

Ubiquitous Smart Space Management

Sven van der Meer, Robert O'Connor, Alan Davy

Telecommunications Software and Systems Group (TSSG)

Waterford Institute of Technology (WIT)

vdmeer@tssg.org, roconnor@tssg.org, adavy@tssg.org

Abstract

This paper introduces the M-Zones approach for the integrated management of distributed networks, services and applications within Smart Spaces. The main objective of this approach is the realisation of software, systems and services that address composition, scalability, reliability and robustness as well as autonomous self-adaptation. It focuses on middleware for management, control and use of fully distributed resources. The term integration refers to the task of unifying existing instrumentations for middleware and management, not their replacement.

The rationale of this work stems from the fact, that the current situation of Smart Spaces and management systems can be described with the term interworking. The actually needed integration of management and middleware concepts is still an open issue. However, identified trends in communications and computing demand for integrated concepts rather than the definition of new interworking scenarios.

The approach is discussed in three steps. First, a general framework is defined based on the hypothesis and an evaluation of the target environments. The second step is the derivation of a conception for ubiquitous management from this general framework. The last step is the realisation of this architecture and its exploitation.

1. The Situation

Smart Space applications need to be prepared to be used and operated in a stable, secure and efficient way. Middleware and service platforms are employed to solve this task. To guarantee this objective for a long-time operation, a Smart Space needs to be controlled, administered and maintained in its entirety, supporting the general aim of a managed zone and for each individual component, to ensure that each part of the Smart Space functions perfectly. Management systems are responsible for this objective. Usage and control

result in mechanisms for mapping information across application, service and network level. Control, administration and maintenance reflect management tasks on each of those levels. Integration of Smart Spaces and management results in a system that provides distributed applications with all of the introduced functionality.

Following the hypothesis of this paper, the target environments are evaluated to extract requirements for the integrated approach. Based on this evaluation, a general framework is developed that clearly identifies the certain levels of integration, their boundaries and their individual objectives. Goal of the framework is to establish a software layer between the applications and distributed technologies that provides integrated management services without losing the advantages of middleware.

2. UOCAM for Smart Spaces

Each smart space is designed to accomplish a purpose. The smart space and its purpose can be viewed from different perspectives, whereas each perspective creates its own requirements. However, all perspectives belong to the same basic understanding: A smart space needs to be prepared to be *used* and *operated* in a stable, secure and efficient way. To guarantee this objective for a long time operation, the smart space needs to be *controlled*, *administered* and *maintained* in its entirety, supporting the general aim of the system and for each single component, to ensure that each ring of the chain functions perfect.

The terms *use* and *operation* describe the part of a system that is seen by users and customers. They are concerned about the system's ability to serve them. A company running a system relies on its efficient *operation* in order to generate revenue. This operation is supported by *controlling* the system. The term *control* describes the brief but permanently reoccurring task of keeping the system stable to serve its customers and to generate revenue. This includes e.g. the configuration of

system components and the record of data for accounting.

Administration and maintenance reflect long term operation and control of a system. This general task is divided into several individual procedures. Administration starts with the permanent monitoring of the system and the logging of all occurring events to analyse the behaviour of its components. The second aim of monitoring is the detection of system failures.

Two different types of failures can be identified. Technological failures are a result of hardware problems in computers and components and communication errors (e.g. broken link or overload). Management systems are already able to detect those failures and to follow (predefined) procedures to re-establish the communication infrastructure. The other type of failures relates to content and semantic issues. The forecast of those failures is still a difficult task. These failures occur when objects are initialised with wrong data, when data is changed during transmission, because of mistakes from programmers, or problems in the design of an application. They are especially difficult to handle when the distributed system is of intrinsically complex nature.

2.1. Areas of Concern and Activities

Three areas of concern can be identified. Applications provide the interface to users and customers enabling them to utilise services. Services are software assemblies of components that offer functionality for applications and provide the access to resources. Resources are software and hardware components needed for the provision of services and for their execution. In other words, a distributed system employs resources to complete services and applications.

