Ericsson Review

The communications technology journal since 1924

2015 • 4

Design, architects, and complex communication systems: painting the bigger picture

May 13, 2015

ERICSSON

Design, architects, and complex communication systems: painting the bigger picture

The task of building, maintaining and developing communication systems is complex. The level of complexity rises as the number of stakeholders involved in creating these systems increases. As a result, vendors, system integrators, operators – and increasingly their business partners – need to communicate more. And so, besides understanding how their own systems work, modern designers and business developers need to grasp how other stakeholder systems work, and to have an appreciation of the various possible approaches to architecture design.

ULF OLSSON, TONI SILJAMÄKI, AND FRANCIS BORDELEAU

The ability to grasp complex structures can be greatly facilitated by using visualization tools. A common illustration approach enables modern system architects to share design concepts. Support tools that help designers to maintain, communicate and discuss structures are a fundamental part of modern systems architecture. This article presents Ericsson's methodology for developing such support tools.

Designing the best system

Experts who build and maintain very large systems are continuously looking for ways to increase productivity and improve quality. Raising the level of abstraction involved in system design is one way of doing this. It releases designers (and others) from the need to keep track of an ever-increasing number of details and dependencies – a number that rises exponentially with system size and complexity, product range and stakeholder count.

Model-based engineering has been used successfully for many years to achieve the best combination of compute power and design knowledge. To capture the structure of a system, this methodology uses a logical model of aggregation and dependencies – which are visualized in a graphical format. In this way, proposed models can be validated and modified easily.

Owing to the complexity of modern communication networks, systems architecture is often split into various domains, each with an assigned group of architects. In addition to designing and modeling their respective domains, these architects are responsible for ensuring that all of their peers have a common understanding of the domain interfaces and functional allocations.

Model-based engineering offers the level of person-to-person information transfer needed to design large, modern, complex systems efficiently. However, designing modern systems requires a toolset: one that is capable of being adapted to a wide range of constantly evolving demands, posed by many different stakeholders – including producers, systems maintainers, and the actual users of the designs.

The wanted output is a complete, coherent, consistent and revisable network description – one that enables the network to evolve in a controlled manner, address new challenges, and absorb new technologies.

But to create the best system, facilitating a mutual understanding among architects is only one key ingredient. Architects also need a set of companion tools – tools that can, for example, help them to validate proposed models, analyze the potential impact of suggested changes, detect inconsistencies, and ease the implementation process.

Support tools for software and hardware design have existed for decades, but, unfortunately, they do not address how to bridge the gap between the abstract representations used by designers to model the real world, and detailed ones, where every aspect of a process or a structure must be described in full to enable automation.

In the context of network programmability, the ability to bridge this gap becomes even more significant.

