

ComVantage

284928

***Collaborative Manufacturing Network
for Competitive Advantage***

**D3.1.2 – Specification of Modelling Method Including
Conceptualisation Outline
(public)**



Grant Agreement No.	284928
Project acronym	<i>ComVantage</i>
Project title	Collaborative Manufacturing Network for Competitive Advantage
Deliverable number	D3.1.2
Deliverable name	Specification of Modelling Method Including Conceptualisation Outline
Version	V 1.0
Work package	WP 3 – Secure Information Model
Lead beneficiary	UNIVIE
Authors	Dimitris Karagiannis (UNIVIE), Robert Buchmann (UNIVIE), Patrik Burzynski (UNIVIE), Jasmin Brakmic (UNIVIE)
Reviewers	David Orensanz (BOC), Patricia Ortiz (INNO)
Nature	R – Report
Dissemination level	PU – Public
Delivery date	29/01/2014 (M29)

Executive Summary

This report presents the conceptualisation approach and results for the development of the *ComVantage modelling method*. It is the second iteration and it is a support document for the implementation phase developed in task 3.4, thus it outlines a set of method concepts and constructs, to be implemented by the modelling prototypes. For the current iteration, the method refines the conceptual coverage given by the first iteration (D3.1.1) and its initial adaptations (D6.2.1, D7.2.1, D8.2.1).

Methodologically, the refinement takes a top-down integrative approach, filling identified gaps, removing redundancies and extending the method scope with additional coverage. The goal is to support the modelling of requirements for mobile apps, Linked Data and access control, within a business context described with the means proposed in the initial iteration. The models created with the proposed method become a knowledge externalisation channel, between stakeholders working on different levels of abstraction and detail, as well as between the design time and run time of the *ComVantage* architecture. Particular emphasis is placed on model interoperability, which is enabled through an RDF serialisation of models using a generic schema, independent of model semantics. Thus, models become themselves Linked Data resources, opening significant potential for model awareness on the run-time side, as reflected by the app orchestration approach proposed in WP5.

In further steps, the method will be adjusted with final adaptations to be developed in tasks 6.2, 7.2 and 8.2.

Table of Contents

EXECUTIVE SUMMARY	2
TABLE OF CONTENTS.....	3
LIST OF FIGURES	5
LIST OF TABLES.....	6
1 OVERVIEW	8
1.1 INTRODUCTION.....	8
1.2 SCOPE OF THE DOCUMENT	8
1.3 RELATED DOCUMENTS.....	9
1.4 TERMS AND ACRONYMS USED IN THIS DOCUMENT	9
2 THE COMVANTAGE MODELLING METHOD SPECIFICATION.....	10
2.1 APPROACH FOR MODELLING METHOD REFINEMENT.....	10
2.1.1 Envisioned Modelling Procedure	11
2.1.2 Assumptions about the Meta ² -model	12
2.2 STRUCTURE OF MODELLING METHOD	12
2.3 GENERAL MODELLING METHOD CONCEPTS.....	14
2.3.1 Aspect-independent Concepts.....	14
2.3.2 Generally used Properties of Concepts	16
2.4 ASPECT-SPECIFIC MODELLING METHOD CONCEPTS	16
2.4.1 Concepts in Motivator Aspect	17
2.4.2 Concepts in Participant Aspect	18
2.4.3 Concepts in Procedural Aspect	19
2.4.4 Concepts in Collaborative Aspect	22
2.4.5 Inter-Aspect Concepts	23
2.5 SCOPE-SPECIFIC MODELLING METHOD CONCEPTS.....	24
2.5.1 Specialised Concepts in Structural Aspect.....	25
2.5.2 Specialised Concepts in Behavioural Aspect.....	38
2.6 ASSIGNMENT OF CONCEPTS TO ASPECT SPECIALISATIONS	46
2.6.1 Business Scope	47
2.6.2 Enterprise Scope	50
2.6.3 Requirements Scope	51
2.6.4 App development Scope.....	54
2.6.5 App execution set-up Scope	56
2.6.6 Evaluation Scope.....	57
2.7 MECHANISMS AND ALGORITHMS.....	62
2.7.1 Determine Instances/Templates for required Capabilities	62
2.7.2 Derivation of Participant collaboration	62
2.7.3 Interaction stepper	63
2.7.4 Derivation of Orchestration.....	63