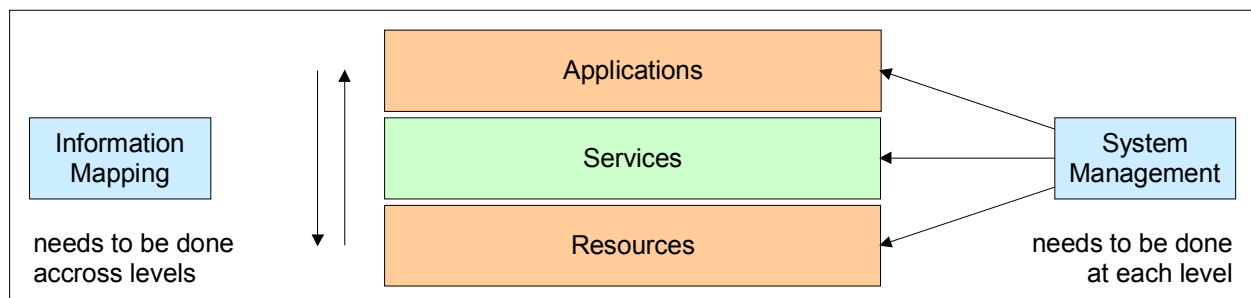
The figure furthermore shows the two main activities inside such a system. Information is mapped across levels, upwards and/or downwards, for usage, operation and control. The system is managed at each level through control, administration and maintenance. The assignment of individual tasks to one activity depends on the system's purpose. This is also true for the separation of the activities. The mapping of

information is supported by management activities and the system's management relies on information mapping.

Both activities have commonalities. The mapping of information as well as the management of the system can be described in terms of presentation, specification, submission, re-specification, triggering, queuing, access and execution. This layered approach is used to model modular client/server applications. It is built upon small and functionally specialised components that can be reused across multiple systems. Each layer of this model provides a specific function in the overall scope of the system.

The first layer focuses on the presentation of information along with the verification of results to support the specification and the submission of individual tasks. The specification answers six questions about a task: Who? (identifier) wants What? (request) Where? (destination) When? (schedule) Why? (purpose) and How? (execution plan). The answers to Who?, What?, When? and Why? are the basis for a submission that is a complete job specification. The re-specification is responsible for the mapping of What? towards a set of commands that is needed to be executed to fulfill the purpose. Triggering activates and deactivates jobs based on date and time information, completion of other jobs, or other available data. Queuing provides load balancing and the prioritising of jobs. Access functions as a mediator between the above layers and the execution layer. It provides interfaces to resources. Execution executes any job that is submitted from submission via access. The type of service executed here depends on the overall purpose of the system and might range from database access, media conversion, user interaction, up to device and network usage. All described layers are supported by navigation, security, metering and logging.

These commonalities between information mapping and system management should lead to a similar design of software that handles them. However, the reality is different. Management activities are outsourced to specialised systems that act independent of the systems they manage. The reason for that is the evolution of network systems.



Starting with basic services (telephony and data transmission), the first two types of networks that existed had had no need for automated management. The networks had been operated manually. Changes in society and on the market have lead to the extinction of this business model. Networks have got connected, the market has demanded more than the basic services and the number of devices has increased. This resulted in the actual need for an automated management of services and resources to continue to run the network in an efficient (and profitable) way. The operation of services and their management became separated areas with different approaches. Today, distributed systems and services run on top of middleware and are managed by dedicated management systems.

