Digital Architecture Framework Handbook

*Digital Engineering Applied*

Author: Giuseppe Platania

giu.platania@doorbusinessconsulting.com

Document Version: 38

License

This work is licensed under the Creative Commons Attribution-NoDerivs 3.0 Unported License.

Digital Architecture Framework Business Manual by Giuseppe Platania is licensed under a [Creative Commons Attribution-NoDerivs 3.0 Unported License](http://creativecommons.org/licenses/by-nd/3.0/).

You are free:

* **to Share** — to copy, distribute and transmit the work

Under the following conditions:

* **Attribution**— You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
* **No Derivative Works** — You may not alter, transform, or build upon this work.

To view a copy of this license, visit <http://creativecommons.org/licenses/by-nd/3.0/> or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

Acknowledgments

The text that you are reading is the result of years of work and exchanges with exceptional colleagues. Without their help Digital Architecture Framework would still be a vague vision.

I have learnt a great deal from those who have worked with me over the past 2o years years.

Let's begin thanking Ingo Herwig, original author of the “Domain” part of Digital Architecture Framework and best Open Source committer. He developed for it his PHP framework wCMF. That is one of the most powerful frameworks around and maybe the most powerful written by one man only. Ingo was the ideal partner for conversations that clarified my thinking on Digital Architecture Framework and other matters.

Niko Stotz was the developer of most part of the original Chronos generator code. He is a sharp mind in pointing out the problem and finding the solutions and an ironic voice when the solution is still to be found . Niko was very good at implementing ideas I expressed verbally.

The venture of applying Digital Engineering concepts in a project began with Fred Meyer, for many years my “twin consultant” at IBM. I am particularly grateful to Fred for his thoughtful and creative comments on the business side of Digital Architecture Framework. I'm indebted with Christian Schiller and Andreas Herdt for modelling “Moma” and “E-Energy”: the most complicated Digital Architecture Framework model ever, discovering “on the job” new requirements and pointing out bugs.

My manager Jürgen Schweickhard has backed the idea behind Digital Architecture Framework for many year, without him would be not possible for me to keep working on it.

Most thanks should go, however, to Pierre Joeris and Detlef Schumann for defending the introduction of Digital Architecture Framework in an Energy & Utility context in difficult meetings.

Thanks to Marco Eilers that wrote the original version of the Digital Architecture Framework Web Browser.

Thanks also to in not special order:

Enzo Failla, Stephan Heller, Christoph Weiss, Jan Laczay, August Ziesemer, Thomas Schuetz, Johannes Hintsch, Gustav Tschirschnitz, Christian Panzer, Paul Robinson, Sai K. Ari, Raul Duczek, Sabine Shentzel.

To all the above colleagues, and to several others whose names I cannot continue listing and who have assisted me one way or another I feel very much indebted.

Last but not least, I am deeply indebted to my family for the all patience and the love.

Document Location

This is a snapshot of an on-line document. Paper copies are valid only on the day they are printed. Refer to the author if you are in any doubt about the currency of this document.

The source of the document will be found under:  
https://github.com/FreeTAKTeam/DigitalArchitectureFramework/tree/main/Docs

Revision History

|  |  |
| --- | --- |
| Date of this revision: Feb 2023 | Date of next revision *Juny 21 2023* |

Introduction

This manual lead to in the step by step creation of a Business Oriented model that can be used in the context of a Digital Engineering (DE) project.

Some suppositions about the reader

Greetings, esteemed Business Analysts, Architects, and Developers. This manual is specifically tailored for individuals who intend to apply Digital Engineering utilizing the Digital Architecture Framework methodology to create an Authoritative Source of Truth capturing requirements, modeling business strategies, and using the result to generate code and documentation. Our audience can be classified into two main groups: modelers with a focus on clear communication and architects who aim to design software in a machine-executable manner. While prior knowledge of computer programming may aid in your understanding, it is not a requirement. However, a fundamental understanding of modeling and architectural principles is recommended.

How this text is organized

After this Introduction the book contains following sections:

* **Concepts**: introduce the foundation of the Digital Architecture Framework methodology and his tool.
* **Handling Projects easily with the Digital Architecture Framework Methodology**: Introduce the Methodology work flow.
* **Additional Tools**: describe the use of the Model browser, the Document and code generators

How to read it

This manual is approached in a practical manner and may be read in any order based on your current needs. You may choose to read it from beginning to end, or jump to specific sections of interest. However, please note that some concepts are repeated throughout the manual for ease of reference. To fully grasp the Digital Architecture Framework, we recommend taking the time to understand its structure. Let's get started

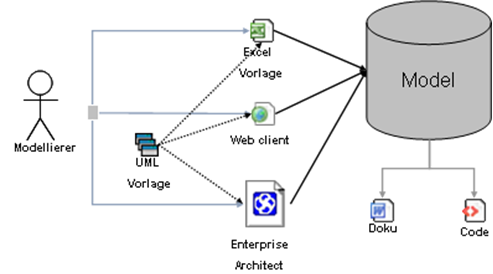
Concepts

In this chapter, we introduce key concepts involved in a Digital Architecture Framework project using Sparx EnterpriseArchitect as the modeling tool. Our goal is to establish a valid model, adhering to a specified meta-model, where the origin and history of the domain information can be traced and effectively displayed as abstract information

About Digital Engineering

Digital Engineering (DE) aims to consolidate organizational’s information, previously trapped in separate systems and files, into a single model. This model serves as an abstraction of concepts that describe a specific domain, and all related knowledge products, such as documents or code, are generated from the model rather than being produced, managed, and stored separately.

DE represents a paradigm shift in IT projects, similar to the invention of the production line in automobile manufacturing. By utilizing a 'knowledge production line,' DE can reduce the time and increase quality in an IT project, potentially leading to savings of 25-63% compared to traditional methods.

Illustration 1: in a DE system all the knowledge in his complexity is stored in a special kind of ontology in form of a UML model. The artifacts, including code and documents, are generated from a model. Different tools can contribute to the manipulation of it.

.

the Digital Architecture Framework methodology

In the past 20 years (2000-2023) , the introduction of Agile methodologies in the software development life cycle (SDLC) has helped to improve the efficiency and flexibility of software development projects, but it has not necessarily addressed all the challenges, especially when applied to complex projects. Agile methodologies can be adjusted to include some of these aspects but they are not designed per-se specifically to handle complexity​. As a result, only 8% of large or Grand projects succeeded.

In the experience of the author one major reason for those failure is miss-managing of knowledge at different levels and from different sources.

The business and the technical language don't fits. Managers, Business Consultants, Solution architects and Developers “speaks” all different languages that needs to be translated again and again. Parts of the original meaning are lost in translation.

The Digital Architecture Framework methodology for Digital Engineering was created as a response to this situation. It describes a how to use the Digital Engineering approach and to create an Authoritative Source of Truth (ASOT). The ASOT provides better communication by serving as a centralized, unified repository of information that is used as the basis for all decisions and actions within an organization. By providing a single source of truth, it eliminates confusion and misunderstandings caused by conflicting or inconsistent information. The ASOT provides a clear and consistent understanding of the organization's goals, processes, data, and technology, which in turn enables better decision-making, collaboration, and alignment across the enterprise. Additionally, the ASOT helps to ensure that all stakeholders are working with the same information, which improves communication and reduces the risk of miscommunication. Overall, the ASOT provides a foundation for better communication by establishing a shared understanding of the organization and its operations.

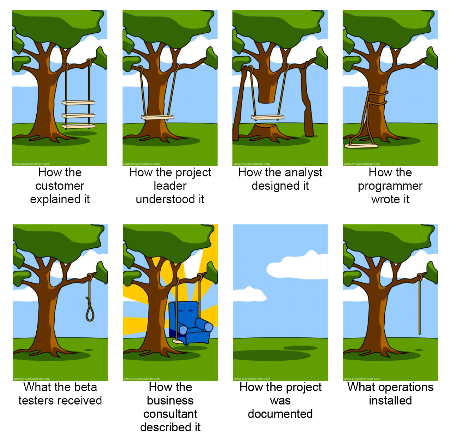


Illustration 2: **WHY** we need Digital Engineering and an Authoritative Source of Truth? Today most projects suffers of poor understanding between different roles

About the DAF Metamodel

Digital Architecture Framework was created as pragmatic way to realize the vision of a DE approach define above.

The DAF Metamodel is a language with a precise syntax and grammar. These rules allows to creates such a precise model (ASoT) that can be also understood from a machine (the DAF generator).

From a formal point of view it is an UML profile[[1]](#footnote-2) which defines the notation for all elements used in the different stages of the project life cycle from the requirement to the application. For details about Digital Architecture Framework consult the documentation.

Some history of the“Digital Architecture Framework”

DAF is the last evolution of the Chronos Metamodel, part of the Olympos family that include Aphrodite (an abstract specification for a server), Dionysos (an abstract specification for a client and the communication with the Aphrodite server) and Ouranos (a metamodel that addresses how to extract models from legacy code). DAF expands the Chronos concepts to include a Computational Independent Model Layer describing strategic concepts.

Introducing Sparx EA

In this manual Sparx Enterprise Architect (EA) is used for create a valid DAF Model. EA aims to provide a collaborative environment to create and manage (much more than) Digital Architecture Framework models. The result can be exported to be used in other UML editors or directly transformed in code and/or a documentation.

EA is a very powerfull tool, so the following is a list of best practices, not an alternative to reading the extensive online material[[2]](#footnote-3).

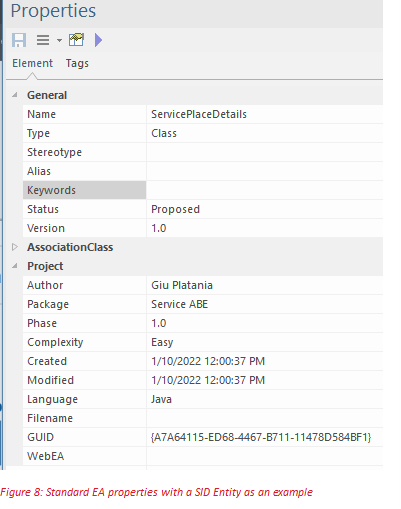
While EA has been selected here, any good modeling tools that support profiling can be used. DAF models where created successfully by using tools like IBM Rational Software Architect, Eclipse Modeling Framework or the Chronos Web Modeler (CWM[[3]](#footnote-4))

EA Special Fields and Features

While EA works pretty intuitively (?), in some case it is necessary to be aware of the functioning of certain characteristics.

Standard EA properties

Each Sparx Enterprise Architect element has a standard set of properties. It’s necessary to understand the meaning of those and use them consistently

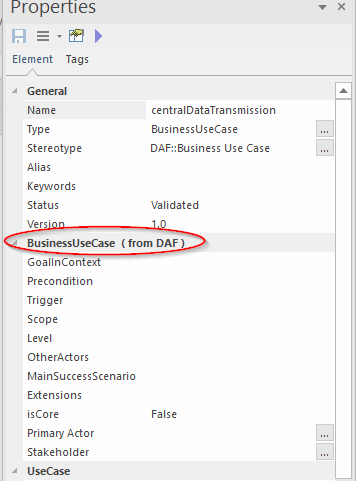


Standard EA properties

| **Name** | **Notes** |
| --- | --- |
| Name | The name of this element |
| Type | The UML type of this element |
| Stereotype | The additional properties for this element e.g. “SID Entity” |
| Alias | An alternative way in which this element is know e.g. Service Location Details |
| Keywords | A set of tags, separated by comma that help to identify the element ‘Service, Place, location, details’ |
| Status | Very important: customizable set of an element conditions  describing his life cycle |
| Version | Very important: The unique version # of this element (as opposite to the Phase that describes a set of elements) for example 21.02.345 where the last number is updated manually every time the element is changed |
| Author | This is the technical author of the original element, namely the user that has physically created the elment in the repository |
| Package | Automatic property: location of the element in the repository |
| Phase | VERY IMPORTANT: free text that describes a set of elements (as opposite to the version describing a single one). For example a new release 22.5 |
| Created | The original date of the element’s creation. Helps to determine the  question “since when?” |
| Modified | Automatic time stamp that will be updated every time the element is modified |
| Language | Each element in EA can be transformed in a target code language |
| Filename | If the element has been generated as code, shows the local path of the file |
| GUID | The unique ID of the element |
| WebEA | If WebEA is installed, the URL of the element |

Any additional property needs to be added as a tagged value, part of a UML profile.

DAF properties

Figure 1: Special DAF properties

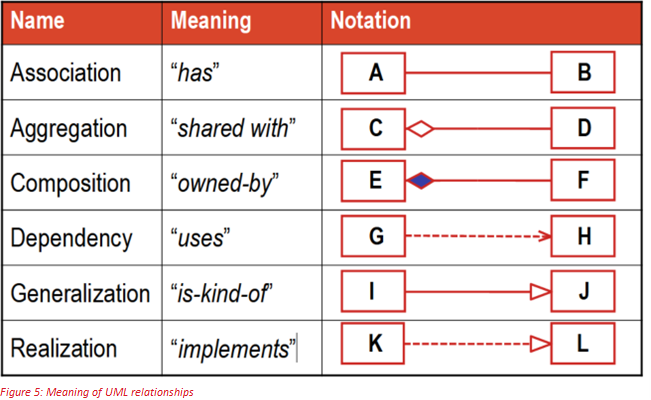
DAF properties are listed under the General section

Relationships

relationships are a key Modelling concept.   This document will not attempt to explain in any detail the rich association options and details,

DAF models makes use of

* Associations
* Generalization
* Aggregation
* Composition
* Dependency
* Realization



In EA Associations contain a number of properties (Name, Source Name, Target Name,  role, multiplicity, navigability) . Relationships can also have a description,

Difference between "Author" and “Creator”.

The “*Author*” field and “*DAF:author*” field help also to keep trace of the object history.

The “Author” field in DAF is regarding the original **business** maker of the object itself. this can be selected from a list of project members (see [#3.2.3.Configuring the model](#3.2.3.Configuring_the_model)).

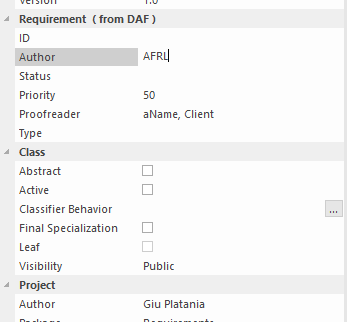
The author is NOT necessary the *technical* author means the EA user that physically inserted the information. This is the conceptual "**creator**" while the "**last\_editor**", as the name suggests, is the last that modified the object. Is a good practice to create technical users with the convention “firstname.lastname” and authors with “FirstName Lastname- Role”

Illustration 12: DAF author and EA Author fields compared

Rich text area

EA offer some facilities to write texts. All the Digital Architecture Framework objects have a property “notes” with a rich text area.

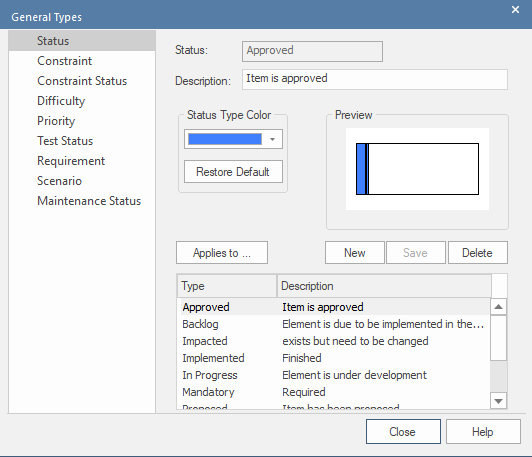
* in the text editing interface EA supports **bold** (ctrl + B), *underline (ctrl + U)*and *italic* (ctrl + I). Bullet Lists and numbering.

In creating very long documents can be more comfortable to use a full features word processor to produce text and the n paste it into the note area. On the other side you should be aware that some formatting will be removed on the server side. By pasting any text in the text area it will be appear exactly as-is in the word processor but by reloading the object you can have some surprises:

* Some special characters like “©¥®Ω√” are supported.
* Images are not supported.
* Formulas created with a formula editor like will not works because they are embedded OLE objects, they will result in to ∑ "a" + "b" + "cifb" ≈ "5"
* Tables are not supported: text in table will be transformed in a list.
* Standard fonts are not supported.
* e.g. “times new roman 13” , “Arial 8” or “courier new 10” will be reset to the default font.
* Those limitation are required to generate documents. You can can control the output of your documents by using the document templates of the generator .

Status types and management

EA objects have a “Status” field that can be configured in theSettings→ reference data User interface in the . Following statuses are delivered with the standard model.

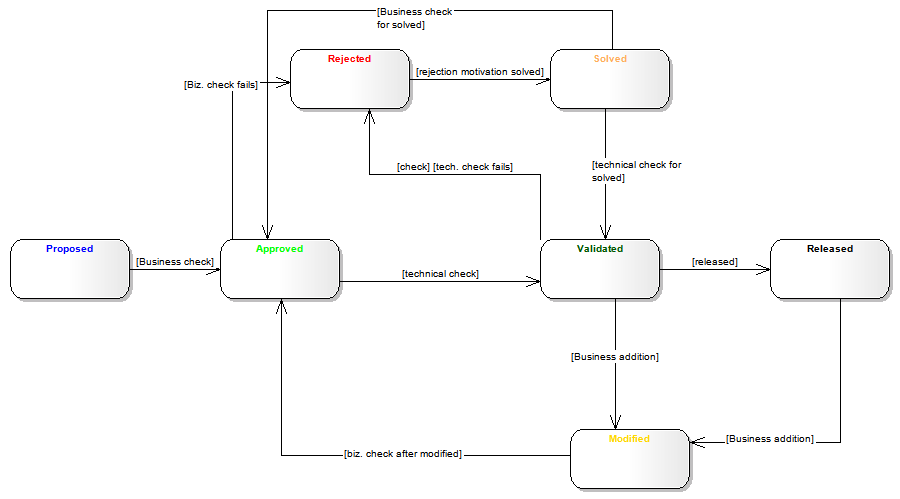
Illustration 13: a list of statuses provided in the Default Model

Notice that Requirements and Features have special Statuses that can be configured separately.

| Status | Notes | Key Color (RGB) |
| --- | --- | --- |
| Proposed | the object has been created and is waiting for review | 0,0,255 |
| Validated | The object has been validated in his quality from a technical/model point of view | 51,153,52 |
| Approved | The object has been validated from a B**usiness** point of view | 0,255,0 |
| Rejected | The object has been rejected from a technical or **Business** point of view | 255,0,0 |
| Solved | The reason for the rejection has been eliminated and the object is awaiting for a further review | 255,128,0 |
| Released | The object has been released and cannot be further be modified in this version of the model | 128,128,128 |
| Modified | The object has been modified from his original status Proposed or released. | 255,255,0 |

Table 1: Statuses in the workflow

The suggested workflow is illustrated Below as map of transitions between those statuses:

Illustration 14: transition workflow between stati

Each object is created in a “*Proposed*” status.

The object is checked from the business point of view. If it is agreed that it make sense his status is set to **Approved**. In the next step the object is evaluated from the technical point of view. If it pass the check is set to **Validated**. In both cases the object that don't pass the check can be set with a status of **Rejected**. The proofreader MUST explain the reason for the rejection. If an element is rejected the original Author if responsible to evaluate the motivation and propose a solution. In this case the object is set to **Solved** and send back for Business and Technical approval. A validated object can be transformed in any artifact (e.g. a deliverable document or a piece of code), in this case is **Released**. A release mean that a physical, non model representation of those object has been created.

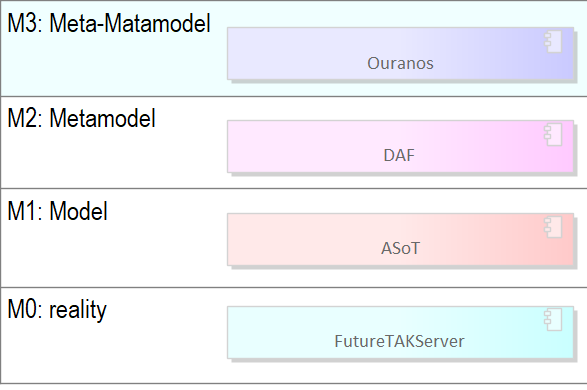
Illustration 15: a version number is added when a released object is changed.

If a released object is changed his status is set to **Modified** and the version number is changed to reflect this.

e.g. as an idea about a requirement is proposed by a Business Analyst. This idea need to be confirmed by a Subject Matter Expert (SME). The SME thinks that the Requirement is fine. The status is set to “*Approved*”. Now the object is checked vs. his “model technical aspect“ by an Architect. Constrains and relationships must be fulfilled, like the Requirement must have a Parent attached. Because this is the case, the status is set to “*Validated*”. In both the cases of the SME and the technical proofreader, an object that is non valid can be “*Rejected*”. When the reason for the rejection has been addressed the status is set to “Solved” and wait again to be approved or validated.

When necessity exists to change it after validation this is set to “*Modified*” and need to be Approved and Validated again. When a collection of objects (e.g a package) achieve a Validated status they can be “*Released*”.

Models and Meta-Models

Figure 2: Models and Metamodels

A Model (m1) is an abstracted representation of of a specific aspect of a system (m0). To describe it, the model needs Meta data describing the concepts used: A Metamodel (m2) is a model that describes the structure of models in a specific domain. Finally the Model that describe the model, is also defined by a Model: A Meta-Metamodel (m3) is a higher-level model that describes the structure and components of metamodels..

Let's explain this.

In the following example “Jon” is a certain concept coming from the reality.

M0 = “FutureTAKServer”

By enclosing “FutureTAKServer” within the tag “dApplicationComponent” we define the type of concept

M1 = “<dApplicationCOmponent>FutureTAKServer</dApplicationCOmponent>”

In a similar way a Meta-model (m2) is an abstract representation of a Model (m1). This is created by adding a definition to “dApplicationCOmponent ”, in our case “Concept”

M2 = <Concept>dApplicationCOmponent</Concept>

finally metaMetamodel is defined self as  
M3 = <Concept>relationship</Concept>

From a practical point of view, the Digital Architecture Framework is maitained as a Metamodel, that is transformed as UML profile. UML profiles allow for the extension of the UML language by adding domain-specific modeling constructs. A UML profile is a set of stereotypes, tagged values, and constraints that are applied to existing UML elements. Stereotypes are used to extend the basic UML elements to provide additional meaning, while tagged values and constraints can be used to specify additional properties or constraints on these elements.

Rweference Model Structure

The DAF Reference Model template features a comprehensive structure, based on The Open Group Architecture Framework (TOGAF) - which is an enterprise architecture framework for developing a consistent, holistic and integrated view of an organization's architecture - and best practices, suited for multi-user and collaboration that can be easily customized for specific needs. The packages have notes, explaining their use and are can be navigated intuitively using diagrams.

