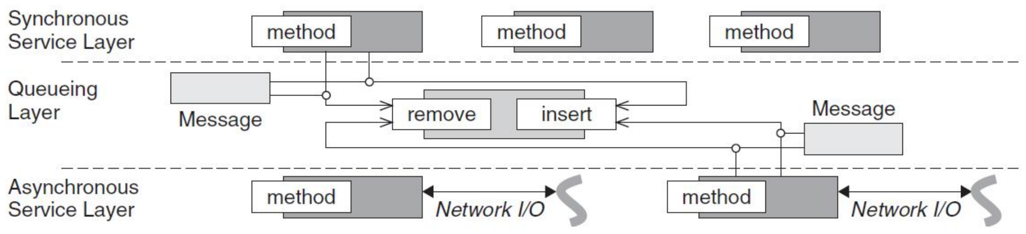

Figure 3.11.: Half-Sync/Half-Async pattern (from [BHS07], p.359)

In this work the Half-Sync/Half-Async pattern is used to obtain a separation between the asynchronous events which are generated by the hardware or by statemachine-MCCs, and the synchronous processes in the VONs, which handle the events and carry out appropriate actions. Figure 3.12 shows how the separation takes place when an actor inserts an event into the *Queue* of the desired VON. If there are already pending events in the queue nothing more happens but if not the *Queue* generates a notify message in order to

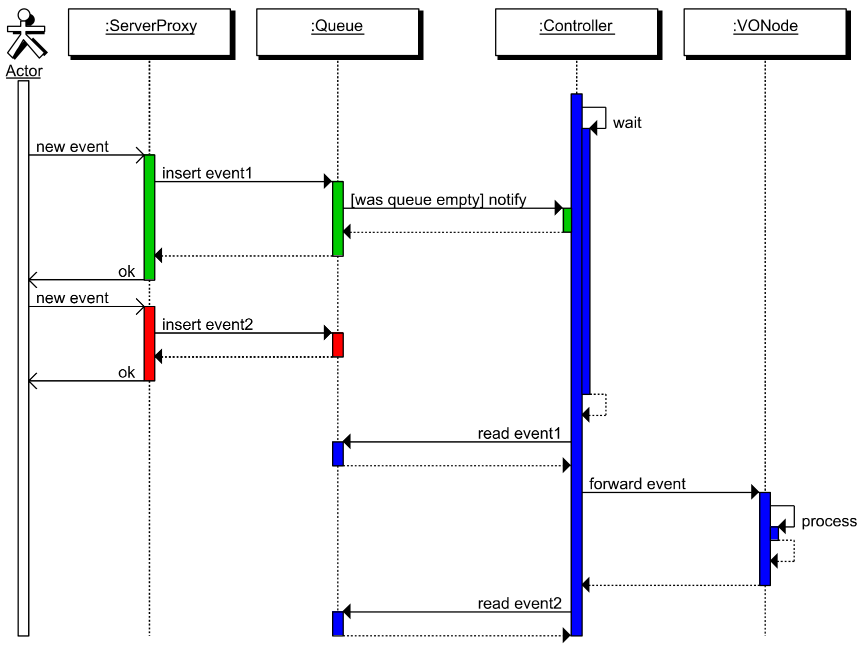

Figure 3.12.: Overview of the event mechanism

wake up the *Controller*. When ready the *Controller* gets an event from the *Queue* and evaluates the address coming with the event. Once determined the *Controller* forwards the event to the appropriate *VONode* which handles the given event.

3.4.4. Component location

Because a potential application using the presented approach is usually distributed over several devices a mechanism has to be found, which shields the application from the complexity of component location an inter-process communication (IPC). To face this challenge the *Broker* pattern is applied. Buschmann et al. explain the usage of the *Broker* pattern as follows [BHS07]:

> *Use a federation of brokers to separate and encapsulate the details of th communication infrastructure in a distributed system from its application functionality. Define a component based programming model so that clients can invoke methods on remote services as if they were local.*

One Broker instance is defined per network component. To invoke methods on a component, a client calls a method on the local proxy to initiate a request. The proxy works together with the local and server-side broker to invoke a method on the component and to receive any results (Figure 3.13).

In this work the Broker pattern is needed to redirect the method calls from the proxy-MCCs to the data-MCCs respectively statemachine-MCCs residing on different VONs. As

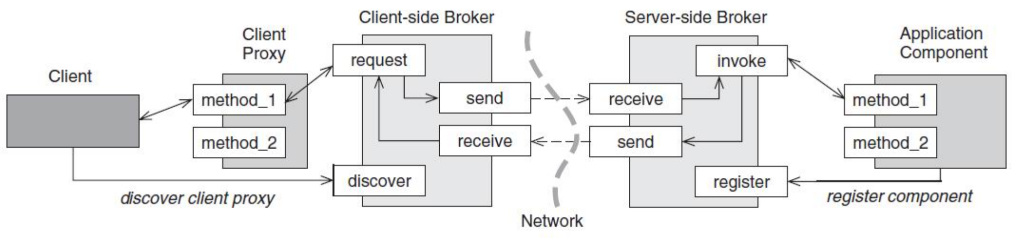

Figure 3.13.: Broker pattern (from [BHS07], p.237)

explained previously accessing a data-MCC or statemachine-MCC on another VON can only be achieved through a proxy-MCC. Therefore, if an application wants to invoke a method of a data-MCC from another VON it has to use a proxy. The proxy-MCC forwards the request to the Broker instance, which in turn evaluates if the VON, wherein the needed MCC resides, is accessible on the same device. If so the Broker forwards the request directly to the server-side proxy of the appropriate VON. Subsequently the server-side proxy invokes the method of the data-MCC.

If the Broker can not locate the appropriate VON on the same device it performs a lookup to the external lookup service which is in charge of managing the locations and access parameters of VONs within a domain. All VONs have to register theirselves at startup within the lookup service to give other components the ability to access them. After retrieving the location of the seeked VON from the lookup service, the Broker forwards the request to the Bridge instance in the *Remoting package*. In the *Remoting package* the Bridge marshals the request and serializes it to a JSON string. The message then gets transmitted over the net to the appropriate device. On the other side another Bridge instance deserializes the message and forwards it to the Broker. The Broker figures out the wanted VON and forwards the request to the server-side proxy of this VON. The server-side proxy finally invokes the method of the data-MCC. The return value of the method takes the same procedure, just in the different direction. Figure 3.14 shows the whole sequence diagram for the case where a client accesses a data-MCC on another device.

3.5. Resource Node

RNs are executed on existing devices and feature a controller having all local resources attached to the architecture. Additional to manage local resources a RN also passes access rights of resources to other VONs created by it. In the presented approach local resources can be either native implemented functions provided by the underlying layers or events which are provided by the underlying hardware. One RN is always created automatically at startup and belongs to the operator of the device. This RN gets initially all access rights to local resources. Furthermore this RN is capable to create further VONs and RNs, and assign to them again access rights to resources.

Figure 3.2 shows a sample RN with three VONs. The RN belonging to the *operator* has

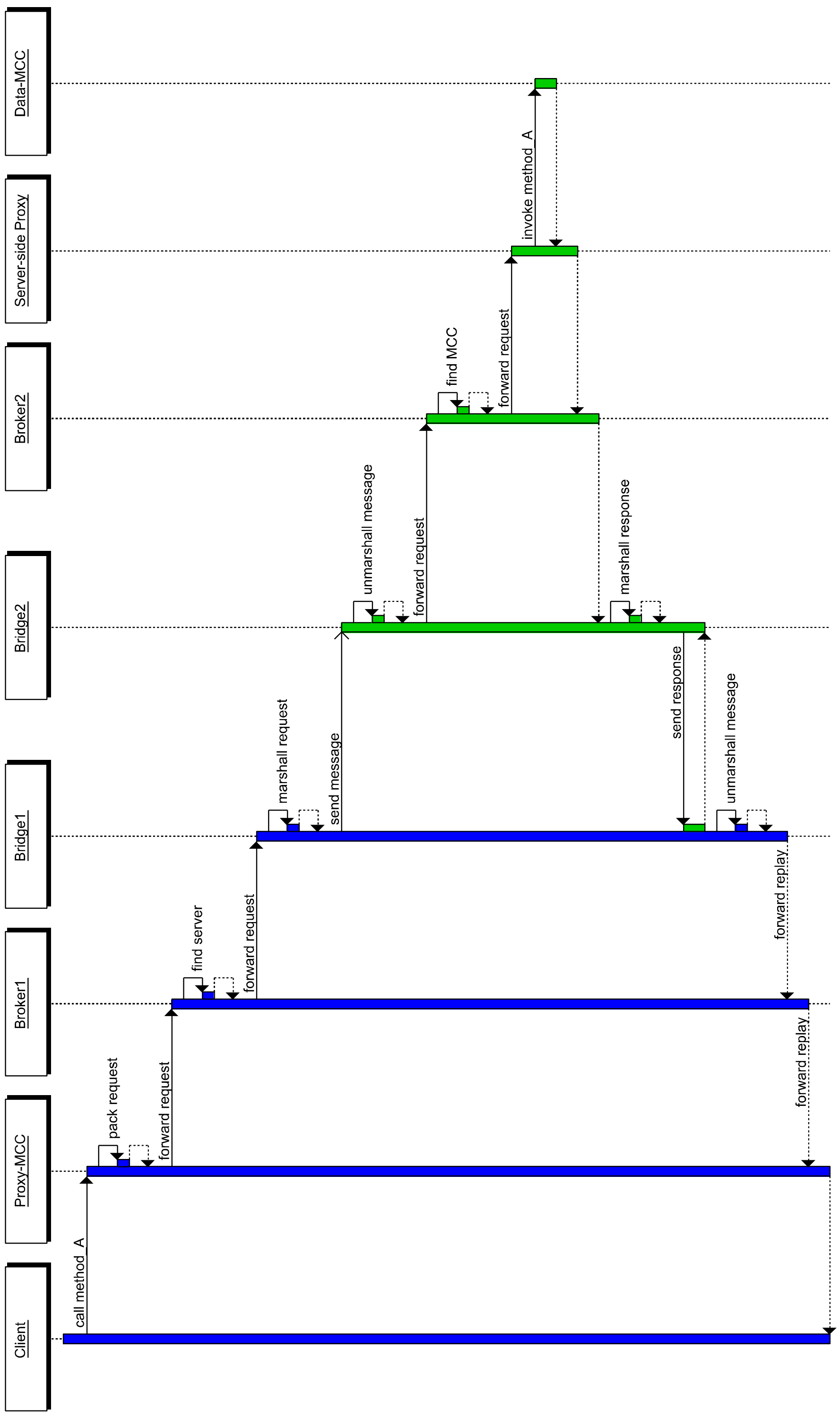

Figure 3.14.: Layered Broker architecture

an additional data-MCC in the controlling unit by which it is possible to define further VONs in the RN. In this case the data-MCC in the RN controller was configured in a way to create three VONs, one for the operator himself and two for potential tenants.