3. Conceptual Model

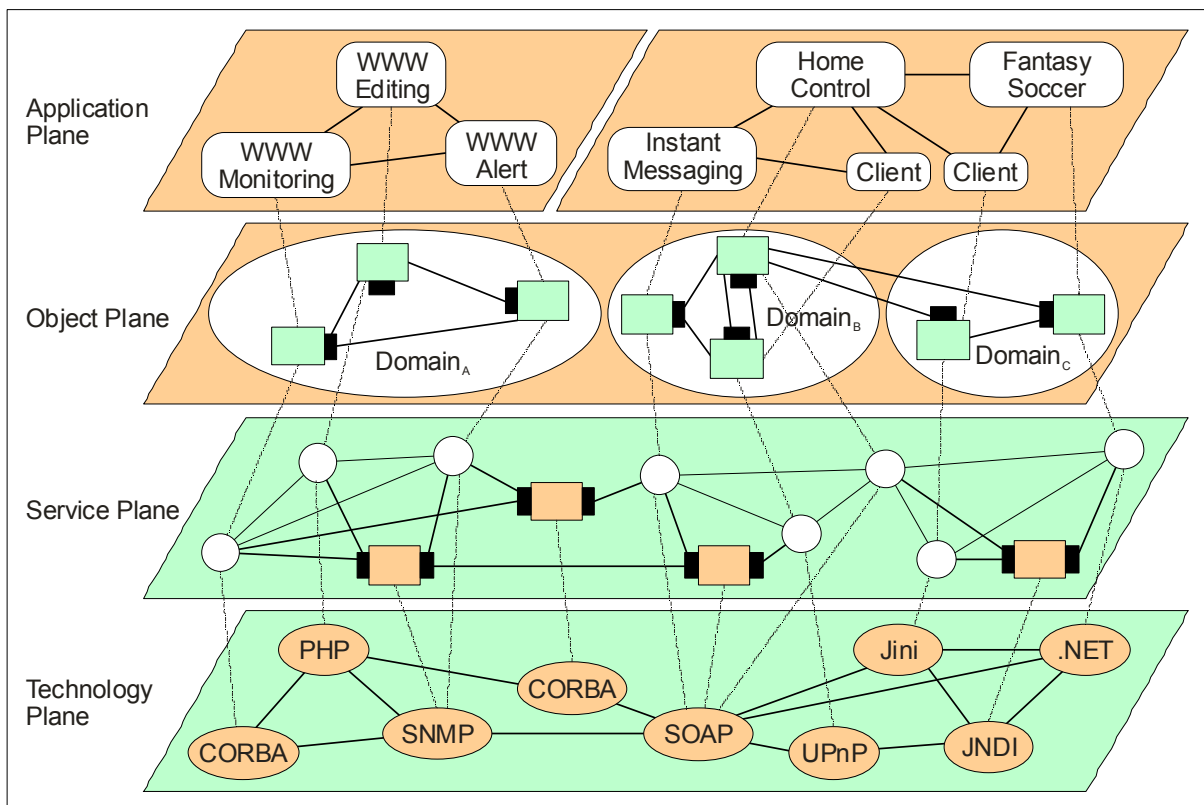
The conceptual model identifies four planes. Each plane is dedicated to a specific problem context. Each problem context describes a dedicated viewpoint to the two areas of concern (information mapping and system management) and the five terms (use, operation, control, administration and maintenance). The planes are used to specify the different types of information that need to be mapped and the different levels of management that is needed. The approach of a conceptual model is taken from the IN standard series.

The conceptual model provides the basis for a specific architecture. It presents rules for components of a specific architecture and describes relationships between those components. The definitions of each plane can be employed to describe particular aspects of the architecture. The four planes are:

- Application Plane, covering functionality within functional blocks;
- Object Plane, modelling functional blocks as computational objects;
- Service Plane, modelling core and application services as computational objects; and
- Technology Plane, identifying the employed technologies and their relationships.

On the first view, the conceptual model separates the applications from technologies. Applications become technology independent. This means, an application is not using a particular middleware or management technology. Those technologies become transparent for the application. The conceptual model allows substituting middleware and management technologies without changing the applications.

However, the conceptual model must not be seen as a dogma. Application might need direct access to technologies. This is especially important for the management of resources. This direct access belongs to the business model and the design of the application. An architecture derived from the conceptual model should not deny such a direct



access.

3.1 Application Plane

The Application Plane focuses on the design and the implementation of applications. An application is an implementation of a set of functionalities that might be distributed over multiple hosts. This set of functionality does not include the support for distribution. Following TINA-ODL, a group of object consists of object specifications that are called components. An application can be seen as a group of objects. An application component is than a single object specification within a group.

The design of an application can be derived from a business model. Appropriate models and tools can be used for design and implementation. The final implementation can profit from a concrete architecture. The actual purpose of future applications can neither be predicted nor foreseen. The conceptual model gives no further recommendations for this plane.

3.2 Object Plane

The Object Plane concentrates on the modelling of (distributed) objects. An object is a part of an application that models a real world entity. It is implemented in a computational, identifiable entity that encapsulates state and operations. Attributes are sets of data with a fixed semantic.