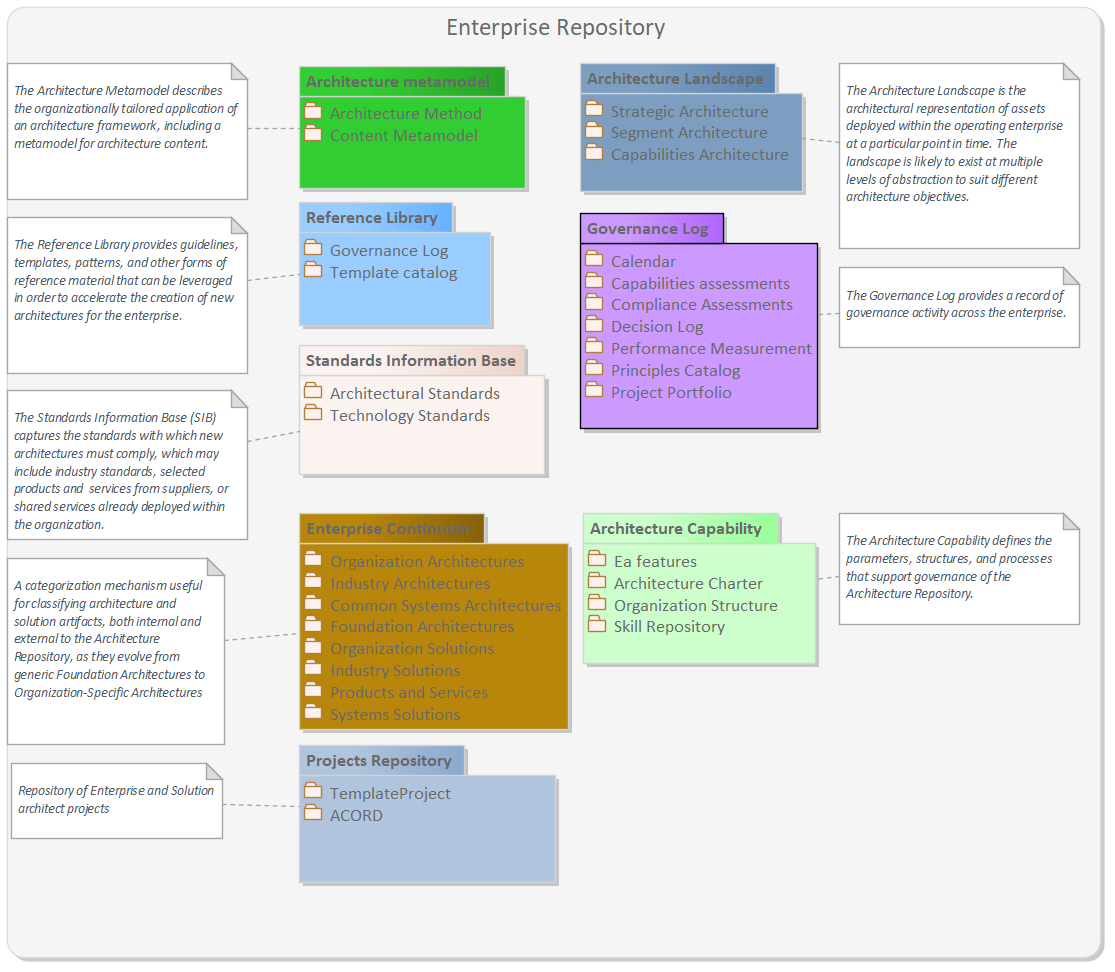
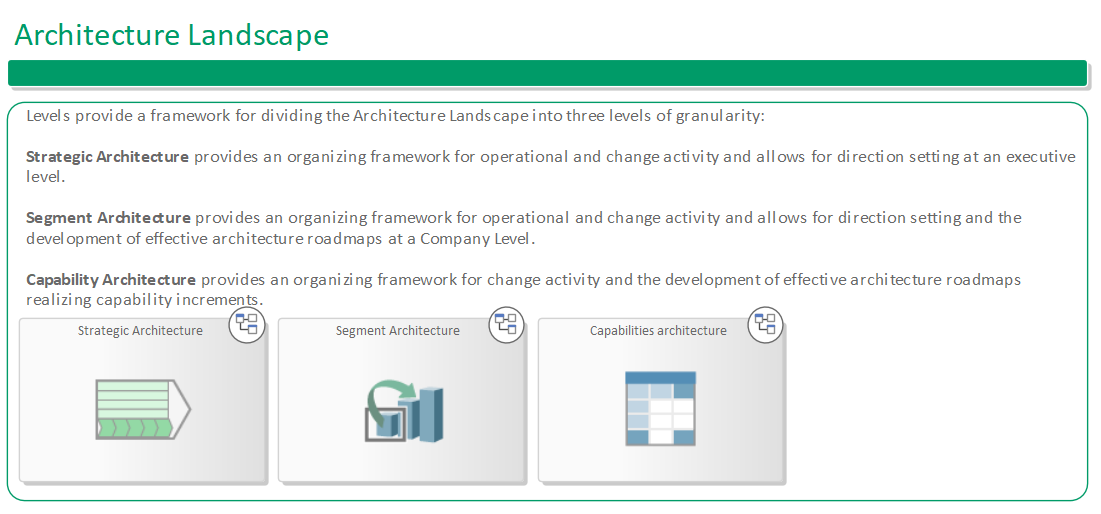


Figure 3: Repository Overview

The main components are the following:

* The **Architecture Metamodel** describes the organizationally tailored application of an architecture framework, including a metamodel and a Methodology
* . The **Architecture Landscape** is the architectural representation of assets deployed within the operating enterprise at a particular point in time.
* **Enterprise Continuum** is a categorization mechanism useful for classifying architecture and solution artifacts
* The **Projects Repository** details the overall projects plan, phases, milestones and resourcing requirements for the all projects. Also hosts all the new created elements that were not yet evaluated and promoted to the architecture Landscape.

The major part of the architectural content is organized in the Architectural Landscape



Levels provide a framework for dividing the Architecture Landscape into three levels of granularity:

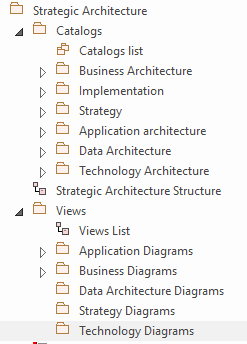
* **Strategic Architecture** provides an organizing framework for operational and change activity and allows for direction setting at an executive level. In general here you will place the elements that are apply to the complete enterprise
* **Segment Architecture** provides an organizing framework for operational and change activity and allows for direction setting and the development of effective architecture roadmaps at a Division or Sub-company Level.
* **Capability Architecture** provides an organizing framework for change activity and the development of effective architecture roadmaps realizing capability increments. You can place here artifacts produced in projects.

Catalogs and Views

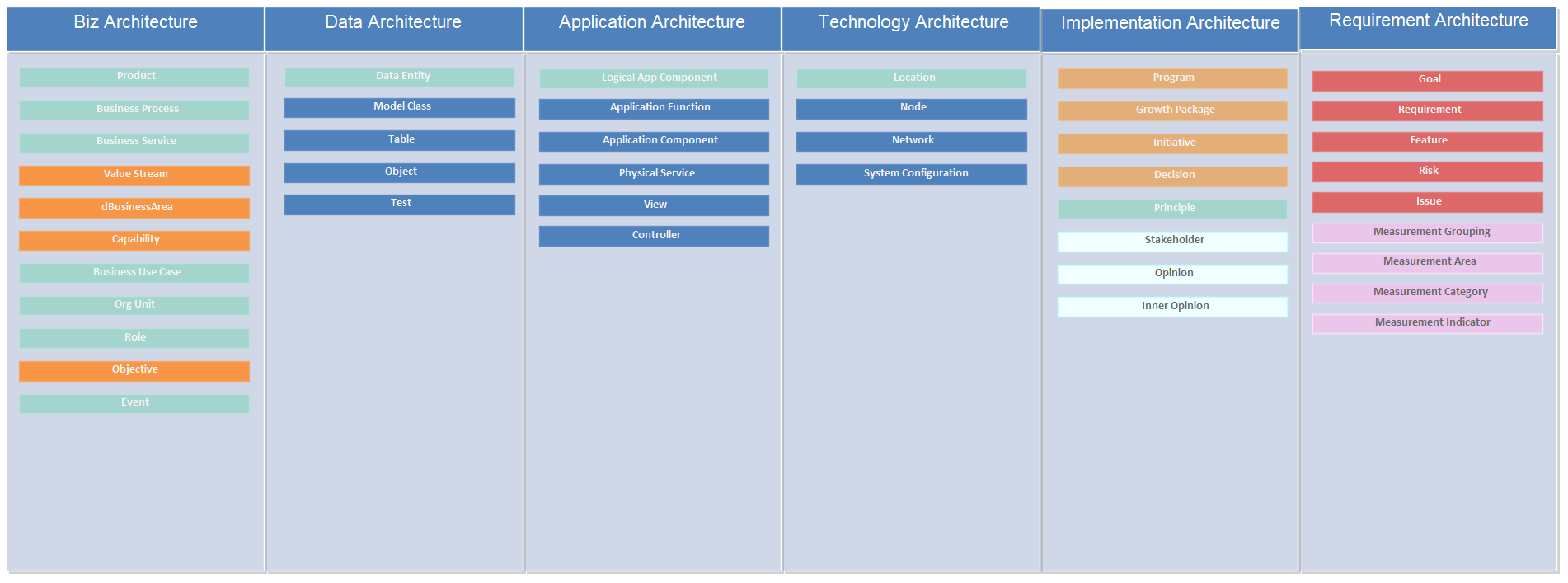
Each Architecture Landscape level is organized in TOGAF Catalogs and Views.

* **Catalogs** refer to collections of information and data elements that are organized and classified for specific purposes.
* A **View** is a representation of a system or an aspect of a system that is tailored to meet specific stakeholder needs. A view is typically depicted in the form of a diagram or matrix.

Views are related to different perspectives or viewpoints . Views can be created at different levels of detail and focus, and can be used to support specific business and technical needs.

Figure 4: Organization in Views and Catalogs

In conclusion, Catalogs are organized in packages containing the Domains with sub-packages for each element in the metamodel

the „20 rule“ for catalogs

As soon the catalog exceed a certain dimension (typically ~20 elements), the need may arise for further organization. A good solution can be to further organize elements inside Catalogs using the Business Capability model.

Alternative structure for Catalogs

As a general rule, elements of one type only should be found in a Catalogs and that is suited for most cases.

However, a question that is frequently asked, is how to organize hierarchies of the same element type (e.g. Application components and sub-components.)

If the “owning” of an element is relevant, the best solution would be to have a single structure that contains them as following:

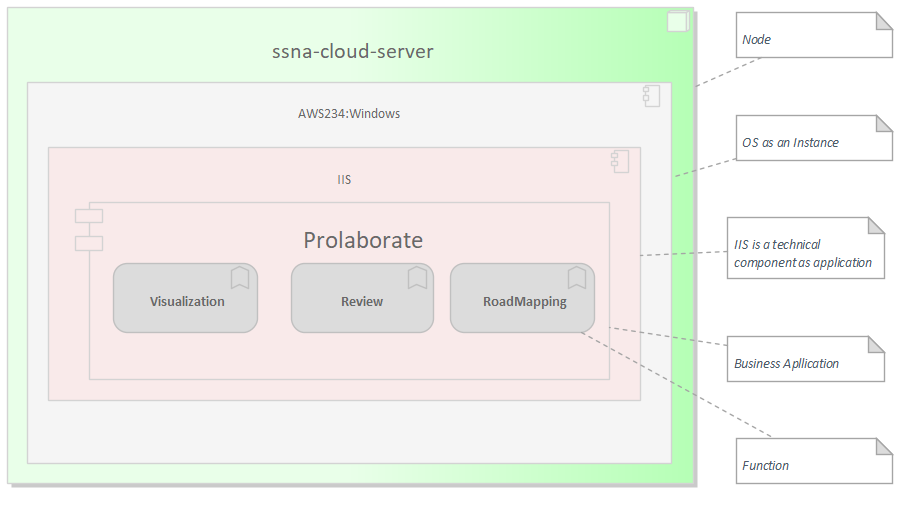
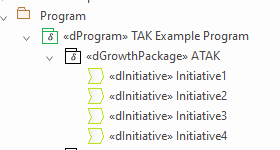


Figure 5: Example of Model embedded structure.

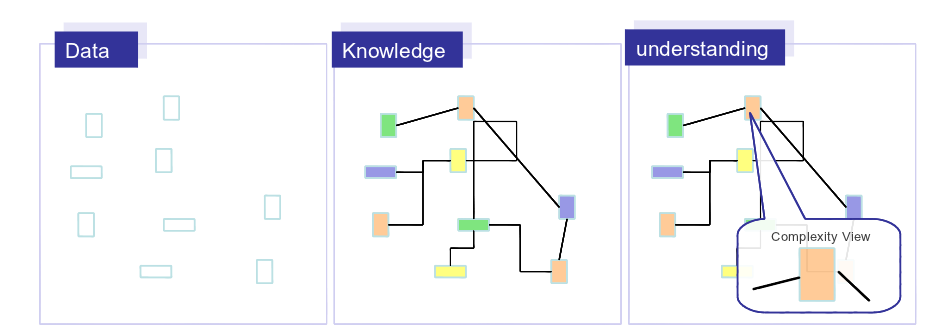
In the example, we have a server (Node), a machine (instance of a classifier), and multiple Application Components and Application functions liked together and residing in the same physical structure.

Figure 6: the catalog of Programs contains dPrograms owining GrowthPackages owining Initiatives

NOTE: Sparx EA will automatically create the structure in the diagram, **if all the elements are in the same package**. In case the elements are in different packages the organization will remain visual only.

About diagrams

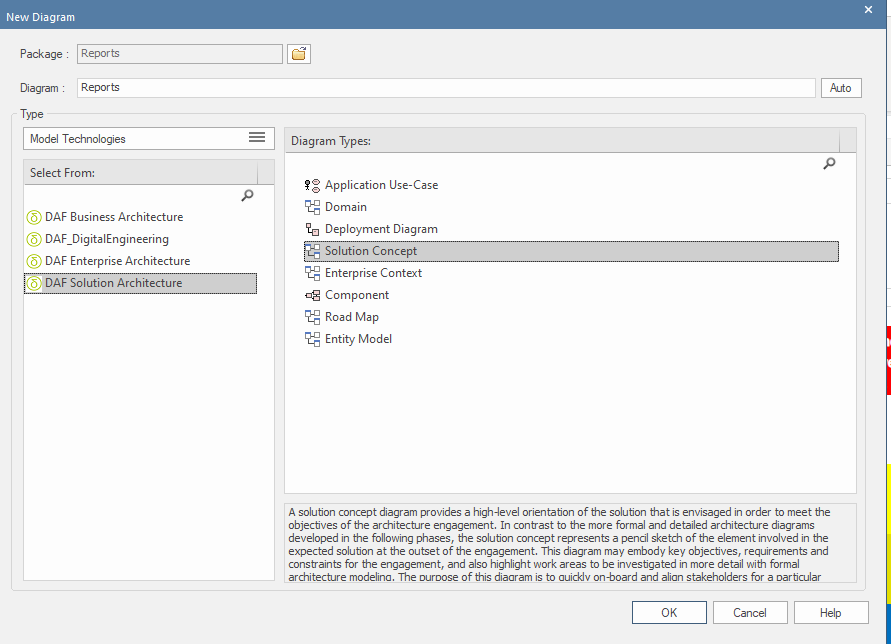
A lose collection of elements is in an entropic status. To order them it is necessary to establish a structure trough relationship. They are the added value that transforms Data in Knowledge. The resulting model, when considered it as whole, can be very complex. For this reason, models are *visualized* in diagrams (views). Diagrams become a form of knowledge representation that helps the Understanding of the complexity. Diagrams are NOT the knowledge model itself. Each of them is a View, that help the understanding from a certain view point, used by a certain actor. The fact that an information is displayed inside a diagram only reveal the original intent of the modeler that can decide to hide certain part of the model.

Illustration 3: Diagrams are views over complexity providing understanding to certain stakeholders

Bottom line: a model can be extraordinarily complex and to display this complexity it is required to have several viewpoints and many more views (diagrams). The modeler can create so many diagrams as he retains necessary to visualize the considered point of view.

Diagrams types in a Digital Architecture Framework Model

Standard diagrams

The DAF Mdg supports a standard set of diagrams organized in ViewPoint represented by the roles:

* Business Architecture
* Enterprise Architecture
* Solution Architecture
* Digital Engineering

Other Diagrams

Additionally we can find following types of diagrams:

* Work diagram
* Package Overview
* Chain Inheritance
* Deep View
* Mapping

Work diagram

A work diagram is anything that you create by simply drag, drop and connect objects in your workspace. Works diagrams are typically created while you add objects to the model and establish relationships.

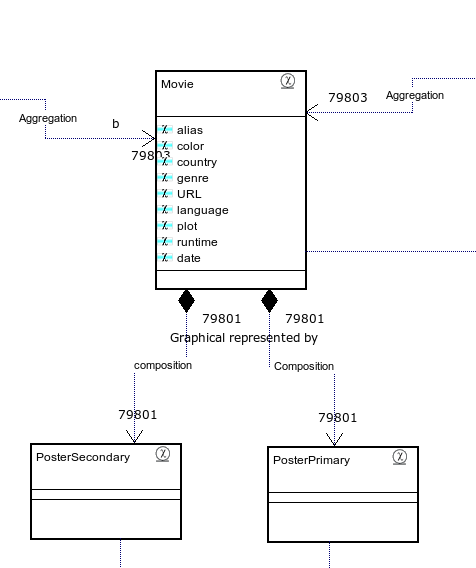
They become easily huge and very hard to understand for an external reader. Conventionally they have the name prefix “<<Name>> Work diagram”.

Work diagrams don’t suit for concepts presentation (see [2.2.2 The seven rule](#2.2.2.The_seven_rule|outline) below).

Relevant nodes

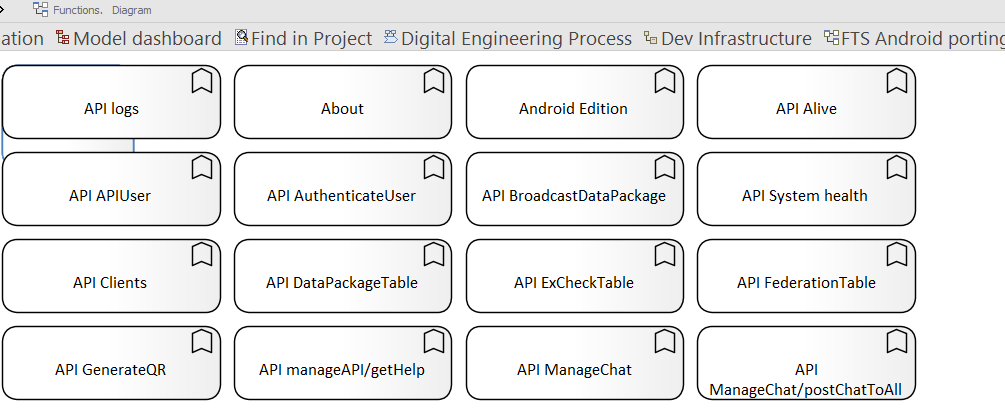
The relevant nodes diagram (see also [5.2 Create the domain model](#_toc2332)) contains all the ChiNodes that are involved in a Use Case (see [#2.1.Modeling Use cases](#_toc913)).

Package Overview

Illustration 4: Example of relevant nodes diagram

The Package Overview Diagram contains all direct children elements of a Package.

As default name you SHOULD use “<<PackageName>> Overview”

Illustration 5: Package overview example

The DAF MdG offers an automatic method to create a package overview diagram.

Right-Click on a package and select “*New Diagram form package”*

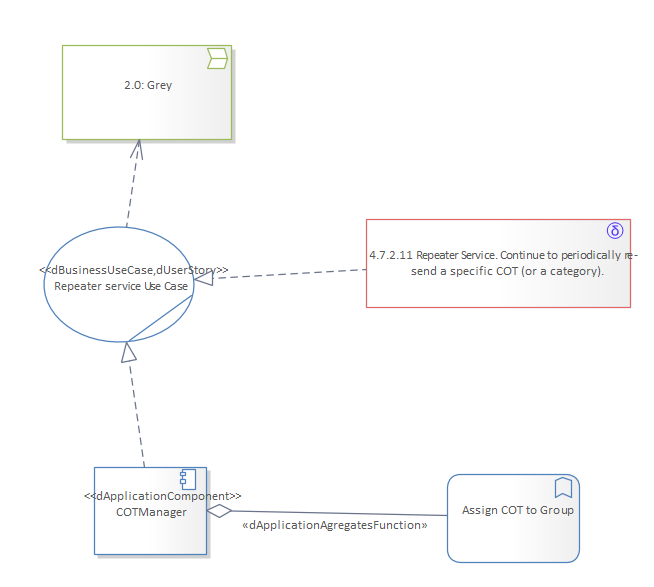


Illustration 6: EA menu for creating a Package overview

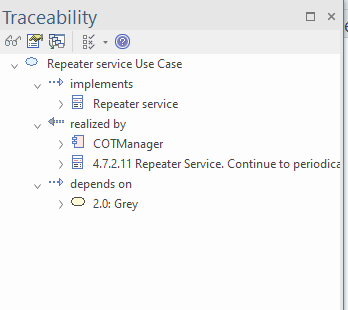
Chain Inheritance

The Chain Inheritance diagram contains the elements related to a single chain of parent- children.

As default the name you SHOULD use “<<ObjectName>> Inheritance”

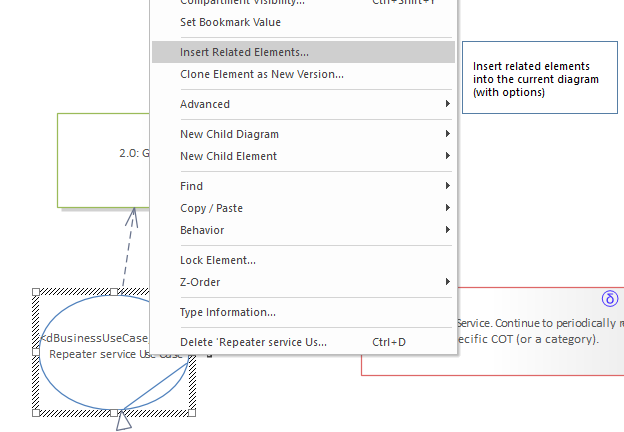
Illustration 7: Chain Inheritance diagram example

EA offers a function to follow the Inheritance chain by using the Traceability Window.

Figure 7: Traceability for the above diagram

You can direct drag objects from the Hierarchy and drop them in the Inheritance chain diagram.

It’s also possible to create such a diagram with Right Click on an object and select “Insert Related Elements*”*

Illustration 8: Show in Hierarchy menu

Deep View

A deep view diagram shows a special aspect of the knowledge model.

You can use any combination of objects here and you are limited from the readability (see seven rule below) only.

Use many deep views diagrams to present complex aspect of the knowledge model.

As default the name you SHOULD use “<<ObjectName>> relevant”

Example of Deep View are relevant objects diagrams (see [#2.2.1.2.Relevant nodes](#2.2.1.2.Relevant_nodes)).

Mapping

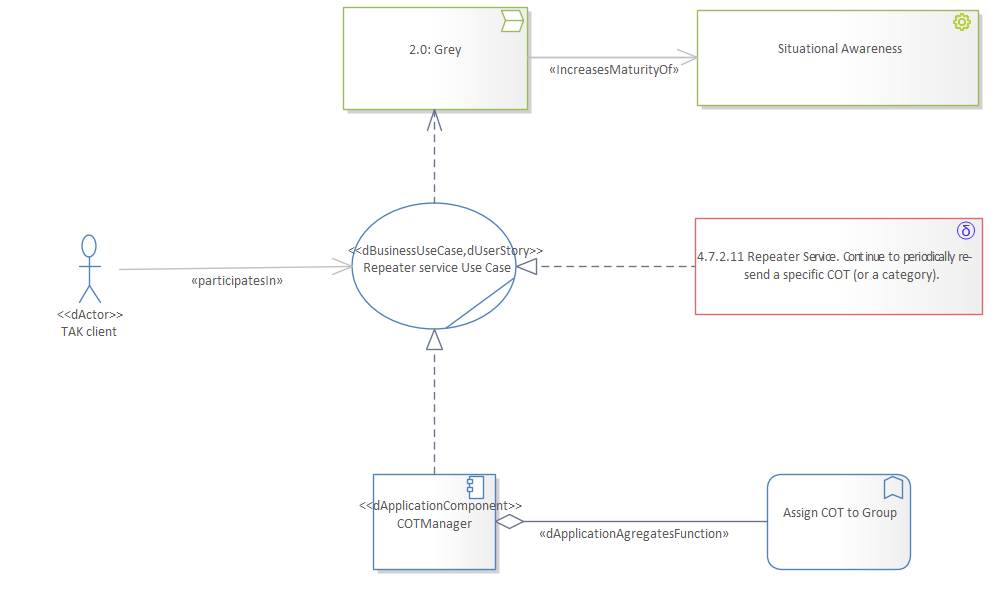
Mapping diagrams show how a certain type of object is linked to another one.

Example of mapping are ChiObject to ChiNode or Container to ChiNode Mapping (see below).

The “seven rule” for Diagrams

A diagram SHOULD NOT display more than SEVEN elements. A good diagram has only five.

See the Inheritance diagram below:

Illustration 10: example of the seven rule

Such a representation is already complex to understand.

Studies have shown that most people have a hard time visually comprehending more than seven elements at a time.

A diagram with more than seven element is a good method to **build** the knowledge model (see diagrams types above) but not to communicate it.

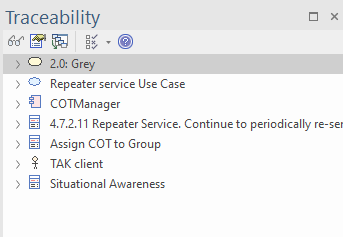
The DAF validator (see below) supports a check for diagrams with more than seven elements.

Traceability

An important feature of the Metamodel structure described above is to support traceability. Following Wikipedia, requirements traceability is defined as:

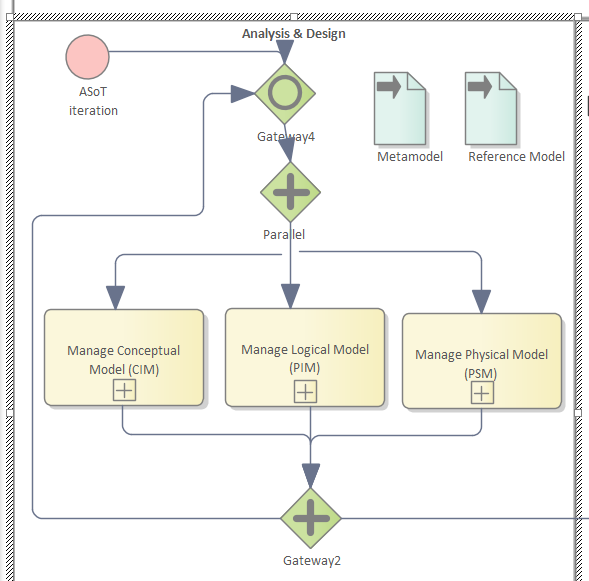
*... the ability to describe and follow the life of a requirement, in both forwards and backwards direction (i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases.*

EA supports this by building chains of model elements

Illustration 11: the traceability allows to explore connected elements

Handling projects easily with the DAF methodology

Overview



The main process of Digital Engineering, “Analysis and Design” consists of three sub-processes that are executed iteratively in parallel, typically by different roles. These processes include:

**Manage Conceptual Model (CIM)**

This sub-process involves creating a Capability Model, defining supporting processes, logical applications, roles, and related data entities.

Following that a Maturity Assessment is performed, and programs, growth packages, and initiatives are designed. Key Performance Indicators are assigned and the model is tested for consistency.

**Manage Logical Model (PIM)**

This subprocess involves creating a Requirement Model, deriving features, managing issues, creating the Use Case Model, and Application Model, and defining functions.