Figure 3.15 depicts the class diagramm of the RN which is part of the VON component shown in the component diagram in Figure 3.3. As mentioned above the RN is a VON with enhanced functionality. This behavior is realized with the additional class *ExtendedController* which inherits from the *StandardController*. The *ExtendedController* hosts one additional data-MCC in contrast to the *StandardController*, thus qualifies the *ExtendedController* to create new VONs and RNs. Furthermore the *ExtendedController* uses the *InternalLookupService* and the *ExternalLookupService* to register the created VONs and RNs.

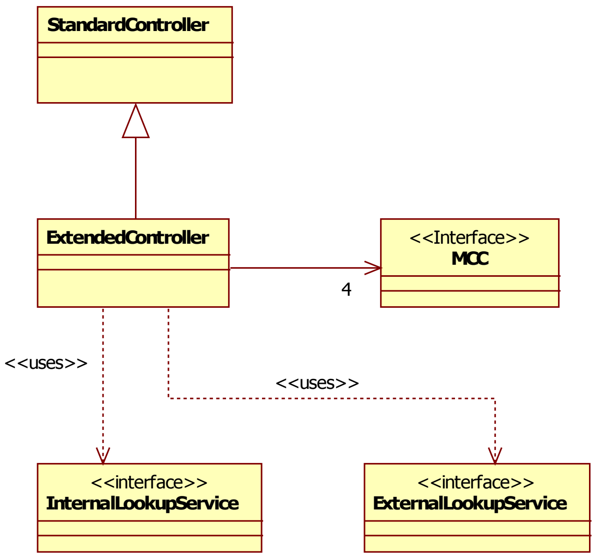

Figure 3.15.: RN class diagram

3.5.1. Resource management

Considering the component diagram in Figure 3.3, the *Resource Mgmt.* component is responsible to manage all local resources. In the presented approach two different resource types, namely CustomActions and CustomEvents are distinguished:

CustomActions are hard coded methods providing additional functionality which can be used within ECQL when defining the behavior of a statemachine-MCC. **CustomEvents** are events which occur in the underlying hardware and can be used as a trigger for transitions in the statemachine-MCC.

CustomActions can define either hardware dependent functionality like controlling an RFID reader or hardware independent functionality such as executing a special algorithm which would be to costly when realizing it with a statemachine-MCC and ECQL. Listening 3.1 shows a signal buzzer with variable frequency and signal duration as an example of a CustomAction. When defining the behavior of a statemachine-MCC, the custom action can simple be used by writing `beep(1000 , 2000)` within ECQL, to define a signal tone with the frequency of 1000Hz and the duration of 2000ms.

Listing 3.1: Signal buzzer

```
public class Beep : CustomAction
{
        private SjjEdkBoard board_;
        public Beep()
        {
            returnType = "Bool";
            arguments = new String[]{"Number", "Number"};
            multiplicity = "1";
            functionName = "beep";

            board_ = new SjjEdkBoard();
        }

        public override Object execute(Object[] parameters)
        {
            double freq = (double)parameters[0];
            double duration = (double)parameters[1];

            board_.beep((int)freq, (int)duration);
            return true;
        }
}
```

The resource management component is so designed to stay independent of the underlying hardware. Hardware independent CustomActions can be added to the runtime architecture by simply adding a dll with the compiled method implementation in it. The ResourceController then adds the additional CustomAction automatically at runtime. Hardware dependent CustomActions have to be defined before compiling and deploying the runtime architecture to the device. The implementation of the hardware dependent CustomAction however, can change during runtime.

3.5.2. Lookup service component

The lookup service component is responsible for managing the location of all VONs. When creating a new VON the *ExtendedController* registers the VON at the internal lookup service as well as the external lookup service. The internal lookup service manages the VON within the actual device, whereas the external lookup service is usually running on a different device or is distributed over several devices like the Domain Name Service (DNS). The external service knows each VON in the domain and hosts information how to access it. Through the usage of a URN scheme as described in 3.3, a location-independent resource identification can be achieved. The lookup services are used by the *Broker* class and the

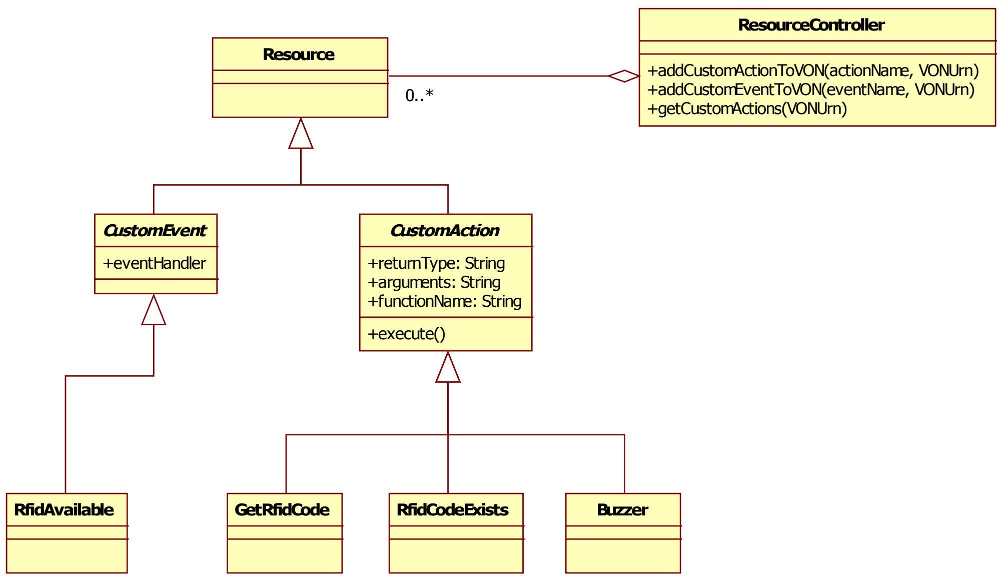

Figure 3.16.: Resource management component

ExtendedController class. The *ExtendedConroller* uses the lookup services to register the VONs at startup and deregister the VONs when shutting down. The *Broker* uses the lookup services to decide whereto forward incoming requests from the Proxy-MCCs.

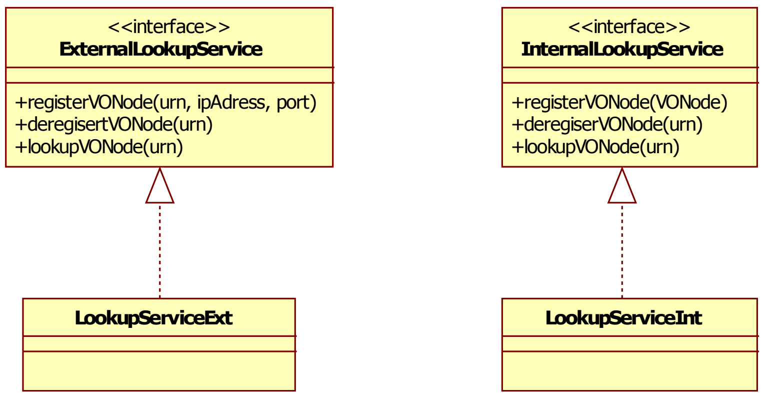

Figure 3.17.: Lookup service component

Figure 3.18 shows how the internal and external lookup services are used in the *Broker*. Each request leaving the current VON is transfered over the *Broker*. First the *Broker* performs a lookup to the internal lookup service to figure out if the seeked destination VON is accessible within the current device. If so it forwards the request to the *ServerProxy* of the appropriate VON. If the VON is not accessible within the device the *Broker* performs a lookup to the external lookup service to figure out on which device the seeked VON resides. Once the location of the VON has been determined, the *Broker* forwards the request inclusively the destination information to the *Bridge*. The *Bridge* marshals the request and subsequently transmits the message to the appropriate device.

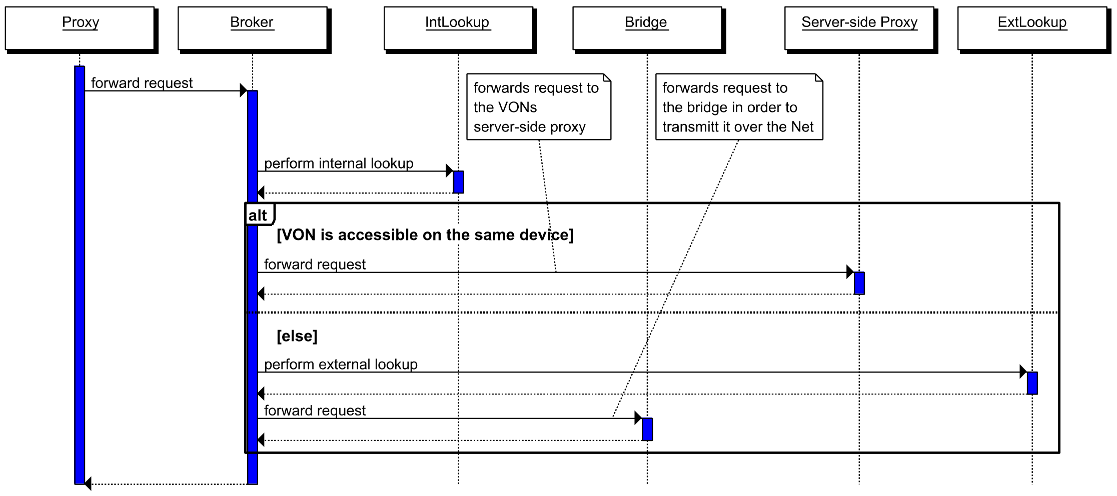
Proxy
Broker
IntLookup
Bridge
Server-side Proxy
ExtLookup
forward request
forwards request to the VONs server-side proxy
forwards request to the bridge in order to transmitt it over the Net
perform internal lookup
alt
[VON is accessible on the same device]
forward request
[else]
perform external lookup
forward request

Figure 3.18.: Performing internal and external lookup

3.5.3. Value Added Reseller principle

Through the distinction between RN and VON the Value Added Reseller(VAR) principle can be realized. In contrast to a VON, a RN has the capability to create new RNs and VONs and manage access rights to local resources.

Figure 3.19 shows the startup procedure of a VAR setup. The *operator* downloads the configuration data which specify one RN within the current RN. When sending the *update* event, the RN controller creates a further RN for *owner1* which in turn has the capability to create VONs. *Owner1* transmits again configuration data to his VON in order to create a VON to which *owner2* can download his application.

In Figure 3.20 the architectural setup of the VAR principle is shown. RN0 which already exists at startup belongs to the operator. The operator configures his configuration-MCCs in such a way to create a VON for himself and an additional resource node, RN1, for *tenant1*. The operator also needs to assign access rights to resources to RN1, which in turn can assign this resources to VONs in his area of responsibility. Although the *operator* is the owner of RN1 (because he is the one who created the RN), *tenant1* has the capability to create additional VONs within the RN. In the presented setup *tenant1* creates two VONs, one for himself and one for *tenant2*.