The interfaces of objects are specified in a certain language. Many middleware and management architectures employ a specific interface language. The languages are combined with tools for automated processing such as compilers and interpreters. The selection of an appropriate interface language depends on the objectives of the specific architecture. Many languages only include the signature of an interface. This signature might not be enough. Languages from the management area often add semantic information to individual parts of a signature. This information can be used to qualify an interface or parts of it. Furthermore, this information can provide the basis for repositories that enable lookup, monitoring and configuration of objects.

A specific architecture can support the application design with a set of basic specifications. This can be done e.g. in form of a core model (like the DMTF core model) or a structure of information (like the Structure of Management Information – SMI).

Beside objects, the concept of domains needs to be introduced. A domain can be used to separate (or to delegate) responsibilities. Domains can be implemented e.g. for the collection of objects that deal with different connectivity services (signalling, packet switching), objects that belong to functional area, or objects that relate to layers of the ODP

Reference Model. A domain can also be used to reflect geographical distribution or hierarchies of an organisation. The reflection of business models is an important factor for modelling domains.

A protocol is needed to define how interactions between objects can take place. The characteristics of the protocol depend on the interface language. The specification of the protocol should enable an easy mapping to technologies of the Technology Plane. The protocol must enable the identification of called objects. Furthermore, it should provide a reliable and encrypted transmission. An Application Programming Interface (API) can be used to realise protocol mechanisms and to provide access to protocol features.

3.3 Service Plane

The Service Plane models a collection of interfaces and objects (services) that provide basic functions for (application) objects. A service is independent from the application domain and from objects of the Object Plane. One of the most important services is a naming service, which enables the identification of (available) objects. The exchange of asynchronous events can be realised with an event service. Monitoring of objects and long-time configuration management relies on logged events. Ad-hoc networking and P2P communication need lookup services to search for objects that offer a specific functionality.

The basic services that must be provided by a specific architecture are a naming service and an event service. Furthermore, the architecture should offer services that combine information about object instances (naming) with information about object classes (specification). This combination allows the assembly of repositories that ease the administration and maintenance of a system. Additionally, the architecture should offer functionality for the visualisation of instances and classes in order to support the manual management of a system.

3.4 Technology Plane

The technology plane is introduced to model middleware and management technologies. This plane functions as a mediator to the actually employed middleware with specific interface languages, communication protocols and services. Management systems can be integrated into the architecture and existing specifications can be reused if available. The technology comprises middleware and management architectures (such as CORBA, Java, Simple Network Management Protocol – SNMP, Telecommunication Management Network – TMN) as well as concrete products.

The conceptual model gives an example that includes eight different technologies. Beside two major middleware platforms (CORBA and Java), the figure shows a Java based directory platform (JNDI) and a Java based ad-hoc networking platform (Jini). PHP, XML/HTTP, UPnP and SNMP are included as examples for middleware that is based on standards from the W3C and the Internet Engineering Task Force (IETF).

The connections between the technologies should indicate that there must exist an interworking between these very technologies. This interworking is needed to enable a communication between the objects and services. To give an example: The application WWW Monitoring is realised in CORBA and the application WWW Alert is based on an SNMP Manager (dashed lines). Both objects should communicate with each other. This communication can be realised with a CORBA/SNMP bridge (horizontal integration within the Technology Plane). Another possible realisation is that the application WWW Alert would also be a CORBA object that communicates with the SNMP Manager and functions as a gateway by itself (vertical integration realised in the Object Plane and the Technology Plane).