The model is tested for consistency.

**Manage Physical Model (PSM)**

This subprocess involves deciding on the Solution Architecture, a technical solution for the previous architecture, and the creating the Object Model, generating the PSM Class Model, enhancing the Class Model, and generating code.

1. Manage Conceptual Model (CIM)
   1. Create Capability Model
   2. Define supporting Processes
   3. Define Supporting Logical Applications
   4. Define Supporting Roles
   5. Defining related Data Entities
   6. Perform Maturity Assessment
   7. Design Programs, growth package and Initiatives
   8. Assign Key Performance Indicators
   9. Test the model
2. Manage Logical Model (PIM)
   1. create initiative’s Goal
   2. Create Requirement Model
   3. Derive Features
   4. Manage Issues
   5. realize the Use Case Model
   6. Model the Domain
   7. realize Component Model
   8. define Application Functions
   9. Test the model
3. Manage Physical Model (PSM)
   1. Decide Solution Architecture
   2. Create the Object Model
   3. Generate PSM Class Model
   4. Enhance Class Model
   5. Generate Code

Digital Architecture Framework Stereotypes and relationships overview

To begin we need to give a deeper look to Digital Architecture Framework.

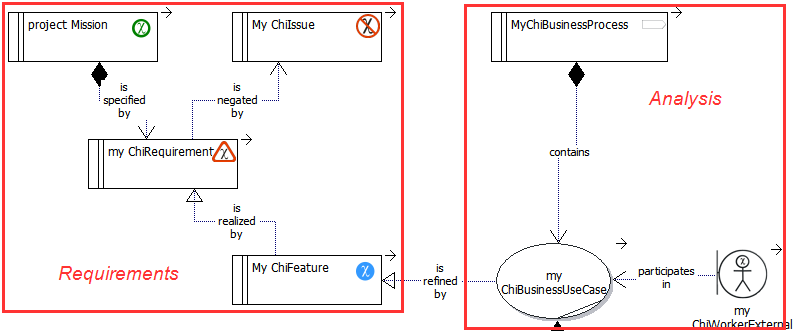
The DAF Metamodel contains the following stereotypes and relations:

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Analysis** | **Domain** |
| * ChiGoal * dRequirement * dFeature * dIssue | * dBusinessProcess * dBusinessUseCase * dWorker (6 types) | * dNode * dController * dView * dConfiguration |

Table 2: Digital Architecture Framework stereotype names

In the next chapters we will discuss in detail the sub models, their sequence and use.

Workflow of the 3 sub models

Illustration 16: Stereotypes and relationships for the Requirement and Analysis model

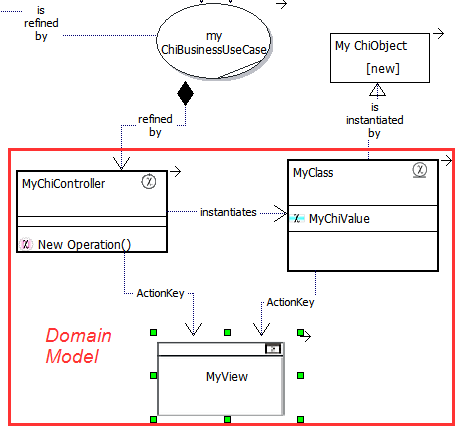


Illustration 17: Stereotypes and relationships Domain Model

Starting a project by selecting a model

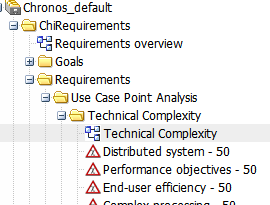
Depending from your settings, when you start a project, you may have different models to start with:

* Digital Architecture Framework Default Model that is freely available
* other default model that maybe available on request

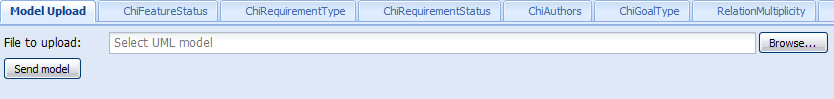
It is strongly suggested to start by duplicating one of them. While it is possible to create a model from the scratch, certain structures are necessaries to the generator.

Structure

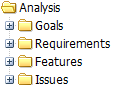
* (if not already done) Start the project by duplicating the provided Digital Architecture Framework\_default model.



* Optional: Import the provided Digital Architecture Framework UML default model.

Illustration 32: the "model upload tab" in the "admin" perspective allows to import a valid Digital Architecture Framework model

* Works in the given package structure.
  + Put every object in a package with the object name (e.g. all the goals under the package goals)



The Digital Architecture Framework Default model

The Digital Architecture Framework default model, is a template that permits to rapid start any Digital Architecture Framework project. It offers a precise structure of packages where place Requirements, Use cases and Domain objects. A single object of each type is available for customizing. It is provided in UML format.

See below for details about its structure.

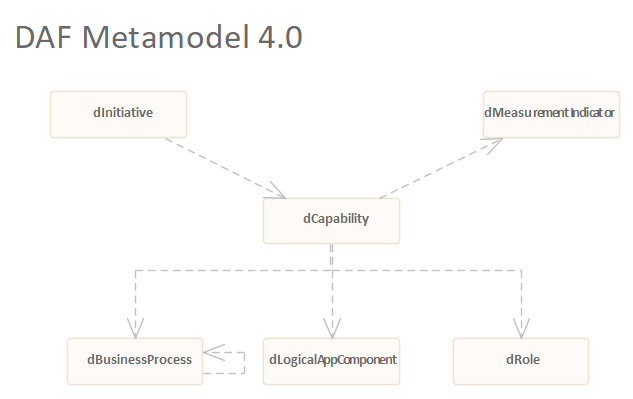
CIM Model

CIM Metamodel: overview

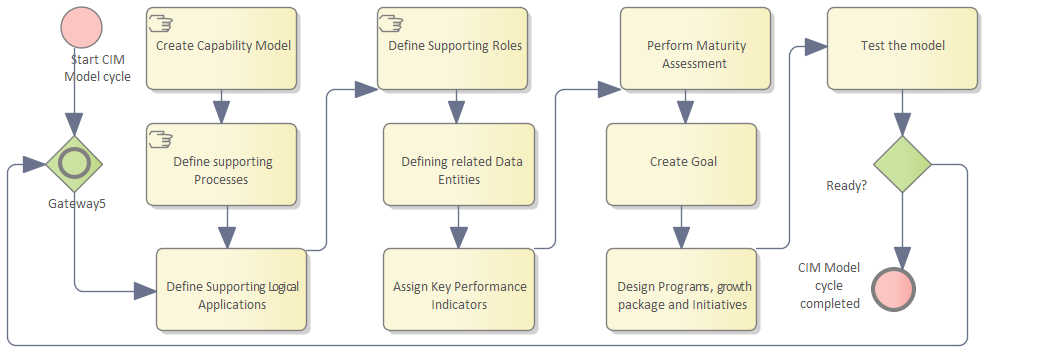
in the DAF model a **Capability** is supported by **Applications**, **People** (role) and **Business Processes.**

An **Initiative** is a set of actions that impact the supporting elements, typically aiming at improve the Capability Maturity Level.

In order to measure the maturity improvement, a **Measurement Indicator** (KPI) is attached to the Capability.

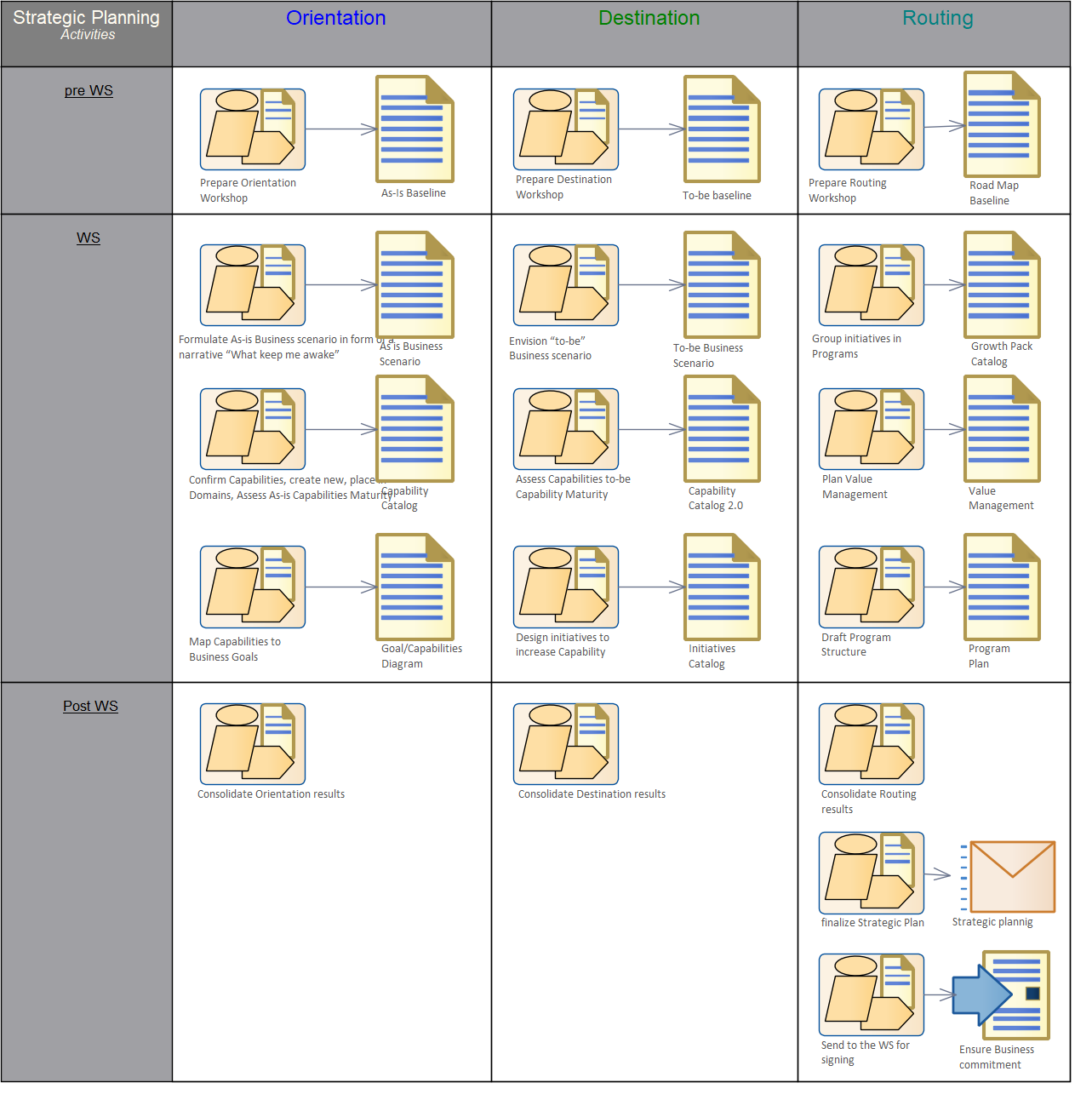


CIM process



Maturity Assessment

The Maturity Assessment is the part of Atrategic Planning value stream. It delivers a Program containing initiatives to move the current Capability’s Maturity to the desired level, considering which Concepts (a.k.a. Taxonomy) Tools, Process and Roles are currently involved, considers organizational aspects such as ‘Readiness for Transformation’, identify KPI’s for measurement before and after the proposed initiatives.



* + 1. Figure 8: DAF maturity assessment process

PIM

Using the PIM model in procurement

The Digital Architecture Framework methodology is designed to lead you trough a typical project. Before the delivery part starts, other activities like providing a quote need to be performed, typically as “bid budget”. Digital Architecture Framework helps you to minimize this effort. To complete the estimation two experienced consultant needs 1-2 days for a middle project with 10-20 Use cases. Additionally the created model can be reused during the project's execution.

Creating a Request for Proposal

Introduction

A Request for Proposal (RfP) is a collection of functional, non-functional and environmental requirements. Those are expected to be fulfilled by Features contained in the Response. Sparx EA supports collaboration and management of RfPs.

RfP: walk-trough

To prepare the RfP using EA:

* Begin modeling with the inner goal (Vision) of the project (optional)
* Describe the which kind of image (Mission) that you want to communicate to candidate contractors
* Describe up to 7 most important goals of the considered project
  + Assign priorities
  + validate goals by stakeholders
* Illustrate for each of the above goals 1 or more requirement. Requirements can be functional, non functional and environmental
  + assign priorities to Requirements
  + validate goals by a proofreader Subject Matter Expert (SME)
* Generate the RfP Document using the document generator
* Ask the contractors company to describe a Feature for each of the considered Requirements and an Issue for each of them that cannot be realized..

working with the Response to Proposal

* Let a SME proofreader validate each Feature: each of them must be response to a certain requirement fulfilling it. In case a feature a no requirement you can consider to add one or to simply ignore it as non relevant for you.
* Compare the different features in a matrix.

Introduction to Project’s estimation

Typically, before a project starts the team is requested to provide a quote to the client. The proposal team gets a so called Request for Proposal (RfP) documentation from the client. The quality of this can be very variable: from a simple list of requirement to more gigabytes of documents. The task that the team must perform is to read (and understand!) the RfP documents providing a technical and legal response. But the most important part is to produce a forecast of the effort involved in the realization of the client wishes.

In this chapter we describe an approach that uses the Digital Architecture Framework Methodology to categorize client requirements and produce a quality estimation using Use Case Point Analysis (UCPA). This is a method for project sizing and measurement based on Use Cases[[4]](#footnote-5) created by Gustav Karner.

How Use Case estimation works

Project's complexity is a function of the different factors.

* **Use Cases**: The number and complexity of the use cases in the system
* **Actors**: The number and complexity of the actors on the system
* **Technical complexity**: Various non-functional requirements (such as portability, performance, maintainability) that are not realized as use cases.
* **Environmental Factor**: Various non-functional requirements regarding the environment in which the project will be developed (such as the language, the team’s motivation, and qualification).

Use Cases Complexity Definition

As we build a project using UML, use cases to describe the proposed functionality. We assign a rating to each use case following those guide lines:

* **Easy** (5 points): fewer than 4 main success scenarios or execution paths in the UC. The use case is considered a simple piece of work, uses a simple user interface and touches only a single storage entity; its success scenario has less than 3 steps; its implementation involves less than 5 classes
* **Medium** (10 points): 4 or more key scenarios, but fewer than 8. The use case is more difficult, involves more interface design and touches 2 or more storage entities; its main success success scenario has between 4 to 7 steps; its implementation involves between 5 to 10 dNodes.
* **Complex** (15 points): The use case is very complex, involves a composite user interface or processing and touches 3 or more storage entities; its success scenario has over seven steps; its implementation involves more than 10 dNodes.

Those values are given for orientation only. Every Use Case could have a rating between 2 and 30 points. The amount of points can be calculated also for every package or set of functionalities (dBusinessProcesses)

Actors complexity definition

Karner's UCP method also calculates project effort by considering project actors, and their contributing complexity. An actor can be a human or a technical system.

As guidelines to this assignment we use the information supplied below:

* **Easy**(1-5 points): The actor represents another system with a defined API
* **Medium** (2 points): The actor represents another system interacting through a protocol or a human with a command line interface
* **Complex** (3 points): The actor is a person interacting via User interface.

Technical factor complexity definition

The technical complexity measures which non functional requirements are relevant for you project. The Digital Architecture Framework default model provides a set of Requirements and Features couples that you can customize. The weight factor is described in the Requirement that poses a question the is answered in the Feature Title and with a Value. Value and weight are multiplied and the result the summed as technical factor “Tfactor”. Tfactor is normalized as 0,6+(0,01\*Tfactor)

* **Requirement Weight** (2 points): this factor weights a strong negative impact on your project more that others. For example the definition of “secure” in a bank system very significant and costs effort.
* **Requirement Weight** (1 point): this factor weights a negative impact on your project .
* **Requirement Weight** (-1 point): this factor weights a positive impact on your project .
* **Requirement Weight** (-2 point): this factor weights a strong positive impact on your project more that others. For example in a bank system is not so important that the system is “easy to install“

Depending from the positive or negative Weight, the Feature's value can impact on the context. Generally speaking :

* **Feature Value (5 point)**: implementing this requirement represent a complex task
* **Feature Value (3 point)**: implementing this requirement represent a medium task
* **Feature Value (1 point)**: implementing this requirement represent a Easy task

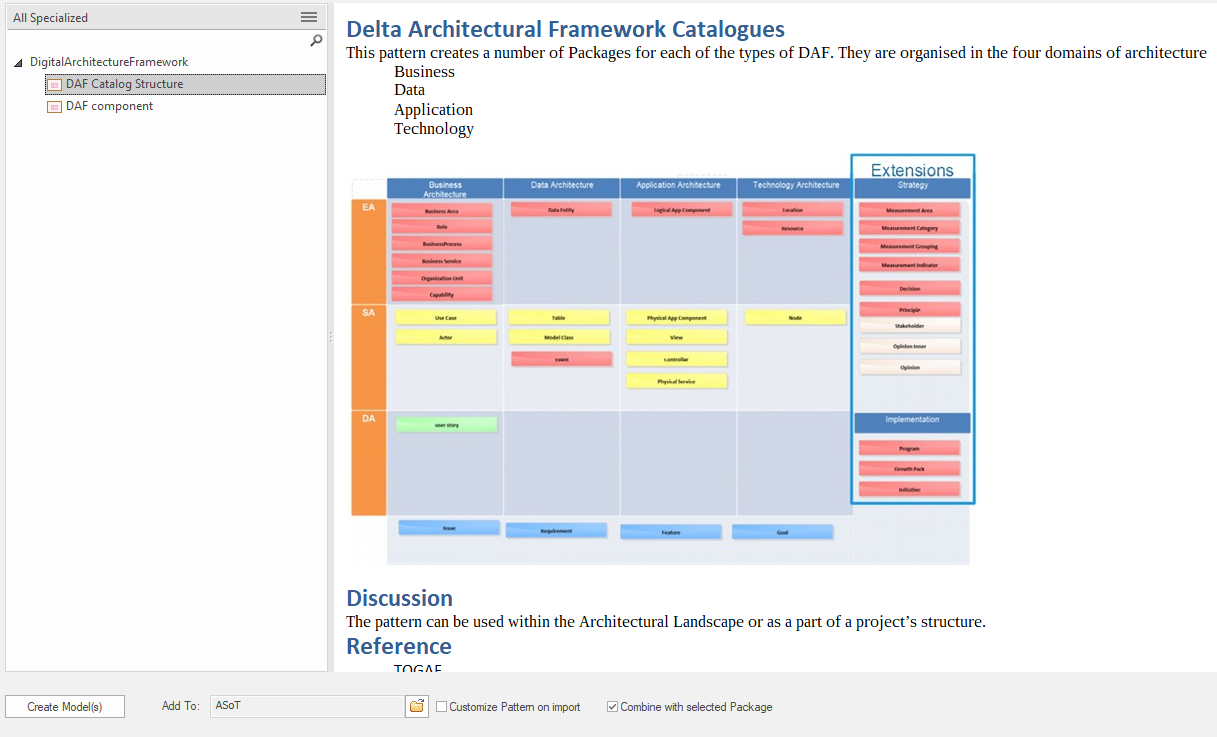
Environmental Factor complexity definition

This is one of the most under estimated (if not ignored) factors in project estimations. Its describe how qualified your team is, how committed to the project and . Notice that the provided set of Requirements and Feature are set to be used with DE. So they are more optimistic that in a traditional project. The weight factor is described in the Requirement that poses a question the is answered in the Feature Title and with a Value. Value and weight are multiplied and the result the summed as technical factor “Efactor”. Efactor is normalized as 1,4+(0,03\*Efactor)

see above for definition of value and Weight.

Use Case point Analysis Walk trough

Start by using the DAF project template (see [#3.2.Starting a project](#3.2.Starting_a_project) for details).



Estimating the Functional requirements

Find the **Goals** of your potential client: what is he intending to do? How will he measure your common success?

Now identify the R**equirements** that specify the **Goals**. After this, describe for each of them at least a realizing **Feature**. Features realizing functional requirements, needs to be further refined by Use Cases. If you have luck the client has already defined his use cases in the RfP document, if not you need to find them. The difference between a Requirement and a Feature sometime is very subtle. As general Rule a requirement leave always space for interpretation and realization while a feature describes it specifically.

For example saying “*the system must have an acceptable performance*” is a requirement while “*each business functionality will run in less that 1 second”* is his feature. Typically requirements are very high level and undefined, especially at the beginning of a project. Sentences like “*The client don't know what he need*” describes this situation. So in the client documentation you can expect to find more requirements for features that features self. It is the job of an IT architect to find the right solution.

For each functional Feature create at least a Use case (see also [Finding candidates for Use Cases](#1.12.2.Finding_candidates_for_Use_Cases)) and link them.

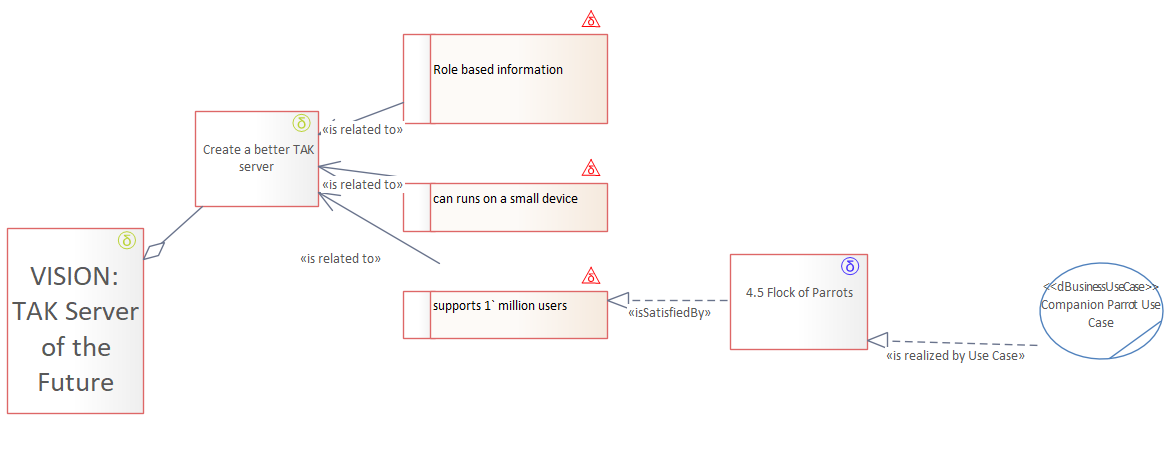
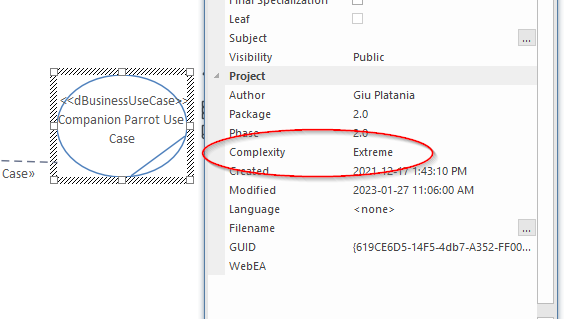


Illustration 25: Example of requirements model for estimation

Now for each use case estimate the complexity factor (see ).



The sum of the complexity factors is normalized against environmental factor and technical complexity.

Estimating the Non-functional and environmental requirements

The technical complexity is the estimated impact of Non Functional Requirements (NfR) over the project.

A standard list of NfR is provided in the Digital Architecture Framework Default model:

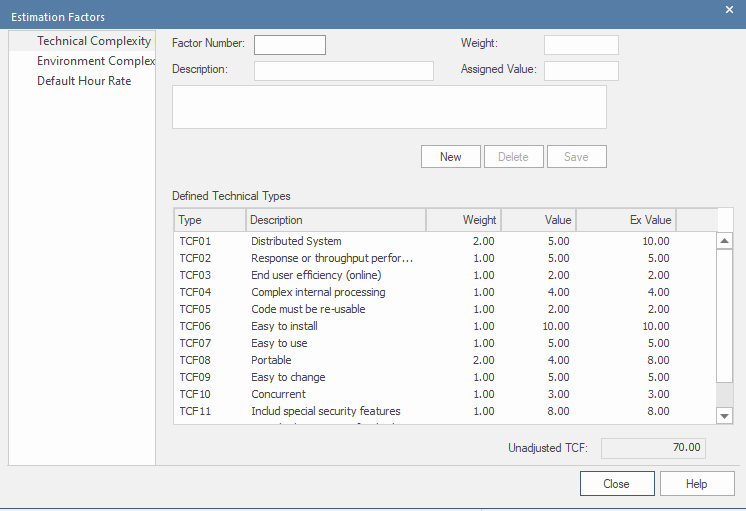
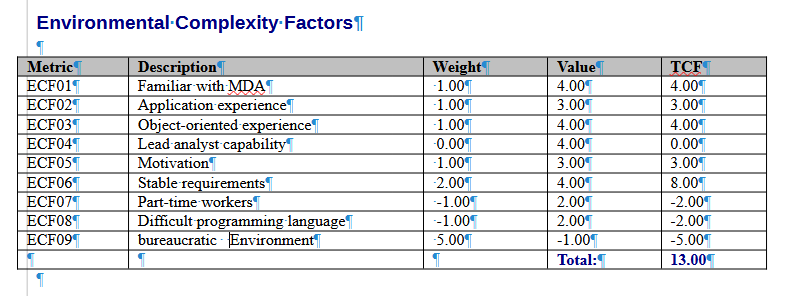


Illustration 26: a standard list of Non functional requirements and their weight is available in the default model.