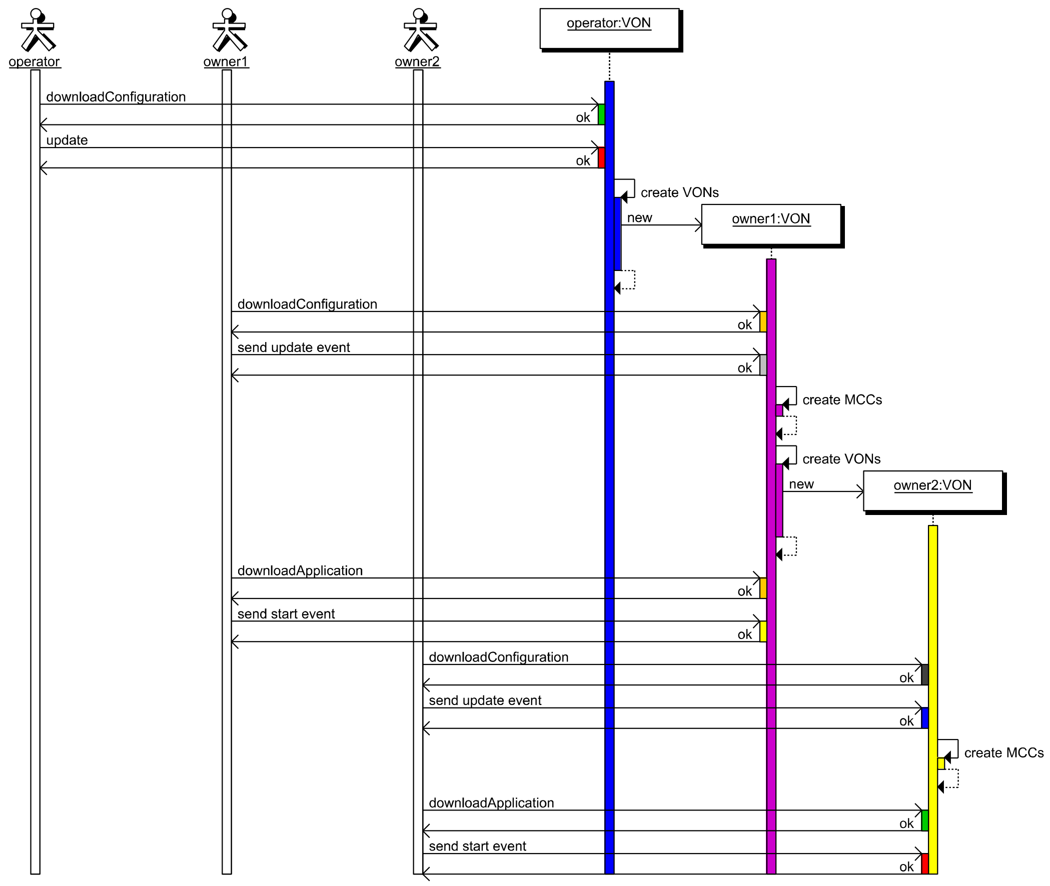

Figure 3.19.: Value added reseller principle

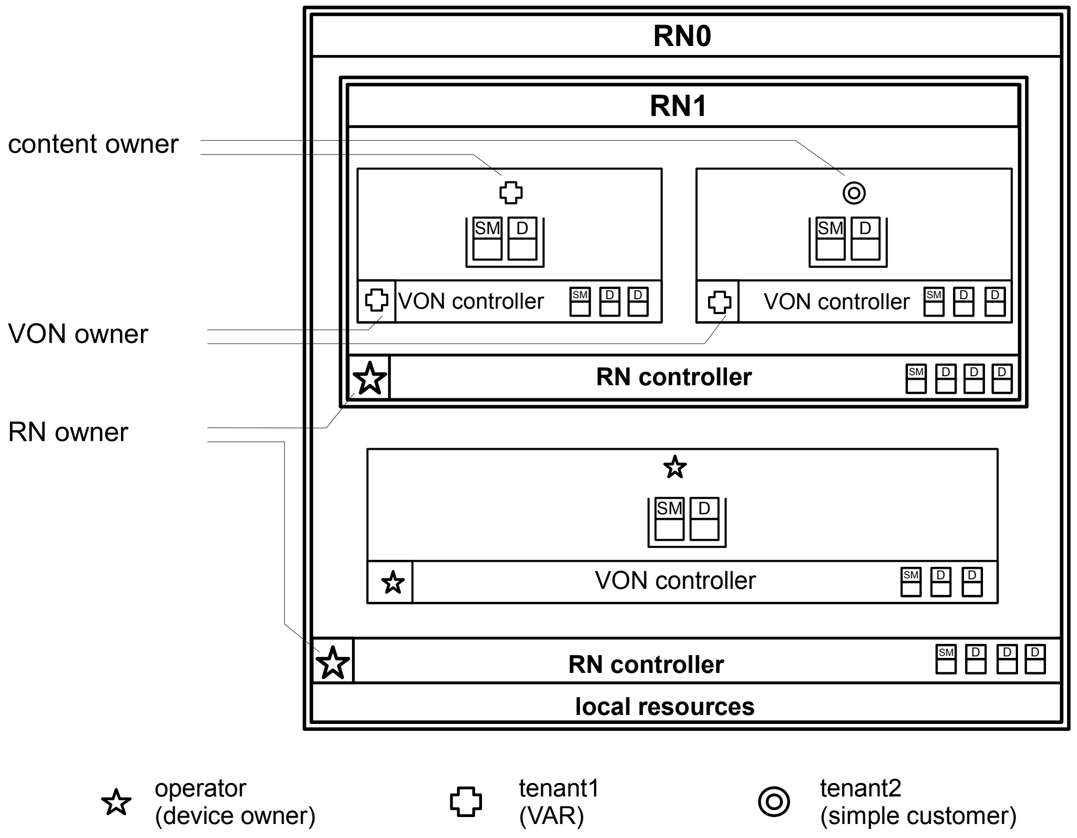

Figure 3.20.: Sample setup of the VAR principle

4. Implementation of a model-based middleware for virtual organizations

Due the fact the project targets not only standard systems but also embedded and mobile devices one requirement was to have an implementation which can be executed on different .Net runtime environments. The following section gives a short overview about the different .Net Frameworks which are currently available.

4.1. .Net Framework

The .NET Framework is the most powerful .Net Framework from Microsoft and can be installed on computers running Microsoft Windows operating system. The .NET Framework consists of two main parts.

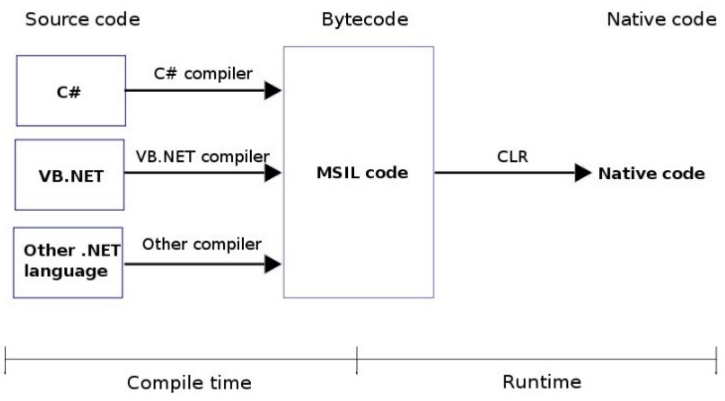

Figure 4.1.: CLR diagram

The **Common Language Runtime** (CLR) is Microsoft's implementation of the Common Language Infrastructure (CLI) standard, specified and published under ECMA-335 and ISO/IEC 23271. The CLR is the foundation of the .Net Framework and manages thread execution, code execution and other system services. Developers using the CLR write code in an high-level language like C# or VB.NET. At compile time the compiler converts the code into a Common Intermediate Language (CIL) and at runtime the CLR's just-in-time compiler generates native code targeting the operating system. The process from source code to native code is illustrated in figure 4.1.

CLR makes object interaction across language boarders possible. Therefore objects written in different languages can communicate with each other and their behavior can tightly integrated. A good example would be that you are able to define a class in one language an can derive from that class in a class written in another language. Through the common language runtime you gain benefits like *strong type safety, garbage collection and a good blend of visual Basic simplicity an C++ power*, when writing code using the C# language. [Mic09a]

The **.NET base class library** is a collection of reusable classes which are tightly integrated in the common language runtime. These classes provide a number of common functions like file reading and writing, graphic rendering and database interaction. In addition to this common functions the library supports a variety of specialized development scenarios such as Windows GUI application, Windows Presentation Foundation (WPF) applications and Web services.

4.1.1. .Net Compact Framework

The .NET Compact Framework from Microsoft is a scaled down version of the full .NET Framework specially targeting mobile/embedded devices such as PDAs, mobile phones, set-top boxes, etc. The Compact Framework is designed to run on the Windows CE operating system and provides native functions through interoperability with the OS to an application. It is a subset of the full .Net Framework and implements about 30 percent of the class library but also contains specific classes and features for embedded development. Some main differences between the two frameworks are:

- **Common Language Runtime.** The CLR of the .Net Compact Framework is about 12 percent the size of the full .Net Framework CLR. But the CLR in both Frameworks benefit from just-in-time compilation, managed code execution and garbage collection.
- **Delegates.** Asynchronous delegates are not fully supported.
- **Languages.** Application development using Visual Basic and Visual C# are supported but not C++.
- **Math.** Not all math methods are supported on all device platforms.
- **Memory.** Due the optimization for battery-powered systems the .Net Compact Framework avoids heavy use of RAM and CPU cycles.
- **Reflection.** The .Net Compact Framework does not support the System.Reflection.Emit namespace.
- **Sockets.** Not all socket options are supported.

An exhaustive list containing all differences between the two frameworks can be found on the Microsoft homepage [Mic09b].

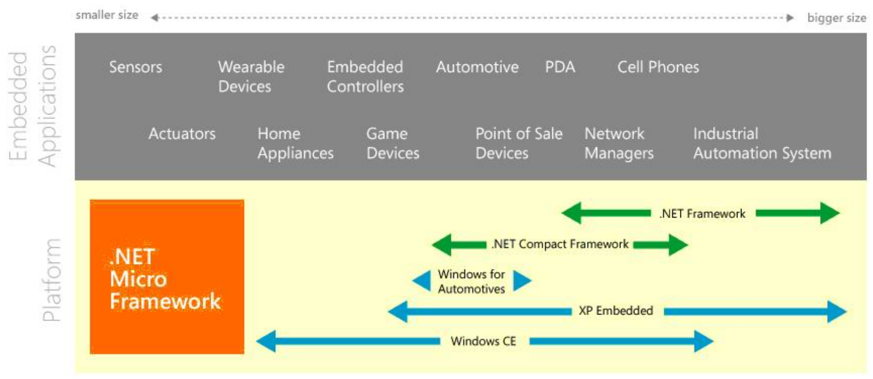

Figure 4.2.: Microsoft Embedded Products [TM07]

4.1.2. .Net Micro Framework

The .NET Micro Framework from Microsoft is a small and efficient .NET runtime environment used to run managed code on small and resource constrained devices. These devices are mostly too small regarding processor power and memory to be capable for Windows CE and the .NET Compact Framework. The .Net Micro Framework enables one to write within C# and Visual Studio application for small embedded devices and use the same tools as if developing for desktop applications. The .NET Micro Framework is not a full-featured operating system but a bootable runtime environment for embedded development. A scaled down version of .NET Common Language Runtime, called Tiny-CLR resides directly on the hardware. This, and the fact that the runtime only needs a few hundred kilobytes of RAM, makes it possible for the Micro Framework to run directly on small and inexpensive 32-bit processors [Kö08].