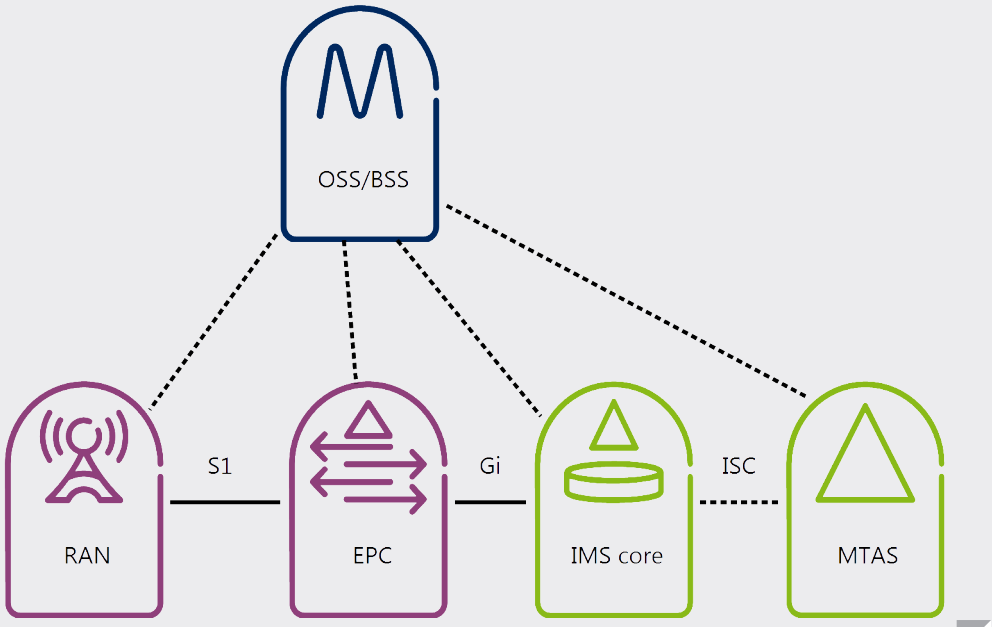
BOX A Terms and abbreviations

CSS	Cascading Style Sheets
DSL	domain-specific language
DSML	domain-specific modeling language
EGit	Git with Eclipse
EMF	Eclipse Modeling Framework
Git	open source control model
GMF	Graphical Modeling Framework
NWA DSL	network architecture DSL
OCL	Object Constraint Language
PaaS	platform as a service
SVG	Scalable Vector Graphics
SysML	Systems Modeling Language
UI	user interface
UML	Unified Modeling Language
Xtext	framework for developing programming languages

Identifying the capabilities that can be invoked, and determining how to control them from outside the network proper, becomes more challenging as the level of automation rises.

The approach we took to develop the Ericsson toolset demonstrates how open-source technology enables developments in modeling technology to be added as they become available. Collaboration – within the telecom industry, and with other industries faced with similar system design challenges – has been crucial factor in developing a toolset that can serve modern multi-stakeholder environments.

FIGURE 1 Top-level network diagram



Primary use case: modeling network architectures

But what is network architecture? One definition might be: the sum of all the components needed for operators to produce their services. Such a definition includes a number of aspects: functional descriptions, implementation components, products, topological architecture, as well as business and responsibility relationships.

Functional descriptions

A system comprises a set of logical functions and how they interrelate through logical interfaces. Deconstructing a system into its various elements facilitates an understanding of how the system works. Modern models use recursion to simplify systems into network elements and component parts to a point where a person with some system knowledge understands them.

Ideally, architecture design should be decoupled from implementation. In practice, variants of a system tend to exist to address the needs of different deployment contexts. Systems can, for example, be provided on a dedicated platform, through a cloud infrastructure, or using PaaS. In a sense, the high level architecture is the variant that is still valid after a system has been significantly re-implemented (for instance, after a change of platform technology).

Implementation components

Logical functions are a great way to explain what a system does, and to some degree how it does it. However, to deliver actual products, implementation components that correspond to logical functions are needed. Implementation components can be combined and reused to produce the behavior described by the logical functions. Sometimes, a direct correlation exists between logical functions and implementation components, but this relationship is typically many-to-many.

A logical function can be executed by several implementation components, although reusability goals suggest that this should only be the case when certain constraints, such as choice of execution environment, create a need for several implementations. An implementation component can be responsible for implementing several logical functions, and again, sound design principles suggest that this should be the case only when functions are tightly coupled.

Products

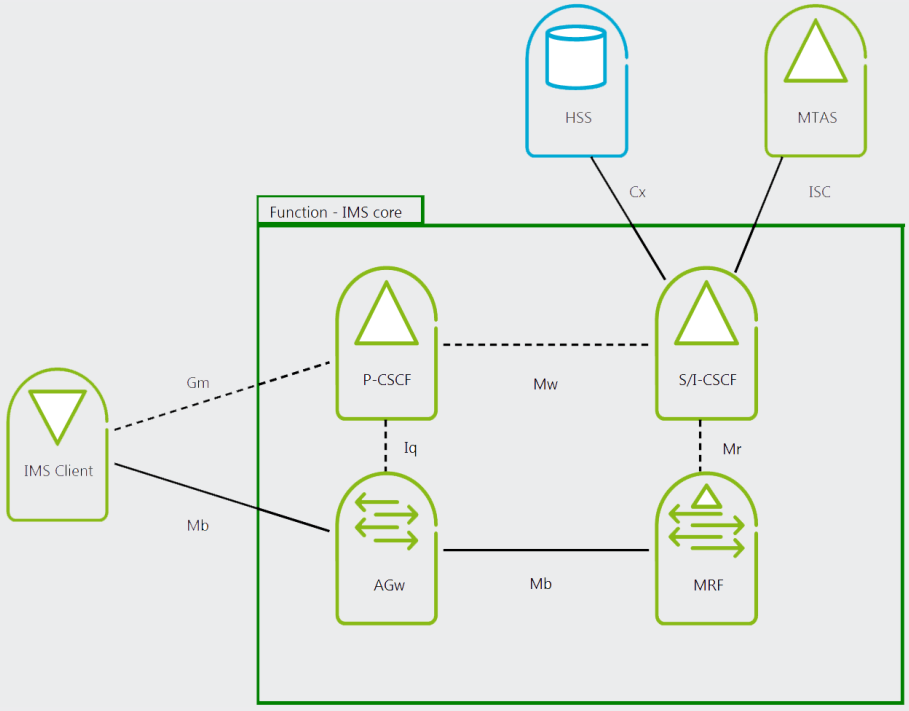
Products are aggregates of hardware and/or software components. Extending the model from logical functions through implementation components links the logical architecture – which does not change if the functional requirements are stable – to product life cycle management (including market adaptations), with its processes and tools. In a virtualized environment, products will, to a large degree, be implemented as prepackaged virtual machines (or increasingly using the container concept, as exemplified by Docker containers[1]), although optimized hardware and software components used in high-performance elements of a radio base station, for example, will continue to play a significant role in designing the best system.