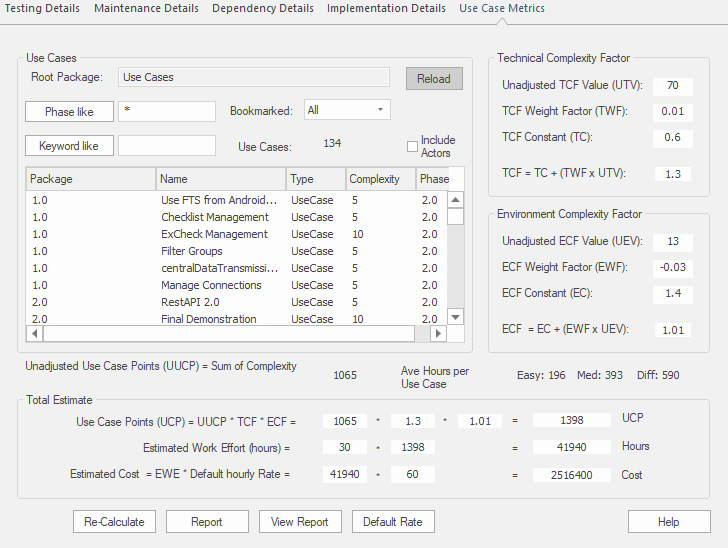
​Technical requirements and their features are all them that are not related to business functionalities (Non functional requirements). The weight of the requirement is expressed as integer by using the requirements field “priority”. Typically you don't need to modify the given default weight. Every feature that is not directly described by a use case must be attached to such a type of requirement. In the feature's note field write the actual Value of the feature expressed as floating number (“1,4” or “-2” are valid examples).

Figure 9: The weight and values are estimate assuming the use of Digital Architecture Framework as DE methodology.

The environmental factor is often underestimated in project calculation. It includes concepts like the motivation of the team, the stability of requirements or the methodology used for the project.

Generating a UCPA Document

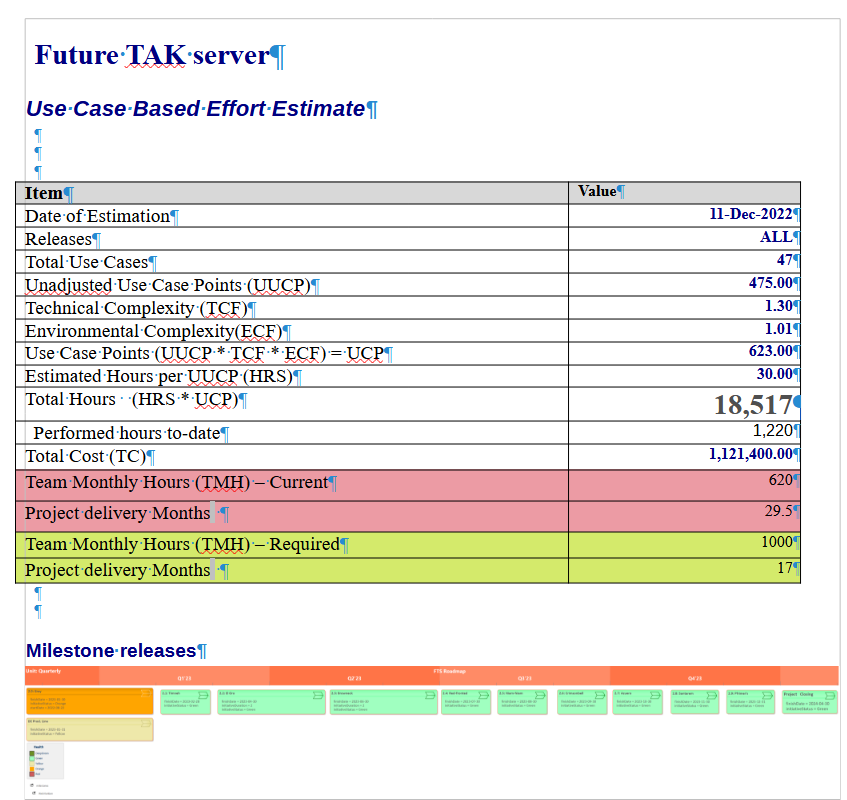
Now you are ready to generate the forecast.

Illustration 29: EA Use Case Metric view

The standard EA installation includes a UCPA analysis. It generates a document containing all use cases, actors, and Requirements with type "*Technical*" and"*Environmental*". Additionally, all formulas for calculating Use Case Point Analysis are included.

To generate a UCPA go to Construct > Project Management > QA > QA Reports > Use Case Metrics and press the “report” button

The result look similar to this:

Illustration 31: Use case point generated documentation

The generated document contains information regarding prices, total hour, person day and so on. Those are average data for the consideration of the user only. This must be considered a template that still need the attention of one or more subject-matter experts.

UCPA estimated tasks

After a Use Case Point analysis you get a first concrete estimation of the hours necessary to realize a certain set of functionalities.

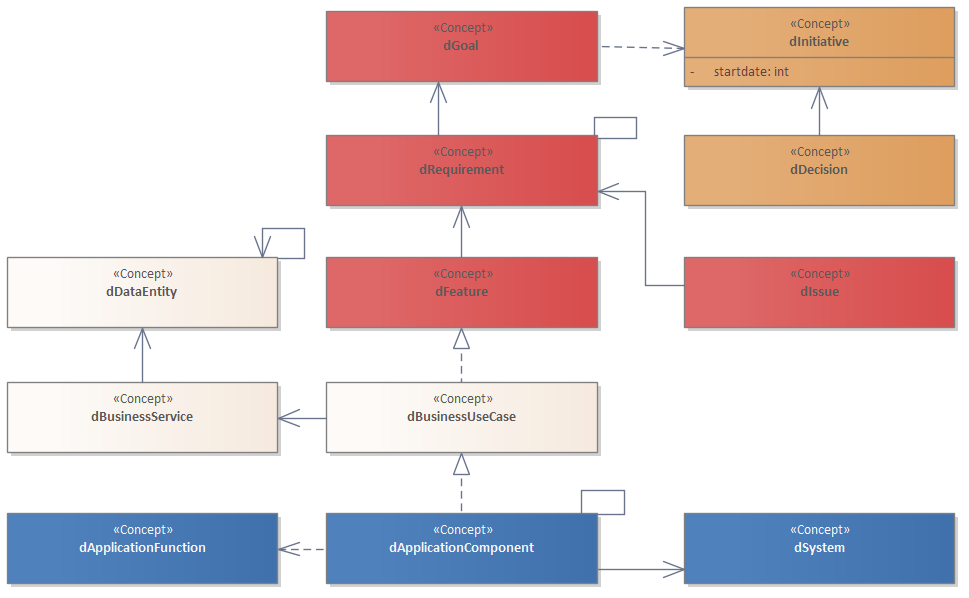
A frequent asked question related to this time is what activities are *inside* that calculation. UCPA includes the time requires for project management, design, development, testing and deployment. All the “standard” phases that are required to complete a project are included. Nevertheless you can decide to add times for special activity if is required by the special situation.

UCPA Conclusion

The use case point analysis methodology is not the silver bullet that “*automagically*” solves the challenge of estimating the unknown. As always the experience of the estimator is the decisive factor for the final quality. At least as “second opinion” cannot be wrong to apply it. Nevertheless the normalization parameters and the precise approach offer a defined path for project estimation.

Within the Digital Architecture Framework project's strategy the Use Case Point Analysis offers an additional advantage: being the project itself based on DE (Digital Engineering), the same model created during the proposal phase can be reused.

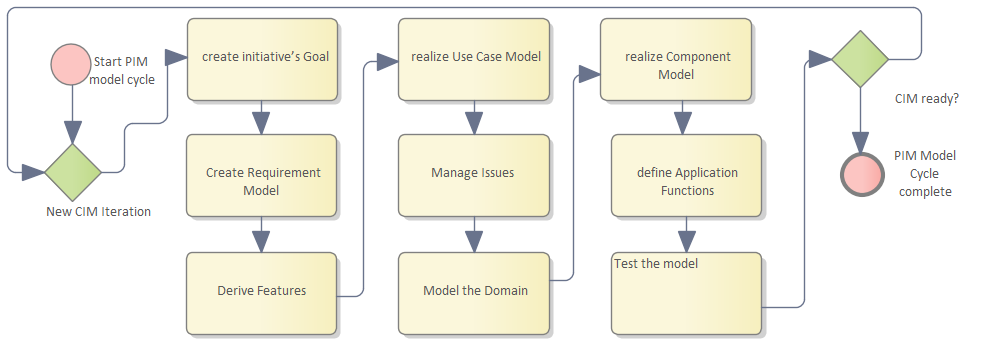
PIM Metamodel Overview



PIM Process

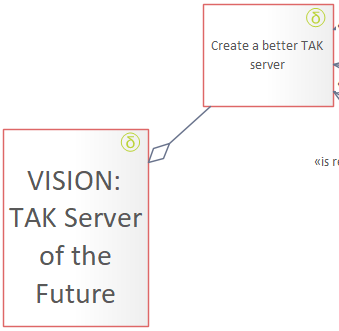
In this section we presents the Digital Architecture Framework elements used in the CIM. Depending from your role, you can use all or only a some of them. The objects in the hierarchy's bottom are more defined that the one in the “top”.

For a complete reference of them consult the “Digital Architecture Framework Profile Documentation”.



Create initiative goal

Begin by defining the Initiative Goal(s). A goal is a desirable (and measurable!) status.



Create Requirement Model

The next steps is to define Requirements specifying the Goal. A dRequirement

is a condition that can be true or false.

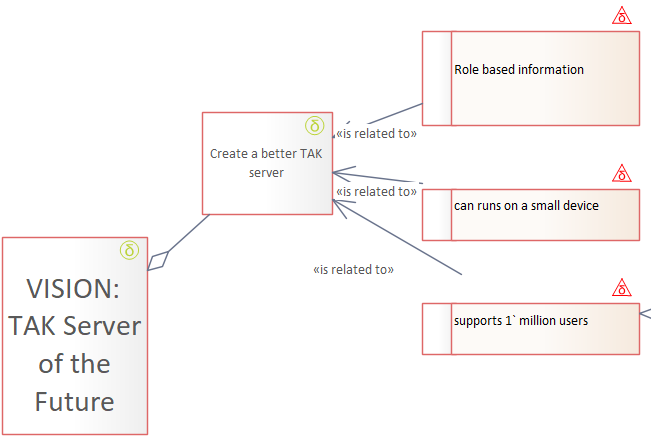


Illustration 18: this goal is specified by 3 different Requirements

Derive Features and Manage Issues

Requirements are realized by Features. A **Feature** is a fact that set his requirement to true. In an opposite way the **Issues** are facts that set the requirement to false.

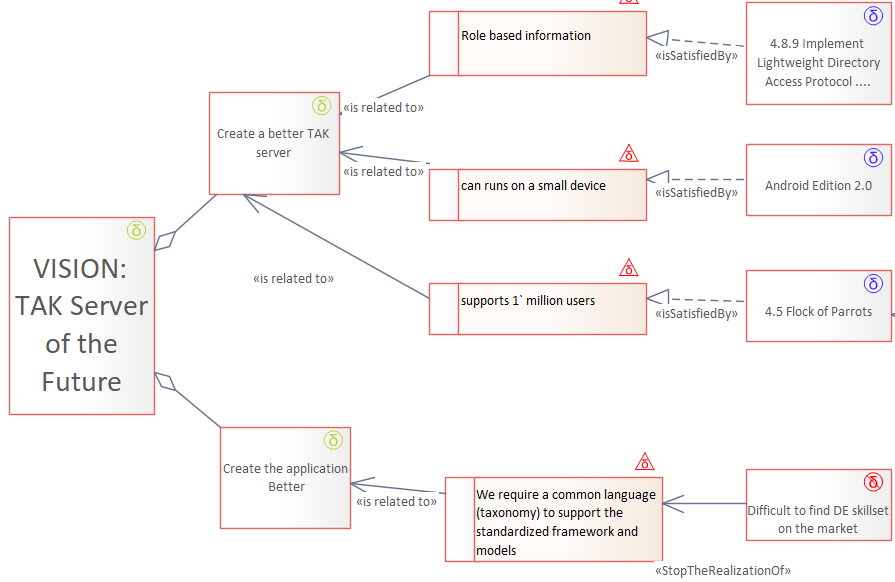
Realize Use Case Model

Illustration 19: elements of the Requirement sub model

A Feature is realized by one or more Use cases. dBusinessUseCases are short stories, containing a step by step descriptions how a feature works.

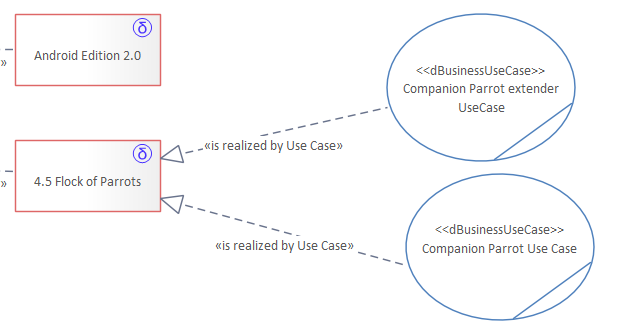


Illustration 20: two use cases refining the same Feature

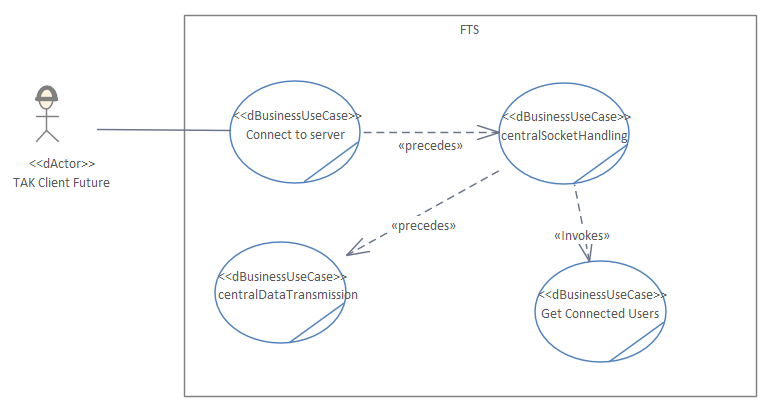


Illustration 21: Example of a system FTS containing four Use cases

Use Cases are execute within the scope of a system. You can imagine the system as telling the complete story where a Use Case is a chapter. Actors like Workers and

Partners are the protagonists of the story that participate in Use Cases.

Defining the Use case flow

The “story” of a Use Case is told in his “note” field; additionally the use case may include an activity, that visually describe the story, for facilitating the reading and generating documentation. The activity can be created from a diagram of by using the scenario

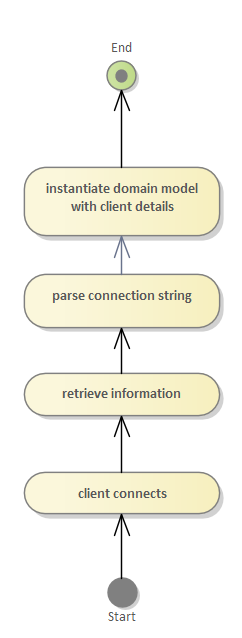
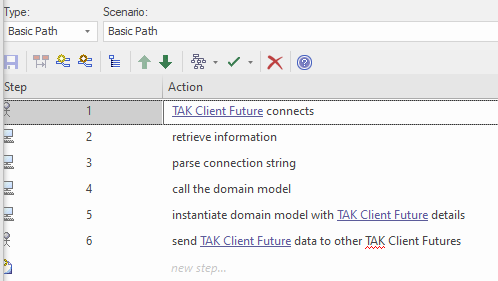
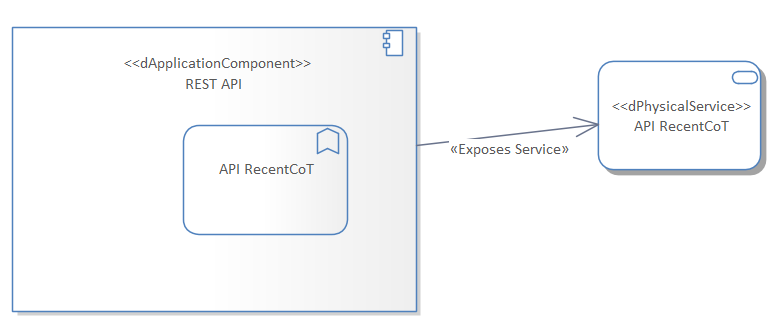


Illustration 22: Activity of the "insert new movie" Use case

Figure 10: With the Properties window open, switch to the Scenarios section.Click on the Structured Specification tab.Review the existing steps for the Basic Flow.Select Event Full from the Scenario drop down list.Review the steps for this alternate flow.Press Generate Diagram on the toolbar and select Activity from the drop down.EA generates an activity diagram based on the scenarios in the structured specification.

Defining Components and Functions



A function that is set as a „public“, should be related to a technical service

Using Organizational Principles to take project’s decisions

Architectural principles are universally considered to be crucial in providing high level guidance to Architecture practitioners, however often they remain an exercise in futility because they are not applied to any concrete case.

In order to avoid this lack of applicable effects in DAF Principles are associated to Architectural Decisions, the latter being taken during Initiatives.

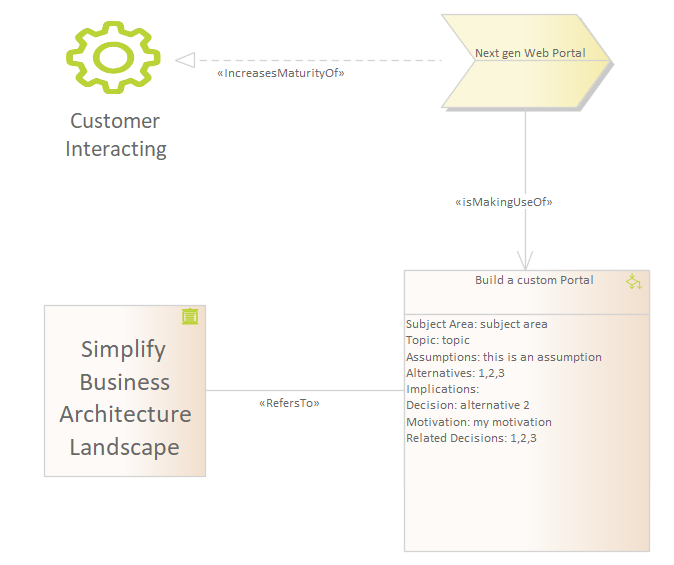
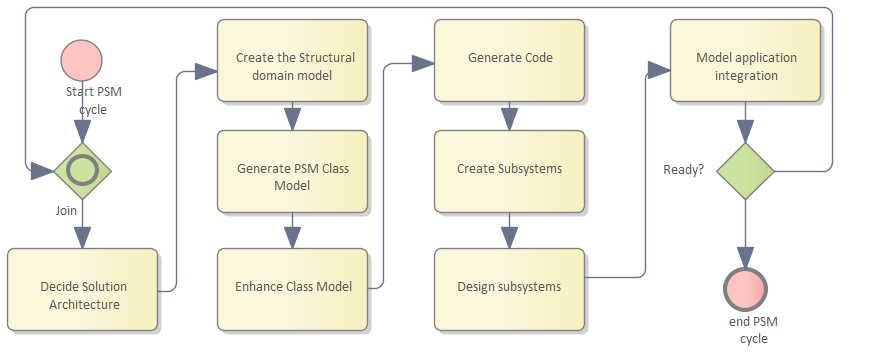


Figure 11: The Principle of Biz architecture simplification is applied in the decision to build a custom portal within the initiative “Next Gen Web Portal” that would enhance the maturity of the Biz Capability “Customer Interacting”

PSM

The scope of this sub model is to host the identified business domains that is required to instantiate the analysis model.

PSM Process



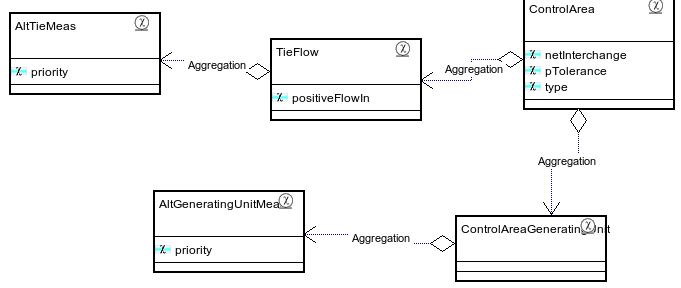
* Create activity model
* Create partitions
* Add actions
* Add objects
* Add object flows
* Add control nodes
* Add control flows and guards
* Create Subsystems
* Design subsystems
  + Create sequence diagram
  + Add lifelines
  + Add messages
  + Model subsystem participants
  + Describe subsystem dependencies
* Model application integration
  + Create information flows
* Organize model

Introduction: Model View Controller

The Digital Architecture Framework Meta model uses Model View Controller (MVC) as an approach to conceptual model domain knowledge. The MVC pattern is a widely used approach for describe concepts separating functionality among objects as to minimize the degree of coupling between them. To achieve this, the pattern divides the domain into three layers: Model, View, and Controller.

Digital Architecture Framework expand this pattern to use it outside from a technological environment so that an abstract domain can describe from a business point of view.

Each layer handles specific tasks and has responsibilities to the other layers:

Illustration 74: CIM domain objects as ChiNodes

* The **dNode** (Model) represents domain objects, the “nouns”. In the SID standard model, classes like ControlArea or TieFlow are dNodes. It describes the identity, property of the BO. Operations, apart common object manipulation (CRUD), are included only if they govern exclusively the access and modification of the considered dNode[[5]](#footnote-6) .
* The **dController** (controller) represent the action of your domain, the verb.

It defines application behavior and the Business Logic that involve more that one dNode. It dispatches requests and selects views for presentation. It interprets user inputs and maps them into actions. A **dController that is has the visibility set as “public” exposes a service to external system**

* The **dView** (view) represent the visible context, the page where the “sentences” are displayed. It displays the contents of a Model and the results of Controllers operations. Any kind of visualization can be modeled as a view.  
  It gets data from the model and specifies how that data should be presented. It updates data presentation when the model changes. A view also forwards user input to a controller. A view cannot execute business logic apart syntactical validation.

Create the Structural domain model

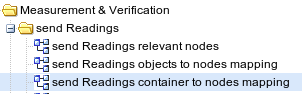
* This section describes how to create the Domain Sub model
* Browse to the Domain Model package



* Create a package structure composed by

*Universal company URI*, project specific definition , Business Process, **use case Name**  
for example:

*org.olympos.filmology*

Illustration 75: required Diagrams in the Use case package.

* For each use case package you should create 3 diagrams:
  + **Relevant Nodes to display the relevant dNodes (see** [**#2.2.1.2.Relevant nodes**](#2.2.1.2.Relevant_nodes)**)**
  + **Object to Node**: to map the dObject to the dNode (see [#2.2.1.6.Mapping](#_toc1279))
  + **Container to Node**: to link relevant dNodes with the Request and Response Containers.

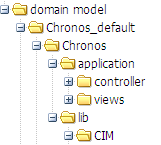
Mapping dObjects to dNodes

In a previous section you created Activities containing dObjects. Scope of this section is to link each of them with a dNode.

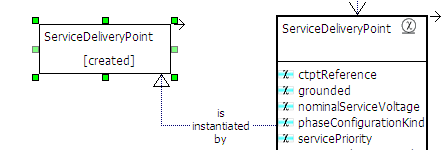
* Create a new diagram call “*Mapping object to node* ” (see also [2.2.1.6 Mapping](#_toc1279))

Using the alphaCORE default model

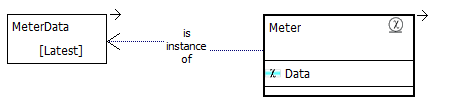
* Browse the CIM Lib

Illustration 76: the CIM lib in the default model

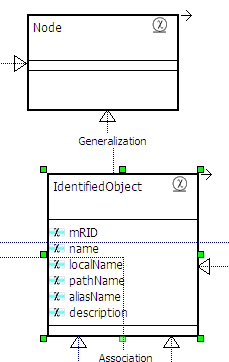
* Consult the list of existing CIM classes.
* Link a dNode from CIM library with every dObject in the Use Cases ActivitySet diagram.

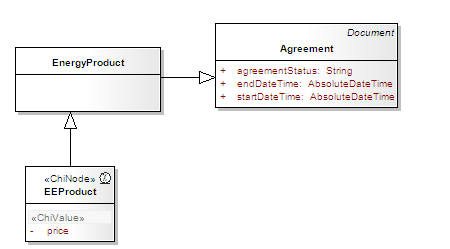
Illustration 77: example of instantiating a ChiNode from CIM library with a ChiObject

* In alternative you could create a new specific dNode for every UML::dObject that you have identified in activities. This should be done in conjunction with an architect.

Illustration 78: a new meter ChiNode is created to support the instance of the ChiObject

* See below an example of class inheritance hierarchy
* (if a new dNode was required) check if is possible to add library inheritance. In case the new object is a logical extension of the library. Most CIM objects inherits from IdentifiedObject that is already connected to Node in the standard model

Illustration 79: IdentifiedObject Inheritance from Node

Illustration 80: In this example EEproducts extend the CIM class EnergyProduct to add the property "price"

* (if a new dNode was required,) that is unrelated to CIM it SHOULD inherits from org.olympos.energy.common.core.EntityBase
* (if a new dNode was required, see above) Consult the list of existing dNode to establish relationships.

