Table of contents

MESH Release 2 implementation at CTIT

Nikolay Diakov, Marten van Sinderen, George Koprinkov

Centre for Telematics and Information Technology
PO Box 217, Enschede, The Netherlands
{diakov, sinderen, koprinko}@ctit.utwente.nl

Abstract

This document contains a description of the development done at CTIT on the MESH services platform, a TINA-based platform for the deployment and exploitation of services to support teamwork. It provides an overview of the results and the usability of architectural solutions and technologies used during this development.

1. Introduction

Distributed software applications are continuously evolving, both because of increasing user demands and because of growing capabilities of enabling technologies. Although this will lead to new and better services offered to the end-users, there are also the issues of separate technology platforms (islands), growing complexity of technology infrastructures, and costs of upgrading to the new and better services. What is needed to resolve these issues is an open services platform that:

- provides interfaces for accessing, using and managing the different services, hiding as much of the underlying complexity as possible;
- supports service development and deployment, in the sense that it facilitates customization, composition and extension of service components and re-use of software;
- is based on a general purpose 'distributed processing environment', which is independent of specific computing or network technology platforms.

In addition, the architecture and mechanisms of such a platform should be able to cope with the direct user requirements on scalability, mobility, performance, security and privacy.

One possible candidate for such a platform is a TINA-based platform. TINA (Telecommunications Information Networking Architecture) is an open architecture supported by many of the world's leading network operators, telecom equipment manufacturers and computer equipment manufacturers [TINA]. The TINA architecture consists of a set of principles, concepts, architectures and interface specifications for the development, deployment and operations of telecommunication systems. The architecture uses an object- and component-oriented approach, following the principles of the ODP-RM (Reference Model for Open Distributed Processing), and defines a distributed processing environment that can be (partially) substituted by other standards, such as CORBA (Common Object Request Broker Architecture) [CORBA].

Whether TINA actually meets the above requirements has to be demonstrated by implementations of TINA-compliant systems. This will also reveal the quality of the TINA specifications themselves (completeness, consistency, etc.) and the effort needed to derive conforming implementations.

During the MESH (Multimedia-services on the Electronic Super Highway) project, a TINA-based platform has been designed and implemented.

The MESH project was carried out by a Dutch consortium of industries, research institutes and end-users with financial support from the Dutch Ministry of Economic Affairs. The consortium partners were: Lucent Technologies, KPN Research, SURFnet BV, Telematica Instituut - Central Organization (TICO), Centre for Telematics and Information Technology, Academic Hospital of the University of Amsterdam, and

Roessingh Research and Development. The overall goal of the MESH project was to develop a platform for supporting teamwork in such a way that the natural, dynamic communication process between individuals remains intact or even improves [MESH]. In particular, high quality video and audio, white board sharing, document sharing and group editing, and multiparty session management had to be supported and demonstrated in business applications such as tele-consultation in the health care sector and tele-learning in the higher education sector. The potential of TINA was identified during the early phases of the MESH project, and in the subsequent phases of the project a MESH services platform was developed following the principles, concepts, architectures and interface specifications of TINA.

The results of the design phases of the MESH project have been described in, e.g. [SiFe98], [BaBa98].

This document describes the work that has been done in the implementation phases of the MESH project, in particular the work done at the Centre for Telematics and Information Technology (CTIT). It shows the relevance and practical applicability of the TINA architecture. The document tracks what implementation decisions have been made and what lessons have been learned about different implementation tools and development platforms. The document gives a detailed picture of the hands-on experience gathered during the implementation of the MESH services platform.

The remaining of this document is organized as follows:

- Chapter 2 describes the context of the MESH services platform implementation, the (learning) activities that had to be done before the actual implementation work could start, and the distribution of work over the different partners involved in the implementation phases.
- Chapter 3 describes in more detail the development activities carried out at CTIT. It presents the division of work in terms of tasks related to the different architectural components that had to be implemented, and discusses the history of the implementation process.
- Chapter 4 focuses on some design and implementation issues that turned out to have particular value to the project. The chapter discusses some of the major decisions about development tools and the problems that were encountered during the implementation.
- Chapter 5 presents the current status of the MESH project. The functionality of the implemented prototype of the MESH services platform is illustrated with some typical scenarios of use, involving the roles of end-user, service developer and service provider.
- Chapter 6 summarizes the conclusions of the MESH implementation effort.