2.7.5 Gathering access requirements	64
2.7.6 Access requirement coverage check	64
2.7.7 Calculation of KPIs/Variables	64
2.7.8 Simulation of Procedural models.....	65
2.7.9 Business model evaluation	65
2.7.10 Serialisation of models as Linked Data	66
2.7.11 Comparison of model serialisations in Linked Data.....	66
2.7.12 Model querying.....	66
3 IMPLEMENTATION SPECIFIC RECOMMENDATIONS.....	68
3.1 RECOMMENDED CLASSES, RELATIONS AND ATTRIBUTES	68
3.2 PROPOSED NOTATION GUIDELINES	73
3.3 RECOMMENDED APPROACHES FOR MECHANISM AND ALGORITHM IMPLEMENTATIONS	89
3.3.1 Recommended approach for Determine Instances/Templates for required Capabilities	89
3.3.2 Recommended approach for Derivation of Participant collaboration	90
3.3.3 Recommended approach for Interaction stepper	91
3.3.4 Recommended approach for Derivation of Orchestration.....	93
3.3.5 Recommended approach for Gathering access requirements.....	95
3.3.6 Recommended approach for Access requirement coverage check	95
3.3.7 Recommended approach for Calculation of KPIs/Variables.....	96
3.3.8 Recommended approach for Simulation of Procedural models	96
3.3.9 Recommended approach for Business model evaluation	98
3.3.10 Recommended approach for Serialisation of models as Linked Data	98
3.3.11 Recommended approach for Comparison of model serialisations in Linked Data	103
4 OUTLOOK AND CONCLUSION.....	104
5 REFERENCES.....	105
6 APPENDIX	106
6.1 EXAMPLE QUERY FOR CAPABILITY MATCHING.....	106
6.2 METAMODEL DIAGRAMS	107
DISCLAIMER	114

List of Figures

Figure 1: Positioning of the current document in the <i>ComVantage</i> task flow.....	9
Figure 2: Process of refinement from previous specification	11
Figure 3: Understanding separable decomposition	16
Figure 4: Possible assumption based on proposed Action/Event alternation for followed by relations.....	21
Figure 5: More intuitive interpretation of multiple incoming followed by relations.....	21
Figure 6: Global Procedural elements with context assigned to followed by relations.....	22
Figure 7: Proposed solution for element reuse using Representative elements.....	22
Figure 8: Dependency control of AND type, where occurrences x and y should be the same	50
Figure 9: Example mockup for the Value structure group	81
Figure 10: Example mockup for the Market structure group	82
Figure 11: Example mockup for the Business structure group	82
Figure 12: Example mockup for the Location structure	82
Figure 13: Example mockup for the Value exchange flow group.....	83
Figure 14: Example mockup for the Business model group	83
Figure 15: Example mockup for the Enterprise structure group.....	83
Figure 16: Example mockup for the Business process group.....	84
Figure 17: Example mockup for the Participant collaboration group	84
Figure 18: Example mockup for the Mobile support structure group	85
Figure 19: Example mockup for the Information space group.....	86
Figure 20: Example mockup for the Interaction flow group	86
Figure 21: Example mockup for the Navigation map group.....	87
Figure 22: Example mockup for the Orchestration group.....	87
Figure 23: Example mockup for the Notification exchange group.....	88
Figure 24: Example mockup for the KPI structure group	88
Figure 25: Example mockup for the Evaluation process group.....	89
Figure 26: Example mockup for the Interaction stepper	92
Figure 27: Metamodel for high abstraction level (General concepts)	107
Figure 28: Metamodel for middle abstraction level (Aspect-specific concepts).....	108
Figure 29: Metamodel for low abstraction level (Scope-specific concepts)	109
Figure 30: Metamodel concept hierarchy (Templates).....	110
Figure 31: Metamodel concept hierarchy (other).....	111
Figure 32: Metamodel relation hierarchy	112
Figure 33: Metamodel for low abstraction level (Scope-specific concepts) of the recommended implementation	113