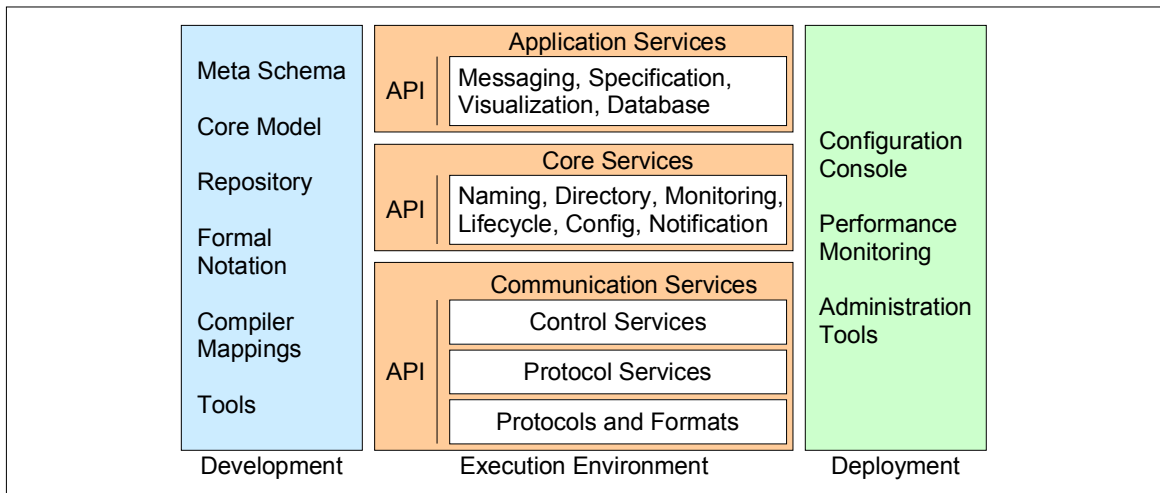
4. Components

The conceptual model defines the basic concepts and the conceptual model specifies the basic rules for a concrete architecture. The architecture developed within this work is the M-Zones Management Architecture (MMA). This architecture concentrates on the two middle planes of the conceptual model. The rules of the Object Plane are used to define the basic components of MMA. The rules of the Service Plane already identify services that need to be realised. The architecture is not going to define methods for business models. Those models are related to the

business logic of application, which should be supported but cannot be defined by the architecture. The Technology Plane depicts important realisations and products. The architecture must recognise those technologies and should reuse existing approaches for interworking between different technologies.

The components of MMA belong to the three parts of development and operation of a distributed system. They are development, execution environment, and deployment. The development step follows a business model. MMA should support the development with an appropriate formal notation for the specification of objects. This specification is based on a meta schema (or object model), which needs to be designed in a way that reflects the needs of the target environments. A core model can be specified using the formal notation. Task of the core model is to provide basic definitions for a distributed system, such as often used data types and generic objects. The formal notation defines syntax and semantic of a language that is derived from an object model. Here, the mapping of specifications to concrete middleware technology, programming languages and compilers must be specified by MMA. This mapping can be supported by tools for an automated processing of specifications.

The second step is the execution of the distributed system. This is supported by MMA in form of an execution environment, which provides an interface to the Technology Plane. The execution environment comprises three layers. Each layer provides a unified access to its functionality via one or more APIs. The lowest layer realises the communication between objects and between objects and services. For this communication, MMA needs to define a protocol along with data formats, and control services. The protocol defines the communication behaviour, including the transmission of data. The data formats are used by



the protocol for the transmission of information specified in the formal notation. The control services can be used to include additional functionality for addressing, security, and transactions. The protocol must be specified in a technology independent way. This should allow the usage of many different technologies for the actual exchange of data.

The services of the Service Plane are further separated in two groups. The first group collects core services, which must be present in the execution environment for usage and operation of a distributed system. The second group depicts services that might be present. This second group of services should improve the architecture's ability to support control, administration, and maintenance.

The final step is the deployment of the objects and the distributed system itself. Here, a number of tools should be provided for the configuration of the system. These tools can be offered in form of an administration application. This application should be able to visualise information about the actual state of objects, including instantiated objects, request counts, runtime behavior, monitoring, and log information. The tools should be based on core and/or application services. Following this approach, an administration tool by itself is a distributed application that can be modeled using the conceptual model and the mechanisms of the architecture.

