Use Case Maps as an Extension of UML for System Integration and Verification

Arkadiusz Rys[1], Czeslaw Jedrzejek[2], Andrzej Figaj[2]

[1] Centre of Excellence in Telematics, Institute of Control and Information Engineering, Poznan University of Technology,
5 Maria Skodowska-Curie Sq. 60-965 Poznan, Poland,
czeslaw.jedrzejek@put.poznan.pl,
WWW home page: http://cdtelema.put.poznan.pl
[2] ITTI Ltd., Palacza 91A, 60-273, Poznan Poland

Abstract. This paper presents application of use case maps as an extension of UML for system integration. The methodology is used in context of scenario based integration and testing for a very large system - Daidalos [3] system. The Use Case Maps are used not only for requirements specification. The description has many advantages for integration, namely for verification of interfaces between components, and also for verification of correct component activity. This is particularly important for flow of global parameters, such as identifiers, and user profiles. The method extends use of UML for architecture specification and refinement and in next step is amenable to adopting MDA paradigm.

1 Introduction

The MDA paradigm of software engineering [1] is currently most promising. For various reasons, mostly steep entry barrier, and existence of large legacy base, it is still not widespread. The use of modelling, namely with UML 2.0 is more accepted. There is an issue what is a proper use of modelling depending on type of a project and software developing organisation.

We present methodology and initial experience related to system design, modelling and implementation of software for a large research project, namely Daidalos IP project [2].

The Daidalos system consists of about 50 subsystems, each having 5-20 components. In the beginning of the project the architecture was developed using textual specification, informal functional block representations and logical MSC diagrams. Complexity of the architecture makes it difficult to even do inventory

[3] The work described in this paper is based on results of IST FP6 Integrated Project DAIDALOS. DAIDALOS receives research funding from the European Community's (EC) Sixth Framework Programme. Apart from this, the European Commission has no responsibility for the content of this paper. This paper may contain forward-looking statements relating to advanced information and communication technologies. Neither the DAIDALOS consortium nor the EC does accept any responsibility or liability for any use made of the information provided in this paper.

of elements. Consistency and interoperability of components is a major difficulty in Daidalos.

Facing difficulties with integration, the project managers have decided that the level of formalisation had to be increased.

Basically this corresponded to enhancing architecture view starting from logical view and ending up with development view, which is equivalent to the more formal approach such as Kruchten's "4+1" view [3] (Rational) (Fig. 1). The

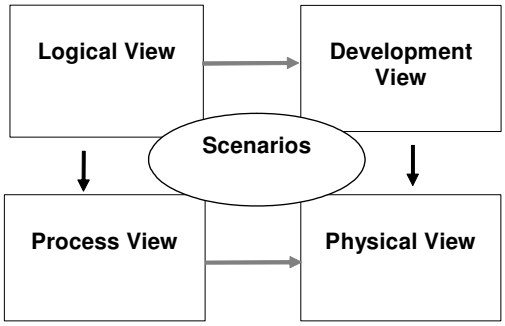

Fig. 1. Logical View, Development View, Process View, and Physical View, "4+1"

UML includes a set of models that are able to provide different levels of system view and accuracy for component modelling,

For a large system there is a need to have an "overall system" view and more detailed subsystems views. Building scenarios depends on both views. The most difficult problem is analysis and handling of content dependence relationships (i.e, how component interfaces and events interact). Wu, Chen, and Offutt [4] specified several strategies how to approach these problems, and relate them to testing:

1. User specification based (structure diagrams and naming conventions)
2. Collaboration diagram based
3. Statechart diagram based
4. Program analysis based.

This paper claims that above approaches fulfil needs for relatively smaller systems, or subsystems of a large system. Collaboration diagrams and sequence diagrams representing interactions among different objects in a component, allow to develop corresponding interaction graph that will be used to evaluate the control flow of a component. However, they are too detailed for "overall" system view, and are not flexible enough, particularly for a research system.

The question is how to proceed in these circumstances is crucial for system engineering. We present the design, development and implementation procedures that are adaptable in context of architecture-centric evolution. The paper is concerned with integration and testing in large software development organizations,

comprised of many partners, such as EU funded integrated projects. Extended Use Case Maps (EUCM) are proposed for the integration, and UML 2.0 and its extension are evaluated for this purpose. The approach is demonstrated using selected elements and functions of full Daidalos telecommunication system.

2 Choosing a View of a System and Requirements

The starting point of our pragmatic approach is the SEI V&B - View and Beyond method stating that "Architecture serves as the basis for system analysis. To support that analysis, the architecture documentation must contain the information necessary for the particular type of analysis being performed" [5].