2. Context and overall work breakdown

The MESH project started in December 1996. The first stage of MESH project comprised two parallel activities. One was the implementation of a platform, called MESH Release 1, that was based on the outcome from another related project, PLATINUM. Because of this, the MESH Release 1 platform could be readily available, but it was not very flexible and, since it used proprietary technology, was not able to interoperate with other platforms [SiCF96]. The other activity was the planning and design of a more flexible and open platform, called MESH Release 2. Already in the early beginning it was decided that MESH Release 2 should be based on TINA. The development of a MESH Release 2 prototype started in March 1998. During this stage of the MESH project we had to interpret the TINA service architecture and learn how it can be implemented, evaluate and select development platforms and tools, middleware technology, and develop software components that will represent the TINA components.

The following preparatory activities ware carried out in the beginning of the second stage of the MESH project:

- Learn about the TINA service architecture design patterns
  We spent a month investigating the TINA services architecture [TSA] and the TINA Service Component Specification [TSCS]. TINA assumes a Distributed Processing Environment (DPE) on top of which service components can be run. For the purpose of the project, this environment had to be extended with a development environment for configuring, composing and instantiating service

components. The former environment could be based on CORBA, the latter environment, referred to as the Distributed Component Software (DCS) framework [BaBa98], had to be developed by the project and required a substantial design effort.

- Selection of development platforms and tools (CORBA, Java, Symantec Café, etc.)
  We evaluated several options for realizing the TINA DCE, and finally selected CORBA middleware technology to provide the necessary distribution transparencies. We also aimed at platform independence and code mobility, so we chose Java as the programming language. The graphical design environment for Java was chosen to be Symantec Visual Café.

- Learn how to use the tools
  We spent some time to learn how to use the chosen technologies and tools. The learning curve was steep since the matter was not unusual and unknown to us.

The partners involved in the implementation phases were Lucent Technologies, Telematica Instituut - Central Organization (TICO), and CTIT. The work was distributed among these partners according to their manpower commitment and expertise:

- Lucent started development of the DCS framework and the implementation of the components related to access session and connectivity session of the TINA architecture.
- TICO was responsible for the development of the application specific components related to the service usage session of the TINA architecture, and several tools that enhance the work of the service developer that wants to extend the system with new services.
- CTIT had to develop the subscription component that manages all user and subscription related information, the administration service that assists the management of the user information model at the service provider, and the subscription service that allows clients to sign in to the system.

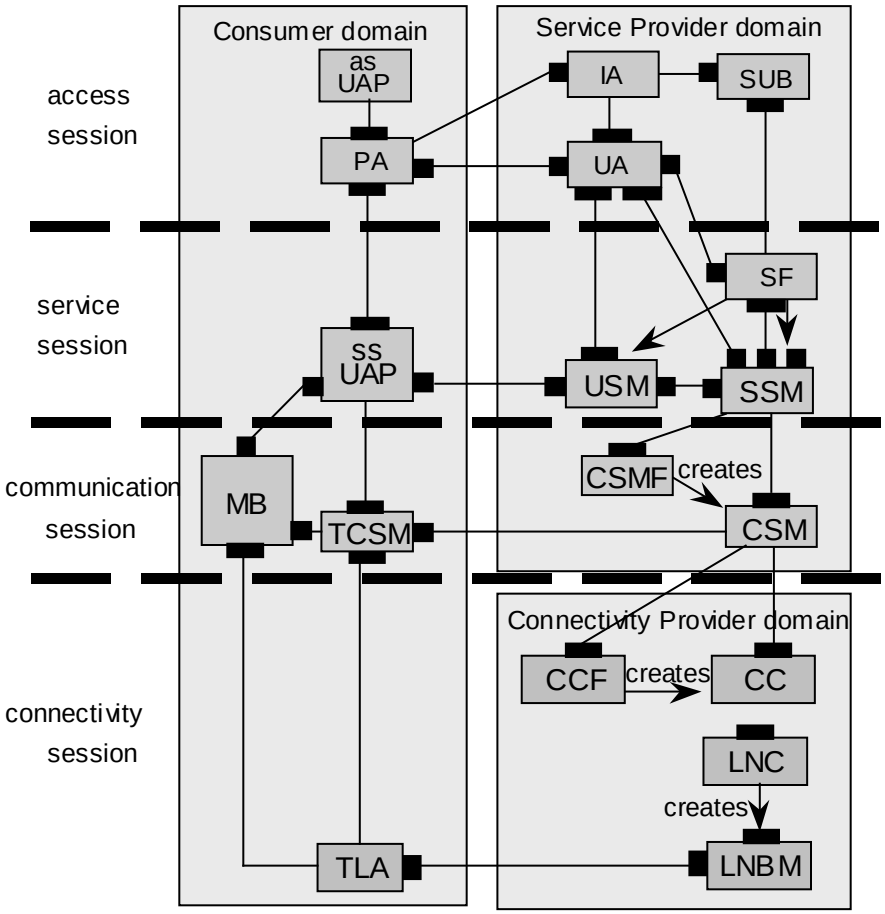

Figure1: General MESH Release 2 architecture, showing software components, sessions and domains.