Microsoft offers a variety of embedded platforms differing in size, ranging from Windows XP Embedded to Windows Embedded CE to the .NET Micro Framework. In comparison to Windows XP Embedded and Windows CE which have a deep support for operating system extension, the .NET Micro Framework was developed of being as power-efficient as possible and resource-light while preserving the benefits of a managed code environment. The strengths of the .NET Micro Framework include [TM07]:

- Lower hardware cost than other managed platforms
- Lower development costs than other embedded platforms
- Lower power consumption

The .Net Micro Framework is made up of different layers as shown in Figure 4.3. The bootable runtime consists of the **Hardware Abstraction Layer** (HAL) and the **Platform Abstraction Layer** (PAL). The HAL is the only layer which is tightly coupled to the underlaying hardware. It is typically 20-30 KB and provides generic access to all the device's peripherals as well as infrastructure for booting applications. Because the .NET Micro Framework enables services and functionality that are usually provided by an operating system it is not necessary to have an underlying operating system, although an operating system may still be used if desired. When running the .NET Micro Framework on an OS the HAL provides functionality by calling through to OS functionality.

The layer above the HAL, the PAL, exposes abstractions such as timers, memory blocks and asynchronous communication. The PAL is responsible to provide software services required by the CLR such as bootstrapping, timers, memory management, debugging and events. If an operating system is available OS facilities may be used for some of these functionality. [TM07]

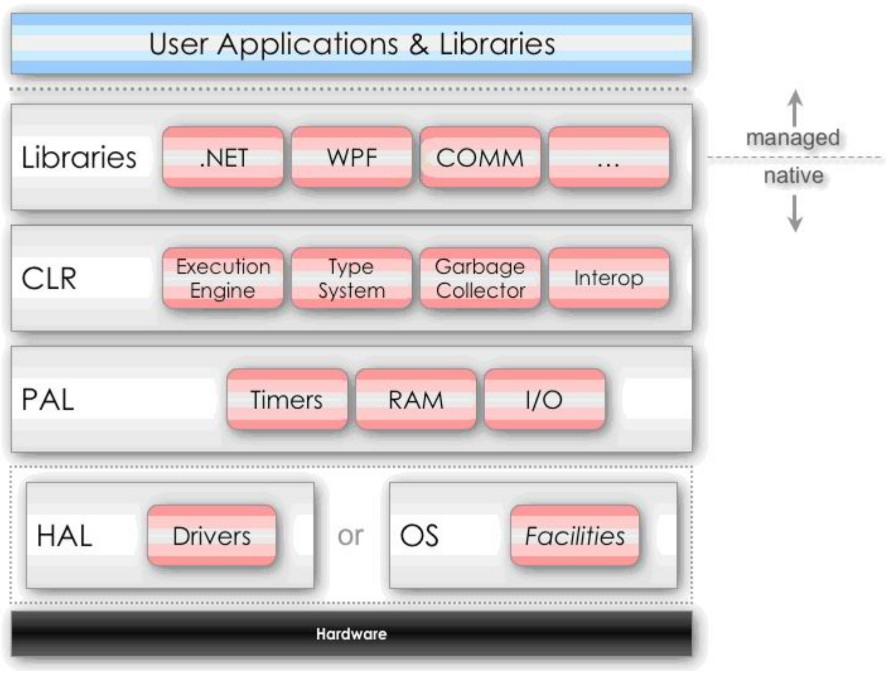

Figure 4.3.: Micro Framework Architecture [TM07]

The .NET Common Language Runtime is a highly optimized managed-code runtime that provides the main benefits of managed code such as safety, security, resource protection, validation, recovery and isolation. **The Micro Framework CLR** also known as tinyCLR supports the C# programming language and includes a class library adapted for the special needs of embedded applications. Furthermore the CLR contains a garbage collector for automatically freeing unused memory blocks. In [TM07] following primary design goals for the CLR are mentioned:

- Minimal footprint
- Able to run from ROM or flash memory
- Optimized for energy-efficiency in battery-powered devices
- Relatively easy portability by running on the HAL

To provide a minimal footprint the CLR is implemented from scratch can run without a traditional OS and provides runtime and library functionalities appropriate to embedded development.

The top of the .NET Micro Framework Architecture is made up off the Class Library Layer and the Application Layer. The Class Library Layer is composed of a subset of the .NET class library tailored to embedded devices. Additionally the Microsoft.SPOT namespace exists, including classes specific to the .NET Micro Framework such as hardware, I/O and cryptography. The uppermost layer, the Application Layer consists of C# managed applications and managed drivers that run on the embedded device.

4.1.3. Comparison

The CLR of the .Net Micro Framework implements the major features of the full .Net CLR such as the execution engine and memory management. Of course some of the features were omitted because they are inappropriate for that sort of devices while other features were added which are specific to embedded applications.
Following features are commonly found in .Net CLRs but have been omitted in the .Net Micro Framework CLR. [TM07]

- Machine-dependent types and unsafe instructions
- Symmetric multiprocessing
- MSCORLIB functionality has been reduced
- Multi-dimensional arrays
- Exception handling in native code
- MMU support

The .Net Micro Framework CLR supports a subset of the Base Class Library of the full .Net Framework. In the System namespace 70 classes are defined which are representing about 420 methods. The full .Net Framework in comparison implements approximately 1450 classes and 22500 methods. Another big difference between the .Net Micro Framework and the full .Net Framework is that the execution engine from the Micro Framework CLR only offers the interpreter mode. A just-in-time (JIT) compilation as it is offered by the full .Net CLR is not supported. The interpretive approach in the .Net Micro Framework CLR offers following benefits: [TM07]

- Smaller code footprint: Managed code is represented in less space than native code and results in a more compact engine code.
- Increased safety by fine-grained control at the instruction level and less complex engine code.

Because the .Net Micro Framework is the smallest framework supporting the C# language and also low-cost hardware for the framework is available it has been decided to implement the middleware based on the Micro Framework. The benefit of implementing the project within the .Net Micro Framework is that the solution will also run with slight modification on the .Net Compact Framework and the full .Net Framework.

Feature	**SJJ Micro**	**GHI**
Processor	200MHz	200MHz
RAM	8MB	64MB
Flash	8MB	256MB
MF Version	4.0	4.0
Network	TCP/IP	TCP/IP
Serial Port	2 RS-232	1 RS-232
USB	not supported	dual USB host ports
SPI/I2C	supported	supported
A/D	not supported	5 channels of 16bit A/D
PWM	2 PWM	not supported
GPIO	16 lines	80 lines
Memory Card	not supported	SD/MCC
Touch Screen	not supported	4.3" TFT

Table 4.1.: Key features of the .Net MF boards

4.2. Hardware

To test the application in real environment two .Net Micro Framework boards have been bought. The first one is a low-cost board from the company SJJEmbedded Micro Solutions with an ARM9 processor and the second one is a development system from GHI Electronics also with an ARM9 processor. A comparison of the key features of these devices is given in Table 4.1.

SJJ EDKplus for .Net Micro Framework

The EDKplus by SJJ is an iPac-9302 from EMAC with a .Net Micro Framework port directly on hardware. The iPac-9302 is a low-cost, embedded Single Board Computer (SBC) based on the Cirrus EP9302 processor. The Cirrus processor is an ARM9 based processor running with 200MHz and a System Bus of 100MHz.

GHI ChipworkX Development System

The GHI ChipworkX Development System is a development kit for the ChipworkX Module. This system exposes all peripherals and includes a 4.3"' touch screen. The ChipworkX Module features an ARM9 processor that hosts the .Net Micro Framework. Beside of the standard .Net Micro Framework features like FAT and USB it supports GHI specific features such as USB host, PPP, GPRS and 3G. Furthermore it supports a SQLite database enabling logging and retrieving of queries.

4.3. Scenario

This section describes a scenario realizing a door access control system that is composed by several organizations. Each organization has the ability to configure and reconfigure application parts, falling within their scope of responsibility, dynamically at runtime. The following organizations are involved in the szenario.

- Facility management
- Security service
- DepartmentA
- DepartmentB

The facility management organization is in the first place the hardware provider equipping the building with an RFID auto-identification, but it also provides basic services to departmentA and departmentB. DepartmentA and DepartmentB are two organizations whose offices are located inside the building. Each organization pursue its own authentication strategy and due to data privacy reasons, only the particular department has access to the employee's personal data and can therefore decide if a person is allowed to enter the building or not. The security service is hired by the facility management organization and is responsible for security affairs.

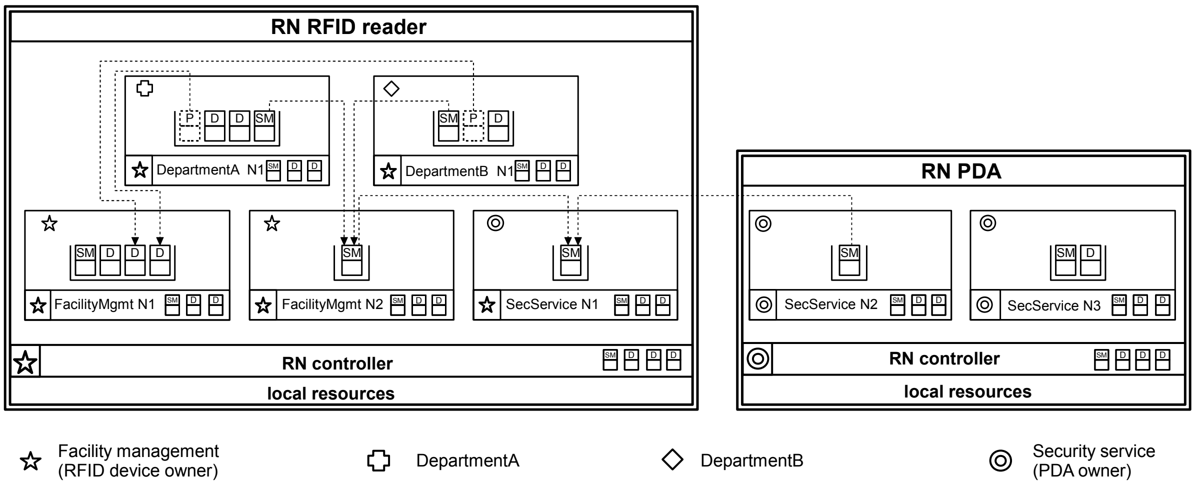

Figure 4.4.: Overall configuration and VON deployment

In this scenario furthermore two devices are involved. The main device is an **RFID reader** running the .Net Micro Framework, which is mounted at the building's door. Through locally attached resources the RFID reader is qualified to read RFID tags and also to control the door's locking mechanism. Furthermore it features a built-in buzzer, an LC display and LEDs to communicate with the user and to indicate the door's status. The second device is a **mobile PDA** running the .Net Compact Framework, which is owned by the security service enabling the guard duty to remotely trigger the door events *open*, *lock* and *unlock*. Furthermore a second application component on the PDA, the logging component, enables the guard duty to get notified whenever a person enters the building.