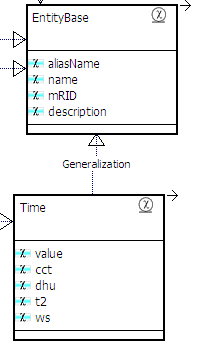


Illustration 81: EntityBase contains the same properties as the CIM class IdentifiedObject. Must be the super Class of all the new created dNode within a Energy & Utility project.

* (if a new dNode was required, see above) Model dNodes class's properties as dValues
* set the fields of the dValue. See table below:

| Tagged value | Meaning | Values | Default value |
| --- | --- | --- | --- |
| app\_data\_type | The attribute's application datatype. This can be used in the application to group attributes and execute special logic on them. | 0 | DATATYPE\_ATTRIBUTE |
| db\_data\_type | The atribute's database type. This will be used in the table definition. | according to the database  e.g. INT, VARCHAR, TEXT, ... | VARCHAR(255) |
| is\_editable | Declares, if the attribute is editable. | false | true |
| input\_type | Definition of the attribute's input control in the HTML form. | text, password, textarea, select, radio, checkbox, file, fileex, fckeditor, filebrowser, linkbrowser, date  + additional definitions | text |
| display\_type | The HTML display type for the attribute e.g. image[[6]](#footnote-7). | 0 | text |
| restrictions\_match | Regular expression, which must be matched by the attribute value. | e.g. [1-2][0-9][0-9][0-9] for years |  |
| restrictions\_not\_match | Regular expression, which must not be matched by the attribute value. | e.g. [a-z] for years |  |
| restrictions\_description | A text describing the restrictions, which will be shown in case of an error. | e.g. “please insert a year in YYYY format” |  |
| column\_name | The name of the database column. If not given the attribute name will be used. |  | attribute name |

Table 3: fields of a dValue

* DAFgen cartridges support a “convention instead of configuration” pattern. This means that also if you let all those properties to the default a running application will be generated from the model. On the other side, for fine tuning will be necessary to edit most of them.
* (optional) Model dNode statuses as UML Enumerations

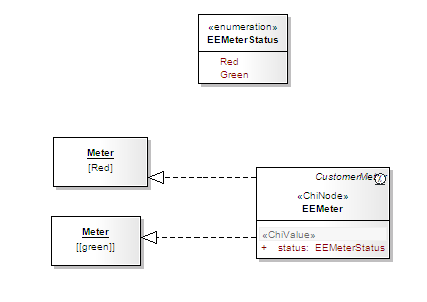


Illustration 82: enumeration are not directly supported by EA

* Model dNodes classes operations: Operations, apart common object manipulation (CRUD), are included only if they govern exclusively the access and modification of the considered dNode only .
* Describes the relation between dNodes

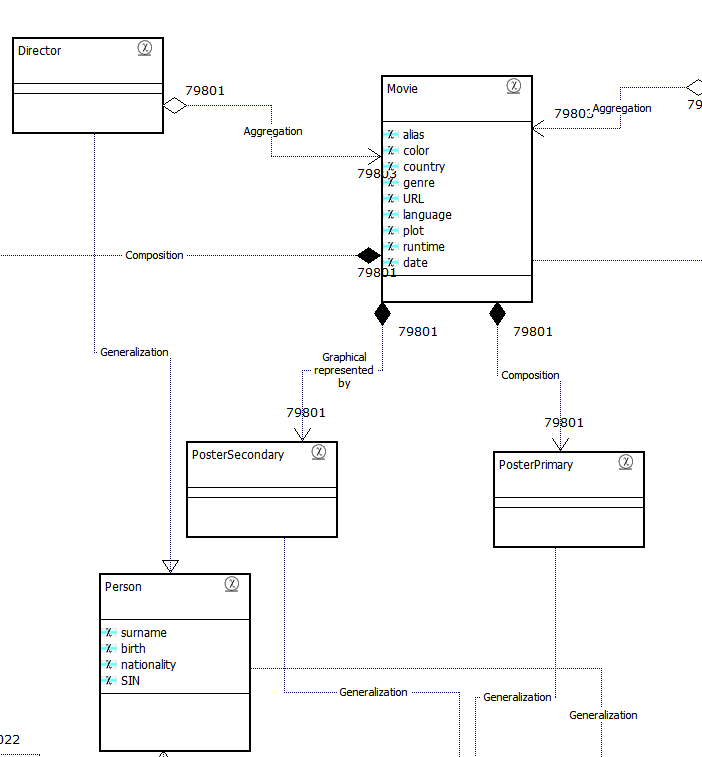


Illustration 83: ChiNodes with different relationships

* set the properties of the nodes

| Property | Meaning | Values | Default value |
| --- | --- | --- | --- |
| initparams | Name of the configuration file's section, in which the initial parameters for the corresponding mapper are defined |  | database |
| is\_soap | Define if the type should be exposed to the SOAP interface. | true|false | true |
| orderby | Definition of default sorting. Possible values: 'none' (no order), 'sortkey' (generates a 'sortkey' column, that is used for explicit sorting) or any the name of any dValue defined in the node optionally followed by [ASC|DESC] e.g. 'name ASC'" default="none" | none|sortkey|any value | none |
| is\_searchable | Indicates whether this type should be included in the default search. | true|false | true |
| table\_name | The name of the database table. If not given the name of the class will be taken. |  | class name |
| pk\_name | The name of the primary key column. The generator will add this automatically if there is no appropriate attribute. | a single value or '|'- separated list of values (e.g fk\_user\_id|fk\_role\_id) | id |
| child\_order | The order of the associated children e.g. for Recipe: Image|Info|AdminInfo. | a single value or '|'- separated list of values |  |
| parent\_order | The order of the associated parents. | a single value or '|'- separated list of values |  |
| display\_value | The dValue(s) to display in a list. | a single value or '|'- separated list of values |  |

Table 4: fields of dNode

* Distribute the dNode in packages that reflect the consider domain (see above). A mirrored CIM package structure can be used here as start.
* (Optional) Decide which Domain objects needs to be exposed as services. The domain object that are involved in UC activities are exposed as service.
* Create a new diagram called <USECASENAME> relevant Nodes (see also [2.2.1.2 Relevant nodes](#_toc1248) )

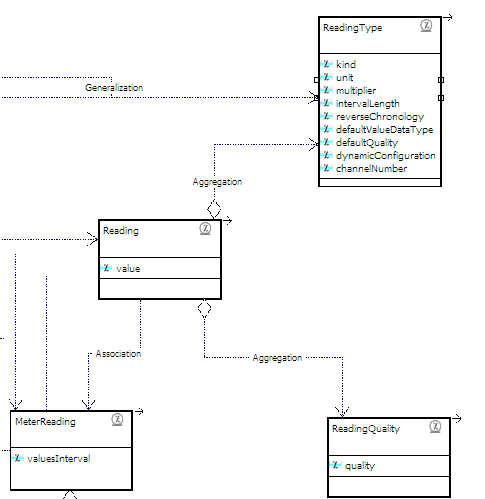


Illustration 84: relevant node diagram example

* (if required) Gather exiting systems, applications, and databases information in a deployment model.
* Plane a deployment model of your servers.
* Gather planned systems, applications, and databases information in a deployment model.

Mapping two dNodes

The Digital Architecture Framework Web Modeler supports the mapping of dNodes. This is useful in case you need to map between different concepts (E.g. an external and internal system). In the alphaCORE architecture the message broker is responsible for the physical mapping between the internal and the external system.

In the model the mapping is done by linking two dValues with each other. By doing this an additional stereotype, call dValueRef, is applied to the source dValue.

How to Maps Two Domain

1. Model the Source Domain.
2. Create a Container that hosts the Source Domain (see [#2.2.1.2.Relevant nodes](#2.2.1.2.Relevant_nodes)).
3. Model the Target domain.
4. Create a Container that hosts the Target Domain.
5. Create a new mapping diagram (see [2.2.1.5 Mapping](#_toc1279)).
6. Drag and drop in the Diagram a class from the Source and one from the target Model in the new created mapping diagram.
7. Select an attribute as source, drag the arrow to the target attribute. Your diagram will looks like this:
8. In the source Attribute 2 additional tagged values are automatically inserted: reference\_type contains the Entity type, whose attribute is referenced; reference\_value contains the entity type's referenced attribute.

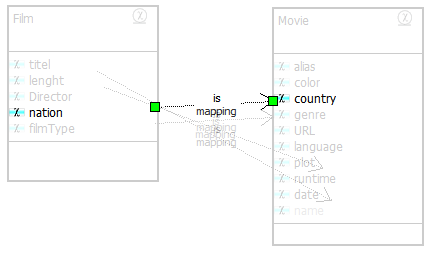
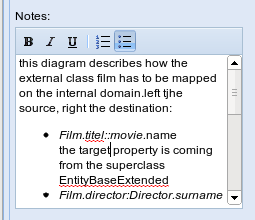


Illustration 85: example of mapping between properties: “nation” is mapped to “country”

1. Describe in the diagram “notes” special characteristics of the mapping.



Create the Behavioral model

A service model is a part of the Domain model. Following steps needs to be accomplished:

1. Develop common semantic based service payload definition in the form of the UML diagram
2. Write model to services transformations in WSDL form.
3. Document the service definition artifacts in one document for a given business integration (interface) area
4. Develop/Elaborate To-Be service orchestration aggregating existing dControllers

Implement Controllers

A use case is implemented by a dController, that is the domain element that host the flow of the story depicted in his Use case, each uml::Activity element becomes an operation of the controller.

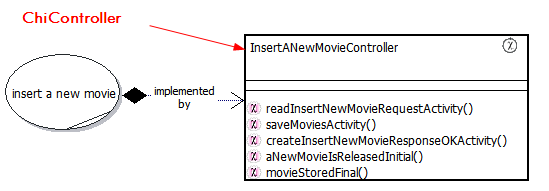


Illustration 23: the ChiController that implement the Use case has operations that correspond to the UC's the activity elements.

dObjects within the activity are instantiated by dNodes

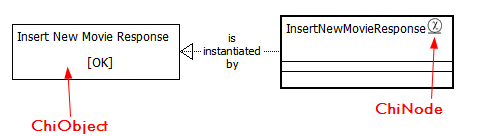


Illustration 24: a ChiObject instantiated by a ChiNode

Digital Architecture Framework and Logic

In the Digital Architecture Framework Metamodel a d**Requirement** is *realizes* by a **dFeature**. The **dFeature** is *refined* by a **dBusinessUseCase** (or **dBusinessUseCaseCore**). The **dBusinessUseCase** (or **dBusinessUseCaseCore**) is *implemented* by a **dController**.

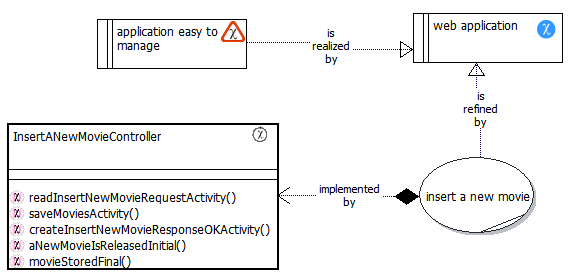


Illustration 86: flow of the implementation from a Requirement up to his Controller

dControllers host one or more dRules. a dRule can be of 4 types (actually only one is supported, see below).

An OOperation represents a condition in which one (or more) OEntity is modified. It can contain a Rule.

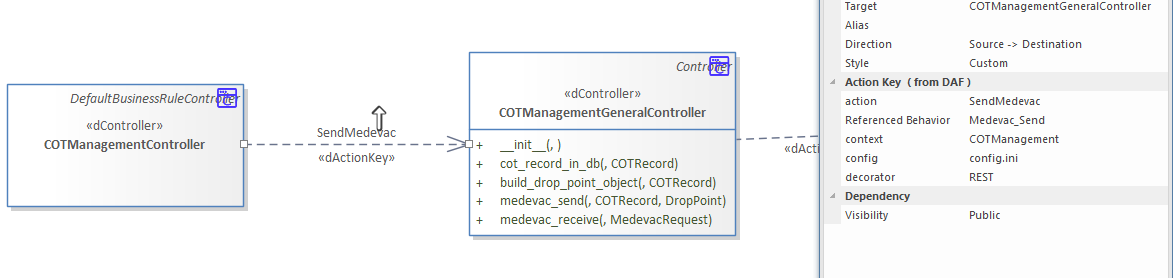
In the Implementation Digital Architecture Framework we divide logic in 3 parts:

* External Workflows
* Internal Workflows
* Atomic rules

External Workflows

The External workflow, is the sequence of Actions across Use Cases, translated as a sequence of **dController** linked by an dependency with the stereotype **dActionKey**.

| **dActionKey** |  | Associates two ChiController instances (to define a control flow) or a ChiView with a ChiController (to define a view attachment), must be one directional |
| --- | --- | --- |
|  | action | The action, which is triggered by this Dependency. If empty, any action is valid |
|  | context | The context, in which this association is valid. If empty, any context is valid. e.g. when 2 controllers are linked twice |
|  | config | The configuration, in which this association is defined |
|  | Method | a.k.a “behaviour”. The name of the method executed by the action, can be empty if the method name is identical to the action name |
|  | decorator | Allows to dynamically attach special behaviors to the result operation without changing his implementation . e.g. a decorator REST, will expose this endpoint as a REST service. |

Figure 12: example controllers with ActionKeys

This special class diagram is the way that Digital Architecture Framework uses to describe the *implementation of* a **dBusinessProcess**, or sequence of Use cases (see below)**.**

Internal Workflow

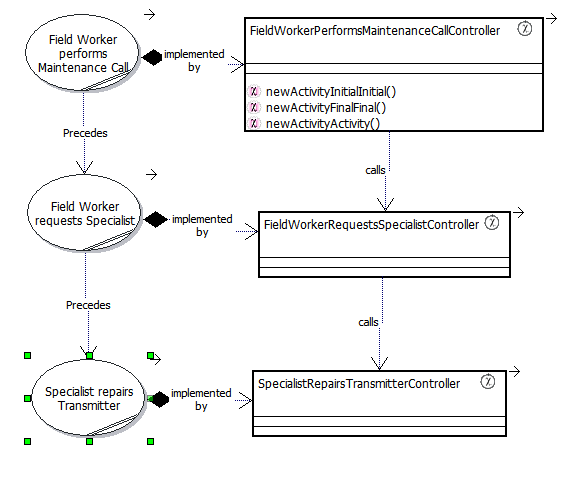


Illustration 87: use cases implemented by controllers

An internal Workflow is a sequence of decisions within a dApplicationComponent.

Controllers are linked to each other with an Action Key.

Rule

The next level is a sequence of rules that are present implemented by an Use Case.

An Action cannot be further broken down or decomposed. The atomic rule part is what is expressed in a single Method within a class. This is modeled as Action with a CallOperation.

| **dRequirements Model** | **dAnalysis Model** | **dDomain Model** | **Java** |
| --- | --- | --- | --- |
| Package | dBusinessProcess | Sequence of dControllers | A sequence of Java Calls defined in a XML Configuration file |
| dFeature | dBusinessUseCase | dController | Java Class that extend a Controller Class |
| dFeature | Text in a dBusinessUseCase | Production Rule | Method within the java class above |

Table 5: definition of logic in different stages

Modeling business rules

This part is not yet (EA 0.9) full implemented in the tool it is included for future reference only.

An dRule in a Digital Architecture Framework model hosts a place a in which one (or more) dNode may be transformed following certain constrains.

Following business rules are partially supported:

* Production Rules

Following business rules are not yet supported:

* Integrity Rules
* Derivation Rules
* Reaction Rules

Definition of dRule (PRR)

A Production rule consist of two parts: a sensory precondition (or "IF" statement) and one or more actions (or "THEN"). The condition evaluated should be part of a business variable. One or more Production rules are embedded in a RuleSet.

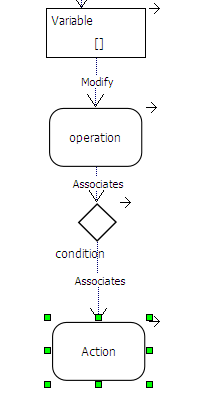


Illustration 88: example of Production rule modeled with activity elements

Production Rule Limits

Production Rules DON’T support AND – OR operators.

modeling controllers walk trough

* In the appropriate package based on *Universal company URI*, project specific definition , Business Process, **use case Name.** for example: *org.olympos.filmology* create a new diagram
* Right click on a Use case.
* Select “Generate Controller” a new controller is generated that refines the Use case

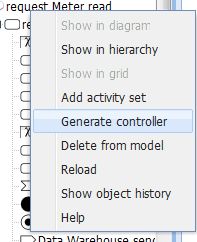


Illustration 89: the Use case context menu offers a generate controller feature

* Draw the original Use case and the Controller on the diagram

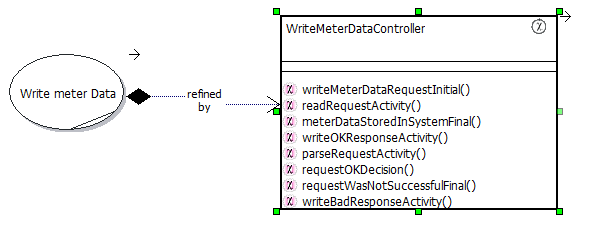


Illustration 90: Use Case refined by a ChiController

* Move the generated controller with drag and drop in an appropriate package (see above)
* Set the controller “visibility” field as *Public*



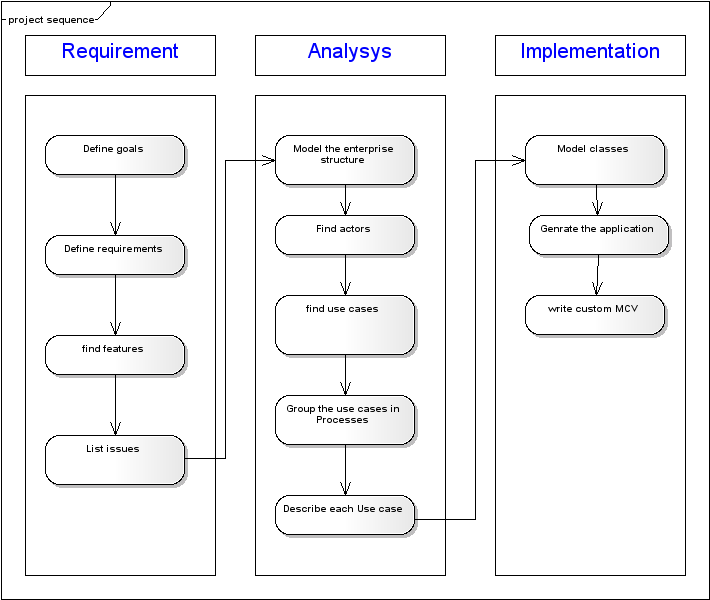
* (If necessary) change the names of the Operation in the new dController
* **NOTE:** Before you change the Controller information check if would be possible to remodel the parent Use case and regenerate the Children dController.

Sequence of use

A project based on this methodology uses the stereotypes described before following the structure of the Digital Architecture Framework Model (see [#3.2.1.The Digital Architecture Framework Default model](#3.2.1.The_Chronos_Default_model)) that is divided in 3 parts:

* Requirements sub model
* Analysis sub model
* Domain sub model

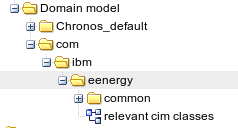
logically they are used in sequence as described in the diagram below.

Image 1: the waterfall Project sequence, starting with the Goals and finishing with the Domain objects, is the exception rather that the rule.

This work flow is constructed like a waterfall project. That is for representation only, rather that a methodology feature. In real life this sequence seldom materialize.

Rather several people will works on the same time on all the part of the sub models. This could appear very chaotic at the beginning. But, thanks to the Model Validator it is possible to connect the different pieces. The Validator (see [) will help you to go toward an increasing finishing of the model.](#7.The_Model_Browser)

Expect to have multiple iterations with different grades of completion.

Illustration 39: location of the e-energy domain objects

Configuring the model

To configure your EA installation for the project some steps are required. Some of them are:

* create users and groups in EA
* Assign rights to the groups
* (Optional) import an existing model
* Customize object statuses for project purpose
* Create a list of model Authors

for more information how to administrate the Business Object Repository (BOR) see his Manual referenced in appendix [#2.Additional reading|outline](#_toc1296).

The Requirements sub model

In this section we will cover in details the creation of the requirement sub model. This contains:

* Goals
* Requirements
* Features
* Issues

creating dGoals

* Define goals for your project.
* Define goal of type “*vision*”. This is what you intend to achieve in you project at a very high level.

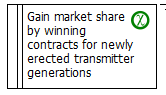


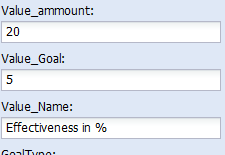
Illustration 40: A goal of type vision is an abstract **inner** representational of the project scopes.

* Define one goal of type “*mission*”. Those aims to “political” communication or “external marketing”. For this reason don't need to be specific a special value.

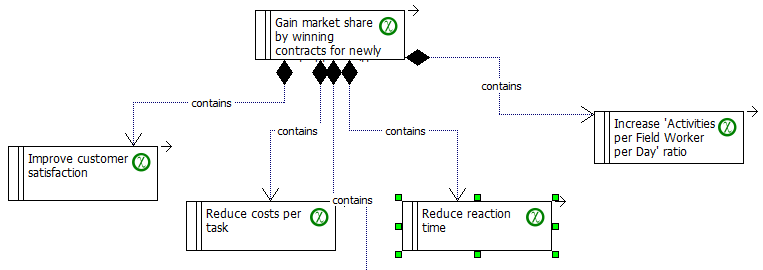


Illustration 41: a Goal of type Mission is the project slogan, how the Vision is communicated to the outside

* Create a number of specific goals that specifies your Vision. For each of them specify a value that you can measure. Specify in Value\_name the name and the type. Add in value amount the actual level and in Value\_Goal what you intend to achieve with this project. Don't exceed with the goals amount.
* Remember that all the goals must be achieved at the project end. 7 is a good number of project goals.

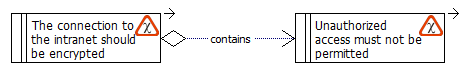
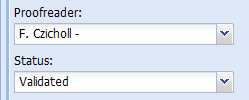
Illustration 42: in this example of goal value the actual effectiveness in % the actual level of 20 should be increased by 5 %

* Link all the goals in a Hierarchy.

Illustration 43: a Hierarchy of goals. the top element of it must be the Vision

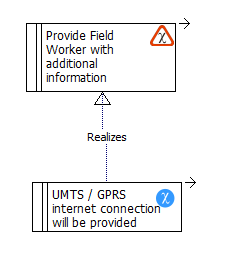
* each goal MUST have as a child one or more goals OR Requirements.
* A goal CANNOT contain both types.

Creating dRequirements

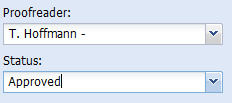
* create dRequirements supporting the goals:
  + - add a new requirement
    - Link the requirement with a parent Goal or Requirement .
* each requirement MUST have as a child one or more requirement OR Features.
* A requirement CANNOT have both children types.
* Review requirement: Each requirement need to be validated by a proofreader
* a proof reader need set the status of each requirement to “Validated”, stating that indeed this is in scope and well understood

Creating dFeatures

* manage Features
* A Feature is a characteristic that set the condition posed by the requirement to be true.
* Propose features satisfying requirements. For each requirement describe a feature and link them.

Illustration 44: a UMTS connection is a valid feature that cover the requirement of connecting mobile workers

* Review features: each Feature need to be validated by proofreader



* Confirm feature

Modeling the Requirements and Feature in the correct way

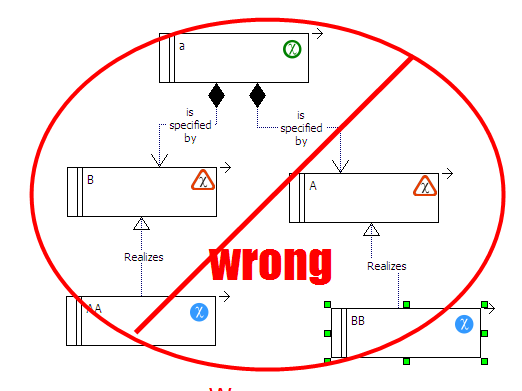
If you want that a Feature is realized in 2 different cases it is necessary to describes 2 different requirements. This is especially true in the case of a **Scenario** situation where not all requirements are intended to be realized.

Illustration 45: to create scenarios use the requirements type field

In this example we have two Features: BB and AA

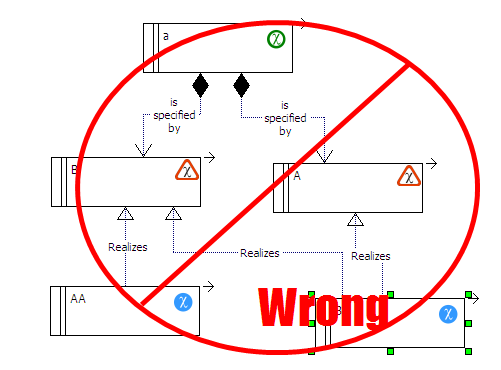


we see that both of them are realized, also if one of the 2 requirements A or B, representing 2 scenarios is no more present in the model. Check the image below out.