3. CTIT implementation activities

Our major goal was to implement the subscription (SUB) component (Figure 2) that maintained the whole subscription model and to implement a subscription service that allows users to subscribe to the system and to manipulate their accounts. As additional functionality we wanted to implemented an administration service that allows power users and system administrators to manage the subscription model remotely and dynamically without the necessity to shutdown components in the provider. For example, the administration service allows new service extensions to be easily incorporated in the system.

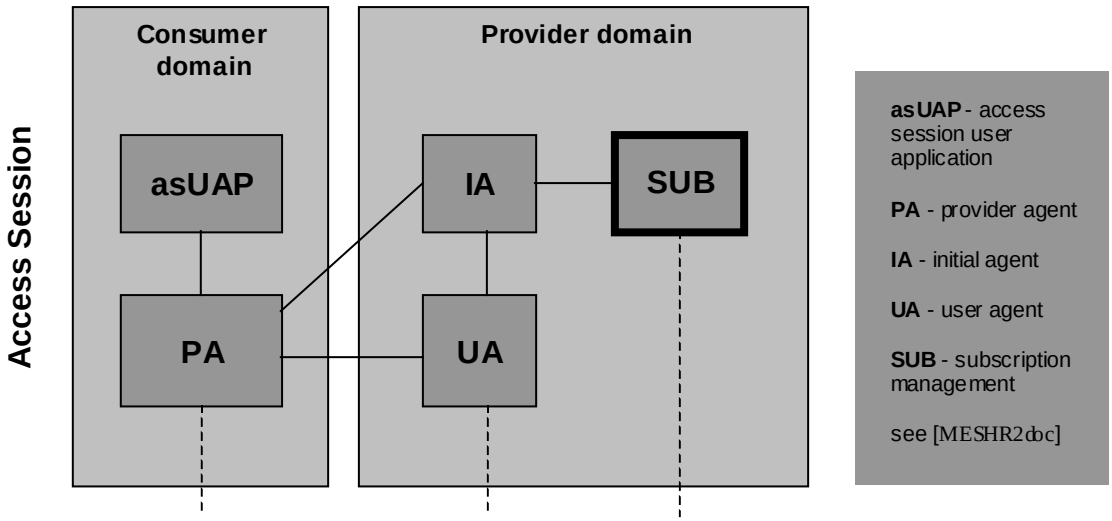

Figure 2: The components involved in the access session.

In order to achieve this goal we divided the work into a number of subtasks, which were carried out sequentially:

- Database tool development
- Database management component development
- Subscription core (SUB) component development
- Subscription service development
- Administration service development
- Testing and fixing bugs

These tasks are elaborated in the following subsections.

3.1. Database tool

This tool utility allows the administrator to easily translate the subscription information model into a database structure of some native database management system (DBMS). The description of the information model should be provided in a form of a text file with SQL statements. The tool is a Java application that executes the statements successively through a JDBC connection to the native DBMS (Figure 3).

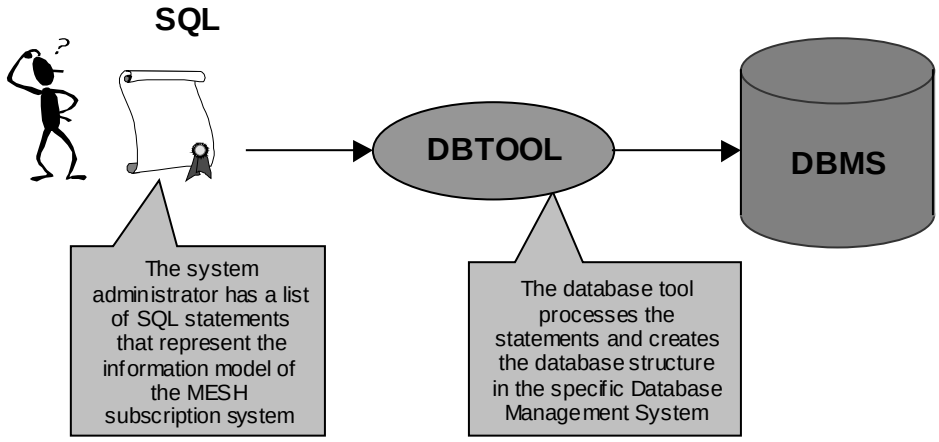

Figure 3: How the database tool can be used to achieve DBMS independence.

The process of creating this tool acquainted us with the JDBC technology as well as some of the advanced programming issues in Java. After completing this tool we continued with the database component.

For test purposes, a Microsoft access database was used together with an ODBC data source and the standard ODBC-JDBC bridge Java driver.

