

Approaches for Architecture Definition

TERJE JENSEN



Terje Jensen is Research Manager in Telenor Group Business Development and Research and Senior Researcher at Q2S/NTNU

A technical architecture is set out to fill specific purposes. Although several expectations may be common or similar across several operations or areas, there would likely exist a number of conditions that are specific for the actual area looked at. Hence, it becomes a fine art to balance the generality and the specificity. Still the experience on elaborating technical architectures can be carried across different areas and operations.

This paper elaborates on methods used for defining architectures within the telecom industry. The method for defining a technical architecture should start out by capturing relevant requirements that are to be complied with. Then, one must specify how to group the functions needed to meet those requirements. Finally, reflections on the resulting architecture should be done to check that it really complies with the requirements, look for potential improvements and capture learning from the activity.

1 Introduction

Two constants in today's world are changes and uncertainties. Some see these as triggers for timely improvements to be swiftly incorporated in the evolution process. Others may consider them as tedious annoyances in a fine-tuned operation. In fact, one claim could be that there is no point at all in defining architectures as, almost for certain, the future will not become exactly as predicted. And should it by some coincidence correspond with what is going to happen, then we could just wait for a while and then the situation will change. Others may state that it is not worth while coming up with idealistic plans as we have to go into the shops to buy the actual systems to be found there anyway.

It is fair to say that these claims miss the main objectives of carrying out architecture work. Besides being incorrect, the claims neglect the benefits following from a systematic evaluation of a company's surroundings and future options. Then, based on those observations, we will figure out how to arrange the company's production means to come out with a sustainable profit.

So, how to observe the changes? A key point is to make a reference in order to capture and categorize the changes and their impacts. For this purpose a *reference architecture* is fruitful. It allows

- Evaluating the changes and seeing which areas are impacted;
- Being able to define a set of goals;
- Comparing today's production means to identify inefficiencies and weak points.

Without a common reference, one may experience quite lengthy discussions and evaluations without meaningful progress. This is just to see that one actually has been discussing different subjects (although using same names) or actually the same subject (while using different names).

The reference architecture is to resolve the matter of names of areas and how the different areas are related. This will then also define the arena for mixing and matching components (or modules/systems). Hence, a common overall architecture may be defined, although local implementations may vary. In particular this is relevant for any multi-national provider¹⁾ as to gain from the scale and scope effect. That is, this allows *reuse of common modules and competence*, while still allowing for *local adaptations*.

The paper starts with an overview of the process of elaborating an architecture (Chapter 1). Some key steps are then further addressed in Chapters 3 to 6, being scope/perspectives, requirements, sub-systems and interfaces, respectively. A key point is to go back to the starting point again to refine the results and check that relevant concerns are effectively complied with (Chapter 7). Some examples of architectures are collected in Chapter 8.

2 Architecture Elaboration

2.1 Basic Steps

Basic steps for elaborating a reference architecture are depicted in Figure 1. Note that these may differ depending on the actual case at hand. A key point is to return to the scope and requirements in order to look for improved descriptions. A main point of the

¹⁾ This can be generalized to any multi-instance operation, where operations of similar types are realized.

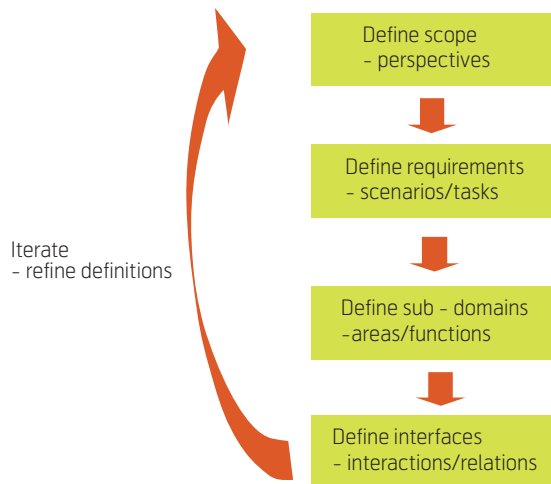


Figure 1 Basic steps in defining an architecture

experience from working with architectures is being able to capture the learning during the iterations. This learning also provides a flying start when the next architecture task comes along.

Note that different flavours of an architecture are requested, such as functional, data, physical, geographical, etc. [Jens09t].

These steps are further elaborated in subsequent chapters. The main result is to arrive at an architecture that could be applied for different purposes. Typically, two variations are requested:

- *Reference architecture* describing how system areas are broken down into smaller areas/subsystems and how they are related. This also defines the main terms/names of the areas.
- *Target architecture* defining how the reference architecture is to be populated.

Basically, the target refers to the strategy for how the relevant systems are to evolve. The reference architecture can be applied on today's, yesterday's and tomorrow's system areas. In that respect, it is needed for evaluating migration plans. These can be related to ensure a holistic and coherent set of system plans.

Conducting the iteration is related to going back to the starting point. This is for example to reflect on whether or not the results actually fit with the initial intention. Validation and verification are two key activities. Validation is set out to answer the question *Are we building the right 'product'?*, while verification wants to answer the question *Are we building the 'product' right?*

2.2 Objectives and Benefits

Motivation for defining and maintaining a reference architecture (adapted from [SPUG09]):

- *Separation of concerns:* Any good reference architecture uses components which are built with the separation of concern principle. This separation of concerns means you can change one component with zero to minimal impact on other components. As a result, you have a flexible and extensible infrastructure.
- *Risk mitigation:* In case an implementation and deployment is done without a reference architecture for guidance the learning is of less value. So, if a new project in the same domain needs to be done, a reference architecture will guide the implementation of key concepts and capabilities. The risks would then be mitigated because of a proven architecture foundation that can be re-used and adopted to the current project needs.
- *Cost reduction:* Since the creation of the architecture does not need to start from scratch, solution development costs are reduced. Usually, critical architecture decisions require time, several rounds of requirements discussions, potential options under given considerations, and the like. A lot of this time and effort can be saved by using a reference architecture as guidance and thus reduce costs.
- *Simplified decision making:* The business view on a reference architecture outlines what benefits can be derived by selecting a solution based on the reference architecture.
- *Improved deployment time line:* The description of the reference architecture also outlines key principles, architecture decisions, deployment scenarios and guidance for developing a solution. It provides examples of architecture building blocks and components that can assist in the selection of systems/products and interoperability requirements. This improves the overall timeline for the deployment of the solution.

