

Daidalos - A scenario based approach from scenarios towards integration

Constantin Werner¹, Yongzheng Liang², Jürgen Jähnert², Michael Ebner¹

¹University of Göttingen, Institute for Informatics – [ebner|werner]@cs.uni-goettingen.de

²University of Stuttgart, Computing Center (RUS) – [jaehnert|liang]@rus.uni-stuttgart.de

Abstract — In the IST Daidalos Project a blueprint of the next generation Beyond 3G Systems is designed, developed and demonstrated integrating mobile and broadcast communications. Daidalos delivers pervasive end-to-end services across heterogeneous technologies. This requires the development of key technologies as demonstrated by the Daidalos five key concepts which are MARQS (Mobility Management, AAA, Resource Management, QoS and Security), VID (Virtual Identity), USP (Ubiquitous and Seamless Pervasiveness), SIB (Seamless Integration of Broadcast and Federation)

The Daidalos results – architectural framework, conceptual solutions, software components and sub-systems, and the overall demonstrators based on two scenarios – will contribute to the next generation of networks.

Having this goal, an appropriate methodology on how to guide the overall development process is required, which relies on formal methods and tools which closely follow a scenario based overall design defining a “red line” along the overall work of any activity from the formal definition of a scenario down to the final integration.

Index Terms—System Beyond 3G, Broadcast, Mobility, Scenario-based design

I. INTRODUCTION

The Daidalos mission to empower customers and users towards mobile, seamless and ubiquitous access to the widest variety of services following targets the optimal connection of mobile users “Anywhere, Anytime and Anyhow”. This is strongly supported by the Daidalos pervasiveness concept focusing on pervasive access and services for the user, where the scenario-based approach will be a cornerstone to assure that such user needs will be driving the technical solutions. The Daidalos mobile integration and service architecture has the potential to enrich and improve the everyday life of the citizen in all its aspects from work, leisure, health and education to administrative, service and governmental processes, research, development and production structures.

In order to illustrate the potential impact that Daidalos will have, two key scenarios, namely the Daidalos Mobile University scenario which provides for the need to support mobility amongst European academic institutions as a typical

example of mobile people and groups supporting the movement of students and staff amongst universities within Europe.

Further, the Daidalos Automotive scenario provides support for the user for both, business and leisure uses of the motor car. The pervasive service environment enables here a continuum in service provision in which the user can move between buildings (e.g. home, work, airport, shops, etc.) and the vehicle, and travel around in it with flexible handover of services.

After an extensive overview of Daidalos this contribution describes the Daidalos scenarios in greater detail including the follow-on drill down process from a scenario-based design perspective, which provides the base for extracting requirements which are then pushed to the architecture definition process. After this, the integration and testing process is described allowing the overall proof of concept of the Daidalos architecture.

II. DAIDALOS OVERVIEW

Having the goal of an integrated IP-based convergence platform, Daidalos covers several specific points to serve mobile users. Daidalos will support a consolidated European approach with several European network operators to enable pervasive applications and services. The Daidalos primary centre of gravity is to provide a “pure-IP” solution targeting a seamless infrastructure that supports mobility. In addition, Daidalos goes far beyond this to integrate Ad-Hoc Networks, broadcasting (DVB-S/T/H, BWA), inter-working with the cellular level (UMTS) to support location-based services and context-aware services with a major goal to influence the most important standardization bodies (ITU, 3GPP, IETF, ETSI and DVB).

Daidalos focuses on combining different access, networking and service technologies that complement each other to provide optimized, pervasive access and services. Specifically, Daidalos enhances existing IP-based technologies and works on solving issues related to combining technologies, such as in the MARQS (Mobility, AAA, Resource, QoS and Security) framework, well knowing that these work individually, but pose new problems when functionally integrated. Daidalos will pioneer new solutions based on applying advanced modeling

and testing techniques in addition to validation via simulations and implementations.

III. THE SCENARIOS

Scenarios are widely used to communicate the value and results of the design process to recipients of design work, including development teams, marketing groups, manager, and even end-users. The basic argument behind scenario-based designs is that descriptions of people using technology are essential in discussing and analyzing how the technology is (or could be) used to reshape their activities. A secondary advantage is that scenario descriptions can be created before a system is built and its impacts felt. Scenarios present possible ways to use a system to accomplish some desired functions or implicit purpose(s) and describe unifying requirements and processes at the task level and make the integration of products in the customer environment easier.