Topological architecture

In telecoms, system topology – the ability to understand where functionality is best deployed – is vital for creating the best system. Telecoms is essentially motivated by the need to connect devices separated by distance. Mobile networks need to connect devices as they move about, and so systems need to be designed to cope with a range of traffic patterns. As a result, topology becomes highly relevant, especially when design parameters include latency, resilience, interconnect points, compute power or cost.

For virtualized scenarios, the location of a software component is not set at deployment time, but is instead a dynamic parameter determined

FIGURE 2 **Exploring a function**



by the operational system. Determining location is a complex cloud-management process, one that requires rapid system response and a high degree of automation. Modeling such system processes is essential to ensure realistic automation. For example, to meet low latency requirements, some network components may need to be deployed close to each other, while redundancy requirements may dictate that certain network components be kept apart.

Responsibility relationships

Systems need to be described in terms of the functionality each business entity (operator, partner, platform provider, device manufacturer, content provider or user) is responsible for. Such division of responsibilities is not necessarily limited to the relationships between separate legal entities, but can also be applied to units within the same legal entity or company. Key resources like access and core networks, computing infrastructure and databases must be protected. At the same time, the pace of business requires that new services can be developed and launched without having to wait for traditional system integration to be completed or for the massive amount of testing to be concluded. Network architecture modeling plays a key role in identifying the key interconnection points, highlighting the interfaces that can be defined and thus protected as exposable services.

Business relationships

The business relationship includes how workflows and commercial processes are described, how they touch, and how they help to define the logical functions in the system.

All aspects, from functional descriptions to business relationships, are linked. The complex job of establishing and maintaining these links is a critical factor behind the need for formal modeling and abstraction capabilities.

Perhaps the most important benefit of a formal model behind the graphical representation of a system is that it allows the architecture to evolve in a controlled way. The ability to analyze a model for completeness and consistency, and the capability to carry out impact analysis on proposed changes, are key factors in the need for model-based engineering.

Graphical representation of the system

Figure 1 shows a typical network architecture illustration. It conveys the main purpose of a system through a set of interconnected logical functions. Each function is portrayed as an icon, which conveys the function's primary purpose. Color-coding helps to support the association of functions to network areas, but color is also used to shift the viewer's focus to specific parts of the system. The elements shown in Figure 1 are examples of network areas – main groupings of functionality.

Besides functional elements, connections are a significant feature of network-architecture diagrams, as they illustrate the direct relationships between functional elements.

Figure 2 illustrates how a network element/function can be described in detail. The bounding box represents the function at a higher level of abstraction. Even though the elements outside the box are functions defined by other network areas, they are included as the information carried over the interface helps the reader understand the role of the function, and its relationships with the rest of the system. By adhering to the principle of need-to-know, context is not lost as the viewer drills down from one level to the next.

This need-to-know principle is fundamental to Ericsson's toolset, which can create a hierarchy of diagrams that show just what the reader needs to understand: in other words, a system that renders illustrations with the main structures and the correct level of detail without being cluttered with unnecessary information.

Details are conveyed through recursive decomposition of the logical functions into sub-functions – a concept that is illustrated in Figures 1 and 2. To illustrate the principle of recursive

architecture descent, Figure 3 shows how a logical function can be rendered.