According to SEI V&B methodology there exist layers of views. The most basic is Primary Presentation of the View in form of textual specification. It contains Element Catalog (block representation of functional modules) supplemented by message sequence diagrams (MSC). Such a documentation is contained in project in deliverables of EU funded projects. When MDA is not adopted statecharts appear only in low-level code.

The next level, level 2 contains [5]:

a Elements and Their Properties
b Relations and Their Properties
c Element Interfaces
d Element Behaviour.

In this work we aim to work out an appropriate depth of views that allows for integration and testing of a complex research system such as DAIDALOS gradually In research type projects the aim is demonstration of novel types of scenarios, and functionality. This means that actually a system may not be even complete, and contain dummy elements, however, it must reach a state for which important pieces of technology work. The attempt is made is to find a balanced solution that would not put impossible demands on a consortium and the same time enforces a minimum degree of formality.

In such circumstances there has to be verification of architecture in terms of a model of process flow for Scenario Based Integration and Testing (SBIT). Only such a view is as able to capture global features (such as analyse VID (Virtual Identifier) transport across Daidalos system, for example).

Our approach is an extension of Use Case Maps (UCM) approach [6][7]. According to [6] use cases are structured descriptions of interaction scenarios between a system to be designed and users of the system. Use cases explain preconditions, postconditions, and the critical scenarios themselves as a starting point for design and, ultimately, for testing the implementation. Use case maps provide a visual notation for use cases and also a means of extending them into high-level design.

A scenario is a dynamic model of a system that depicts the sequence of execution of responsibilities that constitute reactions of the system to a stimulus

and results in a deterministic end state. Use cases and scenarios are applied to describe the functional requirements of a software system components.

The design procedure starts with use cases. They serve to develop architecture (higher level view): structure (higher level components), deployment, MSC and high-level statechart diagrams. Often, rather then than detailed interactions, protocol names are used (such as COPS, Diameter, SIP, or corresponding IETF RFC).

The architecture is then mapped into extended Use Case Diagrams in a form of swimlanes traversing several components. In our approach Extended Use Case Diagrams serve as means of architecture verification, evolution and integration. This is contrast to standard UCM approach that is used in area of formulation of requirements. As an extension of UCM it is possible to generate message sequence charts, MSCs [8-9]. The technical MSCs show how the high level scenarios can be realized by technical means of the system infrastructure.

The procedure presented in Fig. 2 is a basis for SBIT - scenario based integration and testing.

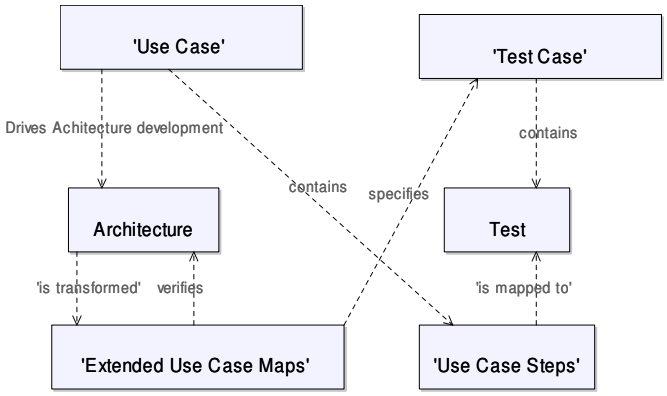

Fig. 2. Mapping of Extended Use Cases Maps to Architecture and to Test Cases

Relation between uses cases and testing are a subject of a number of recent works. In Strembeck and Zdun approach, a test is defined as a formalized and executable description of a scenario. Tests are derived from use case scenarios via continuous refinement. The use case and test information can be associated with a software componentas embedded component metadata [10]. M. Riebisch and M. Huebner [11], S. Pickin and J.-M. Jezequel [12] and L. Briand and Y. Labiche [13] derive test requirements from either system sequence diagrams or UML 2.0 diagrams.

More formal approach related to application architecture and Extended Use Cases Maps to testing will be a subject of a future work.

In order to integrate and test a system integrators need to use the following procedures and information that describes a test bed:

1. Set of nodes needed to build the test bed.
2. The network infrastructure in a form of physical and link layers of the network.
3. The list of IP connections between nodes that are needed to configure IP routing between nodes.
4. The list of subsystems that must be installed and started at each node.
5. The order, in which the subsystems must be started.
6. The list of IP connections between subsystems within nodes (in terms of needed ports).
7. The list of IP connections between nodes (in terms of ports, source and destination addresses, subsystems responsible for handling each connection, information about which protocol will be handled through each link).
8. The configuration data for each node and subsystem.
9. A description of starting state of each node and subsystem.
10. A list of points of observation (log tracing) for communication between subsystems and nodes.
11. List of points of observation to register change of states for subsystems and nodes.
12. A specification of test cases.
13. A specification of logging format.
14. A specification of formats for subsystem and node states.