Daidalos adopts the methodology of scenario-based design. The top-level Daidalos scenarios, the Mobile University scenario and the Automotive scenario are structured in different scenes. Despite these scenario titles, the scenes of the scenarios cover much more than the University and Automotive domains. The scenarios and scenes bring to life Daidalos concepts, especially the Key Concepts, and illustrate sections of the overall architecture and selections of the technologies developed in Daidalos. By basing the design and analysis on scenarios, the project concepts are developed from a user centric point of view. This also avoids the temptation of being technology-driven, which may not be satisfactory for users and operators who need to satisfy their customers. The scenarios will help refine technical approaches in the work packages, thereby allowing new concepts to emerge that both are in line with user needs and friendly to business needs of the operators. For this reason each scene is described from three viewpoints. These are next to the already mention user view a business view which highlights the details from a 3rd party service provider which might offer commercial services on the Daidalos platform and an operator view, which highlights the scene from an Daidalos operator point of view.

The Mobile University scenario describes the daily life of two students in different situations on and off campus in the future University. It demonstrates a platform that allows students and staff establish virtual offices, learning environments and labs across different organizational domains. Technically, the focus is on a virtually global university pervasive network and service infrastructure, which dynamically adapts to the needs of mobile users and is available from anyplace via the Internet.

The Automotive scenario describes the daily life of a user and demonstrates how Cars, people, buildings, streets and other objects are closely integrated and will become a connected world in the near future. It is liaised with the “international automotive projects”, typically at the application service layer architectural issues (especially identity, security, privacy policy framework items).

IV. THE DRILL DOWN PROCESS

Daidalos has 5 Work Packages with about three-hundred experts and researchers throughout Europe. A large system like the Daidalos overall system must be more than just a collection of code modules and should be structured in a way that enables scalability, security and robust execution under stressful conditions. A top-down design is such a method or procedure that starts at the highest level of abstraction and proceeds towards the lowest level. It constructs hierarchies and provides scalability. Each level of a procedure is refined by designing it in more detail and each lower level may then be refined again, defining it in yet more detail until the entire specification is detailed enough to validate the system. In the first phase of Daidalos, a top-down design was partly adopted and is fully considered in the actual running phase 2 of Daidalos.

The following figure shows the relationship between the 5 Work Packages.

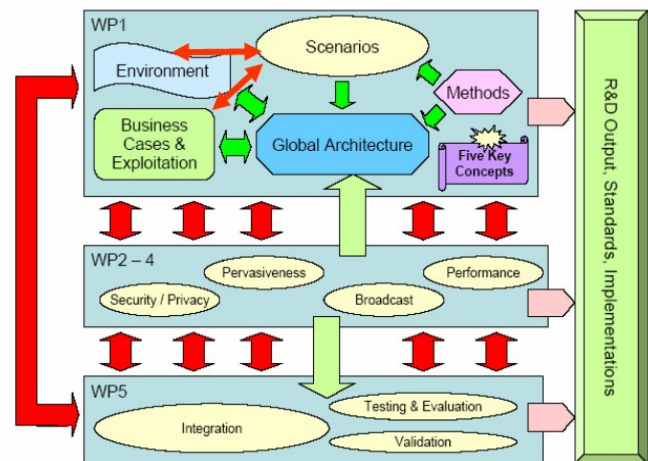


Figure 1: relationship between the 5 Work Packages

WP1: Global Architecture and Scenario Based Design

WP2: Integration of heterogeneous network

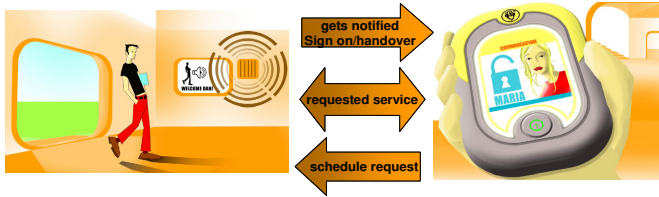
WP3: Context Aware Network Service Provisioning and Management

WP4: Enabling Pervasive Services

WP5: Proof of Daidalos Concepts

As shown the scenarios are the starting point of the system and are able to guide the research and implementation activities.