3.2. Database management component

The subscription component has been decomposed into two functionally independent components, a subscription core (SUB) component and a database management component. This has been done to ensure independence from a particular DBMS and furthermore allow distribution of the workload (e.g,. the database management component can be run on a dedicated machine).

The database management component offers to the other components of the system persistence of data while achieving independence from the underlying DBMS (Figure 4). The principal user of the database management component is the SUB component, which maintains a complex information model.

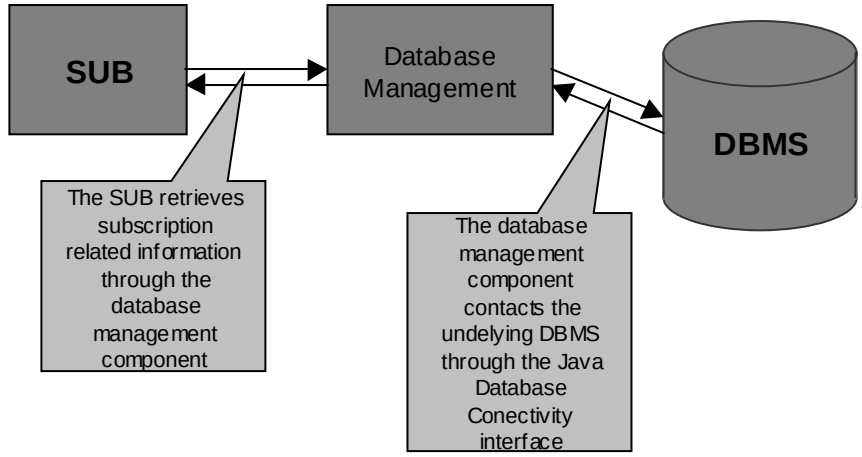

Figure 4: How the SUB and the Database component relate to each other.

The database management component is designed with the DCS framework and inherits all the common features of a generic component [BaBa98].

The database management component implements a number of interfaces:

- i_dbFactory: This interface returns references to the appropriate interfaces for the calling components. Currently, only the SUB component is provided with a specific interface. All others have to use the generic interface I_dboperations_Common.
- i_dboperations_Common: This interface is a generic interface. It allows general operations with the database through SQL statements. The operations from this interface do not have any knowledge about the information model that is represented in the data structures of the underlying DBMS.
- i_dboperations_SUB: This interface is specific to the SUB component. It implements some specific operations that have special knowledge about the subscription information model and according to this model performs unique id generation, multiple query execution, etc.

For the purpose of the MESH, a Microsoft access database was used together with an ODBC data source and the standard ODBC-JDBC bridge Java driver.

3.3. Subscription core component

The SUB component provides functionality to manage the subscription information model for the whole set of services in the service provider domain as defined in [TSA]. It is implemented with compliance to the suggested TINA model for a subscription management component (Figure 5).

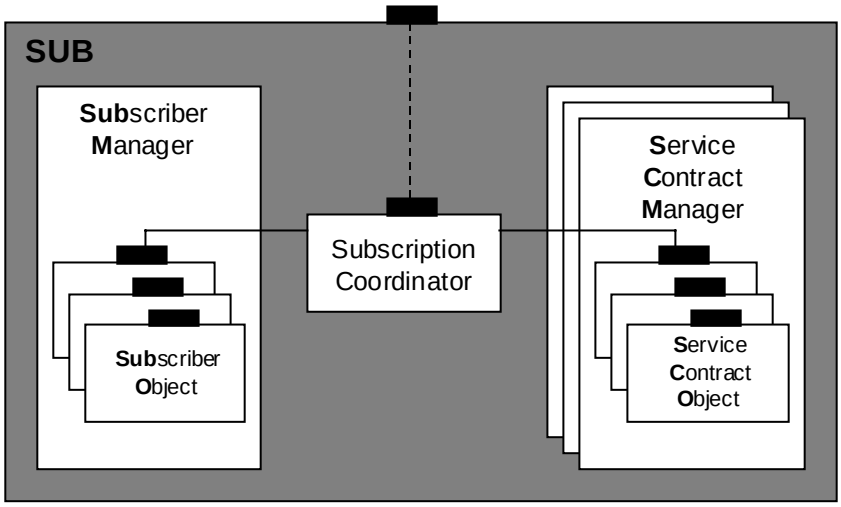

Figure 5: The internal structure of the Subscription component.

The Subscription Coordinator (SCoo) subcomponent is responsible for the management of the other subcomponents as well as being a main control point for the functionality of the whole SUB. It coordinates the subscriber management and the service contract management. The SCoo also implements interfaces that are exported/visible outside the SUB and through which clients of the SUB can initiate interaction with the SUB, i.e. create new subscribers, contract services to a subscriber, list services, etc. The SCoo uses the Subscriber Management (SubM) subcomponent for managing the subscribers and the Service Contract Manager (SCM) subcomponents to manage the service contracts.

