IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4293

A Generic Framework for Defining Domain-Specific Models

Arnor Solberg and Jon Oldevik
SINTEF Telecom and Informatics, Norway
Tel: +47 22067983, Fax: +47 22067350, {arnor.solberg, jon.oldevik}@sintef.no

Audun Jensvoll
EDB4Tel., Norway, audun.jensvoll@edb4tel.com

ABSTRACT

How do you tailor a general-purpose system development methodology to appropriately fit the specific needs of your company and the actual domain or product-family you are working with? Moreover, how do you alter a general-purpose methodology to utilise the domain knowledge possessed by your company? This paper describes a generic framework for tailoring general-purpose, methodologies in order to deliver domain-specific models.

BACKGROUND AND INTRODUCTION

As a result of the widespread popularity of UML, many companies have invested in introducing a UML-based methodology. There are many general-purpose UML-based methodologies on the market today, among the most popular are UP[1], RUP[2], Catalysis[3] and Select perspective[4]. Typically these general-purpose methodologies do not immediately fulfil a company's need. That is why lots of consultants, researchers and others are in the business of helping companies to introduce these methodologies as well as customising the general-purpose methodology to be appropriate for the actual company and purpose. A common way of customising a general-purpose methodology is by removing, adding and/or merging defined tasks, phases, roles and models/artefacts based on different criteria such as domain, customers, market (e.g. in-house or off-the-shelf) and size of company. However, it does not seem to be any standard and formalised way of customising a methodology to produce domain-specific models.

We have for some time worked with customising methodologies to satisfy specific needs. Our customisation has been accomplished by taking a set of different OO-based methodologies (among others RUP, UP, OOram[5], Select Perspective, Catalysis, Open Process[6], as well as self-developed methodologies), methodology expertise and experience as input to a collaborative process with super-users (here users = developers that will use the methodology). By massaging this input through an iterative and incremental process in which we have analysed the company's need, existing methodology (or practice) in use within the company, company culture etc, the output has been a tailored methodology. Some results of this work have been the Business Object Methodology (BOM)[7] and the Magma methodology handbook[14]. Recently we have been working with Telenor **[The major Norwegian telecom-company (http:\\www.Telenor.com)]** and EDB4Tel **[Company offering administrative software-products for the telecom-industry (http://www.EDB4Tel.com)]** with a methodology called TeMOD[8]. TeMOD is now in widespread use within the Telenor group. What we discovered during our work with developing TeMOD for Telenor was that even if we gained substantial benefits from tailoring general-purpose methodologies to the needs of the company, the company itself is pretty diverse. Thereby, a need was expressed of even more tailoring of TeMOD to fit the purpose of different domains and product families within the Telenor group. A main request was to get a methodology that was tailored to utilise existing domain knowledge. However, one of the goals of making TeMOD was to have a common methodology used throughout the company to achieve a common way of developing and specifying systems. So it was clear that we didn't want to end up with a set of proprietary special purpose methodologies, one for each domain and system development group. **Our challenge became to keep TeMOD as the common methodology for the company, producing specifications in a standard way, and at the same time get TeMOD to support specific needs and utilise the existing domain knowledge.**

The most popular general-purpose UML-based software engineering methodologies have both diversities and commonalties. One frequent commonality is that they are model-driven. A model-driven methodology signifies that the methodology prescribes a set of models to be produced during the system development process. TeMOD is indeed a model-driven methodology. Our motion to the above described challenge was a generic framework that provides utilities for tailoring model-driven methodologies in general, in order to utilise the domain knowledge possessed by the company and for producing domain-specific models. Using the framework, the tailoring will only affect the expression of the models prescribed by the general-purpose methodology.

FRAMEWORK DESCRIPTION

By applying the tailoring framework a domain-specific reference-model is produced. The reference-model describes the extensions of the actual general-purpose methodology made for a specific domain. It consists of:

- UML-profiles,
- existing (reusable) models
- and patterns

The set of UML-profiles, existing models and patterns defined in a reference-model are developed, structured and aligned according to the chosen general-purpose software engineering methodology. UML-profiles are used for defining domain concepts and reference architecture, existing models are prepared for reuse and patterns describe standard solutions of recurring problems within the domain. Thus, tailoring a software engineering methodology using the framework, constitutes a leveraging of the methodology in an environment of domain concepts, defined reference architecture, existing models, and patterns. The profiles are the most stable asset in that it defines the overall, reusable concepts within the domain. The existing models and patterns are typically changing, but presumably they constitute a valuable and growing pool of reusable assets.