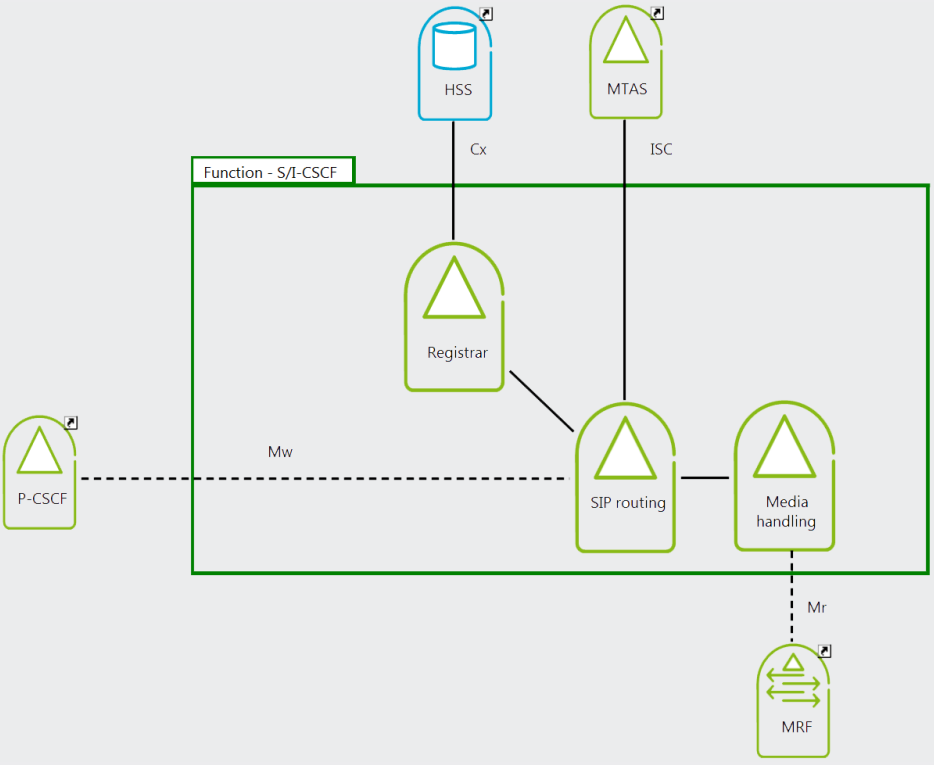
These architecture diagrams are the toolset's graphical description of the logical, multilevel structure of the network area (being described). However, formal representation is also required. Figure 4 shows a formal, tree, representation of (a part of) the same model.

The formal model may contain protocol definitions, which consist of signals and their parameters. Functional components can be associated with such protocols, and detail how they interact with each other. Sequence diagrams, like the one shown in Figure 5, are created using the sets of signals available in the protocols, and aid the understanding of how a system works by adding behavior to the architecture.

In line with good design principles – simplification where possible – recursive descent is also applied to sequences, so that several subsequences can be abstracted into a block, which can then be reused in higher-order sequences.

The goal is to create a toolset that provides a single, consistent and maintainable system description that captures the system's logical functionality, its hierarchical structure, its internal and external relations, and ultimately how it relates to implementation and deployment. In other words, the aim is to build a toolset that supports both human understanding and formal stringency.

FIGURE 3 Drilling down into a logical function



The fundamental enablers

The open source community Eclipse[2] is a good example of how collaboration across industries succeeds today. Initially, the aim for Eclipse was to create a platform for software tool developers. But the platform has since evolved into a sophisticated software system with a specialized support environment for creating a broad range of artifacts.

Eclipse

A basic framework for creating initial workbenches, Eclipse can be tailored, providing network architects with the right support to describe a system.

Eclipse Modeling Framework (EMF)

This framework includes a set of software tools that enable logical model structures to be created and managed. In other words, contained elements, their properties and their relationships can be modeled.

Graphical Modeling Framework (GMF)

This framework includes functionality that renders a model as a customizable graphic.