List of Tables

Table 1: Overview of model specialisations for different Scopes	14
Table 2: Concepts of Motivator Aspect	17
Table 3: Relation concepts of Motivator Aspect	17
Table 4: Concepts of Participant Aspect	18
Table 5: Relation concepts of Participant Aspect	18
Table 6: Concepts of the Procedural Aspect	20
Table 7: Relation concepts of the Procedural Aspect	20
Table 8: Concepts of the Collaborative Aspect	22
Table 9: Relation concepts between the different Aspects	24
Table 10: Scope-specific concepts and relation concepts of the Motivator Aspect	28
Table 11: Scope-specific concepts and relation concepts of the Participant Aspect	37
Table 12: Scope-specific concepts and relation concepts of the Procedural Aspect	44
Table 13: Scope-specific concepts and relation concepts of the Collaborative Aspect	46
Table 14: Concepts in Value structure group	47
Table 15: Concepts in Market structure group	47
Table 16: Concepts in Business structure group	48
Table 17: Concepts in Location structure group	48
Table 18: Concepts in Value exchange flow group	48
Table 19: Concepts in Business model group	49
Table 20: Formulas for calculating occurrences in Business models	49
Table 21: Concepts in Enterprise structure group	50
Table 22: Concepts in Business process group	51
Table 23: Concepts in Participant collaboration group	51
Table 24: Concepts in Permission pool group	52
Table 25: Concepts in Mobile support structure group	52
Table 26: Concepts in Information space group	53
Table 27: Concepts in Requirements process group	54
Table 28: Concepts in the Interaction flow group	55
Table 29: Concepts in the Navigation model group	55
Table 30: Concepts in Orchestration group	56
Table 31: Concepts in Notification exchange group	57
Table 32: Concepts in KPI structure group	58
Table 33: Concepts in Evaluation process group	58
Table 34: Recommended functions for Variables and KPIs	62
Table 35: Recommended classes and attributes	71
Table 36: Recommended relations and attributes	73
Table 37: Proposed notations for classes	79
Table 38: Examples of alternative notations for Points of interaction	81
Table 39: Linked Data constructs recommended for the serialisation	100

Table 40: Recommended transformation of modelling objects into Linked Data	103
--	-----

1 OVERVIEW

1.1 Introduction

This is the second iteration (of two) of a deliverable whose goal is to specify the envisioned *ComVantage* modelling method as knowledge structure comprising several building blocks: a modelling procedure, a modelling language and functionality that takes input from model information. The first iteration had a broad focus on supply chain management, while this iteration is rather focused on modelling business processes with requirements for the technological specificity of *ComVantage*, namely mobile apps, Linked Data and access control. The models aim to semantically connect these requirements with the business context (business model), process evaluations, process motivators and involved assets or entities. Further refinements, capturing the domain specificity of the application areas Plant Engineering and Commissioning, Customer-oriented Production, Mobile Maintenance (WP6, WP7 and WP8) and change requests based on hands-on experience with the intermediary versions of the modelling prototypes (D3.4.1) will be covered by the final adaptation deliverables (D6.2.2, D7.2.2, D8.2.2).

The document is organised as follows: this introduction is followed by a scope statement correlated with the description of work, a positioning of the document in the context of related deliverables and a list of terms and acronyms used in the document. Section 2.1 bridges the approaches of the first and the current iteration, with the envisioned modelling procedure described in section 2.1.1. Section 2.2 gives an overview of the updated modelling stack. Sections 2.3-2.5 present the high abstraction concepts giving the semantic basis for further specialisations across the modelling stack. Section 2.6 groups these concepts according to multiple scopes covered by the method (from the business model to app requirements and evaluation). Section 2.7 suggests functionality that can support a modeller in further processing various parts of the modelling stack. Section 3 shifts the abstraction level to implementation recommendations where concrete guidelines are given regarding modelling notation and functionality. The document ends with conclusions and an outlook to future developments envisioned for the adaptation tasks.