Figure 1 shows the structure of use of the framework. The tailoring framework defines how to build appropriate UML-profiles, patterns and existing models, and it defines how to use the reference-model in correspondence with the chosen general-purpose methodology. The domain-specific reference-model is built using the tailoring framework and is structured according to the chosen methodology. The profile maker and pattern modeller have the main responsibilities of building the domain-specific reference-model. The system modeller uses the chosen methodology's prescribed models and process, as well as the reference-model for the actual domain, to build a concrete model. The system modeller might also feed back reusable models to be part of the reference-model. Those models will then be categorised according to the chosen methodology, and become an existing model of the domain-specific reference-model.

Figure 1 indicates use of a general-purpose methodology, xx-methodology, which specifies three main models as outcome of the development process: A business model, an architecture model, and a platform-specific model. The yy-reference-model (supporting the yy-domain or yy-product-family) is structured according to the xx-methodology and specifies a set of UML-profiles, patterns and existing model in accordance with the model architecture of the xx-methodology (business, architecture and platform-specific). The concrete zz-model consists of a business model, an architecture model and a platform-specific model produced according to the xx-methodology and the yy-reference-model.

In the example of Figure 1, the domain reference-model includes UML-profiles, patterns and existing models at all model levels (business, architecture and platform-specific). This is not required. However, the tailoring framework requires that the constituents of the domain reference-model should be structured according to the model architecture of the chosen methodology.

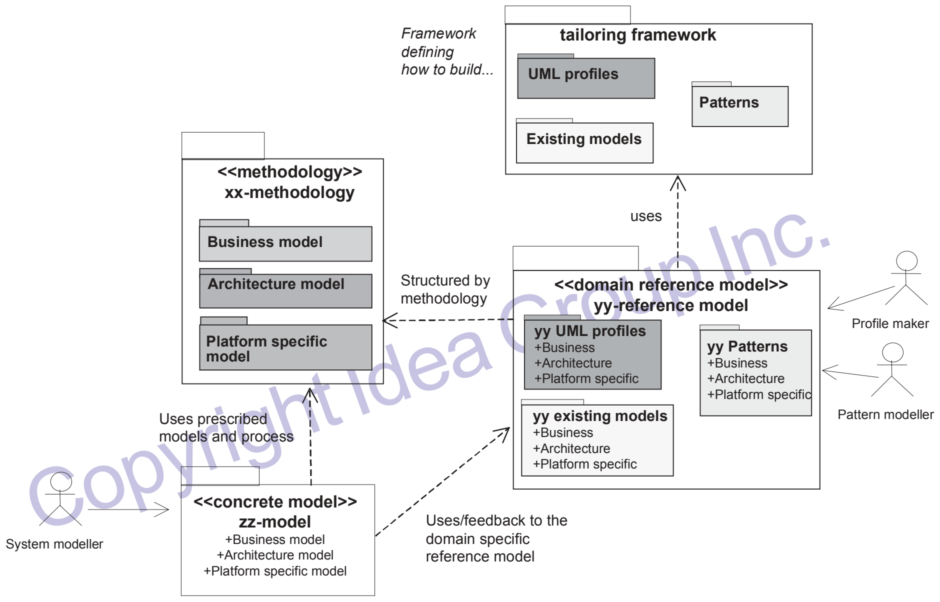
Multiplicity relationships are not shown in the figure. The tailoring framework is generic and might be used to customise all UML-based, model-driven methodologies (including UP, RUP, Select-Perspective, BOM and TeMOD). In principle, an infinite set of domain reference-models supporting a specific domain or product-family might be developed as customisations of a particular methodology (thus, a one-to-many relationship between methodology and reference-model). There are also one-to-many relationships between methodology and concrete model as well as between domain reference-model and concrete model.

It is also plausible to use the tailoring framework to extend or specialise a specific domain reference-model for instance to support a sub-domain.