From an outside, or external viewpoint, the main objectives of defining a reference architecture are to:

- Identify reference points that form key interconnection interfaces, access interfaces or appliance interfaces in a configuration involving a set of providers of services, networks and appliances;
- Identify the set of standards that could be applied at each key reference point;

- Identify key standards development organization industry consortia and other guidance that are relevant for the integration and implementation phases.

This implies:

- Classifying services by type
- Identifying services that can be carried across interfaces
- Identifying end points (reference points) for service delivery
- Classifying interfaces by type
- Identifying necessary data
- Accommodating a profile of all protocols involved, either directly or indirectly, at a given interface.

In summary (from [Y.110]), a reference architecture will:

- Provide a means of checking for completeness of a solution;
- Facilitate the development of common solutions;
- Facilitate comparison amongst solutions;
- Provide a catalogue of standardized solutions to avoid unnecessary reinvention;
- Help identify gaps in the standards repertoire;
- Identify joint interests amongst standards development organizations and areas where collaboration is required;
- Facilitate the investigation of inter-relationships between all elements depicted in a given reference configuration.

Example based on the ITU-T Global Information Infrastructure (GII) project and the Next Generation Network (NGN) architecture. See [Jens09n] for further description of the NGN architecture.

As quoted from [Y.110], "the objective of the GII is to ensure that each citizen may eventually gain access to the information society." This can be enabled by the interoperability of networks, information processing systems and applications. These objectives will be served by the following core principles of the GII, which will

- Promote fair competition
- Encourage private investment
- Define an adaptable regulatory framework
- Provide open access to network

while

- Ensuring universal provision of and access to services
- Promoting equality of opportunity to the citizens
- Promoting diversity of content, including cultural and linguistic diversity
- Recognizing the necessity of worldwide cooperation with particular attention to less developed countries

These principles will apply to the GII by means of

- Promoting interconnectivity and interoperability
- Developing global markets for network services and applications
- Ensuring privacy and data security
- Protecting intellectual property rights
- Cooperating in R&D and in the development of new applications
- Monitoring of the social and societal implications of the information society

3 Scope and Perspectives

There are different viewpoints on a system. One example is depicted in Figure 2. This can also be compared with the different business areas interacting with the technical architecture, see [Jens09b].

The perspectives are also related to the level of detail chosen to describe the architecture. This could also be related to the different abstraction levels and the dif-

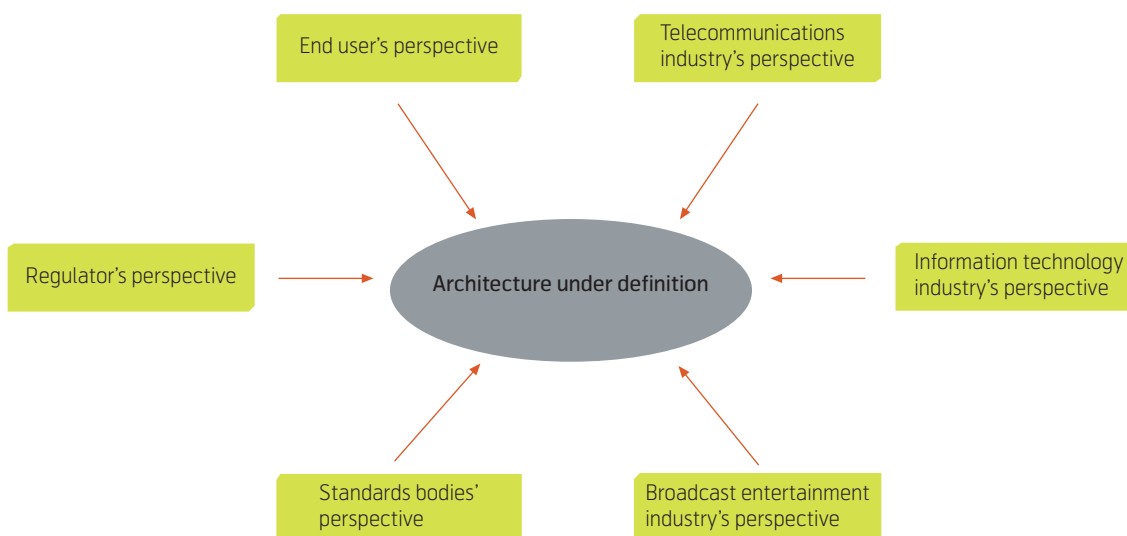


Figure 2 Examples of different perspectives to be taken when deriving requirements and principles of an architecture (adapted from [Y.110])

ferent implementation levels (eg. applications, systems and platforms). As one example, the platform concept (eg. [Sang07]) encapsulates the notion of reuse as a family of solutions that share a set of common features (the elements of the platform). Often we associate the notion of platform to a set of potential solutions of a design problem. We then need to capture the process of mapping functionality (what the system is supposed to do) to the platform elements that will be used to build a platform instance (how the system does what it is supposed to do).

Iteration on different levels of the systems, including applications and platforms, is part of the overall architecture definition to evaluate that the solution in fact meets all the requirements and is made in an effective manner.

Selecting the proper perspectives should answer questions like,

*Who are you? – who and where do you want to be?
– and what steps should be taken to get there?*
as observed from the relevant perspectives.

The architecture is to provide a framework of terms, domains and interfaces. This gives a reference for further detailing as well as providing a target. In that respect there is a grey area between the architecture and the implementation.

One example of the different perspectives is given in [Q.1762] related to objectives of fixed mobile convergence. The objectives are divided into three categories:

- *General objectives:*
 - Seamless service operation from the end-user and service provider perspectives across heterogeneous fixed networks and mobile networks subject to any limitations imposed by the characteristics of the particular access technology being used.
 - Generalized mobility is supported (ie. terminal device mobility, user mobility and session mobility). For a given scenario, different levels of mobility may be needed.
 - Ubiquity of service availability where the end users can enjoy virtually any application, from any location, on any terminal device subject to any limitations imposed by the characteristics of the particular access technologies and terminal devices being used, given that the service has been subscribed to.