The next sections explain in more detail the two hardware nodes and the application components which are running on them. All involved VONs and their associated URNs

Label	Organization	URN
FacilityM N1	Facility management VON1	urn:vo:facilityMgmt:node1
FacilityM N2	Facility management VON2	urn:vo:facilityMgmt:node2
DepA N1	DepartmentA VON1	urn:vo:departmentA:node1
DepB N1	DepartmentB VON1	urn:vo:departmentB:node1
SecService N1	Security service VON1	urn:vo:securityService:node1
SecService N2	Security service VON2	urn:vo:securityService:node2
SecService N3	Security service VON3	urn:vo:securityService:node3

Table 4.2.: VONs and their associated URNs

are shown in Table 4.2. Table 4.3 shows all VONs and the access rights to the particular native functions. The involved events in this scenario can be seen in Table 4.4, highlighting VONs generating such events and VONs subscribing to these, in order to get notified upon the occurrence of the event.

4.3.1. RN RFID reader

The facility management organization owns an RFID reader, running the .Net Micro Framework. On this device five VONs are created which belong to different organizations participating in this scenario.

4.3.1.1. Facility Management VON1

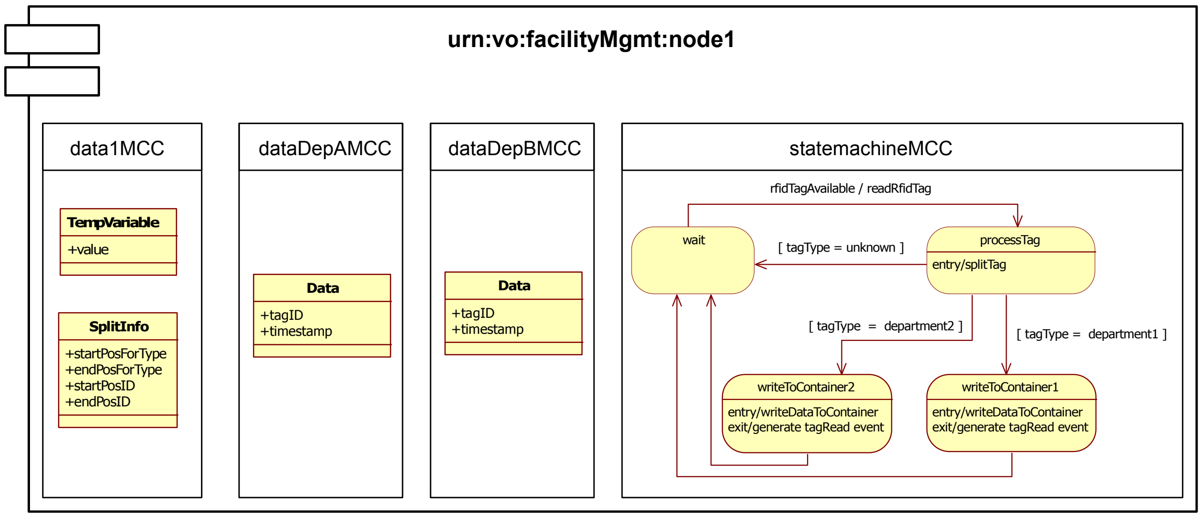

Figure 4.5.: VON1 of the facility management organization

This VON hosts an application reading RFID tags and analyzing the type field of the tag. Depending on the type, the application writes the id of the tag and a corresponding timestamp to different data-MCCs (DataDepAMCC and DataDepBMCC). These containers are used by the application components of departmentA respectively departmentB to retrieve the information of the person who wants to enter.

After inserting the data to the appropriate MCC the statemachine generates a *tagReadDepA* respectively *tagReadDepB* event to notify the corresponding VON that a person wants to access. After tag processing is finished the statemachine goes back into *Wait* state and stays idle until the next *rfidTagAvailable* events occurs. All involved MCCs and their entities in the M1 layer are shown in Figure 4.5.

4.3.1.2. Facility Management VON2

The task of this VON is to forward the *open* event getting send by departmentA or departmentB to VON1 of the security service organization. In this way a separation between the departments and the security service organization is achieved. To forward events only one statemachine-MCC is needed which is shown in Figure 4.6.

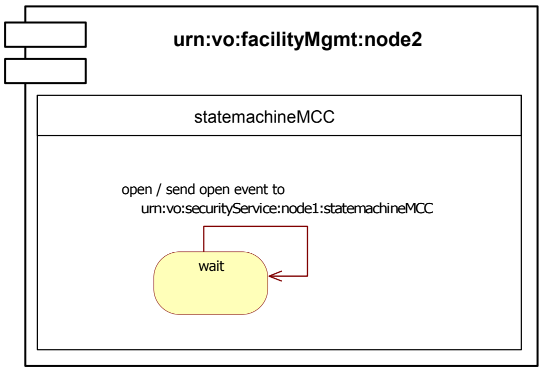
urn:vo:facilityMgmt:node2
statemachineMCC
open / send open event to
urn:vo:securityService:node1:statemachineMCC
wait

Figure 4.6.: VON2 of the facility management organization

4.3.1.3. DepartmentA VON1

This VON is run and managed by departmentA. The purpose of the application component is to read the id-data provided by the facility management VON1 *(urn:vo:facilityMgmt:node1)* and to verify if the id is valid. If so the statemachineMCC sends an *open* event to the statemachineMCC of the facility management organization VON2.

The statemachine-MCC of departmentA gets notified every time a tag is read by the facility managements VON1. After reading and validating the id the statemachine inserts or updates the attributes of the *EmployeeData* entities in *dataMCC2* according to the entering or leaving time. Additional, VON1 of departmentA has the possibility to use the resource *writeTextToDisplay()*. Thus departmentA can specify a personal welcome message for its employees.

Furthermore the VON hosts two data-MCC. *Data1MCC* contains all valid employee ids and their appropriate names. *DataMCC2* contains entities which are inserted and updated by the statemachine according to the time employees are entering or leaving the building. Figure 4.7 presents the M1 layer of the two data-MCC and the statemachine-MCC.

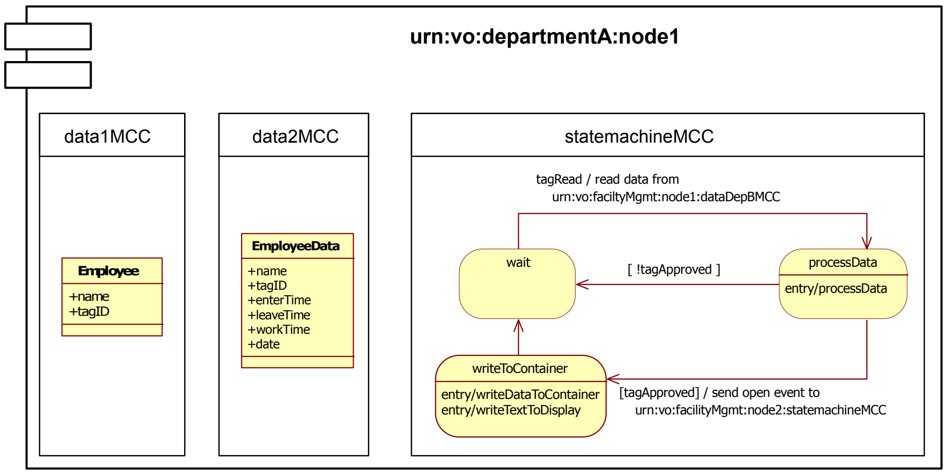

Figure 4.7.: VON of departmentA

4.3.1.4. DepartmentB VON1

This VON is run and managed by departmentB. A data-MCC within the VON hosts all employees which are allowed to enter the building. The statemachine-MCC within the VON executes the authorization steps and if a employee is granted access, it sends an *open* event to the statemachine-MCC of the facility management VON2 *(urn:vo:facilityManagement:node2:stateme*

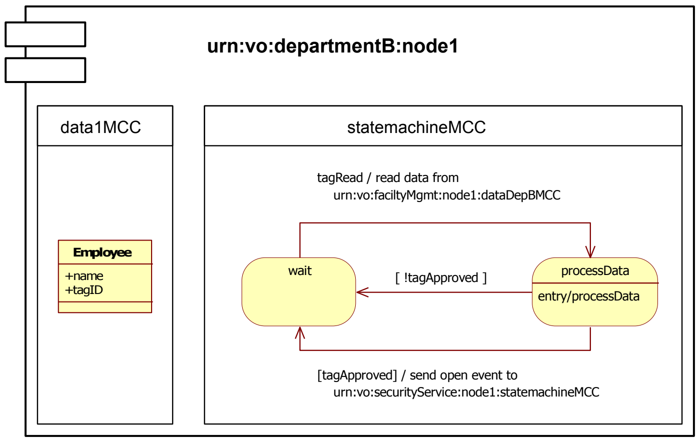
urn:vo:departmentB:node1
data1MCC
Employee
+name
+tagID
statemachineMCC
tagRead / read data from urn:vo:faciltyMgmt:node1:dataDepBMCC
wait
[!tagApproved]
processData
entry/processData
[tagApproved] / send open event to urn:vo:securityService:node1:statemachineMCC

Figure 4.8.: VON of departmentB

4.3.1.5. Security Service VON1

The VON1 of the security service contains only one statemachine-MCC reacting to the incoming events *open*, *lock* and *unlock*. Depending on the incoming event and on the current state of the statemachine different actions are carried out. The M1 model interpreted by this statemachine-MCC is shown in Figure 4.9.

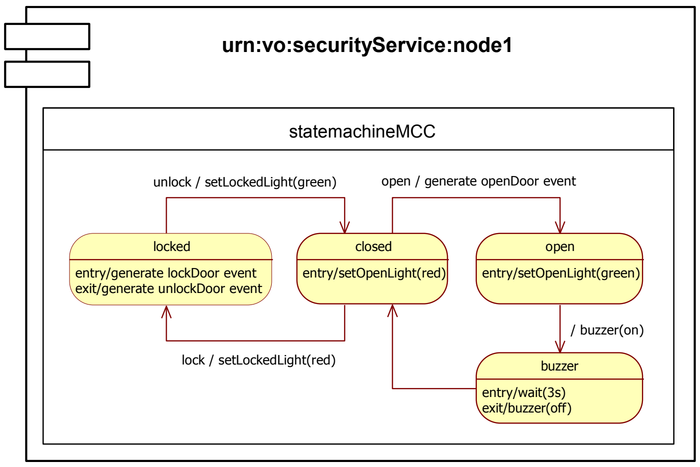
urn:vo:securityService:node1
statemachineMCC
unlock / setLockedLight(green)
open / generate openDoor event
locked
entry/generate lockDoor event
exit/generate unlockDoor event
closed
entry/setOpenLight(red)
open
entry/setOpenLight(green)
/ buzzer(on)
lock / setLockedLight(red)
buzzer
entry/wait(3s)
exit/buzzer(off)

Figure 4.9.: VON1 of the security service organization

Additional the statemachine-MCC provides the following events to which other VONs can subscribe to.