As already mentioned, in Daidalos the two scenarios are structured in different scenes - small fragments in each scenario which are enriched with details (like more precise definition of services occurring in a scene) and assumptions (like what was a user doing before a scene starts). Figure 2 shows the Scene 7 of the Mobile University Scenario



Scene 7: Before he gets to the meeting room Dani gets a notification saying "Maria arrived at Campus", as he has requested for. He invites her to meet, she quickly agrees to meet her friend Dani in one of the campus cantinas at lunchtime.

Figure 2: Scene 7 of the University Scenario

A Daidalos scenario is usually a textual storyboard, a high level conceptual design. Since the Daidalos developers are waiting for the technical requirements and specifications of the system they need to engineer, the Unified Modeling Language (UML), which has become de-facto standard for modeling an Object Oriented system, and is used. A Daidalos scenario considers usually that different kinds of users interact with a system as a whole. This can be mapped in a UML use case diagram which describes the functionality of a system in a horizontal way and interactions between users and a system. The following figure shows the use case diagram of the scene 7 above.

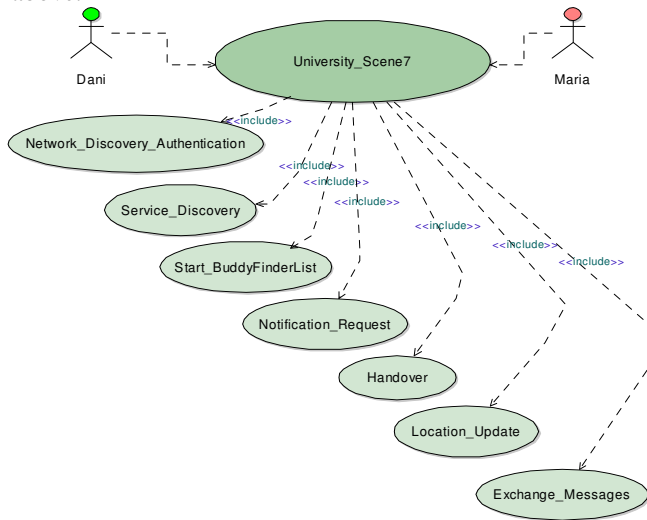


Figure 3: Use Case Diagram of Scene 7 in the Mobile University Scenario

The use cases are used to analyze and identify the high-level software components and the interactions between them.

In the following Figure 4, Table 1, and Figure 5 show the high-level components, interactions between use cases in scene 7 and interactions within one use case.

The high-level components shown in the Figure 4 are: DaniDevice, MariaDevice, AccessRouter, A4C_AccessRouter, A4C_Server, QoS_Broker, QoS_Client, NetworkInfo_Broker, ServiceDiscovery_Agent, ServiceDiscovery_Server, PervasiveService_Management, Rule_Management, SecurityPrivacy_Management, Context_Management, Personalization, Event_Management, BuddyFinder_List

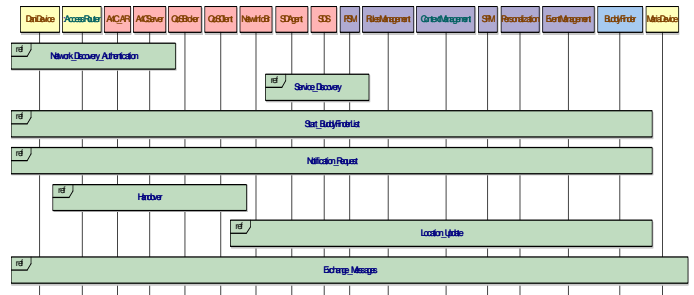


Figure 4: High-level Components and sequences involved in Scene 7

Name	Network discovery & authentication
Actors	Dani
Description	User connects to one available network and authenticates
Services involved	Dani's device, access router, A4C-AR, A4C-Server
Exceptions	No AN available
Pre-Condition	User switches on terminal
Post-Condition	User is connected

Table 1: Use Case "Network discovery & authentication" in Scene 7

The behavior of the system and system components, including the scenarios, use cases, functional and non-functional requirements, will be summarized in an adapted Software Requirements Specification (SRS) document for a research project.