For application of our approach we selected Nidaros: a simplified version of the Daidalos system that executes 7 steps. We present 2 first steps of the Nidaros scenario.

3 Diagrammatic Representation of Architecture Artefacts Leading to Integration and Testing

3.1 Methodology of Scenario Based Integration and Testing

Our methodology is based on standardised solutions with extensions. UML 2.0 delivers formalism of deployment modelling that allows to present:

- system nodes and their interconnections,
- which subsystems are deployed at which nodes,
- execution environment description.

This way of modelling can be used to build top level models that show test bed structure. The nodes from such system provide anchors for already developed models. The main goal of the outlined modelling methodology is first systematise and collect all information concerning a system (here Daidalos) that are scattered around different documents and models. The analysis of this information that is needed to build up these models enables to discover deficiencies and

inconsistencies in the system description. Of course, the model does not deliver directly all data enumerated in above mentioned 14) points. For example, the order of subsystem installation and starting must be described in a separate document that describes in detail the deployment specification.

Throughout this work we present realistic case studies based on Daidalos project. Here, we present structured class diagram of the mobile terminal node.

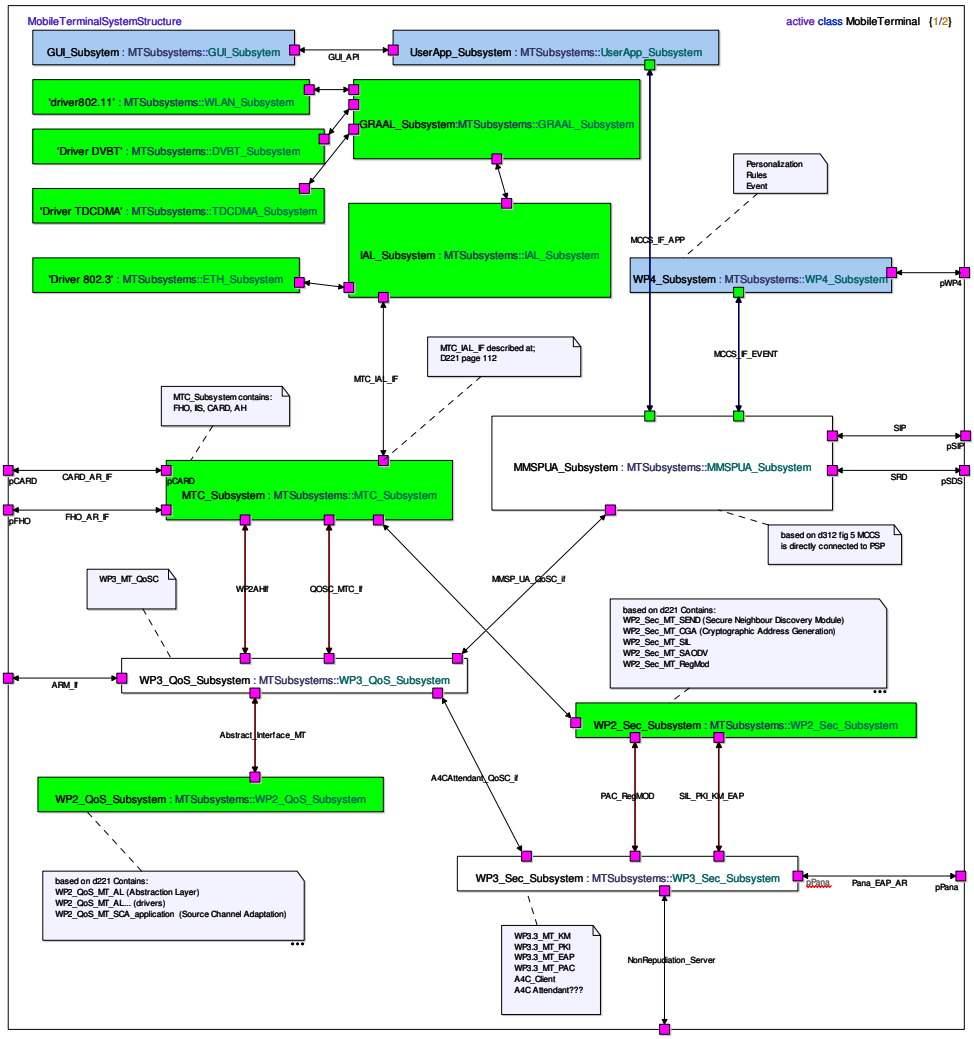

Fig. 3. Mobile Terminal node structural model

The scenario must contain the following elements:

- start state,
- end state,
- sequence of events,
- sequence of responsibilities.