- Support of multiple user identifier and authentication/authorization mechanisms.
- Objectives seen from *end users' perspective:*
 - Converged services that function consistently independent of the access technology or terminal device being used with associated, reduced complexity of telecommunications operation and subscription.
 - Convenient access to and usage of a wide range of services of multiple service providers through technology that enables easy configuration, allowing an end user to obtain only one bill if so desired by this end user, even when multiple service providers are involved.
 - Seamless service experiences, whenever desired and wherever the end user may be, limited only by the capabilities of the access technology and the terminal device being used, and with the expectation that the end user will have access to multiple terminal devices with different ranges of capabilities.
- Objectives seen from *service providers' perspective:*
 - Ability to offer a wide range of services.
 - Simplified network deployment and operation through the ability to offer access and terminal device independent services, as opposed to having to provide multiple access-specific and terminal device-specific instances of the same service.
 - Ability to reuse the fixed-line assets from traditional fixed service providers and mixed service providers in a Fixed Mobile Convergence (FMC) context.
 - Ability to provide better coverage and QoS to end users compared to traditional mobile service providers in an FMC context.

4 On Elaborating Requirements

Providing an extensive list of requirements tends to inspire for a complex system. The whole task becomes rather tedious when the requirements are done together with the consideration of inter-system relations and embedded functionality of individual systems. Hence, in effect the results may become:

- Heterogeneity and complex platforms and systems; that is a direct consequence of the requirements and interconnected systems.

- Application complexity; eg. due to the fact that systems should be customized by reconfiguration and/or by software programmability.
- Integration complexity; as a common approach to dealing with complexity is to decompose the system into subsystems, the task of integrating subsystems increases.

An overall requirement of *reusing subsystems/components* also implies that a challenge in developing solutions is to combine modules that have already been pre-designed or designed independently. Unless there is an overall understanding of the interplay of the subsystems and of the difficulties encountered when integrating complex parts, the integration tasks would become overwhelming.

Requirements have to be elaborated for the specific situation at hand. There are however several examples found in the literature, in particular used when defining standards.

5 Subdivisions/Subsystems

A user may have varying insight into the level of detail. This could even vary for the different areas such as who are the other users, applications and service components, see Figure 3. These relate to the conceptual overview as presented in [Y.110], see Figure 4. Four basic elements are shown:

- Persons that create, produce, use and operate information;
- Information appliances that are used to store, process, and allow access to information;

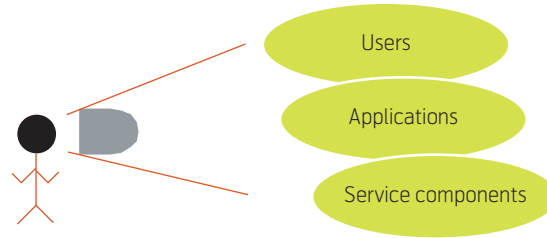


Figure 3 A user may have different insight into the level of detail of a system

- Communications infrastructure which transports the information between geographically separated information appliances;
- Information which includes application software such as home shopping systems, games, etc. This includes any video, audio, textual and graphical information that may be converted from an existing medium into electronic form.

An architectural model is realised using a set of interconnection of computational and communication components. Each component is characterized by the relevant set of parameters. Mappings must be defined between the different models to ensure that every functionality has been captured.

Based on the overall requirements different roles could be identified. This is also referred to as an *enterprise model* in some cases. The purpose of the enterprise model is to identify interfaces that are likely to be of general commercial importance. The roles have to be identified to describe a reasonably well-defined (business) activity. A role is unlikely to be further subdivided between several players. A role should also have a reasonably long time of existence. Interfaces surrounding a role must persist for some

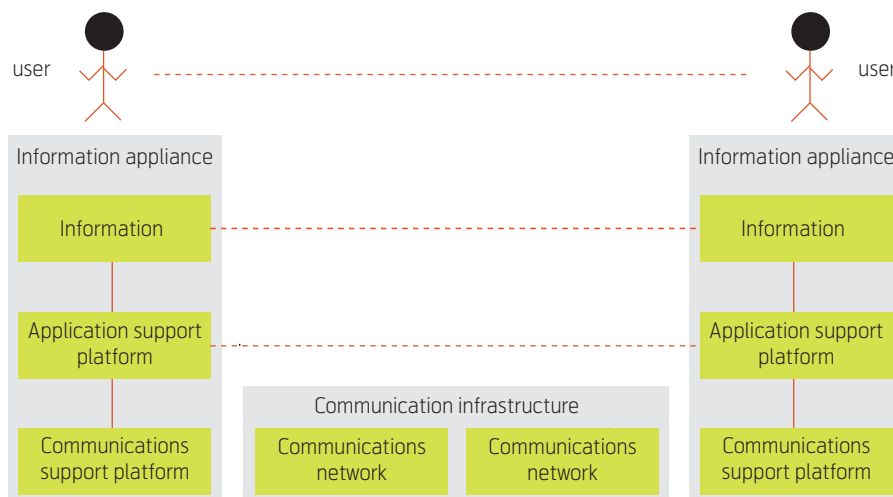


Figure 4 Conceptual overview of areas (adapted from [Y.110])

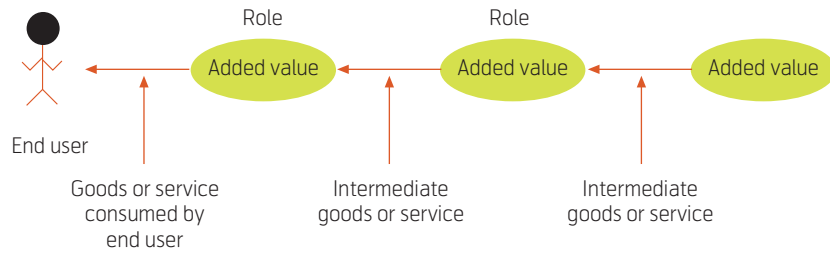


Figure 5 Enterprise model depicting roles and flows of goods/services in a value chain (adapted from [Y.110])

time in order for other roles to successfully interact. Several players may choose to take on the same role in case of competition.

A role can be thought of as adding value to the various inputs it acquires from suppliers. The role then passes its output to its users. The roles are typically linked to form a value chain, see Figure 5. Further details on these roles and relationships are shown in Figure 6.

The result will often be a distributed platform on which information applications and services can be supported. In essence it may provide the means by which an application and its users can be distributed

without being aware of the nature of the distribution. An application will be invoked when requests are sent to it. The platform also supports messaging to and from the applications and between components of an application. It will further include capabilities to support directory service, navigation, security, payments, browsing and searching, etc. These are features to support communication between components.

In order to offer services or supply an application, a role must bring together a number of resources and package them into a service that is applicable to its clients. Each resource brought in, or operated by a role, may well be supplied by another role and therefore be the service of another role.