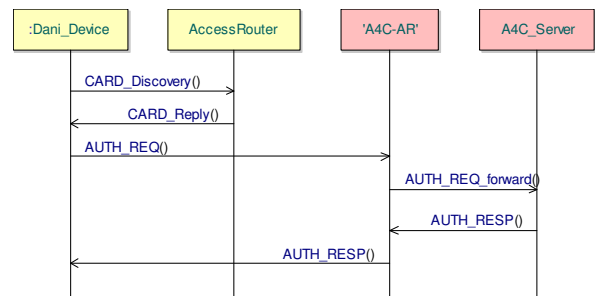


Figure 5: High-level sequence in Use Case "Network discovery & authentication" of Scene 7

An SRS document is an IEEE standard (ANSI/IEEE Std 830-1998) and recommends a document structure covering system description, interfaces, system properties and requirements. This document serves as a reference for development, testing and integration.

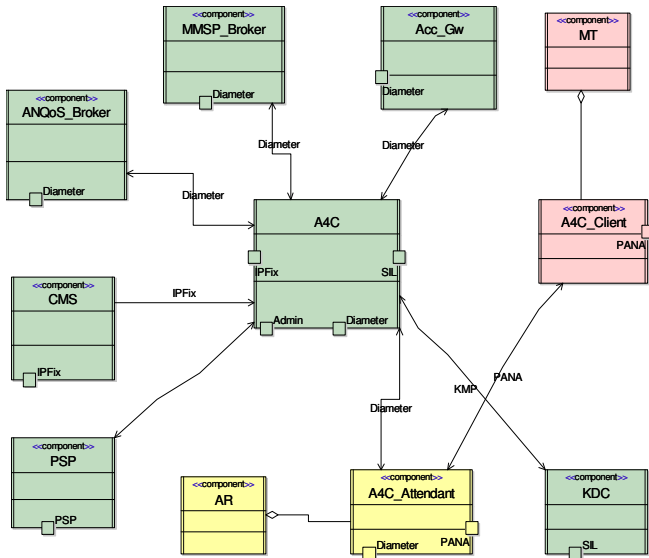


Figure 6: Detailed design of A4C Server

The process can be broken into a lower level, and describes the function, interfaces (Figure 6), dependencies and implementation of enforcing functions (Figure 7).

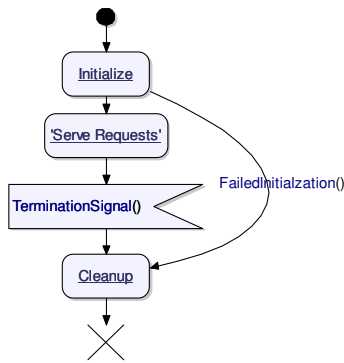


Figure 7: A state diagram for detailed design

V. SYSTEM INTEGRATION

The Daidalos system integration will closely rely on the use case specification and modeling describing the scenarios.

Here, the overall workload as assigned from the overall architecture to three different implementation workpackages (WP2, WP3 and WP4 – see Figure 1), which in turn define several subsystems, to be integrated and tested according to test specifications. Each test will be documented and attached to the delivered software towards the “higher layer” integration.

In order to facilitate this overall and very complex process, the integration has to be planned carefully and is supported by several collaborative working tools and concepts comprising versioning control, overall guidelines definition.

VI. TESTING

The Daidalos project consists of a heterogeneous and possibly volatile collection of developers. Developers will leave and join over the lifetime of the project. Some of them

have a broad background and experience of software engineering. Some of them never participated in such a huge software project yet. Many goals are vaguely defined; the overall system is not well understood by each developer. This may lead to unexpected result und erroneous feature interactions when integrating all developed components. The Daidalos project will implement a combination of several design and development techniques, which aim to cope with the common problems in huge software development projects.

A. Process Model

The overall process model of Daidalos is a top-down approach, shown in Figure 8. Basically, top-down and bottom-up is a traditional concept which has been partly superseded with the upcoming of the object-oriented paradigm. Top-down emphasizes planning and a complete understanding of the system. This implies that no coding can begin until a sufficient level of detail has been reached in the design of at least some part of the system. This delays testing of the ultimate functional units of a system until significant design is complete.