There are following problems that should be solved for weaving a testing scenario:

1. Choosing a system scenario to be mapped into a test scenario.
2. Precise definition of a system infrastructure on which a test case will be executed in terms of a component model.
3. Mapping thesystem requirements into starting and end states.
4. Mapping a system scenario onto infrastructure (Use Case Map preparation).
5. Definitions of subsystems' responsibilities for each scenario
6. Mapping testing case to the technical infrastructure.
7. Definition of test acceptance.

The system works in several layers: layer 2 of OSI (corresponding to work-package 2, WP 2), layers 3-4 of OSI, (corresponding to WP 3) and pervasive network (layers 5-7, corresponding to WP4).

There are following steps needed to model Nidaros in a way to achieve above mentioned goals (these will be illustrated with selected Nidaros diagrams):

1. **Preparation of an inventory deployment model.** The main goal of this model is to show how many functional nodes we have within a system and what kind of software artefacts are allocated at given nodes. This model should be closely related to real hardware equipment accessible in a test bed site. The equipment and operating system requirements must be identified in this model. The main goal of this model is inventory of system hardware and software components.
2. P**reparation of a deployment model with low level communication paths.** Deployment model shows what kind of functional nodes we will have at our test bed. This model should contain communications paths between nodes at the level of physical routing. If two nodes are connected by physical routing communication path it means that the communication is done in one hop. There is no additional routing and there is direct address resolution between layer 2 and layer 3. This information is crucial to identify how routing must be configured at test bed network.
3. **Preparation of a deployment model with layer 3 communication paths.** (Fig. 4) The model prepared in step 1) does not show the communication paths that will be used by software components interfaces and protocols (e.g. Diameter protocol between A4C client and A4C server). Therefore, it is necessary to show what kind of IP links exist in test beds, to identify traffic transmitted through communication paths modelled and step 1), also for a purpose of traffic sniffing and looking for specific communication patterns between components. This way packages of specific protocols and target destinations are identified. For example, Mobile Terminal can keep up many layer 2 connections that can carry several IP connections, and a specific connection has to be resolved.

4. **Preparation deployment diagram on application level.** In the scope of a pervasive computing system developed in WP4, we have several distributed applications that reside above the infrastructure presented in steps 2) and step 3). Therefore, it is necessary to show how this infrastructure is mapped onto physical and logical network infrastructure. The main goal of this model is to determine which components of WP4 are used in Nidaros and how they are interconnected. Then network architecture on application layer determine pervasive computing environment requiring adequate network resources. This is also needed to set up observation point for application to trace WP4 application behaviour.
5. **Preparation of physical deployment model for each node.** Each node may contain many physical processes (running applications) that communicate each other within a node. These applications are developed by different partners, very often in different technologies. Therefore, to correctly deploy a node we need information about relations between software subsystems and components running on a specific node. This model needs to understand:
   - how software subsystems are interconnected,
   - what is a mapping between software components interfaces and communication paths identified in steps 2), 3) and 4),
   - where inter-process communication should be traced,
   - what is an order of starting these processes (e.g. Mobile Terminal Controller should be started before other processes within Mobile Terminal),
   - where inter-process coordination is needed,
   - how interfacing between different work packages is realised.

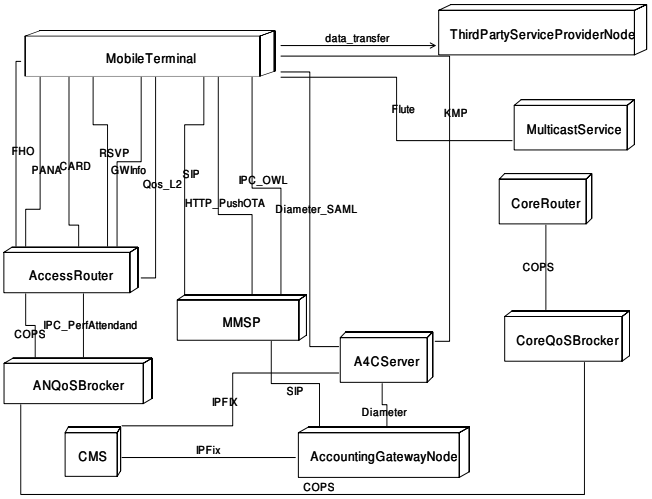

Fig. 4. Deployment model with higher level (layer 3-4)communication channels

6. **Preparation of node behaviour.** Nidaros test bed is scenario driven. It means that for integration process we need information about behaviour of each node. It is crucial for integration to identify if subsystems and obviously the whole test bed correctly goe through the assumed scenario. The best way to present node behaviour is to use a statechart notation (see, Fig. 5). Then one can show what kind of stimuli and reaction we can expect from a node during realisation of test bed scenario. Having this model we can identify discrepancies during acceptance tests. It is desirable to work at different levels of this modelling. The first level is to present a model that translates free text description of the scenario into a more formal statechart notation. Such a description can then be used for development of technical scenarios and statecharts for processes, running on nodes (this is complementary to step 5).