Illustration 46: In this example A and B are alternative scenarios. But we seek a way to realize booth feature in any case.

The above modeling would be wrong, because Feature BB is only realized as if the Requirement A is present in the model.

An alternative solution would be to attach the feature BB also to the requirement B. See below

Illustration 47: In this diagram a Requirement is realized by two different features. This is not allowed by the Metamodel.

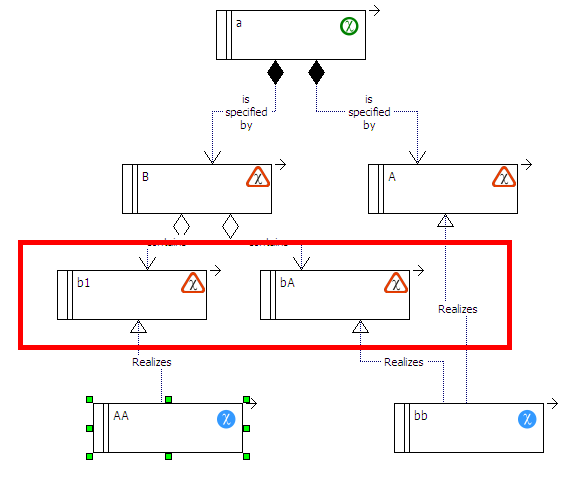
This is OK from the point of view of the feature, but not from the point of view of the requirement. In the Digital Architecture Framework Metamodel a Requirement can have only a realizing feature.

If you have the impression that a requirement requires TWO features means, that it contains 2 children requirements.

Illustration 48: A ChiRequirement has a ChiFeature only. A ChiFeature can satisfy multiple requirements

Imagine a requirement like a hole of a certain dimension and a feature like a screw. You cannot put more that 1 screw in the same hole also if you could use 2 different ones.

We take this as an indication for introducing a new level in our left scenario.

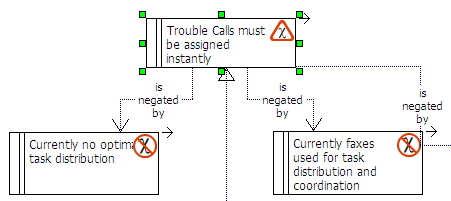
Illustration 49: In this diagram the feature BB is realized in both scenarios A and B

Now also in the case where the “A” scenario is not implemented, the features AA and BB are realized.

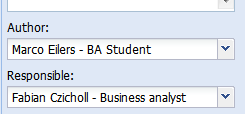
Manage Issues

Issues are facts that stop a requirement to become true. This means that a “Problem” cannot exist if you don't have a requirement behind it. The easiest way to solve issues is to decide the parent requirement is not mandatory.

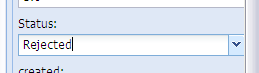
* Discover issues provoked by requirements

Illustration 50: a requirement with two feature attached cannot become true

* Assign responsible to this Issue

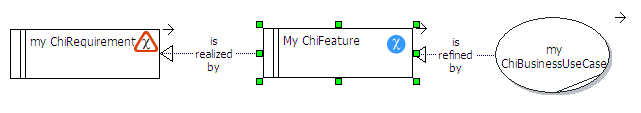
Illustration 51: each Issue must have an author and a responsible

* + Responsible possible evaluation:
    - Delete the requirement from model. It is suggested to don’t keep in the model elements that describes past situations.
    - Change the requirement causing the issue.
    - change the status of the Issue (e.g. to rejected).

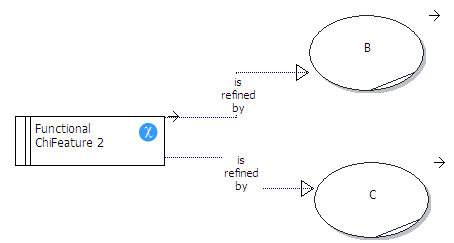


Modeling Features and Use Case in the correct way

In a Digital Architecture Framework Model each dFeature that describes a functional requirement must be refined by a Use Case. A feature can realize a non functional requirement like

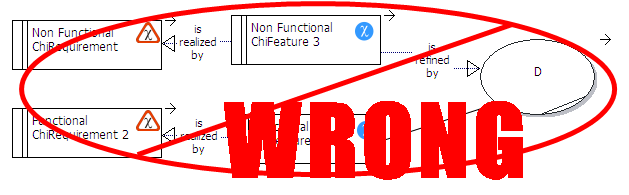
Illustration 52: Digital Architecture Framework prescribes that each feature must be realized by an use case apart the non functional ones

A feature that realize a non functional Requirement (types “technical” or “environmental)” can remain without any Use case refinement. This is because a Business use case describes a functional sequence of events Only. A Feature can be refined by more Use Cases on the same time.

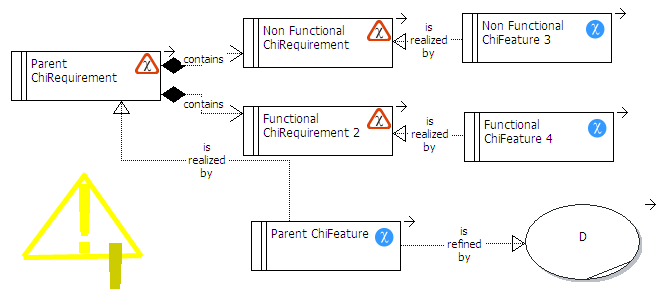
Illustration 53: A feature can have multiple realizing use cases

In some cases you can have the impression that a Use Case realizes more that a feature. This is not supported by Digital Architecture Framework Models. Keep in mind that the relationship between a feature and the use case is similar to the one between the overview of a story and the flow of the story itself.

By having 2 different overview you implicetely affirm that two stories exist, not one.

Illustration 54: a Use case cannot realize two different features

The diagram below is a possible work around but is NOT suggested.

Illustration 55: while a requirement can have both a requirement and a feature as children this is not suggested.

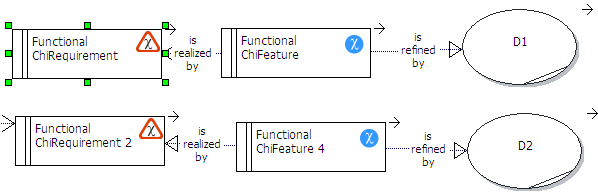
The functional and non functional requirements are both parent of a dRequirement that has a feature that is directly linked. The refining Use Case in this case is covering all 3 dFeatures.

This is possible but is not suggested.

The parent dFeature realize as abstract requirement, meaning that also the use case risks to be not concrete enough.

Typically the cause of the situation explained above is that you have too many Features and to few Use Cases. In a good model the level of abstraction should be higher in the top model types.

If the analysis shows that a use case covers 2 features means that it describes two different portion of business. In this case would be better to split the Use Case in two.

Illustration 56: two use cases are created that belong to the same process

the Analysis sub model

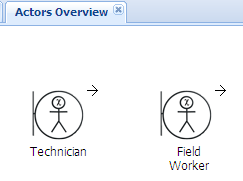
In this section you will learn how to create the Analysis sub model.

Use cases

A Functional feature describes which reaction the business has to a requirement. A use case specified HOW this is working.

Modeling the actors

The scope of this section is to develop an Actor model using d Actors

Illustration 57: ChiExternalWorkers are one of six different types of Digital Architecture Framework actors

1. Model the system's actors using the provided actor package:



1. The Digital Architecture Framework Meta-model defines special type of actors. Answer the following questions to help you find them:
   * 1. which external systems or persons will interact with the considered system? “External” means They are dBusiness Partners. A business partner can be further specified defining his active or passive role (see below).

*e.g. a generic Business Partner is a client of the enterprise.*

* + 1. Of those systems or category of person who will take the initiative to contact the considered system? They are dBusinessPartnersActive.

*e.g. an Active Business partner is the client that place an order in an online system.*

* + 1. Of those systems or category of person who wait for communication from the considered system? They are dBusinessPartnersPassive.

*e.g. a Passive Business Partner is a weather forecast System that offer a public API*

* + 1. In a similar way list of the internal users and systems that will interact with the considered one. They are dWorkers.

*e.g. a generic Worker is an employee of the enterprise that will use the considered system.*

* + 1. Between the Workers find out who has access from or to the outside. Model those as dWorkersExternals.

*E.g. a field sale worker that need mobile access to the enterprise.*

* + 1. Finally model workers that have no direct connection with the outside as dWorkerInternal.

*E.g. a SAP System from which the considered system get master Data.*

Finding candidates for Use Cases

Story-telling is an important activity tat we all learned as children. ask the subject matter expert to tell the “story”. Now divide the story in the his sub parts. Write a list of them: they are the candidates for the use cases.

Another way to find use cases is to consider what each actor requires from the system in scope. Answer the following questions to help you find use cases candidates:

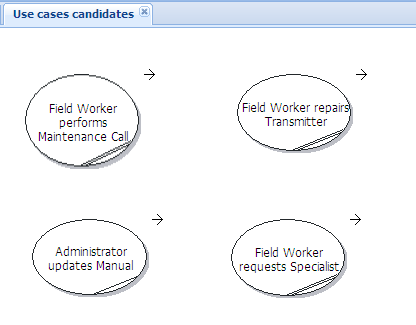
* + For each actor you have identified, what are the tasks in which the system would be involved?
  + Does the actor need to be informed about certain occurrences in the system?
  + Will the actor need to inform the system about sudden external changes?
  + What information must be modified or created in the system?

Independently how you decide to find the candidates follow those rules.

The Use case SHOULD have a name composed a Verb and an Object.

The Use case name CAN contains also the name of the primary actor (see below) but this is not suggested.

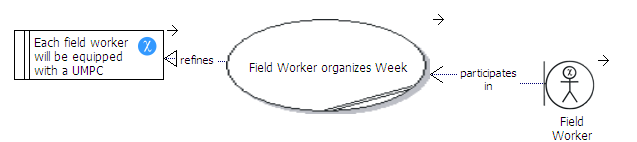
1. Gather candidates for dBusinessUseCases (name only) in a Use case diagram.

Illustration 58: Use case diagram with candidates. The use case name contains the primary actor.

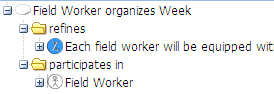
From candidates to final Use cases

* In the previous step you define a list of use cases candidates. This can be very long. This is a good moment to perform a new Use case point analysis (see [#3.2.Estimating Projects efforts with Digital Architecture Framework and Use Cases point analysis](#3.2.Estimating_Projects_efforts_with_Chronos_and_Use_Cases_point_analysis)) to calculate the actual cost. With this information you can decide which ones will be really implemented.
* Share the list and the estimation with business and decision makers.
* They should change the status of the UC to be implemented from “proposed” to “approved“. You can move the rest to another package.
* Transform a “Use cases candidate” in a final dBusinessUseCase by linking actors and describing relationships (e.g. with dFeatures). See also the paragraph “**Modeling Features and Use Case in the correct way”** above

See example diagram below:

Illustration 59: Chain of Feature and Use Case with a linked external worker

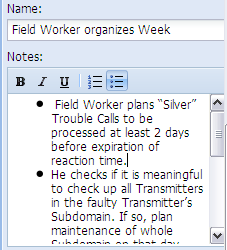
See hierarchy:

Illustration 60: Hierarchy of Use Case, Feature and Actor

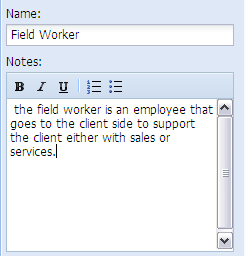
* Polish the **dBusinessUseCase** adding a description in “*Notes*”. This is done by writing in a pointed list of the required steps to achieve the “*main success scenario*”. Detail the flow of events. All "whats" should be answered. Remember that test designers will use this text to identify test cases.

Scenarios

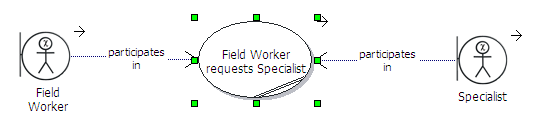
* In a very complex Use case note think about writing a paragraph call “**INTRODUCTION**”. (notice the subtitle special formatting capital bold and underline).
* In very complex use cases you can decides to add scenarios in form of alternative logical ends or set of data. A scenario captures one specific path through his use case. Each complex use case typically could have from a few to several dozen scenarios of interest. This is also important for creating the test cases.

Illustration 61: a scenario describing a field sale business process

* Refine the actors’ model adding a description to their identity (the name). You can achieve it by filling the Notes field with a detailed description text.

Illustration 62: an actor description is necessary to understand how implement this role

* Create a new Use case diagram for each use case named “<<Use Case name>> Use case Diagram”
* Place the *Primary* actor to the left and the other actors to the right of the Use case.

Illustration 63: as a graphic convention the primary actor is placed to the left

Completing the Use case

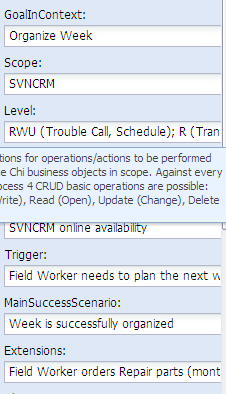
Finish the dBusinessUseCase by filling all the remaining properties:

* Use the “*Scope*” field to describe the business system where this use case is executed. Examples of business systems are “*ERP*” or “*CRM*” or subsystems like “*Billing*”. Basically each use case should be executed in **one** system only. In case of multiple systems divide them with a comma: the use case will require having multiple implementing dControllers (see below).
* The “*GoalInContext*” field should implicitly express the actor's intent or purpose of the use case.
* Fill the “*MainSuccessScenario*” field with the Status that your Use case intent to achieve. this is the normal, error-free scenarios in which the system functions as expected.

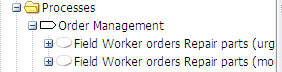


* Fill now the “*Extensions*” field. *Extensions* are additional statuses that the the system would achieve. This include any Business exception. Separate multiple Extensions with a semicolon “;”
* the “*trigger*” field in the use case is the actual action that starts the sequence.
* The *“Level”* field is a comma separated list of involved Business Object and their CRUD access.

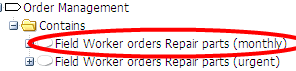


Illustration 64: fields of the use case

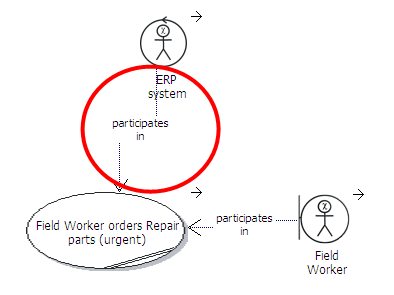
* Now organize the dBusinessUseCases under dBusinessProcesses
  1. Drag the Use case and Drop it in the corresponding Process.

Illustration 65: Use cases can be organized in ChiBusinessProcesses, that are special kind of packages.

* Review the dBusinessUseCase in the business processes: decides which one are eventually dBusinessUseCaseCore. Within a dBusinessProcess you SHOULD define the “most important” Use case as dUseCaseCore.



* Refine the actors model specifying the interaction between the consider system and the externals one as base for required **interfaces**.

Illustration 66: 2 actors participating in an UC

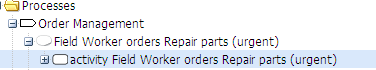
* Refine the Actors model specifying the interaction between human user and the considered system as base for the authorization and right system.

The Use Case's activity

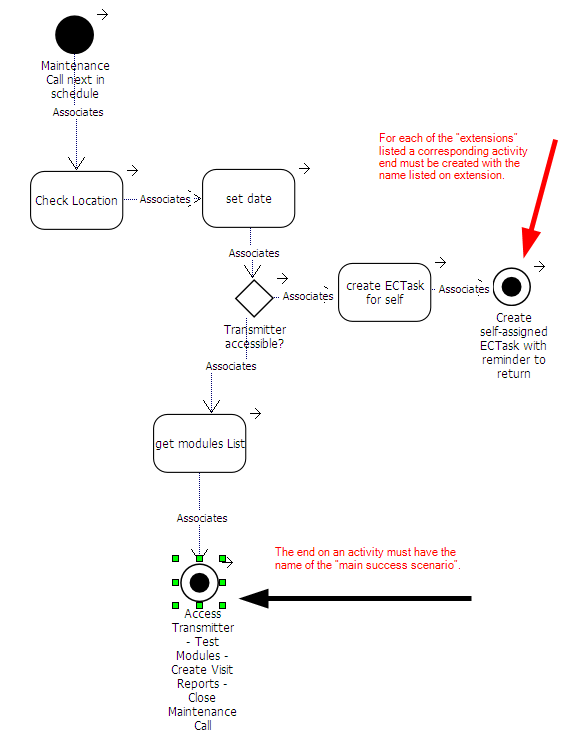
An activity diagram is a visual illustration of the flow of events. It describes graphically each step that is required after the trigger until the achievement of the main success scenario or the alternatives end of the Use case. In the activity you manage instances (dObject) of Classes (dNodes). E.g.”Car” is a dNode while “my Ferrari Testarossa” is a dObject.

In each step you can:

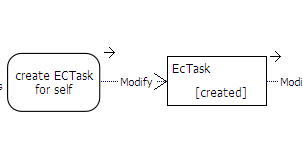
* Create, Read, Update Destroy a dObject
* get or set attributes of dObject
* get or set the parents or the children of the actual dObject.
* Create **Activity Diagrams** for each use Case.

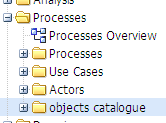
Illustration 67: Activity diagrams are visualized with a special symbol and stored within the Use Case

* Create a diagram with an activity step for each point in the Use cases description pointed list.

Illustration 68: Example of Use case activity

* The name of the “*activity Initia*l” object must be equal to the “trigger” field in the use case.
* The name of the activity end on an activity must have the name of the “*main success scenario*”. For each of the “*extensions*” listed a corresponding activity end must be created with the name listed on extension.
* Consider which Concepts are needed in an activity. Describes them as **dObjects** candidates (identity only).
* Add statuses to Objects.

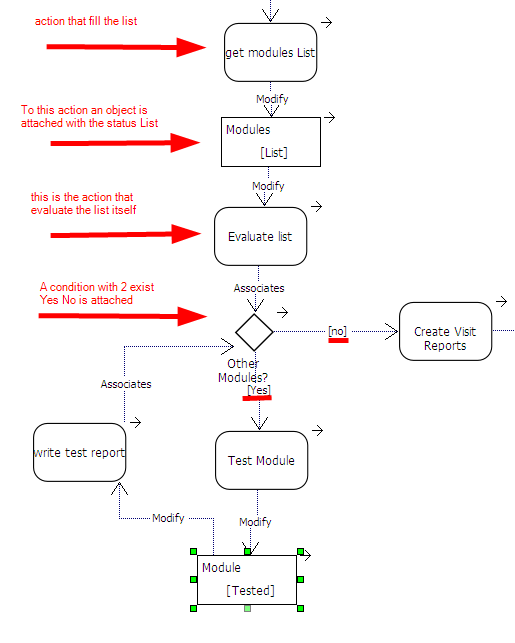
Illustration 69: a ChiObject with a "created" status

* (optional) gather all the object candidates in a single Object Catalog package  
  
* (if required) Consolidate the Objects by deleting the non necessary ones.

Modeling loops

While modeling activities is very probable that you will face the necessity to iterate an action over a list. This is expressed as *loop*.

* A loop is modeled by evaluating within a uml::**Condition** the remaining elements in a list.
* First you need an *activity* that feed the list
* To this *activity* a dObject is attached with the status “*List*”
* An *activity* follows that evaluate the list itself
* A condition with 2 exist Yes No is attached
* The 2 association that comes from the uml:condition element MUST have the names: “*LoopStart*” and “*LoopContinue*” .

Illustration 70: Example of a loop

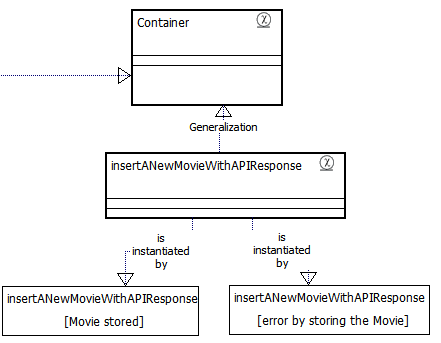
Use case Check List

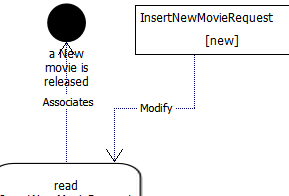
Use the following list of questions to help you complete the job of modeling system behavior.

* **Have you found all of the use cases**? Those you have found must be able to perform all system behaviors. If they don't, some use cases are missing.
* **Do the use cases meet all of the functional dRequirements**? If you have intentionally left any requirements to be dealt with in the object models, such as non-functional requirements, you must mention this somewhere.
* **Does the use-case model contain any superfluous behavior?** Does it present more functions than were called for in the requirements? Each use case must be attached to a feature that is connected to a requirement.
* **Have you found all of the actors**? Have you accounted for and modeled all of the roles in the system's environment? You can't be sure until you have found and described all of the use cases.
* **Is each actor involved with at least one use case?** Remove any actors not mentioned in the use-case descriptions, or any actors without communicates associations with a use case. Note that an actor mentioned in a use-case description is likely to have a communicates association with that particular use case.
* **Can you name at least two concrete people or systems who would be able to perform as a particular actor**? If not, check the role the actor models. If it's part of another role, merge the actor with another actor.
* **Are there actors playing similar roles in relation to the system?** If so, merge them into a single actor. Or create a hierarchy of them.
* Will a particular actor use the system in several different ways or does the actor have several different purposes for using the use case? If so, you should probably have more than one actor.

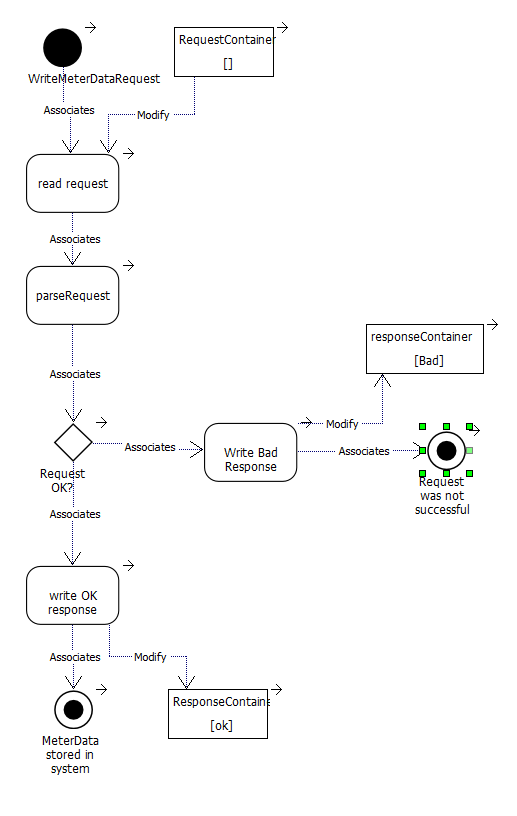
Modeling services

* In a Digital Architecture Framework Model it is possible to model an activity that automatically generate a service (depending from the cartridge for example a Python Flask service, a WSDL or a XSD definition).
* the Import and export parameters MUST be modeled as dObjects, instances of dNode that inherits from type Container.
* The request is linked to the first activity in the ActivitySet. The convention is: <USECASENAME>Request

Illustration 71: each Request or Response ChiObject must be an instance of a ChiNode that inherits from the class Container.

Illustration 72: A Request ChiObject transports the import parameters for the service

* Export Params MUST be modeled as dObjects linked to the last activity in the ActivitySet. The convention is: <USECASENAME>Response

Illustration 73: a finished activity diagram. It includes a start, Request and response containers, a decision main success scenario and an extension.