Bottom-up emphasizes coding and early testing, which can begin as soon as the first module has been specified. However, this introduces a risk that modules may be coded without having a clear idea of how they link to other parts of the system, and that such linking may not be as easy as first thought. Re-usability of code is one of the main benefits of the bottom-up approach.

Modern software design approaches usually combine both top-down and bottom-up approaches. Understanding of the complete system is usually considered necessary for good design, leading theoretically to a top-down approach. Nevertheless, most software projects attempt to make use of existing code to some degree. Pre-existing modules give designs a bottom-up flavor.

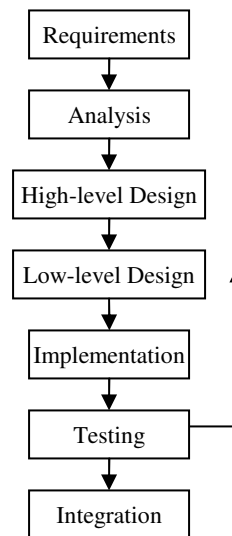


Figure 8: Top-down process

The Requirements phase will define several scenarios, use cases and (non-) functional behavior which will be

summarized in a software requirements specification document. This document is the complete description of the behavior of the system. It is a reference for all designers, developers and integrators. Several viewpoints have to be considered.

The Analysis phase will investigate how the requirements can be fulfilled and describes an initial architecture of the system. The Design phase will concretize the initial architecture which has been developed in the Analysis phase.

B. Feature Driven Development

The common problems for planning work are also threatening Daidalos: that only a minority of people knows what the concrete goal is. There will be well written but lengthy documents which may give answers, but only a minority will read them. Even if known, which track to take remains often unclear as well as the question if one is still on the right track. Success or failures are vaguely defined, and nobody knows whom to ask when problems occur or the current assigned task is unclear. First issued ideas lead to spontaneous activities, delighted that someone takes the initiative. Therefore, it is important to specify a means to give instantaneous feedback on success or failure. In particular, this means feedback within seconds. Developers also tend to become bored working on the same task for months when still the same problems are to be solved, still the same things to be implemented and no new challenges come up.

To overcome these problems, the main intuition is to give an intelligible, precise and clear specification of the goals. However, when facing reality, it will rarely succeed to give unambiguous and precise specifications, amongst others caused by different technical backgrounds of the authors and the readers. Therefore, it is required to define a process model with concrete, simple, short-term tasks, goals and milestones while avoiding unnecessary ballast. Additionally, a responsible person is always identified which forms the expert on a certain topic and assists the developers. When the goals are specified, a continuous, measurable progress should be enforced.

This is introduced by a combination of an adapted Feature Driven Development (FDD) [1], [7] and a Test Driven Development (TDD). FDD is an agile modeling approach and divides the overall goal into a set of *features* which define a small useful step towards a design target which can be completed within roughly about two weeks. A feature itself is a tiny building block for planning, reporting and tracking and its description will be specified by a simple, predefined syntax including some additional description. The detailed combined FDD and TDD process is depicted in the following Figure 9.

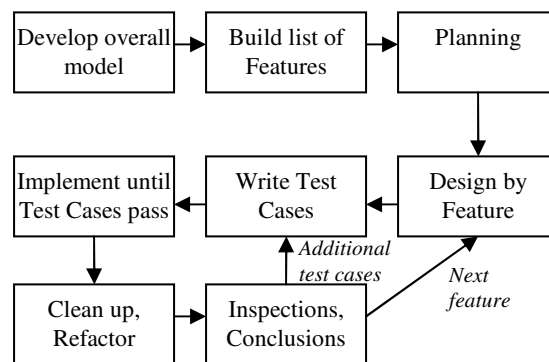


Figure 9: Combined FDD and TDD

An overall model of the system will be specified. This model is used to build a list of features, which, when successively implemented, complete the system definition. Priorities and sequences are assigned to the features by a thorough plan and the classes are assigned to class owners. A class owner is the responsible developer. He has to deliver that feature and is the assigned expert answering domain related questions. The class owner is a member of a feature team, which typically consists of a chief programmer, class owner and developers. The chief programmer leads the following design by feature and implementation phases by gathering a feature team until the feature is completed. He leads the team by examples (design) and mentoring (inspections). The class owner is one of the developers who has to become an expert on this feature. The class owner is responsible for design and implementation of that feature. Developers can join multiple feature teams at the same time. The TDD is explained in detail in the following section.