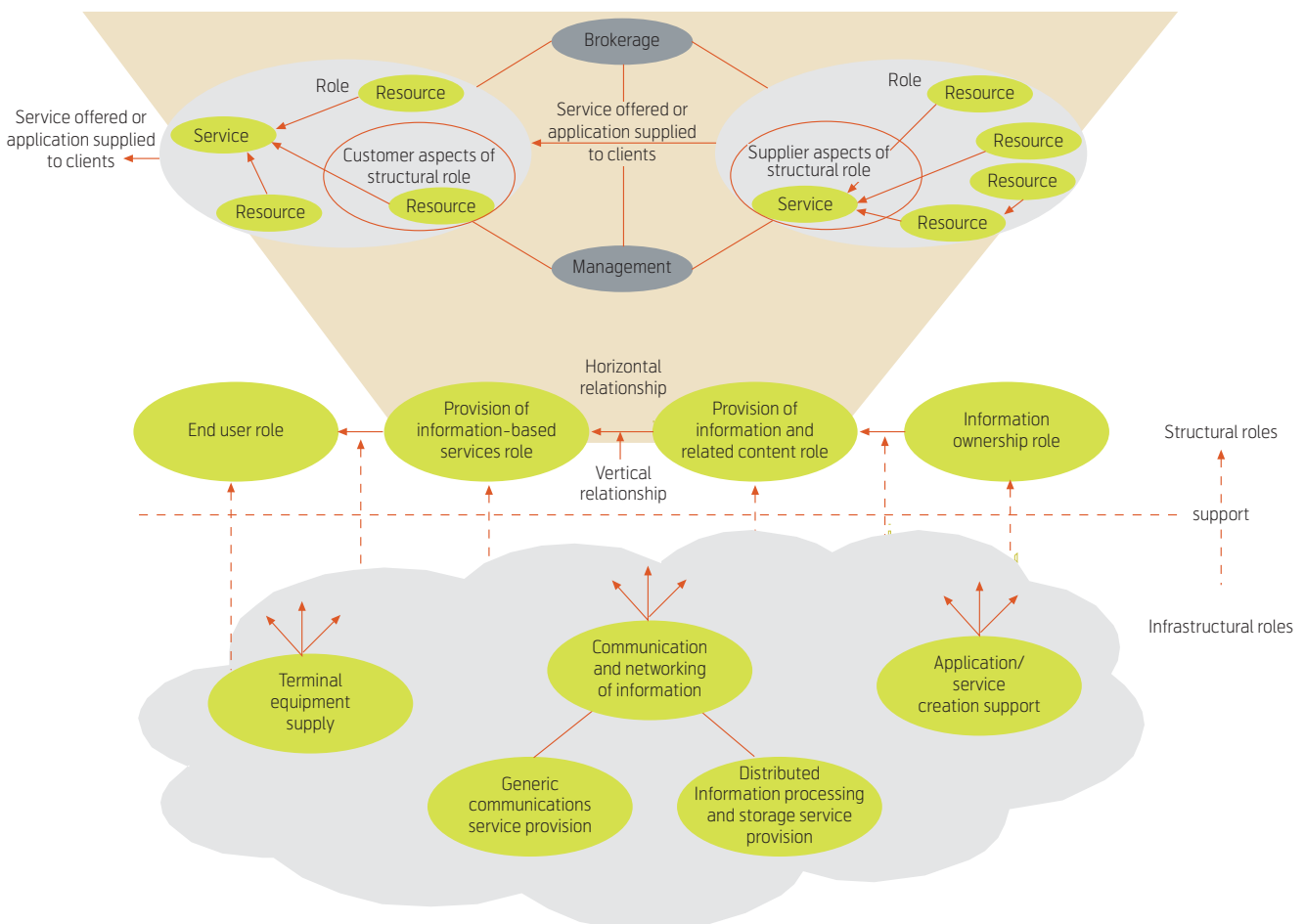


Figure 6 Roles and relationships (adapted from [Y.110])

As a role adds value, it can package together services or applications from other roles and either build them into completely new services or applications or simply provide a packaging service.

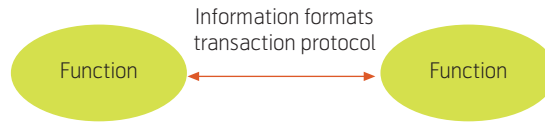


Figure 7 Functional model (from [Y.110])

6 Interfaces

An *interface* is typically understood to be the common boundary between two associated systems (eg. ITU-T Rec. I.112).

A system could be examined by the interfaces it provides (a 'black box' approach). This is also referred to as the reference point method [Gros02]. A major motivation for this is to decompose a system into components in order to make it less complex. Overall, the system should then become more manageable as each of the components could be further detailed. A component is related to other systems through a set of interfaces.

A functional model can be defined when the roles and the structure have been made. According to [Y.110] the functional model is an abstract description of a system and is developed in such a way that it is independent of any implementation of the system. The main purpose is to allow:

- Freedom in method of implementation without affecting the operation of the overall system;

- Large scale functional integration inside one equipment or software module while retaining a manageable and callable description of the equipment or software module;
- The dynamic creation of services which can be tailored to the needs of clients.

Principles of a functional model, see Figure 7, are quite straightforward.

For some standardisation work there is a distinction between an interface and a reference point. For an interface, only the interaction itself is important; which functions that interact is not specified by the interface. Hence, one may say that a given function has an interface without being specific on the function(s) at the other end.

For a *reference point*, however, both ends are specified. A reference point explicitly states which functional elements that interact. A reference point may also include a number of interfaces.