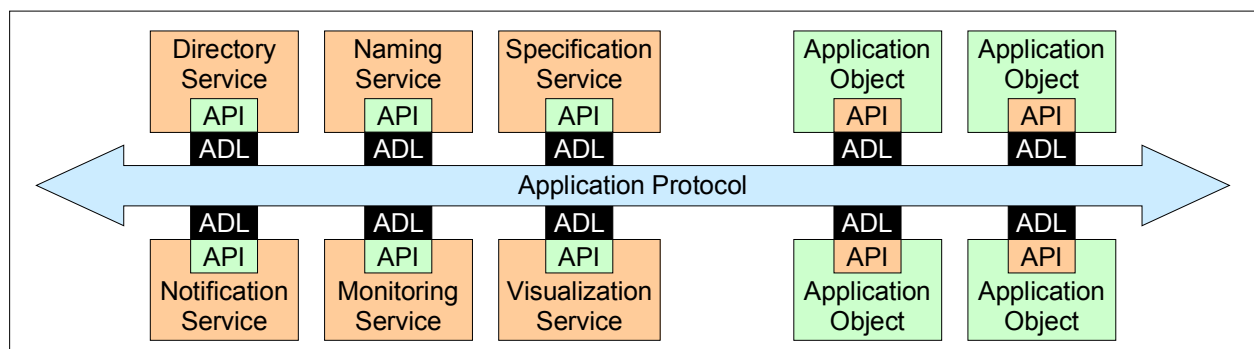
5. Architecture for Ubiquitous Management

The M-Zones Management Architecture (MMA) concentrates on the two middle planes of the conceptual model. The Object Plane of the conceptual model provides concepts and mechanisms for this design process. The Service Plane adds the necessary core services and enhanced services for integrated management. Both planes offer technological transparency to the applications.

The applications are not bound directly to middleware and management technologies. Six

recommendations form the basis to realise this transparency:

1. The basis of MMA is an abstract object model. A concrete object model, called **Meta Schema**, is derived from the abstract object model. The Meta Schema defines the basic specification elements of a MMA application.
2. The **Application Definition Language (ADL)** is a formal notation that is used to express the Meta Schema. This language combines characteristics from middleware interface definition languages, languages for the definition of managed objects, and languages used to specify data that is exchanged between applications.
3. Application objects are specified in ADL. MMA defines a **Schema** based on ADL and a **Core Model** that is relevant for all applications and independent of domain specific specifications. The basic part of the Core Model is the identification of a reasonable set of qualifiers providing meta information about ADL typed application objects.
4. An **Application Protocol** is responsible for the transport of information among application objects including features that enable the construction of management hierarchies like addressing of hierarchies, scoping and filtering, and transactions.
5. An **Application Programming Interface (API)** decouples application objects from middleware technology and enables the seamless integration of management functionality into applications. The API offers a small and simple to use set of operations for the parameterisation of the protocol. Tools are responsible for the automatic generation of parts of the API code that is needed for the exchange of information between applications.
6. **Application services** realise the naming of objects; enable the mapping of ADL specified information to directories, and the usage of type and data repositories for applications and MMA components. All application services can be accessed in a unified way similar to the access to other applications. The API



implements standard jobs like lookup for available services and registration.

The Meta Schema, the ADL and the Core Model take advantage by using existing mechanisms and techniques and, simultaneously, enhancing them to meet the requirements and objectives of the general framework. They are neither specific to middleware nor management. The Application Protocol and the API provide a generic interface to the application. Internally, they map this generic interface to middleware specific interfaces that need to be adapted to any employed concrete middleware technology.

The protocol and the internal realisation of the API are transparent to the applications. Any application developed with MMA is ready to run on any middleware that MMA supports. The support of a new kind of middleware has some requirements. First, the MMA API must support the new kind of middleware. This is realised with a specific implementation of the API that recognises the available facilities of the middleware. Second, the new kind of middleware must be integrated in the existing middleware technologies. The integration reflects the Technology Plane of the Conceptual Model. MMA applications that are designed to communicate with each other must be supported by gateways between the middleware technologies. When these two requirements are fulfilled, the new kind of middleware is available for all MMA applications. It is also possible to substitute the actually employed type of middleware without changing the applications.

5.1 Object Model

The MMA object model builds the basis for the design of a MMA application. It combines characteristics from middleware object models and from object models from management architectures. The object model has the following objectives:

- Define an abstract object model.
- Derive a concrete object model.
- Provide the basic functionality for a formal notation.

The concrete object model identifies the basic characteristics of a MMA application. An application is a piece of software that solves a specific task. An application consists of one or more components, namely application objects. These objects are modelled following the abstract object model.