UML-Profiles

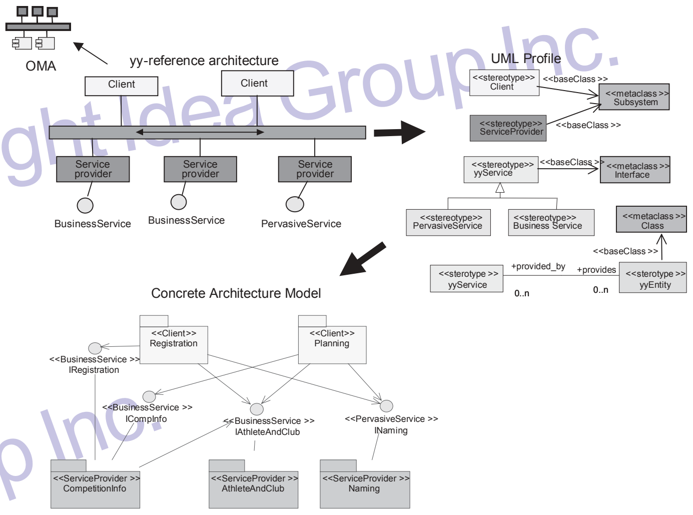
The framework prescribes the common techniques for defining UML-profiles, using the UML extension mechanisms, stereotypes and tagged values, both of which extend the concepts we can work with in the UML world. Using the UML extension mechanisms implies that the developed models still conform to the UML standard.

Figure 1: Example of framework usage



The UML-profiles of the framework describe the essential concepts from the domain in question. These profiles, i.e. the concepts defined in these profiles (the stereotypes), can be used as first class modelling concepts when defining concrete models.

Figure 2: Reference architecture, UML-profile and usage example



The framework suggests to build a profile defining the reference-architecture used within the domain or product-family. Such a profile will give essential support when modelling concrete system architectures. Figure 2 shows an example of defining the reference-architecture for the yy-domain by a UML-profile. The architecture model prescribed by the general-purpose methodology is then customised to support the reference-architecture of the domain or product-family. The figure shows the yy-reference-architecture for the yy-domain or the yy-product-family. This reference-architecture is similar in nature to a standard bus-architecture like the OMA[9](Object Management Architecture defined by OMG (Object Management Group, http://www.omg.org). The UML-profile extends the UML with the appropriate architectural concepts, which then become employed as first class modelling concepts in the concrete architecture model as shown in Figure 2.

Detailing of the UML-profile might be done in a tabular form as shown below (for the yyService and the yyEntity) (Table 1).

Similarly, UML-profiles might be made for all the model levels defined by the chosen methodology. E.g. for the xx-methodology indicated in Figure 1 we might have domain profiles for the business, architecture and platform-specific level defining the vocabulary to be used for modelling each of these levels respectively.

An example of a business level profile for a specific domain concerning business negotiation is shown in Table 2.

An example of a platform-specific profile is the standardised UML-profile for EJB.

Patterns

Patterns represent special kinds of existing models that describe a recurring problem and suggest a general solution to that problem. The tailoring framework is

Table 1: Detailing of the UML-profile

Stereotype	Metamodel base	TaggedValues	Description	Constraints
yyService	Interface	Transactional network-accessible	A service access interface, corresponds to a yyservice from the architecture profile.	self.allOppositeAssociationEnds -> forAll (a | a.type.oclIsTypeOf (yyService) or a.type.oclIsTypeOf (yyEntity))
yyEntity	Class	Persistent	Represents information provided through an yyservice.	self.allOppositeAssociationEnds -> forAll(a | a.type.oclIsTypeOf(yyEntity))

Table 2: Business level profile example

Stereotype	Metamodel base	TaggedValues	Description	Constraints
Service owner	Actor	X	The legal owner of a Fleksit service	x
Service customer	Actor	X	The legal user/buyer of a Fleksit service	x
Service contract	Document	Classification	The contract established between a service owner and a service customer	x
Business agreement	Document	Classification	The high-level business agreement between the business parties.	x
Service Level Agreement	Document	X	The service level agreement	x
Service	UseCase	X	The service in question	x

used to define patterns and categorise them according the actual model architecture of the chosen methodology. For the xx-methodology in Figure 1 there might be business model patterns, architecture model patterns and platform-specific model patterns. A pattern is employed by instantiating it (as a template) into a concrete model, the concrete model defines who/what is fulfilling the responsibilities defined by the roles in the pattern.

The tailoring framework includes a framework for pattern definition and use. This pattern framework includes both some special notation (defined in an UML-profile) and a template for pattern description.