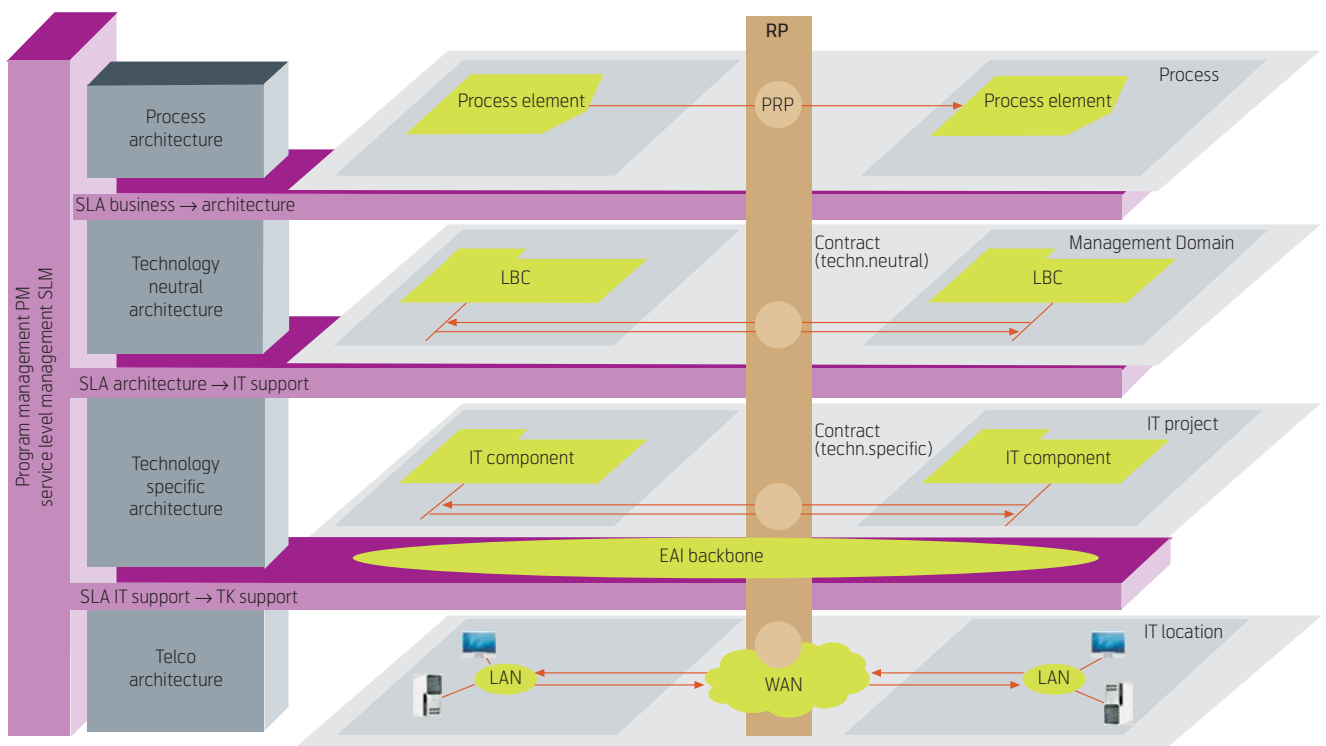


Figure 8 Interfaces between different system areas to define components for further refinements

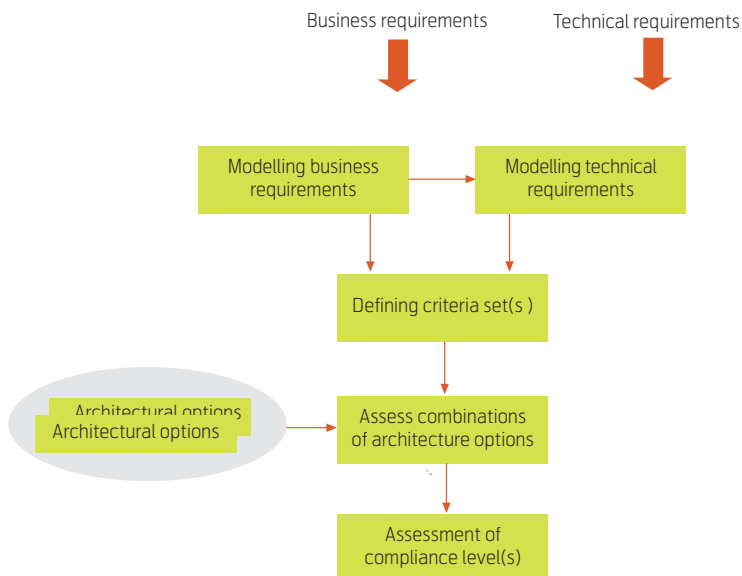


Figure 9 Outline of architecture compliance

As argued in [Gros02] the definition of proper reference points is quite beneficial when addressing the scope of an operation including networks, service platforms, support systems and processes, ref. Figure 8. In that illustration, reference points have been introduced between different layers. Naturally, several sets of interfaces would be present between the different layers. The reference points manifest the decoupling into subsystems and may be defined along several dimensions.

7 Next Iteration – go Back to Refine

7.1 Assuring Architecture Compliance

A set of criteria is needed to derive or evaluate different architectural options. Then, different approaches might be applied for evaluating to what extent the criteria are met. An overall flow diagram may be given as in Figure 9.

Each of these steps are treated above and in accompanying papers (eg. [Jens09t], [Jens09b]). Besides the

linear flow indicated in Figure 9, there may be several iterations. In particular the assessment step will imply looking into different solutions, returning to requirements, and potentially introduce improvements. The learning brought between activities (not included in Figure 9) will also be of vital importance.

7.2 Implementation of Components

The final step is to define the implementation model. An implementation model shows which functions are implemented in which equipment. It also identifies all the protocols which pass across an interface between components, see Figure 10. Commonly, formal standardization would end with a functional model as this should be sufficient to guarantee interoperability between different equipment.

In the NGN architecture (see [Jens09n]) the infrastructural service components can be divided into:

- *Baseware*: i) transport of information, ii) processing and storage, and iii) control and management functions;
- *Middleware*: i) service packaging and cooperation, ii) middleware support, application/service creation, and iii) interworking.

In several contexts the term platform occurs when discussing implementation. A first challenge is that different definitions of platform exist. For example,

- A platform is a flexible system allowing for customization adapted to particular applications by programming one or more of the subsystems;
- A platform is composed of the definition of instruction sets/primitives and interfaces;
- A platform exposes a set of capabilities that can be utilized by applications and used to swiftly develop new applications;

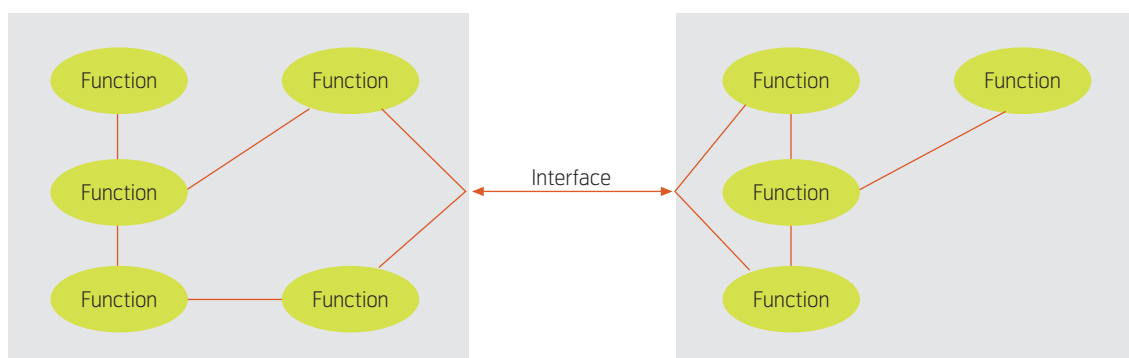


Figure 10 Illustration of implementation model (from [Y.110])