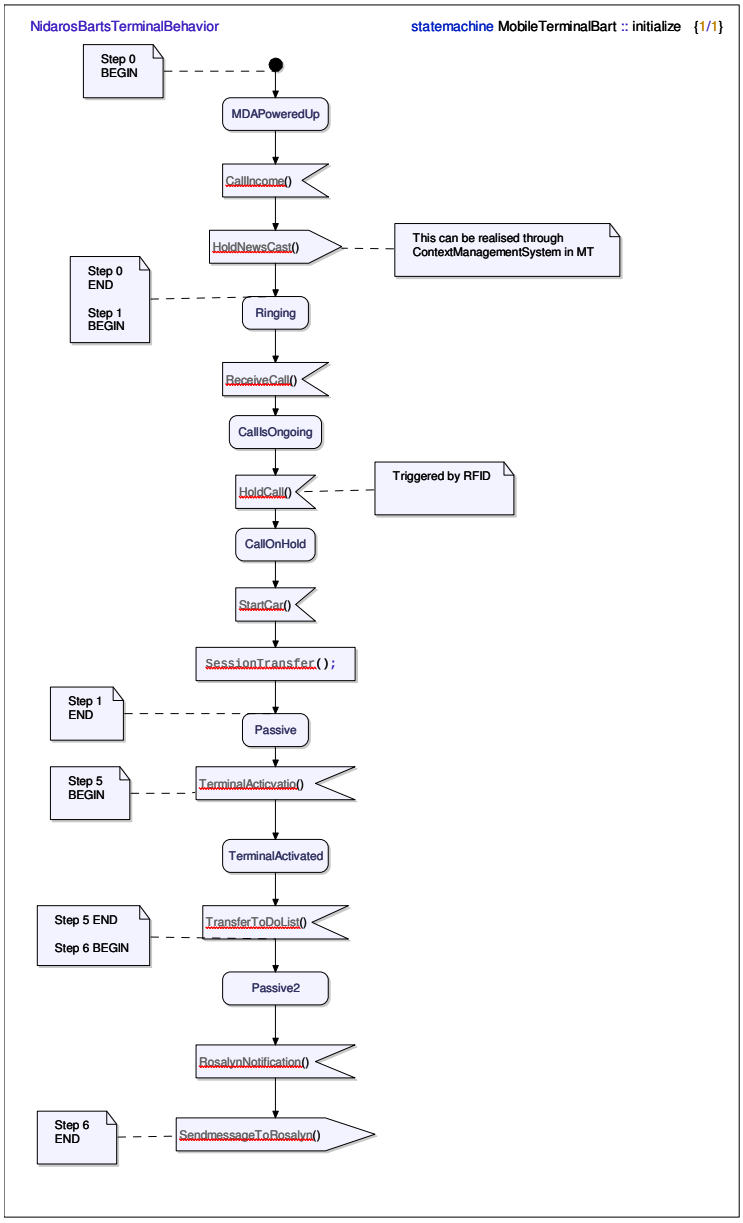

Fig. 5. Statechart for Bart's MT (SDL representation)

7. **Preparation of technical MSC.** This step is crucial to describe dynamics of the system. MSCs developed in this step must be complementary to system structure described by all deployments diagram. This means:
   – it can use as lifelines only components identified in step 1)

- exchange of signals and stimuli is possible only between those nodes that posses communication paths modelled in steps 3) and 4),
- MSCs must fit to statecharts prepared in step 6) .

MSCs must be detailed enough to answer following questions:

- which nodes (identified in step 1)) take part in specified scenarios,
- which processes identified in step 4) are involved in these scenarios,
- what data are transferred between nodes with some concrete data, this is vitally crucial to specify detailed system test cases,
- all interactions must identify a protocol used; this must be consistent with data paths defined in steps 3) and 4).

In Fig. 6 we present MSC for PANA based mobile terminal registration for which an Extended Use Case Map is created in Fig. 7.