The pattern structure technique of UML collaboration is used as basis to define the pattern. The pattern structure technique is used for viewing the collaboration as a single entity from the outside. The collaboration details are further defined using UML activity diagram or UML sequence diagram. A simple example describing a pattern for a naming service and the usage of the pattern is shown in Figure 3.

A UML collaboration describes the collaboration of roles. A role is a placeholder for a set of objects that can fulfil the responsibilities of that role. Roles can be specified in two ways in UML; instance level and specification level. Instance level role collaborations are described in terms of collaboration diagrams with objects, links, and message stimuli. Specification-level collaborations are described by ClassifiersRoles and AssociationRoles.

UML has defined a naming convention denoting roles, which allows a simple way of indicating a role. The general syntax is

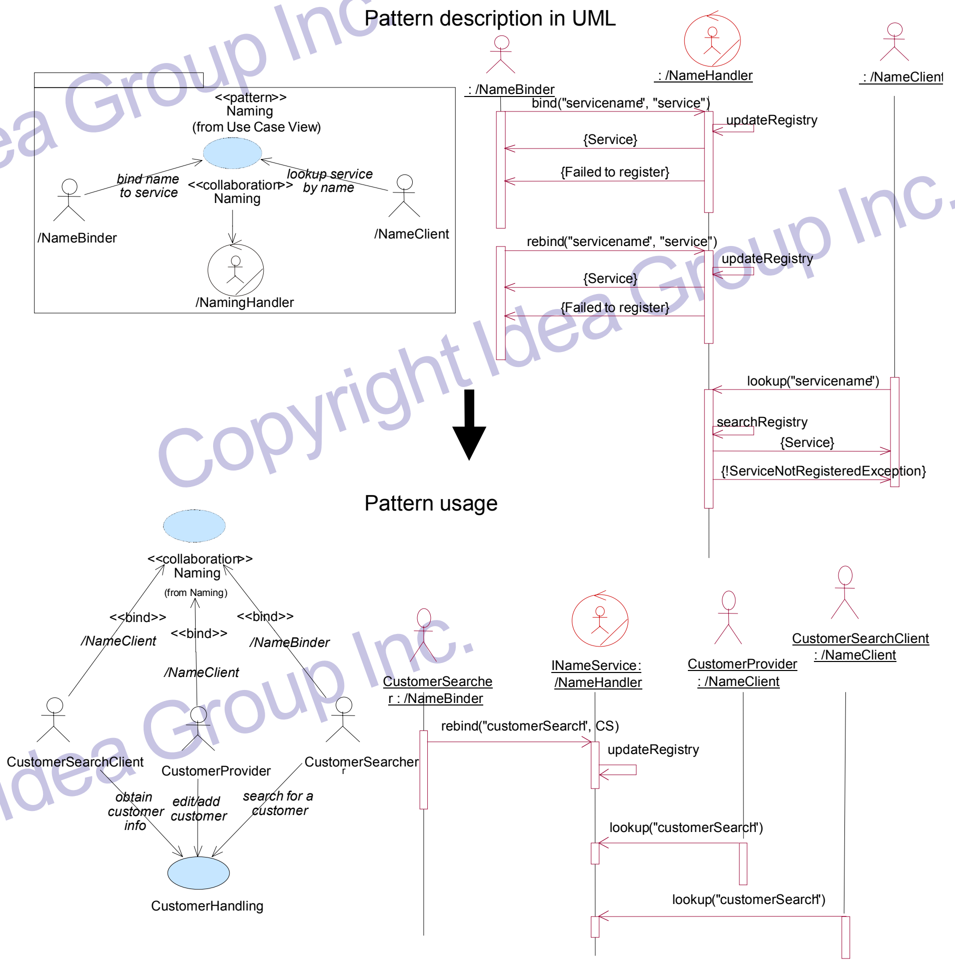
ObjectName '/' ClassifierRoleName ':' ClassifierName [',' ClassifierName]*

This convention can be used on both instance-level and specification-level collaborations.

A collaboration can be considered a set of roles collaborating to fulfil a mission, defined by the unified responsibilities of the roles in the collaboration. Collaborations as a concept has been proven useful in several recognised methodology approaches; Catalysis[3] uses collaborations and collaboration refinements for analysis, design and reuse purposes. OOram[5] uses role models to describe collaborations of roles, and synthesis to refine and reuse existing models. Lately, also the UML community embraces this view of collaborations.