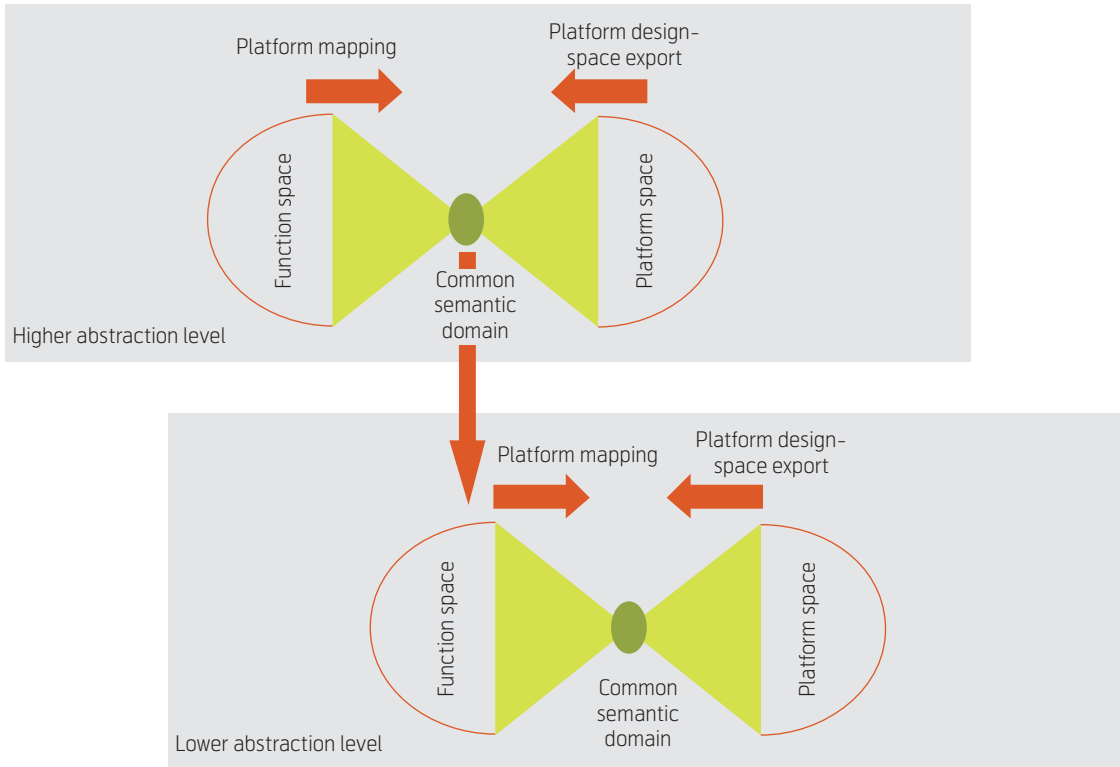


Figure 11 Mapping between functional space and platform space as input to next level of refinement (adapted from [Sang07])

- A platform is defined to be a library of components that can be assembled to generate a design at a given level of abstraction.

According to [Sang07] the principles of *platform design* are to start at the highest level of abstraction. This hides unnecessary details of implementation and only summarises the important parameters in an abstract model. A sequence of refinement steps are carried out to go from the highest level of abstraction to more details by further limiting the design space to a set of components. A platform at a given abstraction level contains both computational units and communication units. Different methods can be used to represent and further refine the communication and the computation units.

A platform instance is the set of components that is selected from the library (the platform class) and whose parameters are set. In the case of virtual components, the parameters are set by the requirements rather than by the implementation. Then they have to be considered as constraints for the next level of refinement. These principles can be compared with the baseware versus middleware, or the application versus system components, as described earlier.

This process is essentially carried out by steps of refinements. Basically two main classes are:

- Top-down: Map an instance of the functionality of the design into an instance of the platform and propagate constraints;
- Bottom-up: Build a platform by choosing the components of the library that characterizes it and an associated performance abstraction.

Both can be applied giving this mapping, see Figure 11. Then the mapping result can be used as input to the next level of refining. Some refers to this as the ‘fractal nature of design’ (ie. repeating patterns on a more granular level).

The mapping process can be interpreted as functionality at a lower level of abstraction where a new set of components, interconnects and composition rules are identified. To progress in the design, we have to map the new functionality to the new set of components. In case the previous step used components that were fully instantiated, that part of the design is considered concluded and the mapping process involves only the parts that have not been fully specified.

The definition of the platform levels allows us also to develop synthesis and verification tools that would have been challenging otherwise. That is, the tools could operate on a given abstraction level.

7.3 Evaluation of Architectures

Selected criteria to evaluate architectures are:

- *Usability*. For example, related to how a person can apply the architecture to make actual decisions.
- *Automation*. Commonly, as many functions as possible should be automated, reducing the human effort to control and analyse the information and network state. This is even stronger for a larger system portfolio.
- *Scalability*. The architecture must allow systems to scale well when the number of instances, nodes, links, traffic flows, subscribers, etc. grows.
- *Stability*. The architecture must include mechanisms for bringing the system portfolio back to a stable operational condition after being exposed to excessive ‘forces’.
- *Flexibility*. The architecture should be flexible both in terms of the optimisation policy and the scope. An example of scope is that additional classes should be considered. Another aspect of flexibility is that some subsystems could be enabled/disabled.
- *Visibility*. Mechanisms to collect information from the individual systems and analyse the data have to be present. These would then allow for presenting the operational conditions.
- *Simplicity*. An architecture should be as simple as possible, that is considered from the outside, not necessarily using simple internal algorithms. Simplicity is particularly important for the human user. It should also be simple to maintain the architecture, in particular regarding future modifications.
- *Efficiency*. As little demanding as possible is requested, in terms of processing and memory resources.
- *Dependability*. Mechanisms should be available to keep the level of dependability requested.
- *Survivability*. Recovering from a failure and maintaining the operation is requested, in particular for the more critical functions. Commonly, this requires that some redundancy is introduced and mechanisms for switching over between working instances are in place.
- *Extensibility*. It should be easy to extend an architecture, eg. when introducing new functions.