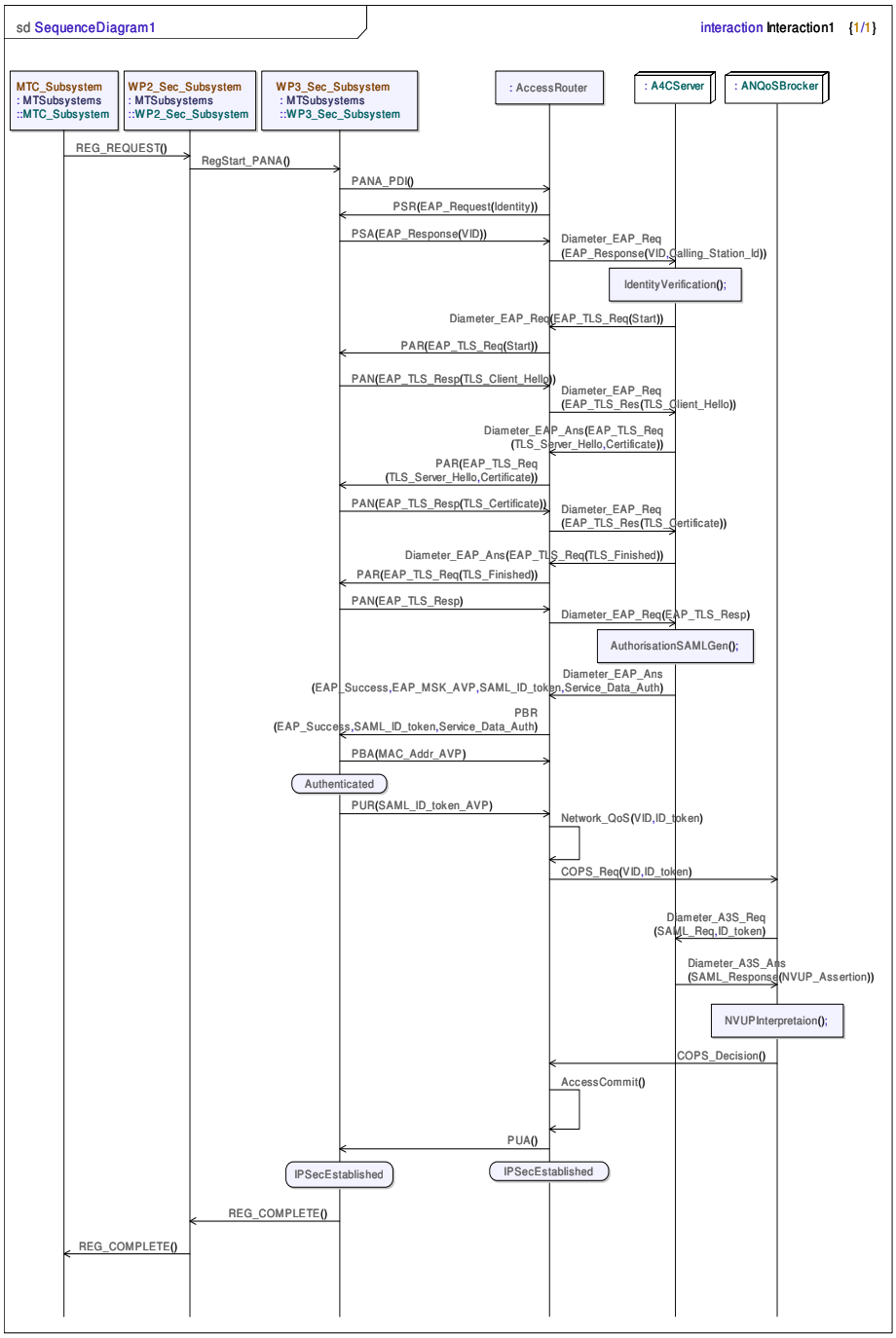

Fig. 6. PANA based mobile terminal registration

8. **Preparation of Use Case Map.** MSCs are very useful to illustrate how scenarios are realised by the distributed system. When we draw MSCs, one single MSC illustrates only one very narrow scenario of a system activity and it is assumed that there are not influences from other scenarios. However, in distributed real-time systems (Daidalos belongs to this class of systems) in majority of cases this not true. A method to identify such feature dependencies is a function of Use Case Maps.. In our approach the goal of using an Extended Use Case Map is to illustrate a scenario directly on the system infrastructure model (e.g. deployment diagram, class diagram, component diagram) depending on an analysed level of a scenario.