The concrete object model represents a formal description of the abstract object model. It follows the facilities of the object model of CORBA and the Meta Schema of CIM. For MMA, the concrete object model is expressed in form of a meta schema

in UML. The UML specification provides the basis for a formal notation, which is used to specify application objects.

5.2 Application Definition Language

The Application Definition Language follows the specifications of the MMA meta schema. Its ability to function as a language for data exchange is based on a specified mapping of ADL to the eXtensible Markup Language (XML). This mapping can be done in both ways. An XML document valid against the ADL Document Type Definition (DTD) can be transformed to an ADL specification and vice versa. ADL offers a solution for common tasks for the specification of a manageable distributed system:

- the definition of distributed applications and objects;
- the definition of interfaces of objects that client objects call and object implementations provide;
- the generation of object- and interface repositories;
- the definition of communication protocols for objects and applications on top of middleware;
- the generation of data exchanged by objects;
- the definition of events sent by objects;
- the definition of derived types;
- the definition of managed objects that describe managed resources;
- the definition of Management Information Bases; and
- the definition of information beyond the scope of this document.

ADL is independent of actual business models, communication protocols, programming languages, and languages that are used to describe information exchanged among applications. Protocol mappings, programming language mappings, and data exchange language mappings can be specified for specific environments in an individual way. Mappings from ADL to programming languages or markup languages depend on the facilities of the target languages. This document defines language mappings for IDL as defined by the OMG, specifies an XML notation for ADL, and provides instructions for the development of other mappings.

5.3 Core Model

The MMA Core Model is an information model that captures issues that are applicable to all MMA applications. It is a small set of type definitions, classes, and other specifications that, in combination, provide the basic vocabulary for analysing and describing MMA applications and services. The Core Model is a specialisation of the ADL schema. While the Core Model might be

enhanced, major changes to the specifications presented in this section are not anticipated. The Core Model follows four objectives:

1. to identify a reasonable set of qualifiers to extend the specification of ADL elements with meta information for repositories;
2. to specify basic type definitions for the MMA protocol, the MMA API, MMA services, and applications;
3. to declare abstract base classes that can be inherited from by any MMA application object; and
4. to define basic rules that apply to all MMA specifications, such as the specification of events and exceptions, recommendations for naming conventions, and possible enhancements of specifications for the Core Model itself.

Furthermore, the Core Model identifies basic types that are useful for MMA applications. Those types comprise a unified specification for time and data information, tickets, and exceptions. Additionally, an abstract core object is included. This object can be used as base class for MMA applications. The type definitions are concluded by a number of miscellaneous definitions.

The Core Model acknowledges definitions for management. For this reason, some general specifications for management of installation and configuration of MMA applications are included (Entity Management). Additionally, generic objects for the management roles Manager, Agent, and Managed Object are made to simplify the specification of management hierarchies.

5.4 Application Protocol

The Application Protocol defines the mechanisms of data exchange between applications and among applications and services. The protocol has the following objectives:

5. Provide a generic mechanism for the communication between distributed objects. This mechanism needs to be independent of concrete middleware technology but should be mapped to employed middleware easily.
6. Realise the communication between MMA objects supported by the API. The protocol defines communication behaviour and the API implements it. The communication between MMA objects is done via the exchange of either ADL typed or xADL typed parameters.
7. Support of management hierarchies. Management systems assume the existence of a management hierarchy that aligns managers, agents, and managed objects in form of a management tree. Since distributed objects are only loosely coupled, this special requirement

needs to be supported by the protocol with mechanisms for hierarchical addressing of objects, scoping and filtering, and the realisation of transactional operations.

The communication between application objects can be divided into three different layers. The first layer describes the relationships between the core objects of applications. These objects exchange information by means of messages or operation calls to realise the aims of the application. The second layer models the transmission of messages and/or operation calls between MMA API objects. The MMA protocol specifications reside in this layer. The third layer is dedicated to the employed middleware and the middleware specific protocol. Further layers are not needed since the middleware protocol already abstracts from underlying network transport protocols.