- lockDoor: This event is generated when the door gets locked.
- unlockDoor: This event is generated when the door gets unlocked.
- openDoor: This event is generated when somebody enters the building.

4.3.2. RN PDA

The security service organization owns a PDA running the .Net Compact Framework. On this device two VONs are created, one to control the door and a second VON for logging all door activities.

4.3.2.1. Security Service VON2

This VON host only one statemachine-MCC receiving user inputs and forwarding events to the VON *(urn:vo:securityService:node1)*, which resides on the RN RFID reader. The user of this PDA has three buttons to control the door remotely (open, lock, unlock). The statemachine running in the VON sends the events to *urn:vo:securityService:node1: statemachine* which in turn carries out the action. The M1 model interpreted by this statemachine-MCC is shown in Figure 4.10.

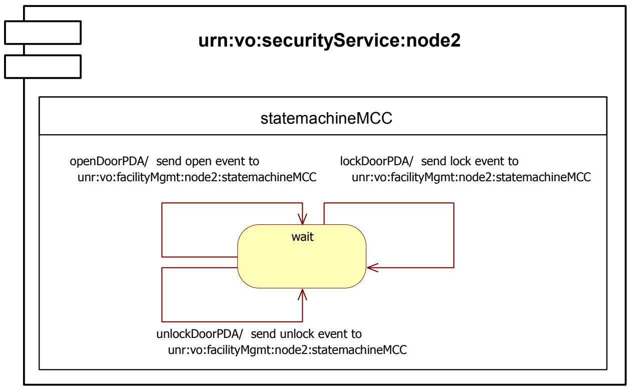
urn:vo:securityService:node2
statemachineMCC
openDoorPDA/ send open event to unr:vo:facilityMgmt:node2:statemachineMCC
lockDoorPDA/ send lock event to unr:vo:facilityMgmt:node2:statemachineMCC
wait
unlockDoorPDA/ send unlock event to unr:vo:facilityMgmt:node2:statemachineMCC

Figure 4.10.: VON2 of the security service organization

4.3.2.2. Security Service VON3

This VON hosts an application for logging. In order to be able to log all events concerning the door the VON must be subscribed to the events *lockDoor*, *unlockDoor* and *openDoor* which are provided by *urn:vo:securityService:node1:statemachine*. Figure 4.11 depicts the statechart executed by the statemachine-MCC of this VON.

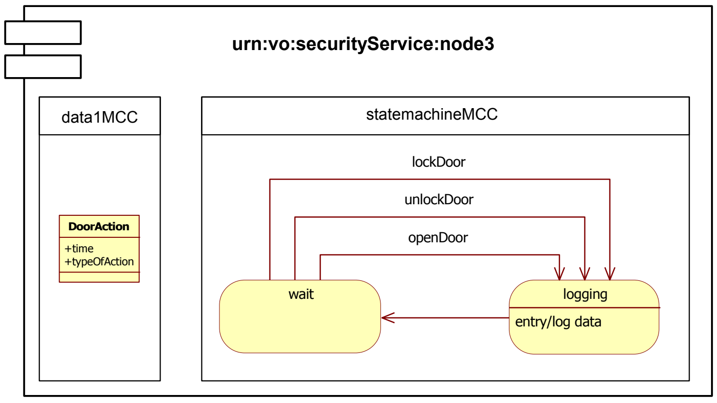

Figure 4.11.: VON3 of the security service organization

Native functions	**RN RFID reader**					**RN PDA**	
	FacilityM N1	FacilityM N2	DepA N1	DepB N1	SecService N1	SecService N2	SecService N3
readRfidTag	X	-	-	-	-	-	-
buzzer	X	-	-	-	X	-	-
writeToDisplay	X	-	X	-	-	-	-
openDoor	-	-	-	-	X	-	-
closeDoor	-	-	-	-	X	-	-
lockDoor	-	-	-	-	X	-	-
unlockDoor	-	-	-	-	X	-	-
setLockedLight	-	-	-	-	X	-	-
setOpenLight	-	-	-	-	X	-	-
getTimestamp	X	X	X	X	X	X	X

Table 4.3.: VONs and their assigned native functions

Native functions	**Source**	**RN RFID reader**					**RN PDA**	
		FacilityM N1	FacilityM N2	DepA N1	DepB N1	SecService N1	SecService N2	SecService N3
rfidAvailable	RFID reader	X	-	-	-	-	-	-
tagRead	FacilityM N1	-	-	X	X	-	-	-
lockDoor	SecService N1	-	-	-	-	-	-	X
unlockDoor	SecService N2	-	-	-	-	-	-	X
openDoor	SecService N3	-	-	-	-	-	-	X

Table 4.4.: Event sources and their subscribers

4.4. Results

This section discusses the outcomes of the work regarding to the code size, the size of the JSON models and the memory consumption at runtime.

Figure 4.12 shows the code size of the different libraries developed in the project. The three main components are the *MCC* which was already fully implemented at the beginning of the work, the *Remoting* component which was adapted to fulfill the requirements regarding the remoting aspect and the *VO* component which was the main part of this work and realizes the whole VO support. The *Device Driver* component is implementing low level functionality to communicate with the Feig LRU2000 RFID reader and the *Custom Function* component is implementing the native functions as shown in Table 4.3.

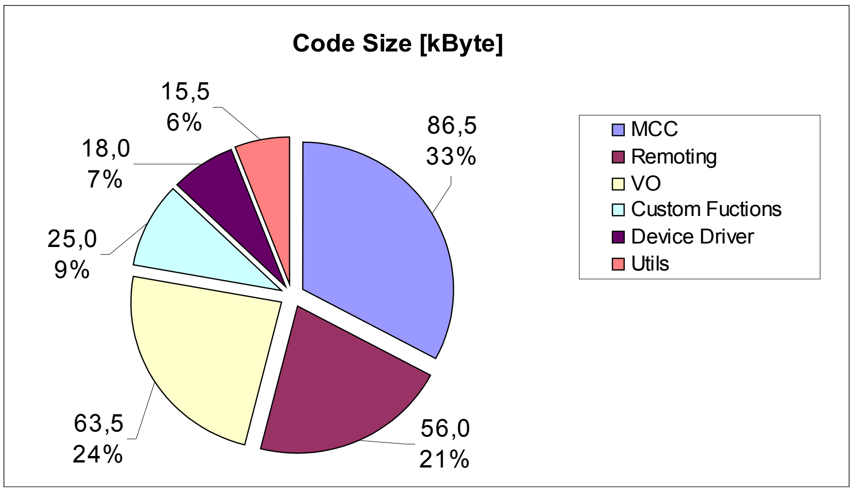

Figure 4.12.: Code size of the different implementation components

The code sizes depicted in Figure 4.12 however are not containing the JSON models which are necessary to realize a specific behavior of an application component. The JSON models listed in Table 4.5 are the M0-layer models for the data-MCCs in the controlling unit and the M0/M1-layer model for the statemachine-MCC in the controlling unit. This models are transferred to the RNs controlling unit as soon as an RN gets created. All these models are treaded as static an can not be modified at runtime.

In the case a new VON gets created only the models 2, 3 and 4 are transfered because a VON is not allowed to create further VONs. After the creation of the RN respectively VON the models (M1 and M0) of the application components are transferred to the VONs. For the above described scenario the sizes of the JSON files containing the models and data for each VON are given in Table 4.6. The size of the JSON models depends mainly on how many MCC are specified in the application area of a VON, how large the class-models at the M1 layer are and how many transition, conditions and actions the statemachine-MCCs have.

	MCC	**Size**
1	data-MCC to specify the VON structure M1	917 byte
2	data-MCC to specify the MCC structure M1	675 byte
3	data-MCC publisher/subscriber mechanism M1	495 byte
4	statemachine-MCC M1 and M0	1647 byte

Table 4.5.: JSON model-sizes of the MCCs in the controlling unit

VON	**Size**
Facility management VON1	8,1 kbyte
Facility management VON2	1,3 kbyte
DepartmentA VON1	5,5 kbyte
DepartmentB VON1	3,4 kbyte
Security service VON1	2,6 kbyte
Security service VON2	2,5 kbyte
Security service VON3	2,0 kbyte

Table 4.6.: JSON models of all VONs involved in the scenario

So far only the size of the static libraries and static JSON models are determined, but due to the fact that the middleware has also to be runnable on resource limited devices an important aspect is how much memory is used by a VON and its appropriate models at runtime. Table 4.7 shows the allocated runtime memory for a data-MCC, a statemachine-MCC, a VON and a RN. It has to be mentioned that these values are just reflecting the frame of a MCC or VON, they are neither containing models (M1-layer) nor data (M0-layer).

VON	**Size**
data-MCC	1,5 kbyte
statemachine-MCC	3,4 kbyte
VON	5,5 kbyte
RN	11,3 kbyte

Table 4.7.: Allocated memory without models

Figure 4.13 shows the allocated runtime memory per VON after all models are transfered in the case of the scenario described above. The VONs of the RFID reader occupies 264 kbyte after transmission of all models. The major part is needed by VON1 of the facility management organization. Because this VON has also the largest JSON model, it is not surprising that it occupies also the most runtime memory. The JSON models for the VONs of the PDA are not that complicated and as a result the allocated memory of these is far less than the allocated memory of the VONs on the RFID reader.

When comparing the values in Table 4.7 and those given in Figure 4.13 it can be clearly

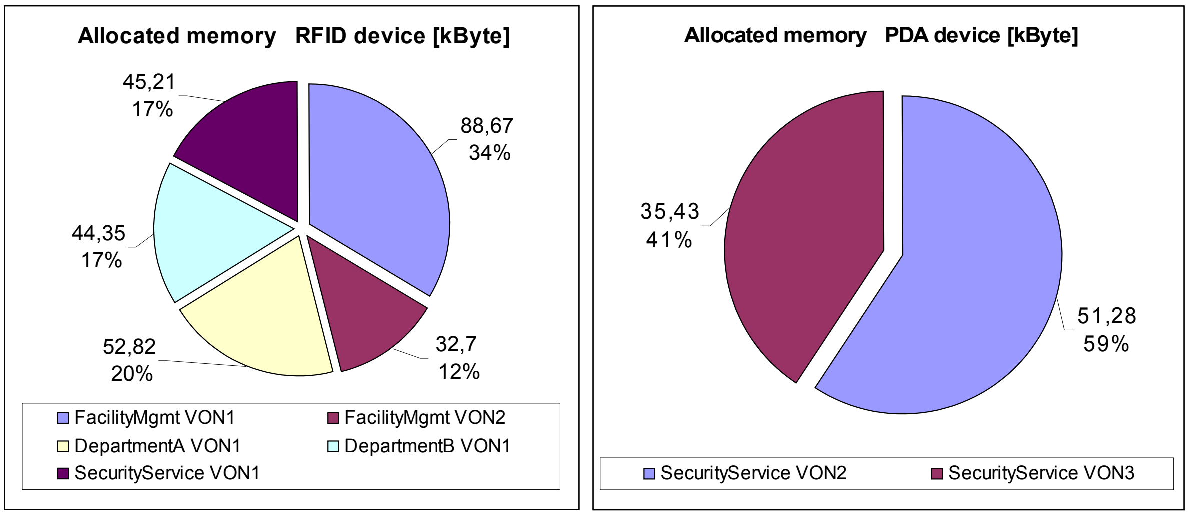

Figure 4.13.: Memory fragmentation at runtime

seen that the most memory is needed by the models which are transfered to the VONs. The larger the JSON files are the more memory is needed at runtime. Because of the overhead in the data representation in the MCC a model needs ten to twenty times more memory at runtime as it is requiring in the JSON representation.