The top left of Figure 3 show the pattern structure in terms of roles collaborating to fulfil a mission defined by the unified responsibilities of the roles in the collaboration. The naming pattern defined includes three roles: Name Binder, Name Client and Naming Handler.

Figure 3: Pattern description and usage



A collaboration is modelled as a use-case stereotyped with <<collaboration>>. The roles are modelled as UML actors with role-names. These roles can either be external or internal to the pattern. It is the external roles that are parameterised when the pattern is used. The worker stereotype defined in RUP is used to denote internal actors. The Naming Handler is an internal role in the example. The semantics of the collaboration 'use case' is the same as a UML collaboration pattern structure (a use-case realisation).

The sequence diagram of the top right defines the behaviour of the pattern by specifying interactions between roles. Certain conventions for describing behaviour in a UML sequence diagram are defined when describing a pattern using the framework:

- Messages sent to a role are described in standard UML manner. These may or may not be messages that exist as a part of that role's protocol. Using '//' as prefix for a method denotes that this method is not explicitly located in the interface, i.e. it may be an analysis operation only, or a reference to existing pattern behaviour.
- Return values are always specified explicitly, with a special message sending convention, the message name is the value of the return and it is packed in curly braces, like this {IObjectType}.
- Errors or exceptions are specified to the extent considered necessary. The convention is the same as for method returns, except for an exclamation mark (!) indicating the nature of an exception, like this {!IOException}.

In order to use a pattern, the desired external roles of the pattern must be instantiated. This is done by using a specialised *'binding'* relationships from the pattern collaboration source to the roles that instantiate the designated roles of the pattern. The role parameters are bound by the role-name specified on the binding relation, e.g. '/ NameClient'. The lower left of Figure 3 shows how a pattern can be instantiated by binding the roles from the pattern. There are no limits as to how many roles can be bound to another role or actor.

The sequence diagram at the lower right shows an example of synthesising the pattern onto a specific architecture.

The framework provides a template for pattern description as shown below.

Name
Names the pattern after the solution it proposes.

Problem description
Provides a thumbnail description of the problem the pattern is solving.

Parameterised structure
The roles participating in the collaboration and the collaboration symbol (the use-case view) (UML use-case diagram)

Behaviour
The behavioural details of the collaboration (UML activity/sequence diagram)

Example
Provide an example of use

Existing Models

Existing models represents already defined concrete models that can be reused by package import and referencing. In the same way as for the UML-profiles and patterns the tailoring framework prescribes that the existing models should be categorised and structured according to the model architecture of the chosen general-purpose methodology. Typical usage of an existing model is to reuse for example an existing interface by inheritance or reference.

An existing model is reused in terms of package import, where the namespace defined by the package becomes available. All public artefacts (use cases, actors, interfaces, classes, etc.) becomes available to the importing package. Alternatively, elements from an existing model can be used simply by scoped naming (UML pathnames), e.g. nnCore::PervasiveServices::NamingService::INaming, which refers to the INaming interface contained in the package structure implied.

The general mechanism for reuse of a model, being it a pattern or a standard model, is by import of the package that defines the model. This makes the model elements defined in that package available (by model elements, meaning interfaces, classes, etc.). In principle, we can then reference interfaces, etc. from that package.

Package import is straightforward in UML, done with the <<import>> dependency stereotype between packages.

CONCLUSIONS

In this paper we have described a generic framework for customising general-purpose methodologies for the purpose of providing better support to specific needs within a domain or product-family. An important aspect of this has been to ensure utilisation of the existing knowledge possessed within the actual domain.

The tailoring framework introduce need for new roles responsible for developing and maintaining reference-models. The developers must also learn to use the actual reference-model appropriately. Thus, successful introduction of the tailoring framework requires a well-defined process and careful consideration of the relevant risks.