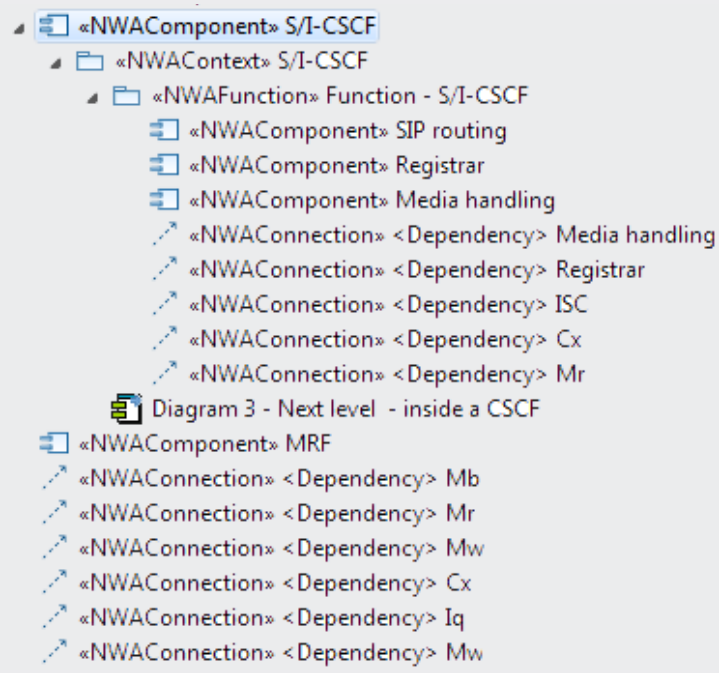
Papyrus

A standard-based modeling tool, Papyrus supports languages like UML and SysML, and is based on a set of Eclipse components, including UML2, GMF, EMF, OCL and Xtext. Designed from the ground up, Papyrus is an attractive implementation solution for the toolset, as it can be extended, specialized, and adapted to appear to the user as a domain-specific language (DSL), or more accurately as a domain-specific modeling language (DSML). This capability enables architects to work using modeling concepts that are natural to the context at hand.

Generic modeling languages – like UML – are powerful and flexible, as they can be applied to a wide range of modeling tasks. However, that flexibility is also one of their drawbacks, as the responsibility lies with the user to perform the mental translation from contextual to generic modeling concepts. By instead using a DSL, this responsibility can be shifted to the toolset, allowing architects to focus on the job at hand: creating the best architecture.

A key component of any design toolset is the ability to share modeling information across a large, and geographically distributed, team. One candidate open source toolset for managing versions, parallel development threads and distributed development is Git. While this toolset is included in the Ericsson solution, Git was primarily developed to support code development, and as such was initially text focused. Fortunately, significant work has been carried out over the past couple of years to make the compare-and-merge functionality model-aware, which is a vital component in the bigger picture.

FIGURE 4 **The formal model**



A domain-specific toolset

The Ericsson toolset for evolving network architecture – network architecture domain specific language (NWA DSL) – is based on Papyrus and supports key aspects of architecture design, such as advanced model validation, model and tool integrations, deployment analysis and validation, architectural exploration, variation points, and product line management. Essentially, the toolset includes the following components:

- UML2 profile – tailored to the network architecture (NWA) graphical modeling language (originally defined independently of tool implementation);
- style sheets (CSS files) – to govern how customized diagrams are rendered;
- graphics library – customized SVG shapes for visualizing different network functions in diagrams;
- palettes – showing the user what elements are available for building network architectures; and
- software – to implement the associated logic, including the additional UI menus needed for the NWA DSL.

However, providing the logic and rendering information is still not enough. To describe the perfect system, domain-specific properties that do not break the fundamental assumptions of Papyrus and UML are needed. This is where the stereotype mechanism of UML comes into play. By extending UML Components (defined in the UML standard) with stereotype designators, the CSS-based graphics rendering process picks up the stereotype and its associated properties to produce the graphical representation. The resulting diagrams that architects use to validate their ideas focus on essential information and use a graphical syntax that is meaningful, rather than the kind of generic syntax used by standard models like UML.

The Ericsson toolset could have been developed from scratch. However, by building directly on the semantic richness of UML, the resulting toolset benefits from years of development conducted by experts in the fields of modeling and tool implementation. In addition, integration with other DSLs based on UML2 profiles – both existing and future releases – becomes easier. In other words, the Ericsson toolset not only benefits from developments already delivered by the Eclipse modeling community, but, and perhaps more significantly, will continue to benefit as enhancements become available.

The open source approach: how and why?