Because dynamic configuration and reconfiguration was a strong requirement the downtime of an application during transferring models to the device is an big issue. Table 4.8 is showing the downtime of a VON during reconfiguration with two data-MCCs and one statemachine-MCC and their appropriate M0/M1 data. This time includes stopping the VON, transferring all models and restarting the VON. Reconfiguring VONs on a Desktop PC or on a PDA is quite fast and the downtime of the application component in the VON is just about one to two seconds. Updating models of a VON on the GHI ChipworkX device takes up to 10 times longer. This can be explained on the one hand through the slower processor and on the other hand through the missing just-in-time compiler (JIT) in the .Net Micro Framework.

Device	**Platform**	**Time**
GHI ChipworkX (200Mhz)	.Net Micro Framework	10s
PDA (600MHz)	.Net Compact Framework	2s
Desktop (1400MHz)	.Net Framework	1s

Table 4.8.: Period of time for transferring models to a VON

5. Conclusion

The presented master's thesis is part of the You-R® Open Next Generation (YRO-NG) project at the Institute for Technical Informatics. The Goal of this work is to provide a middleware supporting the concept of Virtual Organizations (VO) to realize distributed architectures across different administrative domains. A strong requirement was the decentralized configuration of different VOs in a model-based way.

An overview about Grid computing systems, wherefrom the concept of VOs arose, has been given. Furthermore the basic ideas behind Model-Driven Software Development (MDSD) have been presented, which are essential for the understanding of the Model-based Component Container (MCC). Different types of MCCs have been introduced, where each of them defines and implements a concrete model paradigm covering a domain-specific aspect which in turn can be executed at runtime. These MCCs, which were already fully implemented at the beginning of this work, form an integral element in this thesis.

The design and implementation of a model-based runtime architecture for distributed pervasive systems has been presented, supporting the dynamics of virtual organizations (VO) by the separation of resource nodes (RN) and VO runtime nodes (VON). Local resources are controlled by the RNs, and made available to their VONs. VONs are used as execution platforms for MCCs containing a set of models for platform independent specification of domain specific functionality. The models are loaded into several MCCs dynamically which enables a configuration and reconfiguration of application components at runtime. Further communication strategies have been designed to allow data exchange between MCCs residing in different VONs and also different event mechanism have been covered to enable direct event notification as well as a model-based event notification over the publisher/subscriber mechanism.

An important aspect was the design of the VON in a self-contained way to allow the migration between different devices. This has been achieved on the one side by using a location-independent resource identifier, which was realized with Uniform Resource Names (URNs), and on the other side by realizing VON configuration in a model-based way using MCCs. Through this highly dynamic reconfigurable approach also the Value Added Reseller (VAR) principle can be achieved.

Different .Net runtime environments and two different embedded developments board featuring the .Net Micro Framework have been evaluated in this project. The prototypical implementation has been realized with the .Net Micro Framework. This implementation has been evaluated in a building access scenario involving personnel RFID auto-identification, owned by facility management, two departments with independent authentication strategies, and a security service company. This four independent organizations are running application components within VONs on the same RFID reader mounted at the building's door. Although all VONs are running on the same device each organization can configure the application components within its VONs independently.

The presented results illustrate the code size division of all involved components in the middleware, the associated Java Script Object Notation (JSON) files in the case of the presented scenario and also the allocated memory at runtime. The fact that the most memory at runtime is needed by the transmitted JSON files is attributable to the internal data representation in the MCCs.

Future work

Due to the fact that the whole application data as well as the application behavior is defined with models encoded in JSON, writing this models is a tough and error-prone undertaking. To simplify creating these models a broad tool-support is necessary. These tools should provide the possibility to formulate the data- and statemachine models in a graphical way as well as offer the opportunity to integrate the Entity Container Query Language (ECQL) [KTK10] and having a syntax validation before models getting transmitted to their MCCs. Not only tools supporting writing models, also some sort of version control could be a nice feature which simplifies model management and maintenance.

An interesting task for the future would be to include support for additional runtime platforms like JavaME and to construct new MCC model types to extend support for different domain specific aspects. Beside the existing data-MCC and statemachine-MCC, MCC-types could be implemented which are able to process for instance *decision tables* or a *Petri net*.

The security aspect have not been taken into account by this work, which is not a neglectable issue in modern distributed computing systems. Investigations in the area of authentication and authorization have to be made to prevent organizations to connect to MCCs outside their domain and to modify their content. This investigations should take into account the models of the VON an RN specified in this thesis.

A. Event Mechanism

Events are an integral part of the whole architecture and are used for communication and notification. Three different sources of events are distinguished:

- events which occur in the underlying hardware
- events which are send via proxys directly to statemachine-MCCs
- events which are generated by statemachine-MCCs themselves

Events which occur in the underlying hardware are handled as special resources as described in Sec. 3.5.1. A VON which wants to get notified when a hardware event arises must have the permission to get notified. This permission is assigned by the RN with the data-MCC described in 3.3.1.4. If an hardware event occurs and the VON has the permission to get notified the event gets inserted into the *Queue* of the appropriate VON. The *Controller* when ready reads the event from the *Queue* and forwards the event to **all** statemachine-MCCs in the application area because no specific destination address is associated with the event. Figure A.1 shows the sequence diagram for that case.

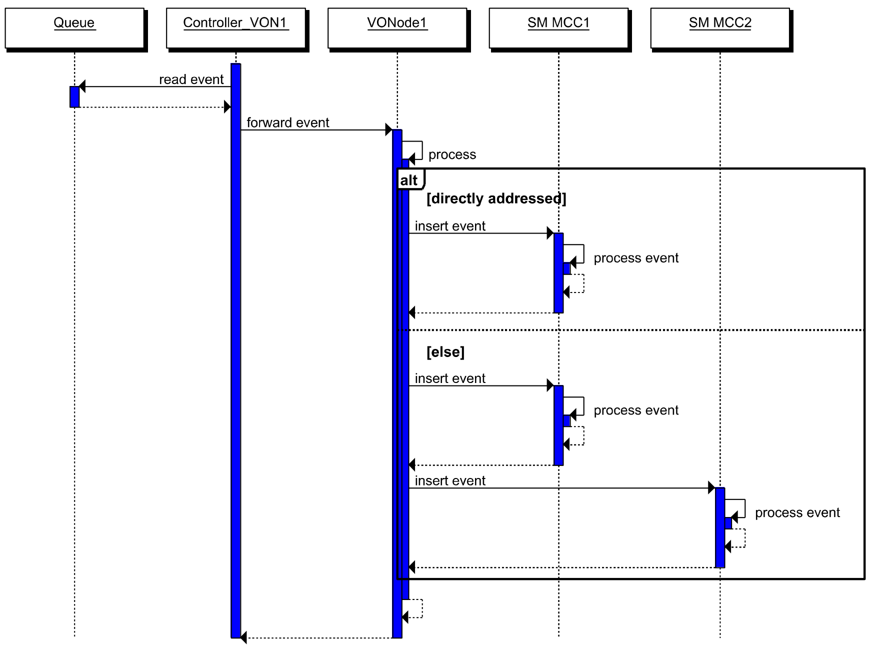

Figure A.1.: Events forwarded to specific MCCs

Events which are send via proxy-MCCs directly to statemachine-MCCs are also written to the *Queue* but there is a specific destination address associated with the event. Through this destination address the controller can decide to which statemachine-MCC the event has to be forwarded. The upper half of the sequence diagram in Figure A.1 shows this case.

When an event is generated by a statemachine-MCC itself two different scenarios have to be distinguished.

- a destination address is associated with the event
- no destination address is associated with the event.

If a statemachine-MCC generates an event during action execution a new event instance is generated and sent to the *Controller*. If a destination address is specified within the event the controller sends the event directly to the appropriate VON where the event gets inserted in the *Queue*. Figure A.2 depicts the sequence diagram for that case.

If the event has no destination address specified two different actions are carried out. The first is that the event gets inserted into the local event queue in order to give the other statemachine-MCCs in the same VONode the possibility to handle the event. The second action is the evaluation of the publisher-subscriber mechanism. The controller evaluates the M0 layer content of the data-MCC, which holds the URNs of all statemachine-MCCs that wants to get notified when the specific event occurs. Afterwards it sends the event to all VONs where the registered statemachine-MCCs reside. Again, before the event is processed by the dedicated statemachine-MCC the event is inserted into the queue to ensure causality. Figure A.3 shows the according sequence diagram.