- *Interoperability*. Open standards should be used for the interfaces in order to simplify interoperation with other systems (outside the considered architecture, ref. discussion on scope in Chapter 3).
- *Security*. Means supporting integrity, information concealment, etc. have to be implemented.

As mentioned, some of these requirements may be mandatory while others are optional for a particular architecture. Several of these concerns will also be forwarded to the implementation phase.

8 Examples

8.1 ETSI 3-Stage Process

An example of the procedure for elaborating a technical architecture has been described by the European Telecommunications Standards Institute (ETSI). This is also referred to as a 3-stage standardization process:

- Stage 1 – *Service requirements*; eg. expressed by text and UML
- Stage 2 – *Functional view*; eg. expressed by text, HLMSCs, MSCs, simple SDL
- Stage 3 – *Detailed protocol*; eg. expressed by text, ASN.1, SDL

After these three stages have been completed, the results could be verified and validated according to correct behaviour. For this purpose, three new stages have been defined for testing, see Figure 12:

- Static summary; eg. expressed by tables
- Test purposes; eg. expressed by text
- Detailed tests; eg. expressed by TTCN

A set of protocols will be applied between different functional entities. It is essential to prevent dubious definition of the protocols to ensure interoperability between the entities. Typical parts of a protocol specification are:

- Explanatory text (not a tutorial)
- Context architecture (relations to other standards)
- System architecture
- Data definitions (separate syntax and encoding)
- Procedural definitions (dynamic behaviour); normal procedures and exception handling
- Profiles (separating normative/optional requirements)
- Static summary of capabilities

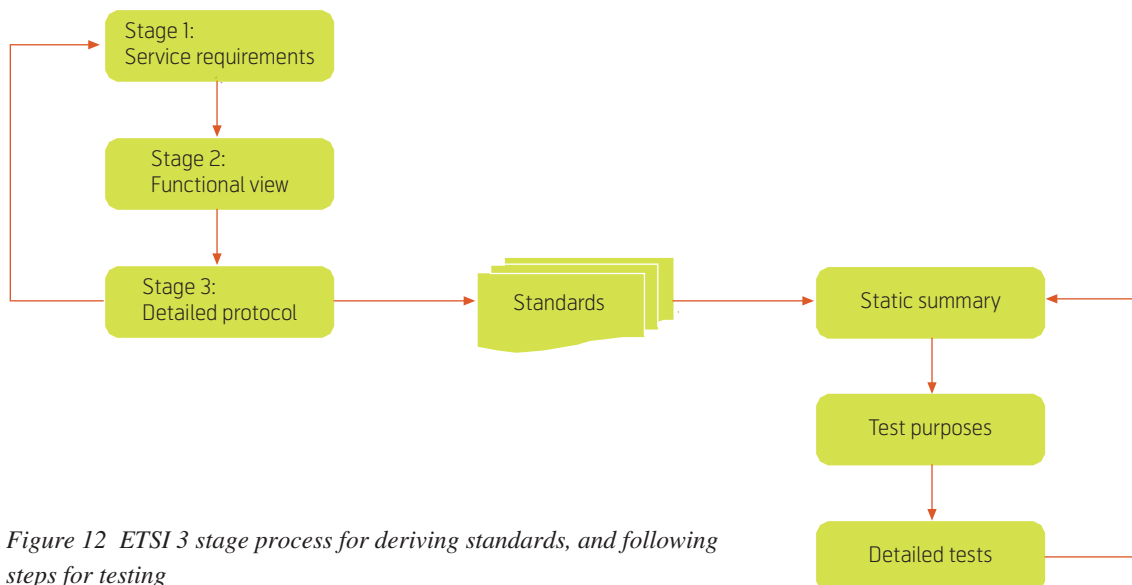


Figure 12 ETSI 3 stage process for deriving standards, and following steps for testing

8.2 Evolutionary, Testing-based

[GENI07] outlines a 5-stage approach for deriving and implementing an architecture:

- 1 *Mechanism design*. Specific proposals for new mechanisms, protocols and architectural components are brought forward and evaluated.
- 2 *Concept integration*. Some set of proposed mechanisms are selected and pulled together to make up a coherent overall proposal.
- 3 Preliminary evaluation and assessment. Some groups of evaluation criteria are:
 - a *Suitability for purpose*. Some initial set of requirements is reviewed to see how the design fits with the candidate system addressed.
 - b *Modularity*. Modularity allows that important capabilities are independent of implementation, independent evolution of critical system parts as system needs change, and independent evolution of different system elements as technology advances.
 - c *Modularity and management of effort* (*'separation of mechanism and policy'*). Concerned with isolation and management of non-technical aspects allowing the system to respond effectively to changes in the underlying social and economic environment in which it sits.
- 4 *Trial implementation*. Build a trial set-up and see how it works.
- 5 *Commercial roll-out*.

In practice the stages overlap and loop-back may later lead to revised design based on the insight gained. Moreover, it is believed that the starting point is some experience from a running system.

8.3 Open Distributed Processing

The Open Distributed Processing (ODP) reference model is put together by five viewpoints, corresponding languages, specifications of required functions, transparency prescriptions and framework for assessing system conformance. The viewpoints are:

- *Enterprise* viewpoint. Focuses on the purpose, scope and policies for the system.
- *Information* viewpoint. Focuses on the semantics of information and information processing.
- *Computational* viewpoint. Enables distribution through functional decomposition of the system into objects which interact at interfaces.
- *Engineering* viewpoint. Focuses on the mechanisms and functions required to support distributed interaction between objects in the system.
- *Technology* viewpoint. Focuses on the choice of technology in that system.

Hence, a viewpoint is a subdivision of a complete system. The objective is to describe the pieces of the system that are relevant to some particular area of analysis or design.