3.2 Example of Scenario Based Integration and Testing with Use Case Maps

For distributed systems it is important to verify if planned technical realization (described by technical MSCs) is compatible with system architecture. In UML 2.0, Sequence Diagrams may be decomposed: horizontally - with reference interaction fragments, and vertically - by decomposing lifelines. Also sequence diagrams have added a number of interactions such as: fragment operators, Naming, Flow of Control, Ordering and Causality.

However, it is quite difficult to work with a large set of MSCs. A change of design is quite disruptive on them. An important part of our approach is to being able to show flow of information through components but with a complete use case line (this contains more information that presented in slide 29 of [14]).

Our methodology is a good technique for such verification with usa of the Extended Use Case Map formalism.

As an example we show in Fig.7 how to transform MSC into Extended Use Case Map.

The following notation is used in this model:

1. A component is described a by pair <Node Name> :: <Component Name>. Node Name must correspond to the nodes from a corresponding deployment diagram, whereas Component Name must be present in a structural diagram of a given node.
2. The components are presented in such a way that the interfaces are visible. They are described by the name of used protocols.
3. The X-crosses placed on the use case flow represent actions that are executed along the flow of actions. They are denoted by symbols like A1..A3, B1..B3 and C1..C5.
4. Each flow starts from dot and ends at bar (e.g black flow starts in WP3_sec_Subsystem by black dot and ends in A4Server:DiameterBaseProtocol by bar-ending).

In Fig. 7 lines have the following meaning :

- PANA pre-phase (solid line flow),
- EAP/TLS (dashed line flow),
- NVUP (Network View of User Profile, a proprietary solution to configure QoS) retrieving (dotted line flow).

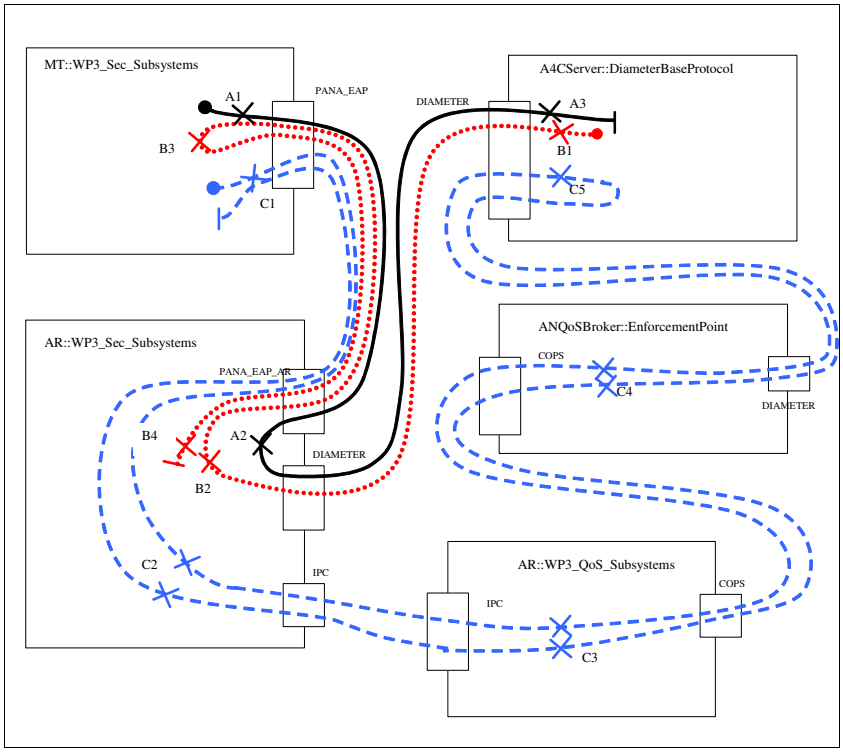

Fig. 7. Use Case Map for PANA mobile terminal registration

4 Conclusions

This work is concerned with issues of software development, particularly architecture verification, system integration and testing for large systems (such as developed in EU IST Integrated Projects).

The biggest challenge in large projects is communication between different workpackages and partners. The methodology developed in this paper allows to present architecture in a holistic view across all workpackages and subsystems. The modelling of Daidalos showed that even UML 2.0 has some inconveniencies when the system architecture must be presented and evolved. This problem has been raised in the the OMG forum and a new UML profile for system architecture, SysML [15] is in progress. In contrast to other UML models Extended Use Case Maps give clear possibility to visualise how system works as a whole. This way, together with other models they provide a good platform for common communication between project members. The method as it was shown in this article is architecture-centric and facilitates reasoning about system architecture features. It allows to discover discrepancies that are not visible in other mod-

els. Very often system architecture contains gaps, but to discover them requires to compare many different models which in UML is not straightforward (e.g. everlasting problem of coupling structural and behavioural models). Extended Use Case Maps deliver powerful and sound method that shows such a mapping, which however needs to be futher elaborated and formalised.