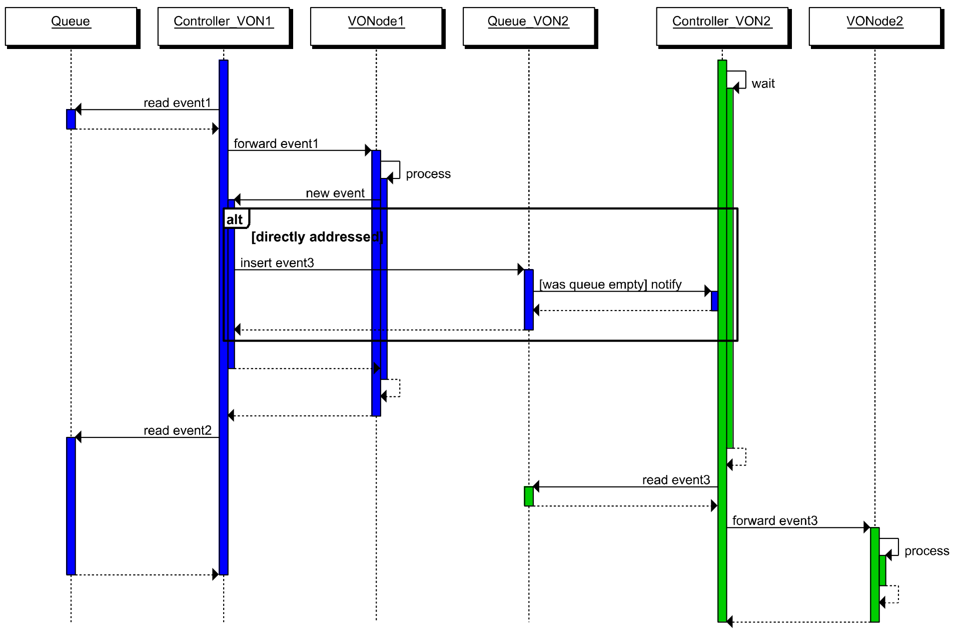

Figure A.2.: Handling events with specified destination address

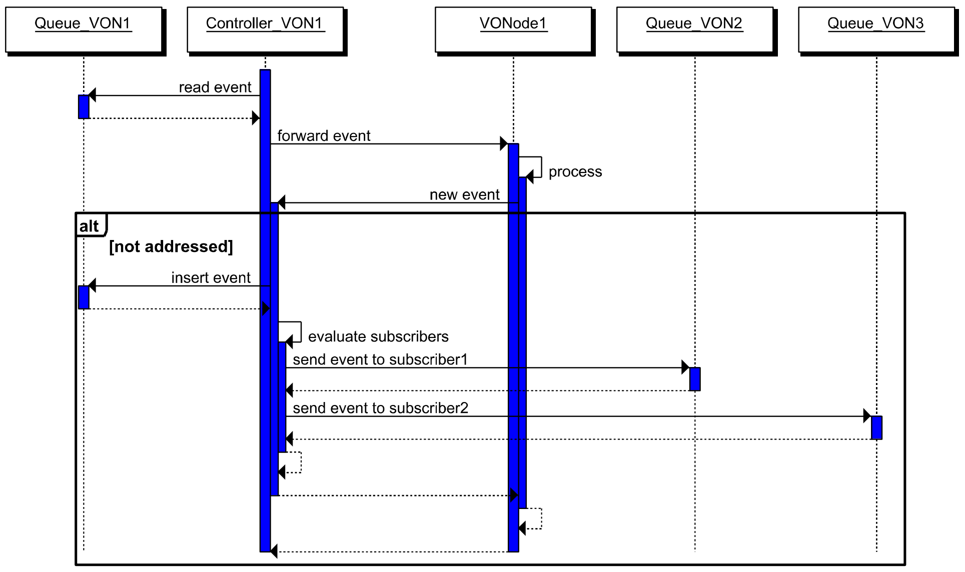

Figure A.3.: Handling events with no destination address specified

B. Development tools

This chapter gives an overview over the tools used during the design, implementation, and writing of this work.

UML

Class diagrams were produced with the *StarUML* tool which can be found at `http://staruml.sourceforge.net`

Sequence diagrams were produced with the *Quick Sequence Diagram Editor* which can be found at `http://sourceforge.net/projects/sdedit`

Microsoft Visual Studio

For code development *Microsoft Visual Studio 2008* was used. In addition to the standard *.Net Framework 3.5* also the *.Net Compact Framework 3.5* and the *.Net Micro Framework 4.0* had been installed in order to be able to develop and distribute programs to the GHI ChipworkX evaluation board and to the PocketPC.

Profiler

Memory and performance profiling was done with the tool *JetBrains dotTrace 3.1* which is available under `http://www.jetbrains.com/profiler/`

Open Office Draw 3.1

Used for drawing various architectural pictures and to post process exported diagrams from StarUML. Available under `http://www.openoffice.org/`.

Bibliography

[Acc10] AccessGrid.org. What is the Access Grid? Website, 2010. Available online at `http://www.accessgrid.org/faq`; visited on February 23th 2010.

[ALM04] Kaizar Amin, Gregor Laszewski, and Armin R. Mikler. Toward an architecture for ad hoc grids. In *12th International Conference on Advanced Computing and Communications (ADCOM 2004), Ahmedabad*, pages 15–18, 2004.

[ANG04] Ashish Agarwal, Douglas O. Norman, and Amar Gupta. Wireless Grids: Approaches, Architectures and Technical Challenges. *SSRN eLibrary*, 2004.

[BEF$^+$00] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch. Design and Deployment of a National-Scale Authentication Infrastructure. *IEEE Computer, 33(12):60-66. 2000.*, 2000.

[BHS07] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. *Pattern-Oriented Software Architecture; A Pattern Language for Distributed Computing.* John Wiley & Sons, West Sussex, England, 2007.

[BJK09] Ian Bird, Bob Jones, and Kerk F. Kee. The organization and management of grid infrastructures. *IEEE Computer Society Press*, 42(1):36–46, 2009.

[DM99] Gerardine Desanctis and Peter Monge. Introduction to the special issue: Communication processes for virtual organizations. *Organization Science*, 10(6):693–703, 1999.

[EPC04] EPCglobal. The EPCglobal Network. Technical report, EPCglobal, 2004. Available online at `http://www.epcglobalinc.org/about/media_centre/Network_Security_Final.pdf`; visited on February 25th 2010.

[FK96] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1996.

[FK99] Ian Foster and Carl Kesselman. *The Grid, Blueprint for a Future Computing Infrastructure.* Elsevier, 1999.

[FK04] Ian Foster and Carl Kesselman. *Grid2, Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, 2004.

[FKNT02] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. Grid services for distributed system integration. *Computer*, 35(6):37–46, 2002.

[FKT01] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[Fos02] Ian Foster. What is the Grid? A Three Point Checklist. Website, 2002. Available online at `http://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf`; visited on February 25th 2010.

[Fou08] National Science Foundation. Virtual Organizations as Sociotechnical Systems (VOSS). Website, 2008. Available online at `http://www.nsf.gov/pubs/2009/nsf09540/nsf09540.htm`; visited on February 23th 2010.

[FZRL08] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. *Grid Computing Environments Workshop, 2008. GCE '08*, 2008.

[GDL+04] Robert Grimm, Janet Davis, Eric Lemar, Adam Macbeth, Steven Swanson, Thomas Anderson, Brian Bershad, Gaetano Borriello, Steen Gribble, and David Wetherall. System support for pervasive applications. In *ACM transactions on Computer Systems, Vol.22, No.4*, pages 421–486, 2004.

[Gmb07] RF-IT Solutions GmbH. You-R® OPEN - The Integration for RFID Projects Version 3.3. Technical report, RF-IT Solutions GmbH, 2007.

[Gro00] OMG Group. Model Driven Architecture, a white paper by Richard Soley and the OMG Staff Strategy Group. Technical report, Object Management Group, 2000. Available online at `http://www.omg.org/cgi-bin/doc?omg/00-11-05.pdf`; visited on February 25th 2010.

[Gro04] NGG2 Group. Next Generation Grids 2: Requirements and Options for European Grids Research 2005-2010 and Beyond. Technical report, Expert Group Report, 2004. Available online at `ftp://ftp.cordis.europa.eu/pub/ist/docs/ngg2_eg_final.pdf`; visited on April 10th 2010.

[Gro06] OMG Group. Meta Object Facility (MOF) Core Specification; Version 2.0. http://www.omg.org/mof/, 2006. Available online at `http://www.omg.org/spec/MOF/2.0/PDF/`; visited on February 25th 2010.

[HHX+05] Xiao Haili, Wu Hong, Chi Xuebin, Deng Sungen, and Zhang Honghai. An implementation of interactive jobs submission for grid computing portals. In *ACSW Frontiers '05: Proceedings of the 2005 Australasian workshop on Grid computing and e-research*, pages 67–70. Australian Computer Society, Inc., 2005.

[HP03] Jaesun Han and Daeyeon Park. A lightweight personal grid using a supernode network. In *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, page 168, Washington, DC, USA, 2003. IEEE Computer Society.

[KÖ8] Jens Kühner. *Expert .NET Micro Framework*. Springer-Verlag, New York, 2008.

[KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[KLAR08] Heba Kurdi, Maozhen Li, and Hamed Al-Raweshidy. A classification of emerging and traditional grid systems. *IEEE Distributed Systems Online*, 9(3):1, 2008.

[KTK10] Ulrich Krenn, Michael Thonhauser, and Christian Kreiner. Ecql: A query and action language for model-based applications. In *Engineering of Computer Based Systems, 2010. ECBS 2010. 17th Annual IEEE International Conference and Workshop*, volume 0, pages 286–290. IEEE Computer Society, March 2010.

[LSV04] Antonios Litke, Dimitrios Skoutas, and Theodora A. Varvarigou. Mobile Grid Computing: Changes and Challenges of Resource Management in a Mobile Grid Environment. *Access to Knowledge through Grid in a Mobile World Workshop, PAKM 2004 Conference, Vienna.*, 2004.

[MHB04] Lee W. McKnight, James Howison, and Scott Bradner. Guest editors' introduction: Wireless grids–distributed resource sharing by mobile, nomadic, and fixed devices. *IEEE Internet Computing*, 8(4):24–31, 2004.

[Mic09a] Microsoft. Common Language Runtime Overview. `http://msdn.microsoft.com`, 2009. Available online at `http://msdn.microsoft.com/en-us/library/ddk909ch.aspx`; visited on March 18th 2010.

[Mic09b] Microsoft. Differences Between the .Net Comapct Framework and the .Net Framework. `http://msdn.microsoft.com`, 2009. Available online at `http://msdn.microsoft.com/en-gb/library/2weec7k5.aspx`; visited on March 20th 2010.

[MMCA02] Muthucumaru Maheswaran, Balasubramaneyam Maniymaran, Paul Card, and Farag Azzedin. Invisible network: Concepts and architecture. *International Workshop on Invisible Computing*, 2002.

[Moa97] Ryan Moats. *URN Syntax*. IETF, May 1997. Available online at `http://www.ietf.org/rfc/rfc2141.txt`; visited on Mach 8th 2010.

[Ri09] RF-iT. You-R® Open Folder. Website, 2009. Available online at `http://www.rf-it-solutions.com/uploads/tx_easydownloadlist/yro-A4.pdf`; visited on February 23th 2010.

[SFF04] Matthew Smith, Thomas Friese, and Bernd Freisleben. Towards a service-oriented ad hoc grid. In *ISPDC '04: Proceedings of the Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, pages 201–208, Washington, DC, USA, 2004. IEEE Computer Society.

[SMK$^+$05] Gernot Schmoelzer, Stefan Mitterdorfer, Christian Kreiner, Joerg Faschingbauer, Zsolt Kovacs, Egon Teiniker, and Reinhold Weiss. The entity container - an object-oriented and model-driven persistency cache. In *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, Washington, DC, USA, 2005. IEEE Computer Society.

[SV06] Thomas Stahl and Markus Völter. *Model-Driven Software Development; Technology, Engineering, Management.* John Wiley & Sons, Ltd, 2006.

[TKS09] Michael Thonhauser, Christian Kreiner, and Martin Schmid. Interpreting model-based components for information systems. In *ECBS*, pages 254–261. IEEE Computer Society, 2009.

[TM07] Donald Thompson and Colin Miller. Introducing the .NET Micro Framework, Technology White Paper. `http://msdn.microsoft.com`, September 2007. Available online at `http://msdn.microsoft.com/en-us/windowsembedded/bb267253.aspx`; visited on March 20th 2010.

[TS07] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems, Principles and Paradigms.* Pearson, 2007.

[Wha10] Whatis.com. What is Cloud Computing? Website, 2010. Available online at `http://searchcloudcomputing.techtarget.com/sDefinition/0,,sid201_gci1287881,00.html` visited on February 25th 2010.