The Subscriber Management subcomponent (SubM) is responsible for the management of a pool of Subscriber Objects (SubO) - one per subscriber - that implement interfaces for managing entities (users, terminals, network assignment points) and subscription assignment groups within a subscriber.

There is one Service Contract Management (SCM) subcomponent per service in the provider domain. An SCM is responsible for managing a pool of Service Contract Objects (SCO), one per subscriber, contracted to the particular service. Each SCO implements interfaces for manipulating service contracts and service profiles.

The SUB component maintains the information model of the subscribers, users, terminals, network assignment points, subscription assignment groups, services, service contracts, service profiles, etc. The data is stored in a DBMS through the database management component.

The SUB component is designed with the generic component framework and inherits all the common features of a generic component [BaBa98]. It implements a number of CORBA IDL interfaces which allows other components to query and manipulate the subscription information model (Table 1).

Type	Module	Name
External	TINAScsSubInitial	i_InitialAccess
-	-	i_Subscribe
-	-	i_ServiceNotify
-	TINAScsSubscriberInfoAccess	i_SubscriberInfoMgmt
-	-	i_SubscriberInfoQuery

-	TINAScsServiceContractInfoAccess	i_ServiceContractInfoMgmt
-	-	i_ServiceContractInfoQuery
Internal	TINAScsSubscriberMgmt	i_SubscriberLCMgmt
-	TINAScsServiceContractMgmt	i_ServiceContractLCMgmt

The following is a history of the implementation of the SUB component:

- Based on the TINA structures suggested in [TSCS] we defined the database tables. The relations and key indices were defined.
- Based on the suggestion for subcomponents decomposition from [TSCS] we defined the structure of the subcomponents and the components specification files for the GSC framework. The following subcomponents were defined: SubM, SubO, Scoo, SCM, SCO (described above).
- Definition and implementation of the interfaces for subcomponent management: i_Pool_xxx and i_KeyStruct_xxx, where the xxx is the abbreviation of the subcomponents. Each subcomponent that has pool of subcomponents implements the i_Pool_xxx, and each subcomponent that is contained in a pool by another subcomponent implements the i_KeyStruct_xxx interface.
- Implementation of the i_InitialAccess and i_Subscribe interfaces of the SCoo subcomponent. These interfaces are also exported as interfaces of the SUB component. i_InitialAccess subcomponent provides operations for establishing and terminating initial access to the SUB component. Under certain conditions it provides reference to i_Subscribe. i_Subscribe is the interface through which access to all other interfaces is granted. It also contains operations for creating a subscriber and contracting services. For more information, see [TSCS].
- Implementation of internal interfaces i_SubscriberLCMgmt and i_ServiceContractLCMgmt. These interfaces provide operations for managing subscriber and service contract related information respectively.
- Implementation of i_SubscriberInfoMgmt interface. It is the main interface for management of the subscriber information. It contains operations to list/manipulate subscriber information, create/delete/edit subscription assignment entities (SAEs), create/delete/edit subscription assignment groups (SAGs), and to list subscribed services.
- Implementation of i_ServiceContractInfoMgmt interface. Through this interface it is possible to manipulate the information about service contracts and to create/delete/adding/remove/activate/deactivate service profiles.
- Implementation of i_SubscriberInfoQuery and i_ServiceContractQuery interfaces, which allow querying the information about subscribers and service contracts respectively.

After the implementation of each interface, several days were spent on testing the implementation.

3.4. Subscription service

We developed the subscription service as a specific extension to the generic UAP component and the USM component from the TINA architecture. The implementation was made in close cooperation with the TICO, who was responsible for developing the components concerning the service session.

The subscription service has two operating modes depending on the user access (Figure 6). If the system has been accessed anonymously, the user is able to start the subscription service to create an account and to activate services for his account. When the user starts the subscription service with named access by authenticating with the system, he is able to alter his user record and change the set of active services.