Large software organizations, like Ericsson, have used model-based engineering as a key business differentiator in many different contexts. Today, modeling is used in a wide range of tasks, including software design, system design, information structure, network architecture, and the mapping of business processes. However, there are a few issues limiting the wider adoption of this approach by industry.

This lack of adoption by industry is mainly tool related. For example, model-based engineering has no proper support for DSML development and customization, and no capabilities to support key development areas like model-based collaborative development, testing, deployment on multicore, and model/tool integrations. These issues, together with the lack of evolution of commercial tools, lead to the conclusion that the traditional approach, based on proprietary technologies, has failed, and that a new solution based on open source is needed.

In light of this failure, Papyrus – an industrial-grade open-source modeling tool – provides the necessary basis to establish a new foundation for model-based engineering; a foundation based on collaboration among users, suppliers, and the research community.

Papyrus (and other related open source technologies such as EGit, EMF Compare, and UML2) has evolved to a new level of maturity, one that has enabled its use at industrial level. Collaboration is the key factor to ensure its continued development and more widespread adoption – not just with suppliers and independent developers but also with other companies.

Interestingly, the needs and requirements of many different companies, with different technology domains, are similar to the telecom domain – underlining the fact that large system development is highly generic. For example, the architecture design characteristics for person-to-person interaction are more or less the same for a telecoms application as they are in the control of electricity grids or remote patient monitoring.

Future development possibilities

Fundamentally, building the best network first requires the capability to model many different aspects of network architecture, and secondly, the ability to capture and maintain relationships among network elements.

The compelling aspect of building comprehensive system models is the ability to rapidly create a functional model of a customer system, with proposals for evolution and transformation paths. As such, the benefits of modern architecture modeling not only apply to a given technology domain, but are a fundamental enabler of collaboration with customers and partners.

To ensure the long-term evolution of the open source modeling solution and the development of a vibrant support community, Ericsson is actively supporting and developing industrial cooperation initiatives in this area. Ultimately, the ability to share ideas and solutions, and contribute to the open source community, are the key factors for successful open source modeling.

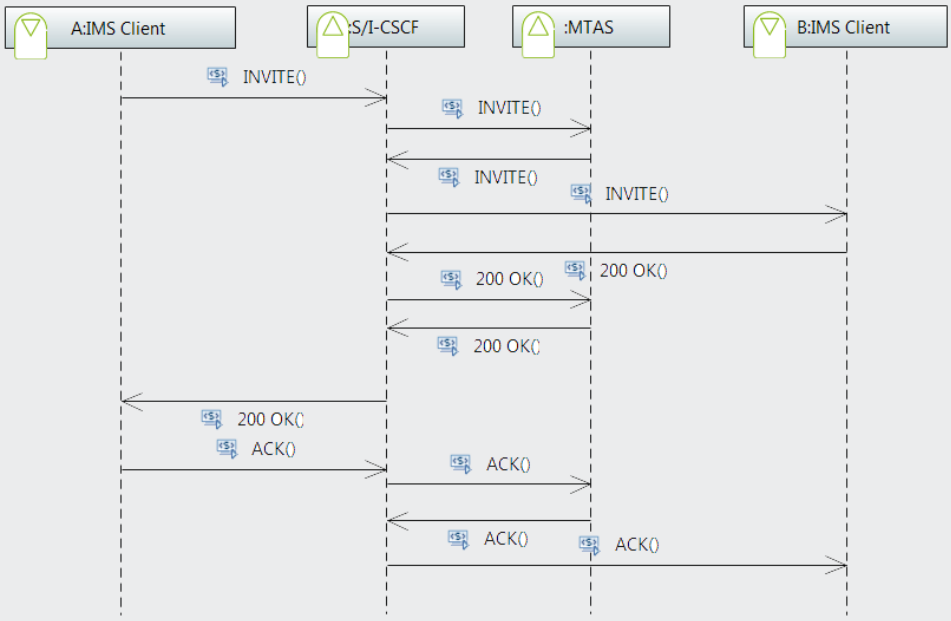
Conclusion

The model-based engineering strategy illustrated by the NWA DSL example lays the groundwork to fulfill the needs of network architecture modeling, which are:

- flexibility in graphical representation – to achieve the right level of abstraction;
- integration potential – efficiency across the development and integration chains;
- a basis in an open source strategy – promoting a community approach that can not only provide benefit to the telecoms industry, but also to adjacent industries experiencing similar architectural challenges;
- ease of use – to lower the threshold for architecture-level users; and
- efficient collaboration – to support the entire organization.