5.5 Application Programming Interface

The MMA API is an interface for application core objects to communicate with MMA services and with other application core objects. The API has three different assignments:

8. Decouple the core object from any concrete middleware.
9. Provide a simple, unified interface to the functionality offered by MMA.
10. Introduce basic management functionality transparent to the core object.

The API is placed between the application core object and the MMA interfaces. The first objective is achieved already by this engineering design. The API mediates between the core object and the middleware specific interface objects and therefore decouples the core object from the middleware. This objective also demands that the parts of the API that are visible to the core object include no middleware specific operations or parameters, at least not for the communication between objects. The API is responsible for the mapping of ADL typed interfaces to middleware specific interfaces described by the protocol. Furthermore, the API is in charge to realise middleware specific operations (e.g. for initialisation) and standard duties (e.g. registration at naming server and event service).

5.6 Application Services

The application services represent definitions and specifications to realise requirements of the general framework and rules of the Service Plane of the Conceptual Model. The requirements depicted by the application services are: naming and addressing and registration of objects, discovery and lookup services, message services, and visualisation. The rules already identify the four services that required for MMA by means of naming, event, repository, and visualisation

services. The following subsections specify the application services that are needed to provide a MMA execution environment.

6. Conclusion

Embedded into the concepts of M-Zones and smart space management, this paper has and related developments have presented a way for the integration of management and smart space concepts. Starting with the identification of major activities, followed by the definition of a general framework and the derivation of a specific architecture, a prototype implementation has been developed. It shows the integration of concepts from both areas middleware and management.

The challenging aspect of this work has been on the one hand to the diversity of available concepts and on the other hand the fact that emerging concepts and technologies are going to change all facets of software development.

The MMA is the specific architecture that is directly derived from the general framework. Six recommendations form the architecture. Furthermore, the MMA describes a method for the realisation of distributed applications employing all six recommendations. Each recommendation starts with specific objectives and requirements. This enables the substitution of the concrete technologies that have been developed with other, more appropriate or environment specific solutions without changing the architecture itself and without a negative impact to the other recommendations and technologies.

The specific architecture was implemented following the objectives of the framework to show that the provided recommendations are a solution that is lightweight, open, smart, and service generic.

- Lightweight as the solution can be widely adopted by vendors and providers of different size and market penetration. This means the solution take into account commonly accepted principles already adopted by service providers and network operators.
- Open as the solution includes well-defined interfaces. Interoperability with legacy systems is a key issue for a smooth integration. This thesis reflects the term open also to indicate that the market demands for an easy adaptation of new technology and interworking with other systems.
- Smart as the solutions must reflect the intrinsically dynamic aspects (particularly for the subscription, deployment, and session set up process of applications, services, and resources), enabling a flexible adaptation to customer

requirements and operator/provider needs. This will be supported through the use of meta-data repositories throughout the whole system lifetime to provide a comprehensive knowledge base improving multi-domain service provisioning and also in the operations/maintenance phase.

- Service generic as the solution is independent of the actual services that are offered. Nobody can predict if there is a killer-application for future services and which one this might be. The preferred applications, services and resources will surely differ significantly within different countries and different cultural groups.

This paper describes a new approach for the support of smart space management. However, some issues have been considered to be out of the scope for this approach. These issues are security, testing, and tool support. All of them are important for the specification and operation of distributed applications. They have been introduced when appropriate to show the places where further investigation on these issues is necessary. Some aspects of security have been recognised in two recommendations of the architecture (API and Services). In both recommendations, a basic set of security options has been included. Testing is supported for the formal processing of ADL specifications with the implemented parser. Further formal test methods have not been developed. The support of tools for specification, development, and deployment is limited to the ADL parser and the visualisation service.

The basic result of this work is the conclusion that an approach that integrates basic concepts of middleware and management provides benefits for the management of smart spaces. The unification of use, operation, and control with the tasks of maintenance and administration minimises the effort that has to be spent for the mid- and long term operation of a distributed application. The independence of concrete middleware and management technologies improves the portability of applications. An application designer and programmer can concentrate on its actual task – the realisation of profitable applications instead of dealing with constantly changing technological issues. The simplicity of the six recommendations of the architecture allowed for a simple and lightweight implementation that can be employed in many different environments, starting from small devices up to complex and huge service platforms.