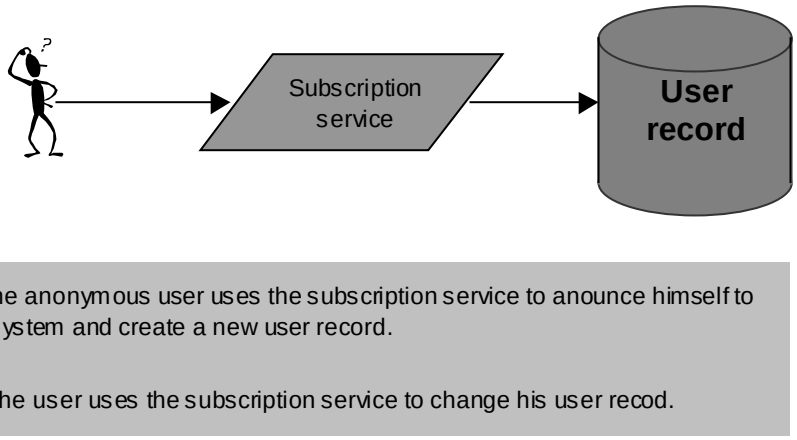

Figure 6: The subscription service can be used in two modes.

The following is a history of the implementation of the subscription service:

- Design of the application functionality.
- Design of the GUI.
- Implementation of the first prototype as a standalone application, to be able to test the functionality.
- Integration with the components developed by the TICO. We used the tools created by the TICO to integrate the service specific components we developed with the service generic components TICO created for the service session.

3.5. Administration service

This service provides a full interface to the operations of the SUB component. It is intended for usage by power users or system administrators to browse and manage the whole information model. It provides a tree-like graphical user interface of the hierarchical structure of the subscribers, users, etc. The administrator user can also alter the information model, e.g. create a service entry, contract a service to a subscriber and perform many other actions on the content of the information model.

The following is a history of the implementation of the administration service:

- Design of the application functionality.
- Design of the GUI.
- Implementation of the 1st prototype as a standalone application, to be able to test the functionality.
- Integration with the components developed by the TICO. We used the tools created by the TICO to integrate the service specific components we developed with the service generic components TICO created for the service session.

3.6. Testing, bug-fixes and small changes

The last phase is testing in order to see whether the functionality is well implemented and does not contain bugs. Also, some of the partners foresaw needs of small changes to the functionality of some of the operations. These changes concern mainly performance and effectiveness.

4. Design and implementation highlights

Some design and implementation issues turned out to have particular value to the project: the software architecture and tools, the component software approach, and the handling of problems that were encountered during the implementation. These issues are further discussed below.

4.1. Architectures, technologies, languages

The key decisions that have been made for MESH Release 2 are adapting the following standards:

- TINA
  The use of TINA as the guiding architecture for software design and implementation has been motivated in Section 1. The resulting software architecture for MESH Release 2 is described in [MESHR2arch].

- Java
  Java was chosen as an implementation language because it is platform independent and as an interpreted language has some features not available in the other imperative languages, like self-describing objects, code serialization and others. Java database connectivity was used for implementing the database management component as the platform independent information repository. There are many other technologies based on Java that we used in the project, including Java Beans component packaging.

- CORBA
  CORBA formed the base of the DPE environment that TINA components would use. Additionally, it contained extra features we wanted to take advantage of, such as implementation repository, security, and others.

4.2. Distributed software component framework

The Distributed Software Component (DSC) framework [BaBa98] supports the development of component software for distributed object oriented environments, and can be seen as a generalization and implementation of the TINA computational object model. One characteristic of the framework is that software components are self-contained packages of code that can be dynamically assembled into a program. External functionality of a component is manifested by way of operational interfaces. The DSC framework supports both dynamic and static operational interfaces. Also, there can exist multiple instances of the same operational interface. Furthermore, components can be grouped together to form compound components. A common control and configuration interface provides configurability of the components with regard to events, properties, operational interfaces, life cycle, and composition. The DSC framework has been applied during the implementation phases of the MESH Release 2 platform to develop the components of the TINA service and network architecture.

4.3. Problems encountered

The major problems that we encountered were the TINA inconsistencies in the documentation and the IDL definition files. Most of them resulted in slowing down the work process and made the reading and interpretation of the TINA documentation difficult.

4.3.1. IDL style inconsistency

During January 1998 we obtained the prescriptive IDL specifications for the descriptive components of [TSCS] and [TNCS]. We noticed three IDL coding styles: one for the service and access level interfaces, one for the communication level interfaces, and one for the network level interfaces. Each style differed in module policy, distribution of interfaces over modules, and in include file approach. As a result, the interfaces were hard to read. Not only were there cosmetic problems to overcome, also the IDL code per style featured different naming conventions. Bot integers (`unsigned long`), single strings, and sequences of strings (`t_TinaName`) were mixed. In order to stick to one coding style and naming convention we modified the IDL files appropriately [BBVD99].

4.3.2. Documentation inconsistencies

Here are some particular inconsistencies with references to the source documents that we encountered during the design and implementation:

- [TSCS] p.42 Table 3-10 : The interface `i_ServiceContractInfoQuery` misses from the table.