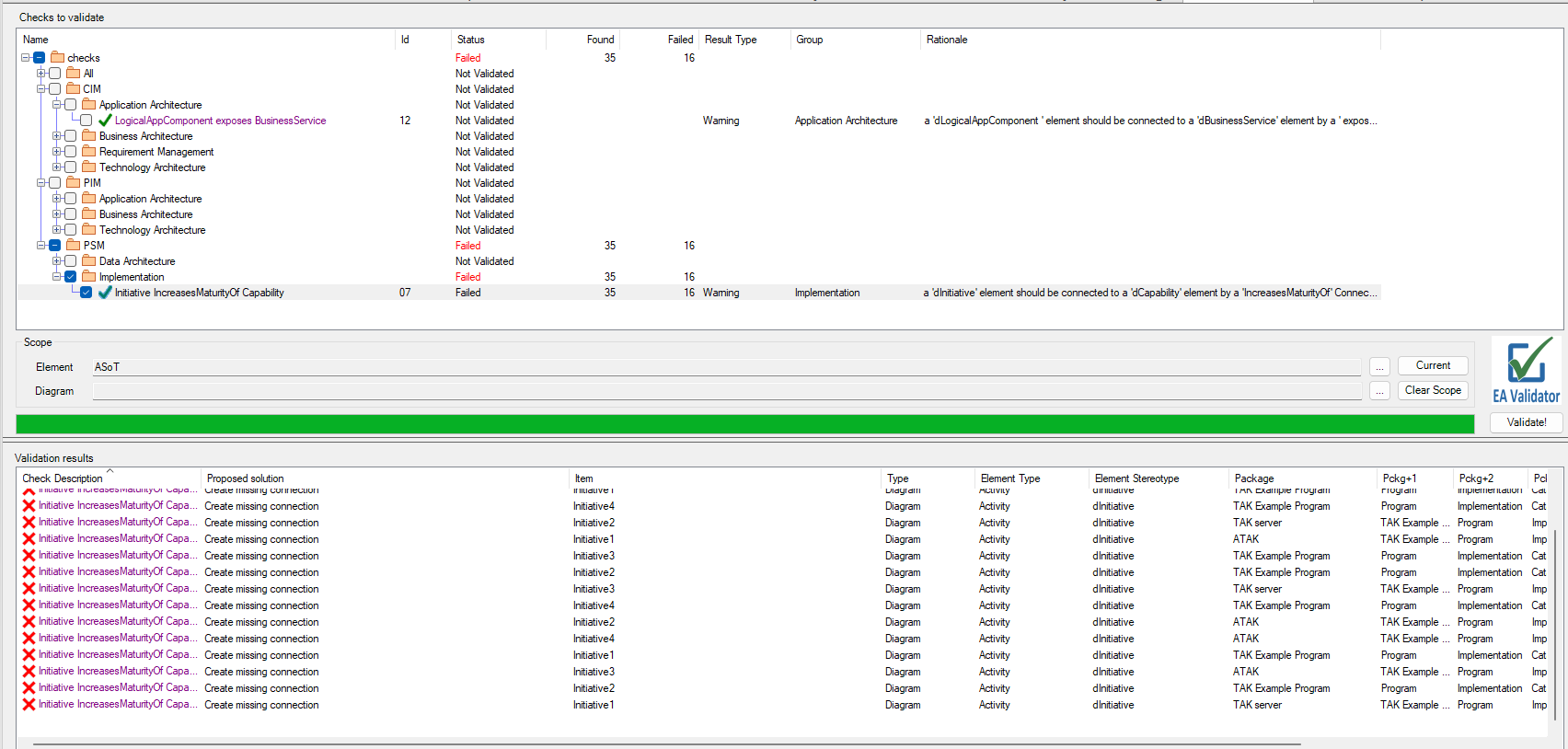
* The final ActivitySet should looks like the diagram below

Additional Tools

This chapter covers the use of the Model browser, the Document generator and the Code generator.

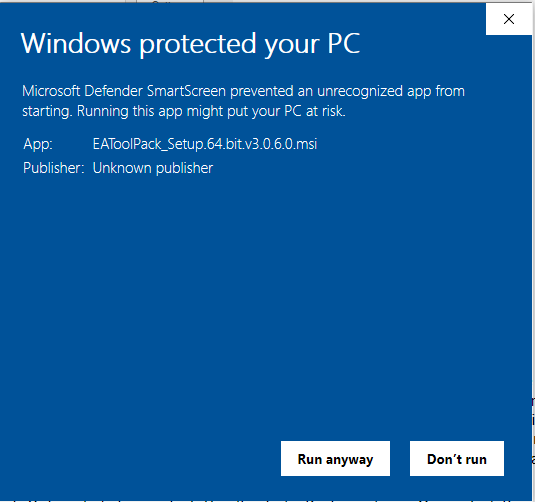
The DAF Validator

The DAF Validator is you ally in determination the next steps model's completion.

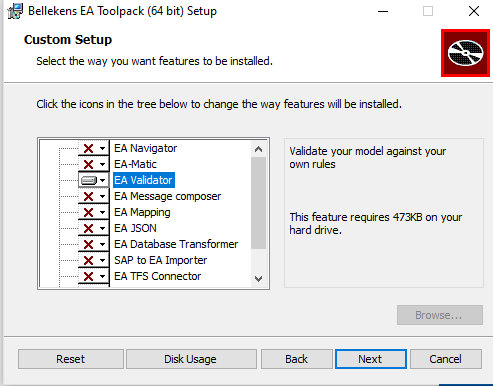
* : example of a model report, 16 Initiatives are not relative to a Capability

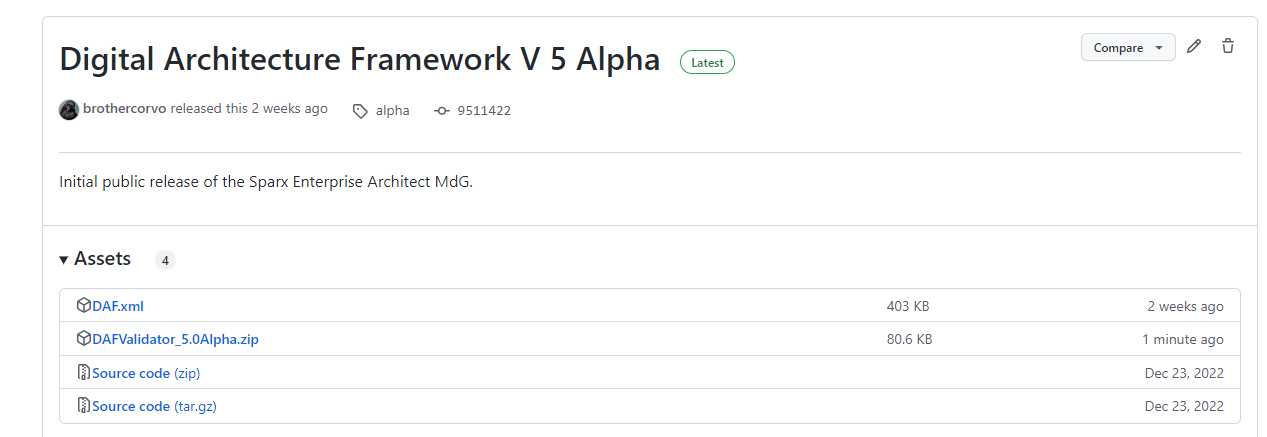
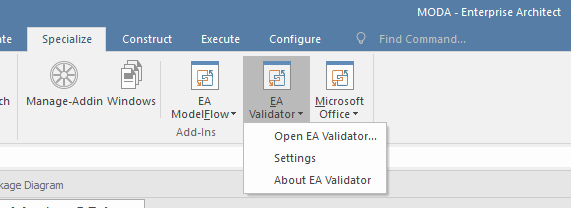
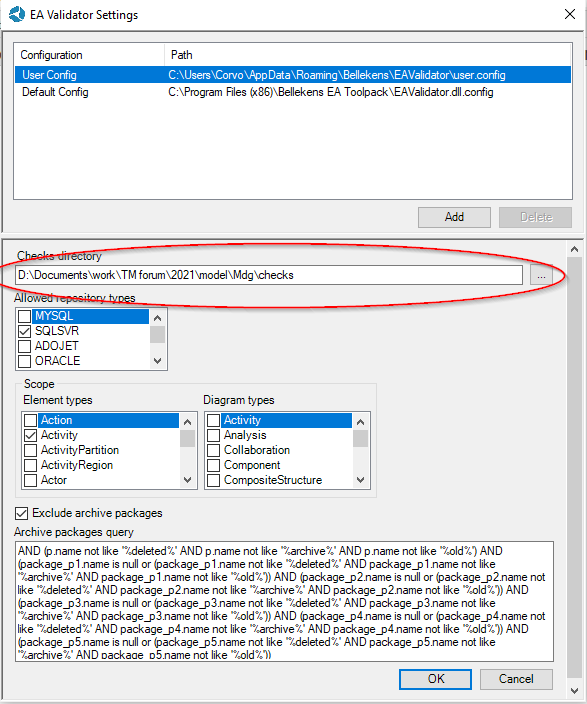
 Installation of the DAF validator

* Download from <https://github.com/GeertBellekens/Enterprise-Architect-Toolpack/releases> installation the installation file EAToolPack\_Setup.32.bit.v3.xx (EA 15.2) or EAToolPack\_Setup.64.bit.v3.xx (EA 16.x)
* Close Sparx EA
* Start the installation
* Click Run Anyway



* Select EA Validator only



* Download the validation Files [DAFValidator\_5.0Alpha.zip](https://github.com/FreeTAKTeam/DigitalArchitectureFramework/releases/download/alpha/DAFValidator_5.0Alpha.zip)  
   from the Digital Architecture Framework repository <https://github.com/FreeTAKTeam/DigitalArchitectureFramework/releases/tag/alpha>
* 
* Unzip in a folder of choice (e.g. c:\DAF\validation)
* Start EA and navigate to Specialize -> Add-in -> model validator -> settings
* 
* Point the check directory to the folder you created previously
* 

Using DAF Validator

* Before you start validating you should check out regards the “most recently edited objects” list. This notify you over modification in the model.
* Start the EA Validator tool using the menu option Specialize | EA Validator | Open EA Validator This will open the EA Validator user interface in EA.

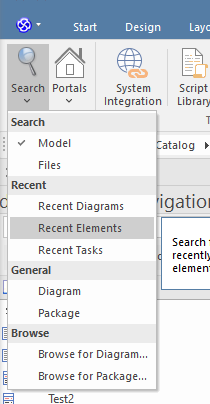
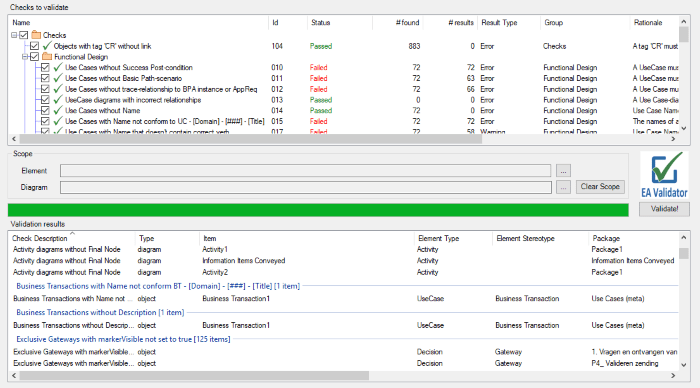
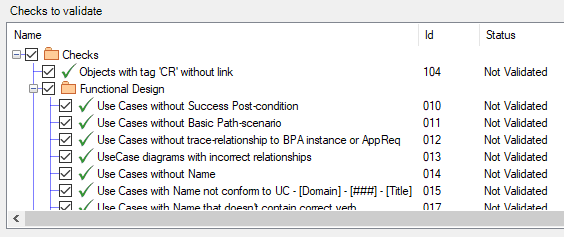


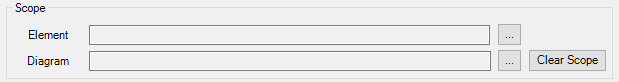
Illustration 92: most recent elements” search .



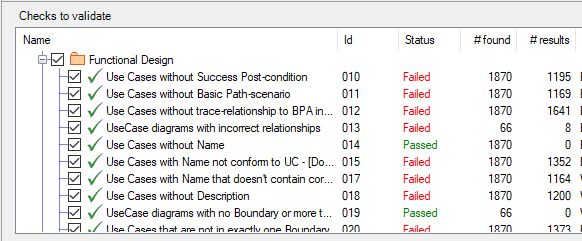
* Select Checks



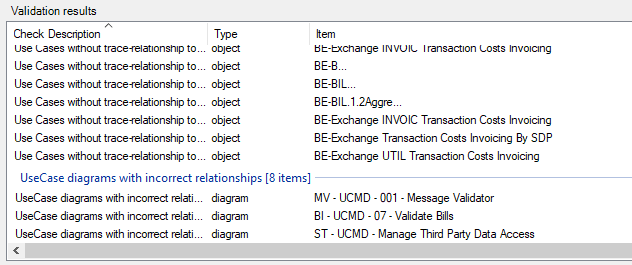
* The top section lists all validation available validation checks. You can check or un-check the checkbox next to the validations in order to enable or disable them.
* Define Scope
  + You don’t have always validate the whole ASoT. In the Scope section you can select which part or the model should be validated. If left empty the EA Validator will validate the whole model.



* In the Element section you can select either a Package or an element such as a Use Case or Class.
  + The Diagram field works a bit differently. You have to select a diagram in the project browser first and then click on the […] button to select that diagram.
  + Press the Clear Scope button to remove any scope selection.
* Validating
  + Press the Validate button to start the validation of the selected checks.   
    Once the validation is finished an overview of the results is shown in the top table with the checks.



* Double-clicking any of the checks that have status failed will show the results for that specific check in the bottom table.



* Double-clicking any of these results will select the validated element in the project browser.

Customizing lists

The list can be customized by clicking on the small arrow near the heading. A list of all the available fields will appear.

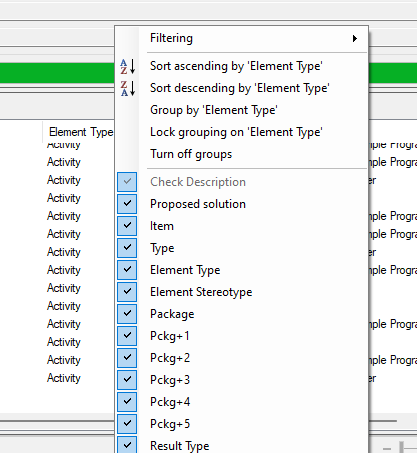
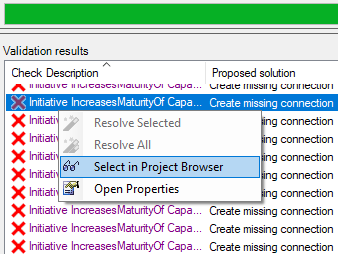
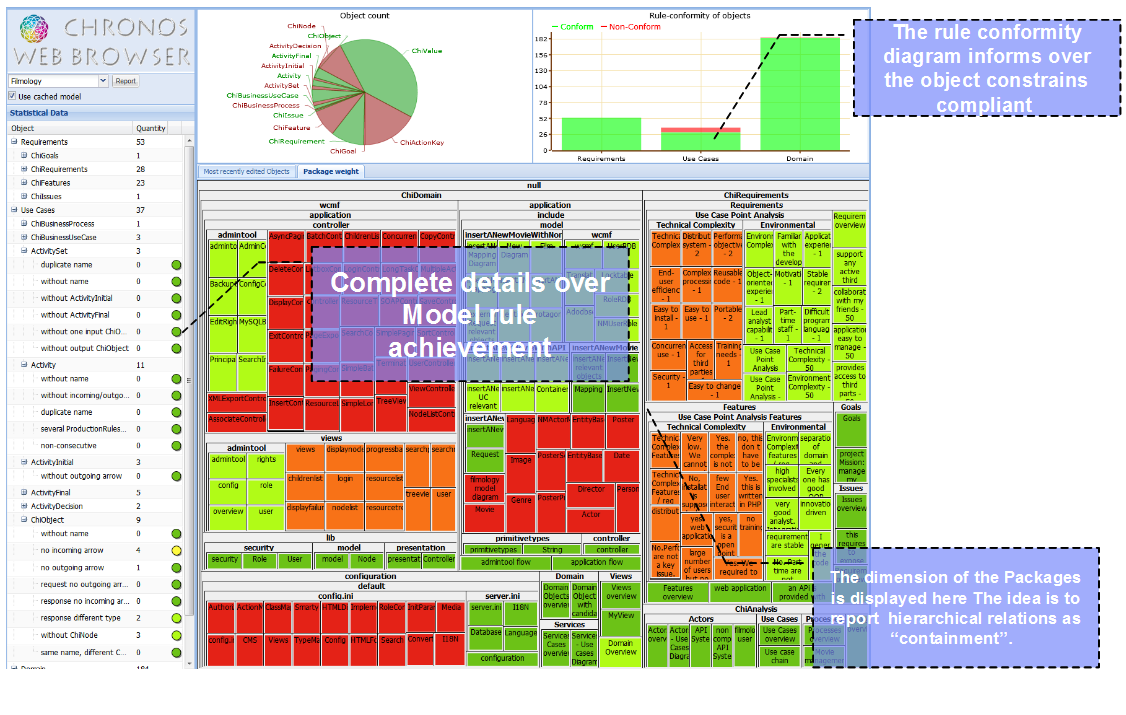
. By clicking on any element in a list the “”model Tree” opens for editing.

Illustration 93: customization of the list's field

Illustration 94: by clicking on a element on the most recently edited objects open in his location on the tree.

Figure 13: By clicking on any element in a list the “”model Tree” opens for editing

Illustration 95: Web Browser: the package distribution in this screen shot reflect that too many elements are in to few packages. this is an hint for restructuring your model

The rule conformity diagram give an overview over the all object constrains compliant, organized in the 3 sub models. By navigating over the red part you get the amout of rules that are actually violated over the total

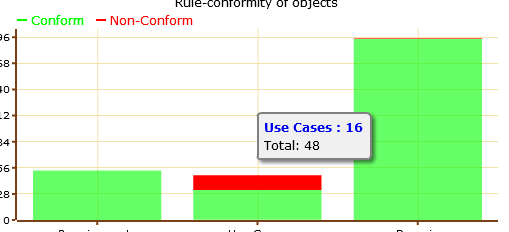
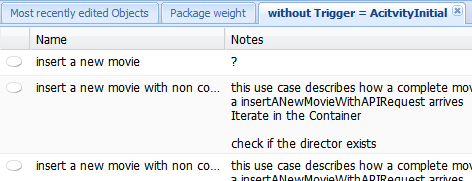


Illustration 96: In this example Use Case sub model a total of 48 objects exists of which 16 are not rule conform.

Complete details over Model rule achievement is listed to the left in the “*statistical Data*”.

The checks are again organized in the 3 Digital Architecture Framework sub models and then in the associated objects .

Illustration 97: in this example 3 use cases don't have an ActivityIntial with a name equals to the trigger

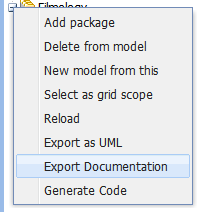
The dimension of the Packages is displayed in the package weight tab. With larges models this requires some time to load. The idea is to report hierarchical relations as “containment”.

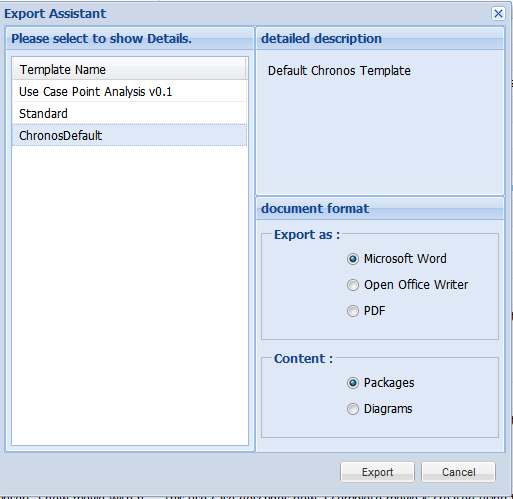
The document's generator

EA integrates a powerful document generator that permits to creates Microsoft Word, OpenOffice and PDF files based on templates.

How to generate a document

To generate a document:

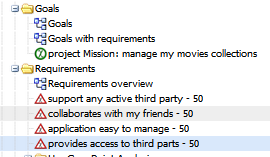
* Click on any package
* select “export documentation”
* a list of the installed templates appear:
* the document format can be selected in this screen.

Illustration 98: the list of available templates can be expanded for your project needs.

* See below for the use of “content”.
* If your project require a certain document, with layout, projecproofreadert logo, etc you will need to create a new template. This requires some programming. See the “HelloWorld\_DocumentGeneration” in the “additional reading” appendix for more information.

Diagram as virtual packages

* Normally all package's and children elements are exported in the documentation. This is useful to generate a complete sub model

Illustration 99: by exporting the documentation starting from package "Goals" only the elements in this package will be present in the document.

* In some case a certain collection of elements need to be documented that are distributed trough multiple packages. EA supports “virtual packages” to accomplish this.
* By selecting the in content option “diagrams” you can generate a document with all the elements in a certain diagram also in there are hosted by parallel packages.

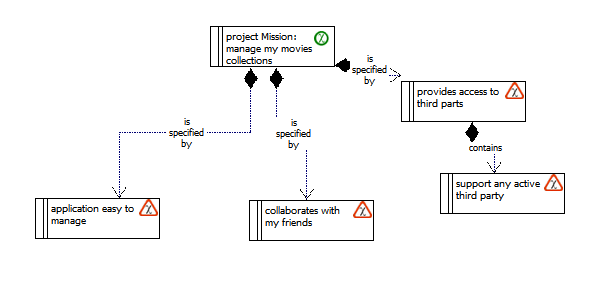


Illustration 100: the diagram "Goals with requirements" shows requirements from another package. If the option "diagrams" has been selected the documentation will include those elements

* In this way, you can build a very flexible documentation. Notice that if you export a package where an element is phisically present and the is also shown in a diagram this will exported two times.
* The best way to use this functionality is to create a separate structure of packages containing diagrams only.

Use case documentation

EA provide a convenient way to generate use case documentation. By selecting a use case element in the tree right click you get a special “export Documentation”:

in the resulting document contains Use Case Description, Objects instantiated, Object to Node mapping, Request & Response Containers, Participating Domain Classes, Use Case Scenario & Hierarchy.

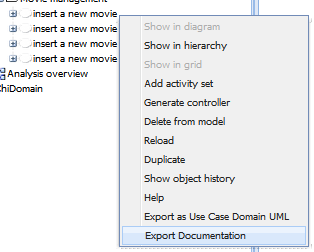


Illustration 101: the Use case element has a special menu for exporting the documentation

DAFgen The code generator

The DAF model is linked to a mighty generation features: It is able to connect directly with a component called DAFgen (Digital Architecture Framework Generator) which can supports several transformations (cartridges). A Cartridge contains the set of transformations necessary to support a certain technology stack

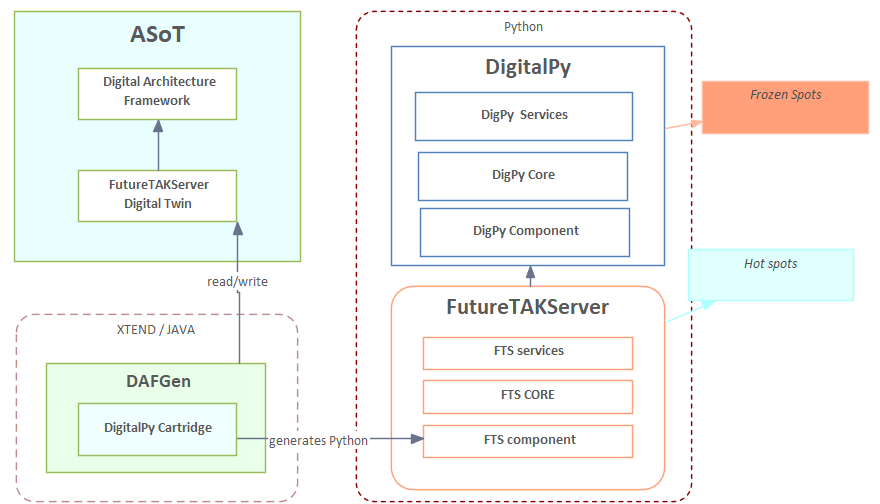


Illustration 102: Architecture of a DAF installation. EA is a front end for the AsoT. DafGen the generator executes a transformation (e.g. Python)

The generator is in fact a completely independent program. In alternative to use it embedded in the EA, you can export a valid UML model, created with any tool and pass it to the generator offline. This is the suggested way in large scale projects.

A complete documentation about the cartridges is available separately (see [#2.Additional reading](#_toc1296)).

Appendix

List of Model Checks

In this section all the constrains are listed that the CWB checks.

Requirements

* dObject has no associated type
* A goal Must have a child requirement or goal
* a goal cannot have both requirement or goal as children
* A goal must have a priority
* A goal must have a Value that’s of integer type
* Every goal with a requirement must be realized by a feature.
* A requirement must have a parent (goal or requirement)
* A requirement must have at least one Child (Feature Issue or requirement)
* A requirement can have a feature Child only
* a requirement cannot have both a feature and an issue as a children
* a feature must have a parent requirement
* a feature with the status implemented must have a children use case
* an issue must have a parent requirement

DAF Analysis

A ChiBusinessProcess must have at least a Use case within

Use Case

* a **ChiBusinessUseCase**Must have a feature
* a **ChiBusinessUseCase**should have a Scope
* a **ChiBusinessUseCase**should have a GoalInContext
* a **ChiBusinessUseCase**MUST have a Trigger
* a **ChiBusinessUseCase**MUST have a MainSuccessScenario
* a **ChiBusinessUseCase**MUST have a PrimaryActor

Activity

* ActivitySet must have globally unique name
* ActivitySet must have a non-empty name
* ActivitySet must have exactly one ActivityInitial
* ActivitySet must have at least one ActivityFinal
* AcitivtySet must have exactly one Input ChiObject
* AcitivtySet must have at least one Output ChiObject
* AcitivtySet must have at least one Output ChiObject
* Activity must have a non-empty name
* Activity must have at least one incoming and one outgoing arrow
* Activity must have a unique name within its ActivitySet
* Activity must have at least one incoming and one outgoing arrow
* Activity must have at most one ProductionRuleSet
* ActivityIntial must have at least one outgoing arrow
* **ActivityIntial**must have the same name as the “trigger” field in the corresponding use case
* ActivityFinal must have at least one incoming arrow
* at least one **ActivityFinal**must have the same name as the “main success scenario” field in the corresponding use case
* All Activites associated with the same ProductionRuleSet must be consecutive
* ActivityDecision must have a non-empty name
* ActivityDecision must have a unique name within its ActivitySet
* ActivityDecision must have at least one incoming and two outgoing arrows
* All arrows exiting from an ActivityDecision must have unique names
* ActivityDecision must have at most one ProductionRuleSet
* An ActivityDecision with no ProductionRuleSet must be preceded by an Activity
* All ActivityDecision associated with the same ProductionRuleSet must be consecutive
* At most one ActivityDecision may be associated with the same ProductionRuleSet
* Loop type ActivityDecision must have at least one incoming *LoopContinue*and exactly one outgoing *LoopStart* arrow
* Loop type ActivityDecision must have the following predecessors: Activity('get list') -> ChiObject('list') -> Activity('*evaluate*list') -> ActivityDecision
* Loop type ActivityDecision must have the following successors: ActivityDecision -> Activity('get list element') -> ChiObject('list element')
* Loop type ActivityDecision must be in same ProductionRuleSet as its required predecessors and successors: Activity('get list') -> ChiObject('list') -> Activity('evaluate list') -> ActivityDecision -> Activity('get list element')
* ChiObject must have a non-empty name
* ChiObject must have a unique name within its ActivitySet
* ChiObject must have at least one outgoing arrow
* ChiObject must have at least one incoming arrow
* Input ChiObject must have at least one outgoing arrow
* Output ChiObject must have at least one incoming arrow
* All Output ChiObjects must be of the same type
* ChiObject has no associated type

Domain

* ChiNode must inherit from Node superclass
* ChiController must inherit from Controller superclass
* ChiNode must have a name
* ChiNode.orderby must be "none", "sortkey" or an attribute name
* ChiNode.initparams must be the name of a ChiSystem class
* ChiNode attributes must not have the name "type", "state", "oid", "value", "none" or "sortkey"
* ChiValueRef.reference\_type must be the name of a ChiNode class
* ChiValueRe.reference\_value must be an attribute name in the referenced ChiNode class
* ChiActionKey.config must not be empty
* ChiValue must have a name
* ChiValue must have a valid name

Appendix : Keywords and Graphic conventions

Keywords

Keyword with a special meaning.