A telecom and Internet-based system will in most cases imply distributed implementations. This means that parts of the solution can be located on different

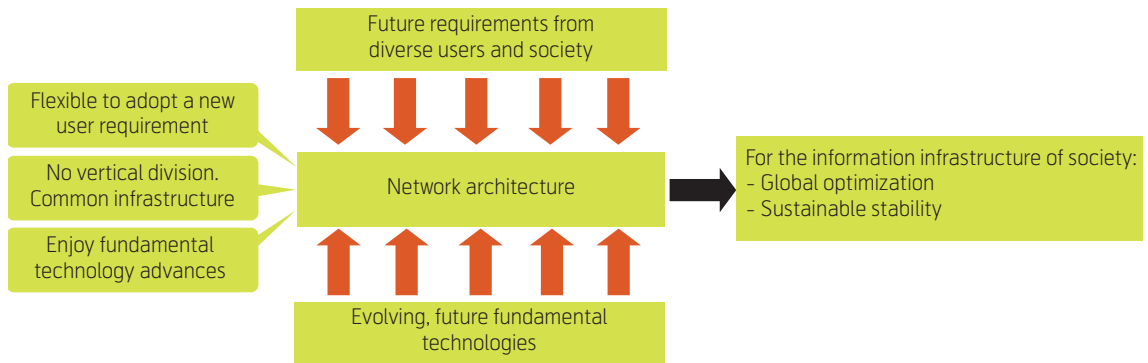


Figure 13 Position of the network architecture, clean slate (from [AKAR08])

physical locations. To avoid that applications need to manage the distribution, mechanisms are introduced in the architecture to hide this complexity. This is referred to as distribution transparencies and may include:

- *Access transparency.* Masks differences in data representation and invocation mechanisms to enable interworking between objects.
- *Failure transparency.* Masks, from an object/application, the failure and possible recovery of other objects (or itself), to enable fault tolerance.
- *Location transparency.* Masks the use of information about location in space when identifying and binding to interfaces.

- *Migration transparency.* Masks, from an object, the ability of a system to change the location of that object. Migration is often used to achieve load balancing and reduce latency.
- *Relocation transparency.* Masks relocation of an interface from other interfaces bound to it.
- *Replication transparency.* Masks the use of a group of mutually behaviourally compatible objects to support an interface. Replication is often used to enhance performance and availability.
- *Persistence transparency.* Masks, from an object, the deactivation and reactivation of other objects (or itself). Deactivation and reactivation are used to maintain the persistence of an object when a system is unable to provide it with processing, storage and communication functions continuously.
- *Transaction transparency.* Masks coordination of activities amongst a configuration of objects to achieve consistency.

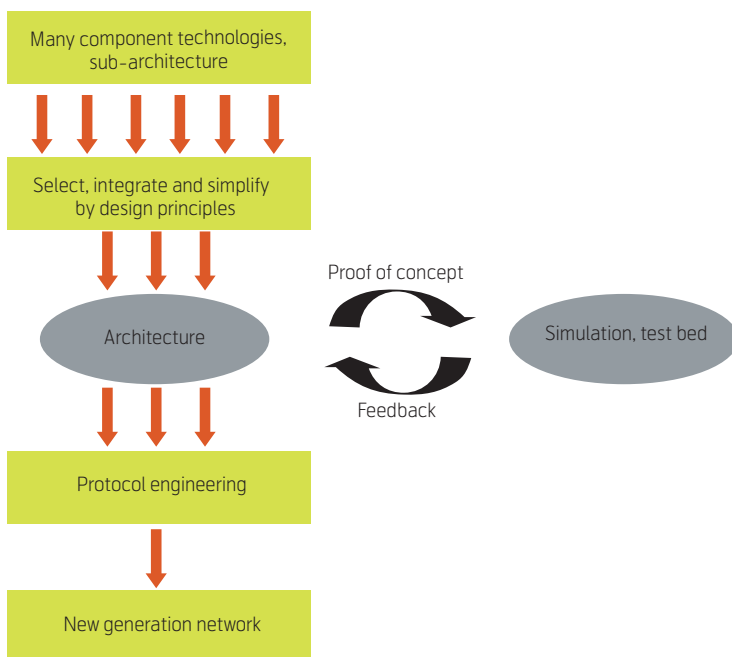


Figure 14 Approach for defining new generation network architecture (from [AKAR08])

8.4 AKARI Project – New Generation Internet

This section is based on [AKAR08]. A network architecture is described as a set of abstract design principles. These principles are used for making decisions when there are several options to choose from. The position of the network architecture is depicted in Figure 13.

Note that a so-called clean slate approach is applied; that is, not necessarily considering any limitations or other characteristics of today’s network. An overview of the approach is shown in Figure 14. The solutions are to be verified by experiments and simulations. On a theoretical level, the architecture is to perfectly match all requirements and providing complete guidance for the implementation.

9 Concluding Remarks

An objective of this paper is to discuss ways to elaborate architectures. As this is a broad area, it by no means provides full justice to every approach or all concerns to be taken. It does however introduce a simple step-wise approach that is often found in standardisation and other literature.

Key steps are to define; i) scope/perspectives, ii) corresponding requirements, iii) relevant sub-systems, and iv) appropriate interfaces. Then multiple iterations can be carried out for refinements and additional adjustments.

A major challenge when devising technical architectures is to balance generality and specificity. This is for the architecture to meet with its purpose, without being overloaded with details or unnecessary features. The balance is by some considered an art, although it will be improved through experience.

References

- [AKAR08] *AKARI ARCHITECTURE Design Project: New Generation Network Architecture AKARI Conceptual Design* (ver. 1.1). Online: <http://akari-project.nict.go.jp/eng/conceptdesign.htm>. 2007.
- [GENI07] *Global Environment for Network Innovations Research Plan*. GDD-06-28. Online: <http://groups.geni.net/geni/attachment/wiki/OldGPGDesignDocuments/GDD-06-28.pdf>. April 2007.
- [Gros02] Gross, W, Mayer, T, Zimmer, F, Herrmann, C. The Reference Point Method: Requirements-based ICT convergence solution Development. *Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE-02)*. IEEE, 2002.
- [Jens09b] Jensen, T. Business aspects impacting technical architecture. *Teletronikk*, 105 (2), 47-62, 2009. (this issue)
- [Jens09n] Jensen, T. Next Generation Network architecture. *Teletronikk*, 105 (2), 134-156, 2009. (this issue)
- [Jens09t] Jensen, T. Technical aspects to consider in an architecture. *Teletronikk*, 105 (2), 4-31, 2009. (this issue)
- [Q.1762] ITU. *Fixed-mobile convergence general requirements*. Geneva, International Telecommunication Union, Sept. 2007. (ITU-T Rec. Q.1762)
- [Sang07] Sangiovanni-Vicentelli, A. Quo Vadis, SLD? Reasoning About the Trends and Challenges of System level Design. *Proceedings of the IEEE*, 95 (3), 467-506, 2007.
- [SPUG09] Subscriber Profile User Group. *Reference Architecture Release 1.0*. February 2009. Available at: www.spugonline.com
- [Y.110] ITU. *Global Information Infrastructure principles and framework architecture*. Geneva, International Telecommunication Union, June 1998. (ITU-T Rec. Y.110)