- [TSCS] p.43 Figure 3-2 : The link between the SAG and the SAG service profile is not possible with the currently defined IDL structures.
- [TSCS] p.47 topic 3.6.3 : The description of the interface i_ServiceContractInfoQuery misses.
- [TSCS] p.47 getServiceContractInfo() : There is a sentence mentioning SAGs specified as parameters while in the IDL files the parameters are Service Profile IDs which is irrelevant. The same inconsistency occurs between the comment in the IDL file (just above the operation definition) and the operation definition itself.
- [TSCS] p.49 topic 3.6.5.1: The name of the interface is given as i_SubscriberSubscriptionMgmt while in the IDL definition files it is i_Subscribe. We chose the latter.
- [TSCS] p.247 Annex 4. Topic 4.1: There is a sentence mentioning two interfaces i_subSCooManagemet and i_subSCMNotify while on the next page and in the IDL definitions they are enlisted as i_SubscriberLCMgmt and i_ServiceContractInfoUpdate.
- The description of the service profiles scheme is not consistent. A special subscription assignment groups are created to group entities (users and terminals) and to assign to these groups service profiles. Then occasionally it is mentioned that the entities could be assigned service profiles directly. Then suddenly on p.244 of Annex 3, it is explained that there is a third way, a user profile assigned to a user which does the same as the service profiles. Here we had to make a decision in order to be able to implement a reasonably sounding profile model. Since the service profiles scheme was not extensively used in the MESH Release 2, we decided to keep the model as simple as possible.
- Changes to the original TINA interface definitions: The i_SubscriberLCMgmt and i_ServiceContractLCMgmt interfaces were only described but not prescribed. We had to fill in a number of operations that we needed for the communication between the internal sub components within the Subscription Component. The i_SubscriberInfoMgmt interface were extended with several operation since the original TINA idl specification was not expressive enough. Additional i_ServiceMgmt was defined to provide additional service manipulation features to the SUB. Originally this role had to be done from the Service Life Cycle Management component (SLCM) but since this component was not implemented the SUB was extended to meet the requirements.

4.3.3. Lack of functionality of subscription management operations

Specifically with respect to subscription management, the following problems were encountered:

- Inaccessible structures (t_SAE)
  Some structures in the model were equipped with operations for creation/deletion but no access operations were provided.

- Performance inefficient operation definitions
  Some of the operations drag big structures from the remote objects. It would be preferable to decompose these operations into several ones, which fetch small parts of the structures, since the most frequent need is not the whole structure.

4.3.4. Organizational problems

Finally, two problems of an organizational nature have to be mentioned:

- Machine upgrading
  The upgrade and purchase of software products for development machines was sometimes delayed due to administrative inefficiency and insufficient synchronization between the engineering and management departments.

- Staff reallocation
  Some staff changes during the project led to loss of expertise and delay due to familiarizing new personnel with the project's topics, results and objectives.

5. Status

MESH officially ended in November 1998. A closing ceremony was organized on December 2, 1998, at CTIT in Enschede. During this ceremony a (successful) demonstration of the MESH Release 2 prototype was presented during a 1 hour session. The demonstration consisted of 4 scenarios, each demonstrating different features of the prototype. The scenarios are shown below.

Scenario 1

In this scenario two end-users interact with each other. User 1 is an experienced platform user and introduces User 2 to the system. User 2 has a PC with only browser software.

Order of features of the platform being demonstrated:

- WWW integration
- Anonymous access session
- Access & subscription session
- User invite and shared white board service
- User preferences
- Session mobility
- User mobility

Scenario 2

In this scenario a new service (shared database access) is introduced. A service developer and service provider interact during the process.

Order of features:

- the powerful extensibility capabilities of the platform, by simply adding new services components to a central repository.
- the service providers administration process for adding a new service.
- the automatic downloading and distribution of new services to end users.

Scenario 3

In this scenario the shared database service is demonstrated by two end-users browsing the requirements database.

Order of features:

- activation of a new service.
- shared browsing through a database.
- read/write/ownership permissions on shared resources.

Scenario 4

In this scenario the addition of audio, video and possibly other R1 media is demonstrated. The two end-users from scenario 4 setup an audio/video conference and evaluate the platform.

Order of features:

- audio & video services.
- MESH Release 1 compatibility.

6. Conclusions

The TINA principles, concepts and architectures proved to be conceptually sound. The CTIT development on the MESH services platform demonstrated that TINA allows an easy and straightforward approach to

the management of user oriented subscription information. The software component decomposition approach of TINA allows a flexible implementation that once it is operational can be easily manipulated.

During the implementation phases of the MESH services platform, however, we encountered several problems as well as inconsistencies in the TINA documentation, mainly concerning the IDL interfaces. In our opinion, the TINA access and service levels are the most mature [BBVD99].