Having an Extended Use Case Map model it is possible to analyse step by step flows to discover implicit system features and also weave integration test cases.

Within our examples we have used a subset of Use Case Map notation with some extensions to UML. In this model there are not as many details as visualised by MSCs. However, by loosing some details we profit from better integrated view of the system. This is particularly important for flow of global parameters, such as VID and NVUP in Daidalos system. The lack of details is a generic feature of Use Case Map notation. Therefore, the requirements analysis is the most often area where Use Case Maps are used. In our examples, we have shown that this notation has excellent expressiveness to facilitate the intermediate and last stages of the system development - namely architecture verification, integration and testing.

In future work we will design explicit test cases based on scenarios. Also we are working to couple EUCM view with low-level practical implementation approach such as bootstrapping of processes.

UMC approach gains popularity, however, work on Z.152 standard for URN: User Requirements Notation - Use Case Map Notation is still in progress. However, there is a growing realisation that for dimensioning scope and size of projects Use Cases happen to be a more consistent artefact than functions upon which to base an early project estimate [16]. This corroborates indirectly our claims. The fact that it is advantageous to employs Extended Use Case Maps for architecture-centric verification and evolution before code execution effect MDA implementation.

The authors thank Daidalos project managers Hans-Werner Bitzer and Amardeo Sarma for elucidating discussions and reviewers of the paper for comments.

5 References

References

1. www.omg.org/mda/ - Model Driven Architecture (MDA) suite of standards include Unified Modeling Language (UML); Meta-Object Facility (MOF); XML Meta-Data Interchange (XMI); and Common Warehouse Meta-model (CWM) and as well as separation of a model (PIM) from implementation (PSM). Developed by the Object Management Group (OMG).DAIDALOS, Designing Advanced Interfaces for the Delivery and Administration of Location independent Optimised personal Services IST 2002-IST1 Contract No. 506997
2. DAIDALOS -Designing Advanced Interfaces for the Delivery and Administration of Location independent Optimised personal Services, Contract Number 506997, www.ist-daidalos.org. [3] Kruchten, "The 4+1 view model of architecture," IEEE Software, 12 (6) November 1995, 42-50.

3. Kruchten, "The 4+1 view model of architecture," IEEE Software, 12 (6) November 1995, 42-50.
4. P.Clements and al "Documenting Software Architectures in an Agile World" CMU/SEI-2003-TN-023 report (www.sei.cmu.edu/publications/ documents/03.reports/03tn023.html).
5. Y. Wu, M.-H. Chen, and J. Offutt, UML-based integration testing for component-based software, In Second International Conference on COTS-Based Software Systems (ICCBSS 2003), pages 251-260, Ottawa, Canada, November 2003.
6. I. Jacobson, M. Christeron, and G. Overgaard, Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, 1992.
7. R.J.A. Buhr, R.S. Casselman, Use Case Maps for Object-Oriented Systems, Prentice Hall, 1996.
8. D. Amyot and G. Mussbacher (2000), On the Extension of UML with Use Case Maps Concepts. In: The 3rd International Conference on the Unified Modeling Language, York, UK, October 2000. Deriving Message Sequence Charts from
9. A. Miga, D. Amyot, F. Bordeleau, D. Cameron and Murray Woodside, Use Case Maps Scenario Specifications, Tenth SDL Forum (SDL'01), Copenhagen, Denmark, June 2001. LNCS 2078, 268-287
10. M. Strembeck and U. Zdun. Scenario-based component testing using embedded metadata. In Proceedings of the Workshop on Testing of Component-based Systems (TECOS), pages 31-49, Irsee, Germany, September 2004. Lecture Notes in Informatics (LNI) P58.
11. M. Riebisch and M. Huebner, Traceability-Driven Model Refinement for Test Case Genertation. In Proc. of the 12th IEEE International Conference on the Engineering of Computer-Based Systems (ECBS), April 2005.
12. S. Pickin and J.-M. Jezequel, Using UML sequence diagrams as the basis for a formal test description language. In Proc. of the 4th International Conference on Integrated Formal Methods (IFM), April 2000.
13. L. Briand and Y. Labiche, A UML-based Approach to System Testing, Journal of Software and Systems Modeling, 1(1), 2002.
14. M. Richardson, Use UML 2.0 for better modelling, NEC - Nohau Embedded Conference 2004
15. "Systems Modeling Language (SysML) Specification Draft 0.9", www.sysml.org, 10th January 2005
16. E. Carroll, Estimating Software Based on Use Case Points, OOPSLA05, San Diego, 2005