The framework has already been used within the Telenor group and we have seen several benefits from its application. It assists in:

- Establishing and maintaining models representing knowledge possessed within the domain.
- Model reuse: The reference-model advocates model level reuse leveraging existing models and patterns describing best practice for solving recurring problems within the domain. Current tool support is rather immature, so there are still potential for gaining substantial improvement of the efficiency of the model development with sufficient tool support.
- Reuse at the right level: It has proven that efficient reuse is easier to gain within a product-family community or a fairly small scoped domain as opposed to general-purpose reuse in widely scoped domains. It is then easier to build a reusable asset library within the reach of the users. The reuse task is also more straightforward (does not imply a lot of tailoring).
- Ensuring consistency of a set of models: Customising a general-purpose methodology with stereotypes, common domain-models, or common patterns will help making the models more consistent.
- Standardisation: The reference-model functions as the "standard" for the specific domain, without contradicting the prescriptions of the general-purpose methodology which function as the standard for a set of domains (e.g. the enterprise as a whole).
- Adding more semantics to the models: The use of stereotypes and the description of patterns can help making the models more powerful and expressive for the readers and the modellers.
- Preparing for code generation: Code generation can be made more powerful when defining reference-models and utilise the defined UML-profiles, patterns and existing models. This might be stereotyping a certain kind of interfaces, data types or other aspects that can help the code-generator doing sophisticated code generation. This requires a customised code generator, which introduces overhead and administration in some aspects of development, but might substantially increase efficiency of development and also make the system development less error-prone.

Another interesting aspect we have experienced is that the tailoring framework might be used as a vehicle for successful introduction of a common general-purpose methodology within an enterprise. Since the customisations makes the general-purpose methodology more appropriate for the different domains within the enterprise it is more acceptable for the different groups of system developers (users). The users are typically also involved in the customisation gaining a feeling of ownership to the introduced methodology

In the future, we plan to work further with the tailoring framework in co-operation with Telenor and EDB4Tel. We'll follow up the usage of the framework within those companies and investigate the experiences attained. General aspects of the framework as well as tool support will be main foci in the COMBINE[13] project and the DAIM project[15]. We'll also work further with the framework as part of the CAFÉ[16] project.

REFERENCES

[1] Ivar Jacobson, Grady Booch, James Rumbaugh, The Unified Software Development Process, Addison-Wesley 1999.

[2] 1996, ISBN 0134529308 Rational: The Rational Development process

[3] Catalysis, Desmond D'Souza, Alan C. Wills, 'The Catalsysis Approach', ISBN 0-201-31012-0, www.catalysis.org

[4] Paul Allen, Stuart Frost, Component-Based Development for Enterprise Systems, Applying the SELECT Perspective, SIGS Book and Multimedia 1998

[5] OOram, Reenskaug, Wold, Lehne: Working With Objects. The OOram Software Engineering Method, Manning/Prentice Hall 1996. ISBN 0-13-452930-8

[6] Ian Graham, Brian Henderson-Sellers, Houman Younessi: The OPEN Process Specification 1997, Addison Wesley, , ISBN 0-201-33133-0

[7] Arnor Solberg, Arne Jørgen Berre. The Business Object Methodology, Deliverable of the OBOE[12] project

[8] Arnor Solberg, Jon Oldevik. Telenor Methodology for Interface Modelling with UML, version 3.02, August 2001

[9] OMG's Model Driven Architecture: http://www.omg.org/mda (MDA is a trademark of OMG)Specification 1997, Addison Wesley, , ISBN 0-201-33133-0

[10] Svein O Hallsteinsen, Geir Skyllstad, Arnor Solberg, Tor Neple, Arne Jørgen Berre. The Magma software engineering handbook, ver 1.0 2000, Magma - A technology development project initiated by PROFF; The Software Industry Association of Norway

[11] OMG EDOC profile – UML-profile for enterprise distributed object computing, http://www.omg.org/techprocess/meetings/schedule/UML_Profile_for_EDOC_RFP.html

[12] OBOE - Open Business Object Environment (OBOE). Esprit IV project 23233, http://www.opengroup.org/public/oboe/Home.html

[13] COMBINE – COMponent-Based INteroperable Enterprise system development, ESPRIT V project IST-1999-20893,

[14] MAGMA: http://www.ikt-norge.no, > Prosjekter > MAGMA (Web pages in Norwegian, but the MAGMA Software Engineering Handbook in English can be downloaded)

[15] DAIM – Distributed Architecture, Internet, and Multimedia, a research project sponsored by the Norwegian Research Council.

[16] CAFÉ – from Concept to Application in system-Family Engineering http://www.extra.research.philips.com/euprojects/cafe/