C. Test Driven Development

The TDD concept requires developing tests before writing corresponding code. The main idea is to implement only components resp. features which are required and working. However, what is really required and how will this interwork with other components? Hence, we have not only to think about component testing and writing testable code. We have to focus the test perspective also on the integration part to get the whole system to work. This focus is guided by the scenario driven development [5].

Thus, the test objective inside Daidalos is integration, consistency, and prototypes. Integration means the interworking between components. Consistency means the development of features required to realize prototypes and the consistency of interfaces between service user and provider according corresponding scenario descriptions. Prototypes means to concentrate test activities only on key scenarios required to implement prototypes and not on exotic scenarios resp. features. To sum up, test activities are done in a hybrid way divided into component testing and integration resp. system testing and they are guided by the scenarios. Other testing approaches are not considered because they are out of scope of the Daidalos project.

The hybrid test approach can also be seen as testing tight to development throughout the process and testing tight to services coupled to integration. Coupling scenarios and integration testing gives a good possibility for discussions between and inside working groups. Furthermore, it binds development and integration teams together to ensure the development of the right system and system right.

To couple scenarios and testing the scenario and test descriptions have to be discussed in detail. The scenario descriptions have to be independent of the used test languages or test systems. Test detail levels differ depending between development and system testing. However, starting point for the whole discussion is scenario description with nearest technical level. If the scenario description is too far away from technical details the generation resp. building of tests from it gets quite difficult. Scenario descriptions for integration testing require interface and data details to start a process inside the system and to proof the reaction and output from it.

For development testing a scenario description requires much more details because it is coupled to a much smaller part of the system. Hence, tests for component testing cannot be generated from requirement scenarios in general. However, they can be seen as a source about the importance of a required feature and therefore, they are an important source for component test developer.

As discussed earlier, requirement scenarios are described on several levels and the lowest level are scenes described via sequence diagrams, use cases, and/or state machines. In comparison, test scenarios contain the most technical details inclusive a tight link to architecture level. Therefore, the important step from scenes to tests requires deep knowledge of the underlying system. For component tests the component behavior in its environment is important whereas for integration tests the overall behavior has to be known [4].

A major problem in (automatic) test generation from UML diagrams is weak semantic of UML diagrams. In addition, diagrams are most time not detailed enough to get all necessary information to generate a complete test. There are many approaches to get tests from UML diagrams but they assume special conditions. Most time diagrams are used as another kind of describing tests. Hence, if diagrams are not directly related to test description, it is quite difficult to get tests automatically. Nevertheless, manual test generation or extending given diagrams to test descriptions is possible and useful. Main task is to choose the right pieces of scenes and to add missing architectural information. A semi-automatic detection of differences between tests, scenarios, and test architecture is possible by comparing interface descriptions in the different representations

In Daidalos the different levels of scenarios are described via UML2 diagrams as described before. To get an easy and test language independent switch from scenarios to tests, we will use the UML Testing Profile (UTP) to describe test cases [3]. UTP allows describing tests on an easy way for participating developer and people responsible for integration

and scenarios which are inexperienced in testing because it is quite easy to read due to its graphical view. Furthermore, tools like Telelogic TAU G2 allow validating the UTP diagrams against the corresponding scenarios described via UML diagrams. Hence, consistency between scenarios and tests can be easily achieved.