Applying a model-based engineering approach to network architecture design results in increased productivity and enhances the ability of all parties to understand the target system. The model-based approach ensures that the level of consistency, performance and adaptability needed by Ericsson and its customers is safeguarded as we progress deeper into the Networked Society.

FIGURE 5 Simple sequence diagram



References

1. Docker, What is Docker?, available at: https://www.docker.com/whatisdocker/
2. Ericsson, 2014, Presentation, UML or DSML?, available at: https://www.eclipsecon.org/europe2014/sites/default/files/slides/Ericsson_NWADSL_at_EclipseCon_Europe2014_0.pdf

Ericsson Review

To bring you the best of Ericsson's research world, our employees have been writing articles for Ericsson Review – our communications technology journal – since 1924. Today, Ericsson Review articles have a two-to five-year perspective, and our objective is to provide you with up-to-date insights on how things are shaping up for the Networked Society.

Address:
Ericsson
SE-164 83 Stockholm, Sweden
Phone: +46 8 7190000

Publishing:
Additional Ericsson Review material and articles are published on: www.ericsson.com/review. Use the RSS feed to stay informed of the latest updates.

Ericsson Technology Insights

All Ericsson Review articles are available on the Ericsson Technology Insights app available for Android and iOS devices. The link for your device is on the Ericsson Review website:www.ericsson.com/review. If you are viewing this digitally, you can:
download from Google Play or
download from the App Store

Publisher: Ulf Ewaldsson

Editorial board:
Joakim Cerwall, Stefan Dahlfort, Åsa Degermark, Deirdre P. Doyle, Björn Ekelund, Dan Fahrman, Anita Frisell, Jonas Högberg, Geoff Hollingworth, Patrick Jestin, Cenk Kirbas, Sara Kullman, Börje Lundwall, Hans Mickelsson, Ulf Olsson, Patrik Regårdh, Patrik Roséen, Gunnar Thrysin, and Tonny Uhlin.

Editor:
Deirdre P. Doyle
deirdre.doyle@jgcommunication.se

Subeditor:
Ian Nicholson, and Birgitte van den Muyzenberg

Art director and layout:
Carola Pilarz

Illustrations:
Claes-Göran Andersson

ISSN: 0014-0171

Volume: 92, 2015

Ulf Olsson

has a background in software architecture for large-scale distributed systems, ranging from military command and control to current and future telecommunications. He joined Ericsson in 1996, working mainly with packet-based systems like Packet PDC, GPRS, UMTS, CDMA2000 and IMS. He then moved on to systems architecture in areas like service exposure and analytics. He is currently a senior expert at Group Function Technology, focusing on overall system architecture issues including how to represent them formally and informally. He holds an M.Sc. in engineering physics from the KTH Royal Institute of Technology in Stockholm, Sweden.

Toni Siljamäki

has a background in modeling and software development for embedded systems in the Swedish defense industry. He joined Ericsson in 1997 to work on bridging the gap between hardware and software design disciplines, and held responsibility for Executable UML modeling support and model compiler development – transforming UML models into executable code in Erlang, Java, Plex-C and C for different platforms.
Since 2013, he has focused on basic core capability and usability improvements of Papyrus, with a special focus on DSML development and customization. He has also designed and developed the NWA DSL for Papyrus described in this article.

Francis Bordeleau

is product manager in the EITTE software design group at Ericsson. His primary focus is model-based engineering and modeling tools. In this role, he is responsible for defining product specifications and roadmaps, developing business cases, managing budgets, running open source initiatives, and collaborating with other companies, researchers, and academia. Before joining Ericsson in 2013, he was founder and CEO of Zeligsoft – a provider of domain-specific model-based engineering. He has held the position of Assistant Professor at the School of Computer Science of Carleton University, Ottawa, Canada. He holds a B.Sc. in mathematics from the Université de Montréal (1989), a Bachelor of computer science from the University of Quebec (1991), and a Master in computer science (1993) and Ph.D. in electrical engineering (1999) both from Carleton University.

Ericsson
SE-164 83 Stockholm, Sweden
Phone: + 46 10 719 0000

ISSN 0014-0171
284 23-3255 | Uen