1.2 Scope of the Document

According to the description of work, "this task aims to design the business process modelling method for *ComVantage* and conceptualisation as a basis for the development on a meta-modelling platform. This includes the design of a hybrid, process-based method for dynamic collaboration processes between different user roles and regarding different types of information".

The business process modelling paradigm is integrated with the technological specificity adopted by *ComVantage*, where activities rely on two key types of IT resources: mobile support and Linked Data. The first iteration of the method specification took a bottom-up approach, by designing a hybrid metamodel integrating concepts identified in application area scenarios (e.g. product, service, role, app, business entity) with broader practices coming from the supply chain and business process management literature (SCOR, VRM, e3 value etc.).

The current iteration refines the conceptual landscape with a top-down approach, driven by a set of concepts of higher abstraction, capturing semantics that recur throughout the initial specification. In addition, new elements come into focus, such as evaluation and information space modelling, and a more cohesive semantic integration is applied on the existing elements. Further adaptations (tasks 6.2, 7.2 and 8.2) are expected to reflect domain specificity from the application areas and changes derived from hands on experience with the currently available implementations of the method.

1.3 Related Documents

The current document's position in the project context and with respect to related deliverables and tasks is presented in Figure 1. The relations expressed in the figure are as follows:

- Application areas' refined scenarios and functional requirements provided concepts to be reflected by the models;
- The modelling method, as it is presented in the current task will be further specialised in the adaption deliverables from WP6, 7 and 8;
- The modelling method also relates to technological work packages through an RDF export of models enabling model analysis and processing outside a modelling tool;
- The modelling method is being implemented in modelling prototypes within task 3.4.