The CORBA middleware technology proved to be useful and very appropriate for a base of an implementation of the DPE environment needed by TINA. The particular implementation - OrbixWeb from IONA - proved its robustness and flexibility.

CORBA combines very well with the Java technology. The platform independence, portability and easier integration provided by OrbixWeb, enabled us to implement many features we would have to sacrifice if using other technologies, given the time frame we had in the project.

Within two years time, we managed to design, implement and demonstrate a prototype of the MESH services platform. To the designers of the platform, this proved the usefulness and applicability of TINA; to the users of the platform, the functionality and flexibility of the platform could be demonstrated.

The results and experience of the MESH project were important inputs to the projects AMIDST [AMIDST] and FRIENDS [FRIENDS]:

- AMIDST (Application of Middleware in Services for Telematics) aims at developing a 'next generation' middleware that exploits component and agent technology to support dynamic service composition and service provisioning for nomadic computing. This project makes use of the knowledge regarding TINA and component-oriented development built up in the MESH project. AMIDST combines the TINA approach with other (e.g., agent-based) approaches in order to solve the difficult problems of service, QoS and resource management for mobile application and wireless network environments.

- FRIENDS (Framework for Integrated engineering and Deployment of Services) aims at developing a 'state-of-the-art' platform that integrates support for service development, exploitation and usage. This project makes use of the MESH services platform itself that is developed in the MESH project, and will improve and extend this platform, primarily to better support off-line service creation and exploitation of services.

Abbreviations

CORBA	-	Common Object Request Broker Architecture, developed by the Object Management Group
CTIT	-	Center for Telematics and Information Technologies at the University of Twente
DBMS	-	Database Management System
DPE	-	Distributed Processing Environment
DSC	-	Distributed Software Component Framework, a framework accommodating the TINA architecture requirement for a component oriented Distributed Processing Environment
GUI	-	Graphical User Interface
IA	-	Initial Agent
IDL	-	Interface Definition Language
JDBC	-	Java Database Connectivity
MESH	-	Multimedia services on the Electronic Super Highway
NAP	-	Network Assignment Point
ODBC	-	Object Database Connectivity
PA	-	Provider Agent
PLATINUM	-	Platform providing Innovative-services to New Users of Multimedia, a predecessor project to MESH
SAE	-	Subscription Assignment Entity
SAG	-	Subscription Assignment Group
SCM	-	Service Contract Manager
SCO	-	Service Contract Object
SCoo	-	Subscription Coordinator
SubM	-	Subscriber Management
SubO	-	Subscriber Object
TINA	-	Telecommunications Information Networking Architecture, developed by the TINA Consortium
UA	-	User Agent
UAP	-	User Application
USM	-	User Session Manager

References

[MESHR2arch] Bakker, J. H. L., H.J. Batteram, H. P. Idzenga, V. Jones, L. F. Pires, M. J. van Sinderen, P. G. A. Sijben, J. P. C. Verhoosel, R. G. de Vries, "System Architecture for MESH Release 2", MESH/WP0.2/N002/V06

[TSCS] TINA-C, Service Component Specification: Computational Model and Dynamic, http://tinac.com:4070/1/97/services/docs/scs/compmod/final/idl/.

[TNCS] TINA-C, Network Components Specification,

http://tinac.com:4070/1/97/resources/network/docs/ncs/v2.2/idl/modules.

[TSA] TINA Service Architecture Version 5.0/URL

[BaBa98] Bakker, J.L., H.J. Batteram, "Design and evaluation of the Distributed Software Component Framework for Distributed Communication Architectures", Proc. of the 2nd Intl. Workshop on Enterprise Distributed Object Computing (EDOC'98), IEEE Press, pp. 282-288.

[BBVD99] Batteram, H. J., Bakker, J. L., Verhoosel, J. P. C., Diakov, N. K., "Design and Implementation of the MESH Services Platform", accepted for TINA99.

[CORBA] http://www.omg.org

[MESH] http://www.mesh.nl/extern/english.htm

[SiCF96] Sinderen, M.J., P. Chimento, L. Ferreira Pires, "Design of a shared whitboard component for multimedia conferencing", Proc. of the 3rd Intl. Workshop on Protocols for Multimedia Systems (PROMS'96), October 1996, Madrid, Spain, pp. 1-16.

[SiFe98] Sinderen, M.J. van, L. Ferreira Pires, "The application of TINA in the MESH project", Proc. of the 5th Intl. Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS'98), Springer LNCS 1483, pp. 77-82.

[AMIDST] http://amidst.ctit.utwente.nl

[FRIENDS] contact Telematica Instituut, http://www.telin.nl

[TINA] http://www.tinac.com