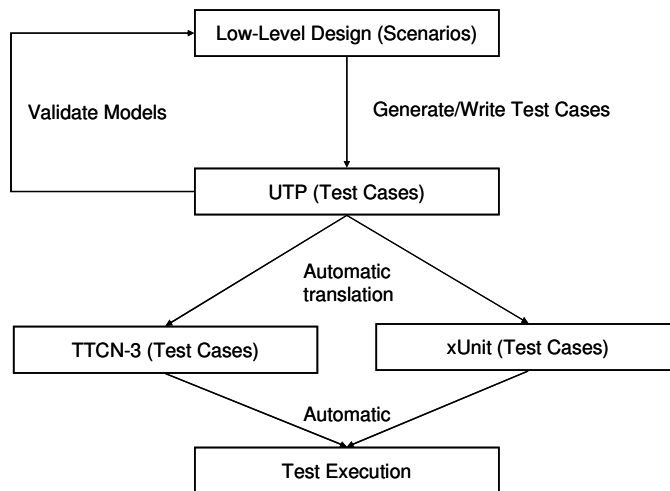


Figure 10: Test Approach

Tests described via UTP cannot directly be executed and therefore, they have to be translated to test languages resp. frameworks like TTCN-3 [1] and JUnit [6]. There is a mapping between UTP and TTCN-3 resp. JUnit defined. Hence, TTCN-3 can be used for integration testing and JUnit (and CUnit) for component testing. Because of automatic test generation from UTP and automatic test execution of TTCN-3 tests and JUnit tests, we have a tight coupling between graphical test descriptions, requirement scenarios, and system model (see Figure 10). And all together inside one modelling language and one tool if the tool supports UML profiles. Furthermore, regular, even daily, testing gets possible.

Nevertheless, the approach focuses on necessary features for scenarios and not to provide satisfactory test coverage. Hence, other test approaches have to be used if more than integration, consistency and focusing on function are wished.

VII. SUMMARY AND CONCLUSION

The described overall methodology describes the formal process of the scenario-based design approach adopted by the Daidalos consortium. It shows the way from the overall objectives, the scenario description to the drill down process down to the system integration and testing. The decision to go for a scenario based design approach which provides a degree of freedom for requirements extraction provides the required input in order to finally design the detailed architecture which finally will guide the technical work. The key goal of the architecture is then to fulfill exclusively the requirements coming from these requirements extraction.

Our approach could be possibly improved by real end-user evaluations and interviews. However, while it is certainly true,

that a scenario-based approach as such does not make the approach user-centric – in a context such as the Integrated Project Daidalos integrating virtually hundreds of technical subtasks, the take-off from human-oriented narratives and scenarios and to force technology to support such scenarios, is certainly a big step towards real user centrality.

Experiences from the first iteration loop of this overall approach has been deployed in the first phase of Daidalos and has shown, that scenarios – acting a red line – towards and integration and testing of the overall concept is very beneficial in the final integration and demonstration phase. The presented report digested already the experiences from the first phase of the Daidalos project which can be considered as first iteration loop.

ACKNOWLEDGMENT

The work presented in this paper was partially funded by the FP6 EU project "Daidalos" (Designing Advanced network Interfaces for the Delivery and Administration of Location independent, Optimised personal Services), and synthesizes the contributions of the Daidalos 1 and Daidalos 2 consortia.

REFERENCES

- [1] Peter Coad, Eric Lefebure, Jeff de Luca, "Java Modeling in Color with UML: Enterprise Components and Process" New Jersey: Prentice Hall, 1999, ch. 6.
- [2] Jens Grabowski, Dieter Hogrefe, György Réthy, Ina Schieferdecker, Anthony Wiles, Colin Willcock, "An introduction into the testing and test control notation (TTCN-3)", Computer Networks, Volume 42, Issue 3, Elsevier, Amsterdam, Juni 2003, 375-403.
- [3] Ina Schieferdecker, Zhen Ru Dai, Jens Grabowski, Axel Rennoch, "The UML 2.0 Testing Profile and its Relation to TTCN-3" Testing of Communicating Systems (Editors: D. Hogrefe, A. Wiles). Proceedings of the 15th IFIP International Conference on Testing of Communicating Systems (TestCom2003), Sophia Antipolis, France, May 2003.
- [4] Cem Kaner, "The power of 'What If...' and nine ways to fuel your imagination: Cem Kaner on scenario testing." Software Testing and Quality Engineering Magazine, Vol. 5, Issue 5 (Sep/Oct), p. 16-22, 2003. <http://www.kaner.com/pdfs/ScenarioSTQE.pdf> (Original, more complete version <http://www.kaner.com/pdfs/ScenarioIntroVer4.pdf>).
- [5] Cem Kaner, "The Ongoing Revolution in Software Testing" Software Test & Performance Conference, Baltimore, MD, December 7-9, 2004.
- [6] Johannes Link, "Unit Testing in Java: How Tests Drive the Code", Morgan Kaufmann, April 2003., ISBN 1558608680
- [7] Stephen R. Palmer, John M. Felsing, "A Practical Guide to Feature Driven Development". New Jersey: Prentice Hall, 2002, ISBN 0130676152.