Word with *italic* are used

words with underline are required and are checked by the generator.

Words between "()" are aliases or previous keywords

Use cases

* *retrieve*: the considered system starts an active data request to a passive external system
* *query*: the considered system passively offers a response for external systems to query data from the System.
* *Send*: the considered system receives external data and persist them.
* *Create*: the considered system get a request to create certain data. Those are created, persisted and send back.

Activities

* *create:* create a domain object in the memory. If parameters are present a new object is created with those parameters. if not present an empty object is created.
* *store:*Store a domain object persistently.
* *read (get):*Read a domain object from a variable (e.g. a Container)
* *load: Load a certain domain object from the persistence.*
* *destroy:*delete a certain domain object
* *iterate (evaluate) :*start a loop (see [#4.3.1.1.Modeling loops](#4.3.1.1.Modeling_loops) )over a list
* *Invoke:*execute a certain action.
* *this*: the domain object in context
* [RS???]: (at the end of the name): Linked to the RuleSet given by the number, e. g. [RS1] or [RS123]

Decisions

* LoopStart*:*required by a Loop, outgoing arrow pointing to the first Activity in the loop
* LoopContinue*:*required by a Loop, incoming arrow pointing from the last Activity in each thread in the loop back to the loop Decision
* [RS???]: (at the end of the name): Linked to the RuleSet given by the number, e. g. [RS1] or [RS123]

ChiObjects

* <USECASENAME>Request: Name of the container with import paramenter
* <USECASENAME>Response: Name of the container with export paramenter
* *list*: a collection of elements, in alternative use name[] to indicate a list.
* <ChiObjectName>.<ChiValueInstanceName>: reference to the instance of a property.

chiObject Statuses

* created
* stored
* ok
* error

Graphic conventions

* The Primary actor should be placed to the left and the other actors to the right of the Use case.

Formatting convention

* **Sub titles**: Capital, bold and under lines
* **Actors:** underline
* **Use Cases:** Italic
* **ChiNodes:** Bold

Appendix: Additional reading

| **Tool** | **Description** | **Role** |
| --- | --- | --- |
| Digital Architecture Framework Doc Package | This mind map that can be run in a browser provide a hierarchical view of all the documents listed below organized by role. | ALL |
| DAFgen Xtend API | JavaDoc style documentation of all the Xtend methods and classes used by the DAFgen. | Architects,  IT specialists |
| EA API | JavaDoc style documentation of all the PHP methods and classes used by the Digital Architecture Framework Business Object Repository. | IT specialists |
| Metamodel | UML class model of the Digital Architecture Framework Profile. | Architects,  IT specialists |
| PIM Model | UML Class model of the EA application | Architects |
| PSM Model | This document in UML format describes the tooling and generators from an architectural point of view using a subset of the UMF | Architects,  IT specialists |
| wCMF API | this is the documentation of the PHP framework used by the EA | IT specialists |
| Draw2D | Documentation of this framework |  |
| extJS API 3.0 | Documentation of this framework | IT specialists |
| CWE API | The Digital Architecture Framework Web Editor API. CWE is a framework for creating rich Internet Application in Java script format. | IT specialists |
| Digital Architecture Framework Business Object Repository | This document introduce how to administrate the Digital Architecture Framework Business Repository. | Admins,  IT specialists |
| Digital Architecture Framework Business Manual | this comprehensive manual presents the Digital Architecture Framework system and lead the user step by step in the creation of a valid model. | Business Analysts |
| What do I need to know about Digital Architecture Framework. | This documents contains frequent asked questions about the Digital Architecture Framework methodology, the Digital Architecture Framework Web Modeler (EA), the Digital Architecture Framework Generator *DAFgen* and their use. | ALL |
| Rational tools and Digital Architecture Framework | This document present the difference between the standard Rational Tools and the Digital Architecture Framework offer . This is addressed to go-to market and managers. | Proj. Manager, Go-to-market |
| alphaCORE: model Driven utility | This article describes a model-based approach for utilities creating IT smart grid solutions based on computation independent definition CIM. This means that the definition of the solution is keepen separated from the implementation. It illustrates how do we speed up the development and bring agile enterprises flexibility in handling and usage. This approach has a positive effect on the price and time of solution development cycle of customized Smart Grid application. | Proj. Manager, Go-to-market, Architects |
| Building\_Digital Architecture Framework\_Web\_Modeller | This document describes how to build the **Digital Architecture Framework Web Tools** from source. | Architects, It Specialist |
| Building And Running Code generators | This document leads through the steps of: checking out the subversion repository, setting up the Eclipse Tooling, building the DAFgen, running a cartridge with test models (headless), running the cartridge from EA | IT specialist |
| Digital Architecture Framework Web Modeller Installation | This document helps to configure and install the Digital Architecture Framework Web Tools, including the Digital Architecture Framework Web Modeller and the Digital Architecture Framework web Browser. It is addressed to system administrators. | Admins |
| HelloWorld\_EA | This document lead to the creation of a simple web application without writing a single line of code. The application is completely defined in UML and is generated using a Model Driven approach. | IT specialists, Business analysts |
| HelloWorld\_generator.odt | This document lead to the creation of a simple cartridge for the Digital Architecture Framework generator. | IT specialists |
| HelloWorld\_DocumentGeneration.odt: | This document explains how to create a document template for the Digital Architecture Framework document generator. A document template can host any formatting and information coming from the model. | IT specialists, |
| Digital Architecture Framework Generator, features list | this spreadsheet contains a list DAFgen of features. | Proj. Manager, Go-to-market, Architects |
| Digital Architecture Framework Programming Guide.odt: | This document describes best practices for programming new cartridges for the DAFgen. | Architects, Businesses Analysts |
| Digital Architecture Framework Asset Report | this presentation aims to document the Digital Architecture Framework assets in terms of scope, functionality, applicability, completion, documentation and level of work effort to get to an asset that reusable by other IBMers | Proj. Manager, Go-to-market, Architects |
| CartridgeDocumentation.odt: | This document contains the documentation for each Digital Architecture Framework cartridge. It includes: cartridge's purpose, how to run a cartridge from EA, how to run a cartridge from command line. | IT specialists, Architects |
| EA\_VS\_restOfTheWorld\_RequirementToolComparison.xls | This Excel file contains one tab that consolidates thirty-four responses to the Requirements Management survey as of August 8, 2010 with an extra tab to define the features of the EA. | Proj. Manager, Go-to-market, Architects |
| Digital Architecture Framework Profile specification | This document explain in details the components of the Digital Architecture Framework Metamodel in form of an UML profile | IT specialists. Architects, Business Analysts |
| Ouranos Profile Specification | This document explain in details the components of the Ouranos Metamodel in form of an UML profile. Its explain especially its links and differences to the Digital Architecture Framework Metamodel above. | IT specialists. Architects, Business Analysts |
| Dionysos Programming Guide.odt: | This document explains how to create Dionysos client applications. | IT specialists |
| Dionysos JSON Interface Specification | This document describes the Specification  of a client working with a specific server  called Aphrodites | IT specialists. Architects |
| Aphodites Specification | This document describes the requirements for a Aphrodites framework to be used with a of a Digital Architecture Framework Model within a Model Driven Software Development | Architects, IT specialists, |
| White paper EA | Reusable ontologies are becoming increasingly important for tasks such standardization bodies that use knowledge-base development. IBM has realized a set of tools and a methodology to support the process of achieving consensus on common shared ontology by geographically distributed groups. These tools make use of the world-wide web to enable wide access and provide users with the ability to publish, browse, create, and edit models stored on an server. In this paper we present shortly some features of the both methodology and tools. | Proj. Manager, Go-to-market, Architects, Business Analysts |
| Methodik Workshop Deutsch.odp | This presentation in German language should be use to introduce the concepts of Digital Architecture Framework to German modeler and/or business consultants | Business Analysts |
| Energy & Utilities made easy with Digital Architecture Framework | This presentation introduce the use of Digital Architecture Framework in the context of the E&U industry. It is addressed to industry consultant and decision makers | Business Analysts, Proj. Managers |
| Digital Architecture Framework introduction for Managers | this presentation introduces Digital Architecture Framework from the corresponding role. It contains all the slides produced during the project that matter for this role. | Business Analysts, Proj. Managers |
| Digital Architecture Framework: introduction for Programmers | this presentation introduces Digital Architecture Framework from the corresponding role. It contains all the slides produced during the project that matter for this role. | IT specialists |
| Digital Architecture Framework Introduction for Programmierer | this German presentation introduce the methodology and tooling approach for technical oriented people. | IT specialists, Architects |
| Digital Architecture Framework: introduction for Architects: | this presentation introduces Digital Architecture Framework from the corresponding role. It contains all the slides produced during the project that matter for this role. | Architects |
| Wenn Das Business „laufen Lernt: | this high level presentation in German language introduces Olympos for managers with a special focus on the “mobile” aspect. | Business Analysts, Proj. Managers |
| Digital Architecture Framework: Use Case Point Estimation | this presentation introduces the Use Case Point analysis for estimating effort | Business Analysts, Proj. Managers |
| Digital Architecture Framework: Integration mit Jrules | This presentation in mix German and English introduces to the (unfinished ) work for supporting rule engines trough JRules. | IT specialists, Architects |

1. Contents

Table of Contents

Some suppositions about the reader 8

How this book is organized 8

How to read it 8

About DE 8

the Digital Architecture Framework methodology 9

About the DAF Metamodel 10

About the name “Digital Architecture Framework” 11

Introducing the Digital Architecture Framework Web Modeler 11

Models and Meta-models 12

About diagrams 12

Diagrams types in a Digital Architecture Framework Model 13

Work diagram 13

Relevant nodes 13

Package Overview 13

Chain Inheritance 14

Deep View 16

Mapping 16

The “seven” rule 16

Traceability 17

EA Special Fields and Features 17

difference between "Author" and “Creator”. 17

Rich text area 18

Status types and management 19

Digital Architecture Framework Stereotypes and relationships overview 22

Sequence of use 24

walk-trough the Digital Architecture Framework stereotypes 25

Before the project start 29

Creating a Request for Proposal 29

Introduction 29

RfP: walk-trough 29

working with the Response to Proposal 30

Introduction to estimation 30

How Use Case estimation works 30

Use Cases Complexity Definition 31

Actors complexity definition 31

Technical factor complexity definition 31

Environmental Factor complexity definition 32

Use Case point Analysis Walk trough 32

Estimating the Functional requirements 32

Estimating the Non-functional and environmental requirements 33

Generating a UCPA spread sheet 35

UCPA estimated tasks 36

UCPA Conclusion 36

Starting a project by selecting a model 37

Structure 37

The Digital Architecture Framework Default model 37

Configuring the model 38

creating ChiGoals 39

Creating ChiRequirements 43

Creating ChiFeatures 43

Modeling the Requirements and Feature in the correct way 44

Manage Issues 46

Modeling Features and Use Case in the correct way 47

the Analysis sub model 50

Use cases 50

Modeling the actors 50

Finding candidates for Use Cases 51

From candidates to final Use cases 52

Scenarios 53

Completing the Use case 54

The Use Case's activity 56

Modeling loops 58

Use case Check List 59

Modeling services 60

Introduction: Model View Controller 63

Create the domain model 64

Mapping ChiObjects to ChiNodes 64

Using the alphaCORE default model 65

Mapping two ChiNodes 71

How to Maps Two Domain 71

Create the Service model 72

Digital Architecture Framework and Logic 73

External Workflows 73

Internal Workflow 74

Rule 74

Modeling business rules 75

Definition of ChiRule (PRR) 75

Production Rule Limits 76

modeling controllers walk trough 76

The Model Browser 78

Customizing lists 79

The document's generator 81

How to generate a document 81

Diagram as virtual packages 82

Use case documentation 83

The code generator 84

Requirements 86

DAF Analysis 86

Use Case 86

Activity 87

Domain 88

Keywords 89

Use cases 89

Activities 89

Decisions 89

ChiObjects 90

chiObject Statuses 90

Graphic conventions 90

Formatting convention 90

1. Illustration Index
2. Illustration 1: in a DE system all the knowledge in his complexity is stored in a special kind of ontology: the UML model. The artifacts, including code and documents, are generated from a model. 15
3. Illustration 2: Today most IT projects suffers of poor understanding between different roles 16
4. Illustration 3: Diagrams are views over dir complexity 18
5. Illustration 4: Example of relevant nodes diagram 19
6. Illustration 5: Package overview example 20
7. Illustration 6: EA menu for creating a Package overview 20
8. Illustration 7: Chain Inheritance diagram example 20
9. Illustration 8: Show in Hierarchy menu 21
10. Illustration 9: Show in Hierarchy example 21
11. Illustration 10: example of the seven rule 22
12. Illustration 11: the object history allow to trace every single change that was performed on a model element. 23
13. Illustration 12: author and creator fields compared 24
14. Illustration 13: a list of stati provided in the Digital Architecture Framework Default Model 25
15. Illustration 14: transition workflow between stati 26
16. Illustration 15: a version umber is added if a released object is changed. 27
17. Illustration 16: Stereotypes and relationships for the Requirement and Analysis model 28
18. Illustration 17: Stereotypes and relationships Domain Model 29
19. Illustration 18: this goal is specified by 3 different ChiRequirements 31
20. Illustration 19: elements of the Requirement sub model 31
21. Illustration 20: two use cases refining the same Feature 31
22. Illustration 21: Example of a Process for movie management containing two Use cases 32
23. Illustration 22: Activity of the "insert new movie" Use case 32
24. Illustration 23: the ChiController that implement the Use case has operations that correspond to the UC's the activity elements. 32
25. Illustration 24: a ChiObject instantiated by a ChiNode 33
26. Illustration 25: Example of requirement model 37
27. Illustration 26: a standard list of Non functional requirements and their weight is available in the default model. 38
28. Illustration 27: Standard technical factors are included in the default Model 38
29. Illustration 28: requirements for the UCPA must be of type "Technical" or "Environmental" 38
30. Illustration 29: environmental factor tab of the Use case point generated documentation 39
31. Illustration 30: the Use Case Point Analysis document template 39
32. Illustration 31: sum up tab of the Use case point generated documentation 40
33. Illustration 32: the "model upload tab" in the "admin" perspective allows to import a valid Digital Architecture Framework model 41
34. Illustration 33: components of the Energy Template Model for alphaCORE 42
35. Illustration 34: the structure of the alphaCORE template model is given by the Digital Architecture Framework organization. 42
36. Illustration 35: ChiRequirements packages within the requirement model 43
37. Illustration 36: ChiFeatures packages within the requirement model 43
38. Illustration 37 The analysis model contains the e-energy, maturity model, open smart grid and smart home use cases 43
39. Illustration 38: position of the CIM 13 ontology 44
40. Illustration 39: location of the e-energy domain objects 44
41. Illustration 40: A goal of type vision is an abstract inner representational of the project scopes. 45
42. Illustration 41: a Goal of type Mission is the project slogan, how the Vision is communicated to the outside 45
43. Illustration 42: in this example of goal value the actual effectiveness in % the actual level of 20 should be increased by 5 % 46
44. Illustration 43: a Hierarchy of goals. the top element of it must be the Vision 46
45. Illustration 44: a UMTS connection is a valid feature that cover the requirement of connecting mobile workers 47
46. Illustration 45: to create scenarios use the requirements type field 48
47. Illustration 46: In this example A and B are alternative scenarios. But we seek a way to realize botth feature in any case. 48
48. Illustration 47: In this diagram a Requirement is relized by two different features. This is not allowed by the Metamodel. 49
49. Illustration 48: A ChiRequirement has a ChiFeature only. A ChiFeature can satisfy multiple requirements 49
50. Illustration 49: In this diagram the feature BB is realized in both scenarios A and B 50
51. Illustration 50: a requirement with two feature attached cannot become true 50
52. Illustration 51: each Issue must have an author and a responsible 50
53. Illustration 52: Digital Architecture Framework prescribes that each feature must be realized by an use case apart the non functional ones 51
54. Illustration 53: A feature can have multiple realizing use cases 51
55. Illustration 54: a Use case cannot realize two different features 52
56. Illustration 55: while a requirement can have both a requirement and a feature as children this is not suggested. 52
57. Illustration 56: two use cases are created that belong to the same process 52
58. Illustration 57: ChiExternalWorkers are one of six different types of Digital Architecture Framework actors 53
59. Illustration 58: Use case diagram with candidates. The use case name contains the primary actor. 55
60. Illustration 59: Chain of Feature and Use Case with a linked external worker 55
61. Illustration 60: Hierarchy of Use Case, Feature and Actor 56
62. Illustration 61: a scenario describing a field sale business process 56
63. Illustration 62: an actor description is necessary to understand how implement this role 57
64. Illustration 63: as a graphic convention the primary actor is placed to the left 57
65. Illustration 64: fields of the use case 58
66. Illustration 65: Use cases can be organized in ChiBusinessProcesses, that are special kind of packages. 59
67. Illustration 66: 2 actors participating in an UC 59
68. Illustration 67: Activity diagrams are visualized with a special symbol and stored within the Use Case 60
69. Illustration 68: Example of Use case activity 60
70. Illustration 69: a ChiObject with a "created" status 61
71. Illustration 70: Example of a loop 62
72. Illustration 71: each Request or Response ChiObject must be an instance of a ChiNode that inherits from the class Container. 63
73. Illustration 72: A Request ChiObject transports the import parameters for the service 64
74. Illustration 73: a finished activity diagram. It include a start, Request and response containers, a decision main success scenario and an extension. 64
75. Illustration 74: CIM domain objects as ChiNodes 65
76. Illustration 75: required Diagrams in the Use case package. 66
77. Illustration 76: the CIM lib in the default model 67
78. Illustration 77: example of instantiating a ChiNode from CIM library with a ChiObject 67
79. Illustration 78: a new meter ChiNode is created to support the instance of the ChiObject 67
80. Illustration 79: IdentifiedObject Inheritance from Node 68
81. Illustration 80: In this example EEproducts extend the CIM class EnergyProduct to add the property "price" 68
82. Illustration 81: EntityBase contains the same properties as the CIM class IdentifiedObject. Must be the super Class of all the new created ChioNode within a Energy & Utility project. 69
83. Illustration 82: enumeration are not directly supported by EA 70
84. Illustration 83: ChiNodes with different relatioships 71
85. Illustration 84: relevant node diagram example 73
86. Illustration 85: example of mapping between properties: “nation” is mapped to “country” 74
87. Illustration 86: flow of the implementation from a Requirement up to his Controller 75
88. Illustration 87: use cases implemented by controllers 76
89. Illustration 88: example of Production rule modeled with activity elements 78
90. Illustration 89: the Use case context menu offers a generate controller feature 79
91. Illustration 90: Use Case refined by a ChiController 79
92. Illustration 91: example of a good model report, most of the checks are green 80
93. Illustration 92: most recently edited objects” list with the standard information. You can customize it as needed. 81
94. Illustration 93: customization of the list's field 81
95. Illustration 94: by clicking on a element on the most recently edited objects open in his location on the tree. 81
96. Illustration 95: Web Browser: the package distribution in this screen shot reflect that too many elements are in to few packages. this is an hint for restructuring your model 82
97. Illustration 96: In this example Use Case sub model a total of 48 objects exists of which 16 are not rule conform. 82
98. Illustration 97: in this example 3 use cases don't have an ActivityIntial with a name equals to the trigger 83
99. Illustration 98: the list of available templates can be expanded for your project needs. 83
100. Illustration 99: by exporting the documentation starting from package "Goals" only the elements in this package will be present in the document. 84
101. Illustration 100: the diagram "Goals with requirements" shows requirements from another package. If the option "diagrams" has been selected the documentation will include those elements 85
102. Illustration 101: the Use case element has a special menu for exporting the documentation 85
103. Illustration 102: Architecture of a EA installation. EA is a front end for the Digital Architecture Framework BOR that call the generator. This can execute a certain transformation 86

Appendix 5: Glossary

| Term | Definition |
| --- | --- |
| Analysis sub model | Part of the Digital Architecture Framework model containing the Use cases. |
| BOR | Business Object Repository |
| Cartridge | A set of workflows, constrains, M2M and M2T transformation that can be run by the DAFgen. |
| DAF | the Greek letter  (see [http://en.wikipedia.org/wiki/DAF\_(letter)](http://en.wikipedia.org/wiki/Chi_(letter)) initial of the name Digital Architecture Framework (Χρόνος). See Digital Architecture Framework |
| DAF Metamodel | Also Digital Architecture Framework Metamodel. A collection of UML stereotypes and their relationship. |
| ChiController | A UML class that describes a certain behavior. |
| *DAFgen* | The Digital Architecture Framework Generator. this is a Java program that runs Cartridges for M2M or M2T. It can be executed with or without the EA. |
| Chi**Goal** | A goal is a desirable (and measurable!) status. |
| ChiNode | A UML class with some additional properties describing the Domain objects. It describes a certain structure. |
| ChiObject | An instance of a ChiNode |
| ChiRequirement | A requirement is a condition that can be true or false. |
| Digital Architecture Framework | In Greek mythology Digital Architecture Framework was the father of Zeus. Name of the Metamodel and the related methodology. |
| Digital Architecture FrameworkBusinessUseCases | a short story, a step by step descriptions how a feature works. |
| CIM | Common Information Model |
| Container | A special class that transports domain objects in an use case. |
| CWB | Digital Architecture Framework Web Browser a web tool that shows the actual completition status of a Digital Architecture Framework Model. |
| EA | Digital Architecture Framework Web Modeler, a program in the cloud for the creation of Digital Architecture Framework UML models |
| Domain sub model | Part of the Digital Architecture Framework model containing the Requirements. |
| E&U | Energy and utilities. The industry where Digital Architecture Framework was extensively applied. |
| Features | A Feature is a fact that set his requirement to true |
| Goal | A desirable (and measurable!) status |
| Issues | A fact that set the Requirement to false. |
| IT | Information technology |
| M2M | Model to Model. A transformation of a Model in another model. For example an XML model can be transformed in a UML model. Or a plan UML model can be enhanced with the stereotypes of Digital Architecture Framework. |
| M2T | Model to Text. A transformation where a Model is transformed to Text. For example a Digital Architecture Framework Model can be transformed in a Java Class, in a HTML file or in a Word Document. |
| DE | Digital Engineering. A conception where all the knowledge is hosted in a model. |
| NfR | Non functional Requirements |
| PRR | Production Rule Representation |
| Requirement | A Requirement is a condition that can be true or false. |
| Requirements sub model | Part of the Digital Architecture Framework model containing the Requirements. |
| RfP | Request for Proposal |
| stereotype | An additional information that can be applied to any UML classifier adding additional information. |
| UC | Use case. A part of a process. |
| UCPA | Use Case Point Analysis. A way to forecast project's effort using a collection of Use Cases. |
| UML | Unified Markup Language. A language that describe how to create valid models |

Table 6: Glossary

1. see [https://secure.wikimedia.org/wikipedia/en/wiki/Profile\_%28UML%29](https://secure.wikimedia.org/wikipedia/en/wiki/Profile_(UML)) for information about profiles [↑](#footnote-ref-2)
2. <https://sparxsystems.com/enterprise_architect_user_guide/16.1/welcome/index.html> [↑](#footnote-ref-3)
3. You can download the last CWM version from: <http://sourceforge.net/projects/olympos/> . [↑](#footnote-ref-4)
4. See also <http://www-128.ibm.com/developerworks/rational/library/2870.html>

   [↑](#footnote-ref-5)
5. for a detailed specification: <https://github.com/FreeTAKTeam/DigitalArchitectureFramework/blob/master/Docs/DafDocumentation.docx> [↑](#footnote-ref-6)
6. The interpretation of the display\_type is done by DefaultValueRenderer or its subclasses. [↑](#footnote-ref-7)