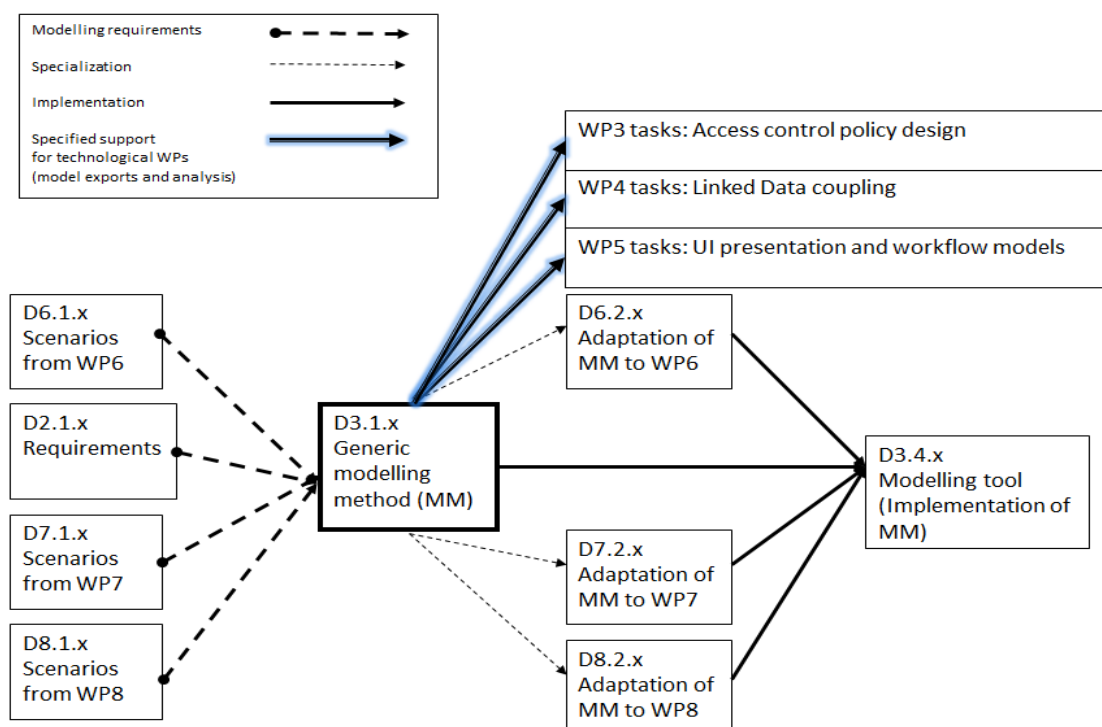


Figure 1: Positioning of the current document in the *ComVantage* task flow

1.4 Terms and Acronyms used in this Document

ERM (ER-Model) – Entity-Relationship Model, a model focusing on describing entity (/object) types, their attributes and relations between them.

KPI – Key Performance Indicator

OMI – Open Models Initiative

POI – Point of interaction (improvised term introduced through D8.2.1 to indicate an app feature of component that enables the user-app interaction)

RDF – Resource Description Framework, provides the format for model interoperability

SPARQL – the standard language for querying Linked Data expressed as RDF

2 THE COMVANTAGE MODELLING METHOD SPECIFICATION

2.1 Approach for Modelling Method Refinement

The first iteration of the modelling method (as specified in D3.1.1 and its adaptations) has been the result of a bottom up integration and a knowledge acquisition effort taking input from the scenario descriptions, the generic project requirements (further reflected in the conceptual coverage of the project ontologies) and the literature. The input was further structured in model types, resulting in a modelling stack and a metamodel developed incrementally in the first stage of the project.

The gained domain insight and an analysis of the resulted semantic landscape lead to the identification of a set of recurring high level generic concepts. For example:

- control flow aspects have been recurring on various levels of abstraction or detail – in business process models and thread models (D3.1.1), in interaction flows or SIPO models (D8.2.1);
- decomposition-driven modelling approaches have been identified in product models, market models, organisational models (D3.1.1, D7.2.1), service models, machine state models, app models (D8.2.1);
- collaboration views have been recurring in business models, scope models (D3.1.1).

This inspired a top-down revision of the modelling stack, driven by a specialisation of the high level generic concepts across several scopes dictated by the *ComVantage* multifaceted domain: the business scope, the requirements scope, the evaluation scope and others, to be detailed further in this section. An improved stack, structured somewhat similarly to Zachman's framework (Zachman, 1987) has been designed in order to capture the new cohesion and to enable the identification of gaps to be filled. Multiple changes have been thus triggered in the metamodel, for example:

- merging product modelling (D3.1.1, D7.2.1) and service modelling (D8.2.1) in a common value structure model type that covers any mix of products and services (allowing “product servitisation” models), including the variability of their decomposition (inspired by feature-oriented analysis in software production lines (Kang et al., 1990);
- splitting the SIPO model type (D8.2.1) in order to obtain finer granularity of app and data requirements;
- replacing purely visual models (causality diagram from D3.1.1) with semantically richer models (KPI influence structures);
- creating new model types, especially those reflecting interactions between different types of entities or assets;
- shifting the focus of simulation from a system dynamics approach (hinted at, but not specified in D3.1.1) towards a process-based approach in order to satisfy the description of work requirement for deriving workflow efficiency indicators from models.

This refinement process is depicted in Figure 2. The structure of this document will reflect this top-down refinement and a modelling tool implementer may stop at the desired level of abstraction (reading lower specialisations and implementation guidelines as recommendations). However, specialised versions of the abstract concepts are more useful than generic ones, and should be preferred. In some cases, generic concepts are not further specialised, meaning that the lowest available specialisation should be used.

We also emphasise the fact that users should be aware that they are modelling their own perspective, hence the data stored in models should reflect this perspective (for example, market shares for market segments).

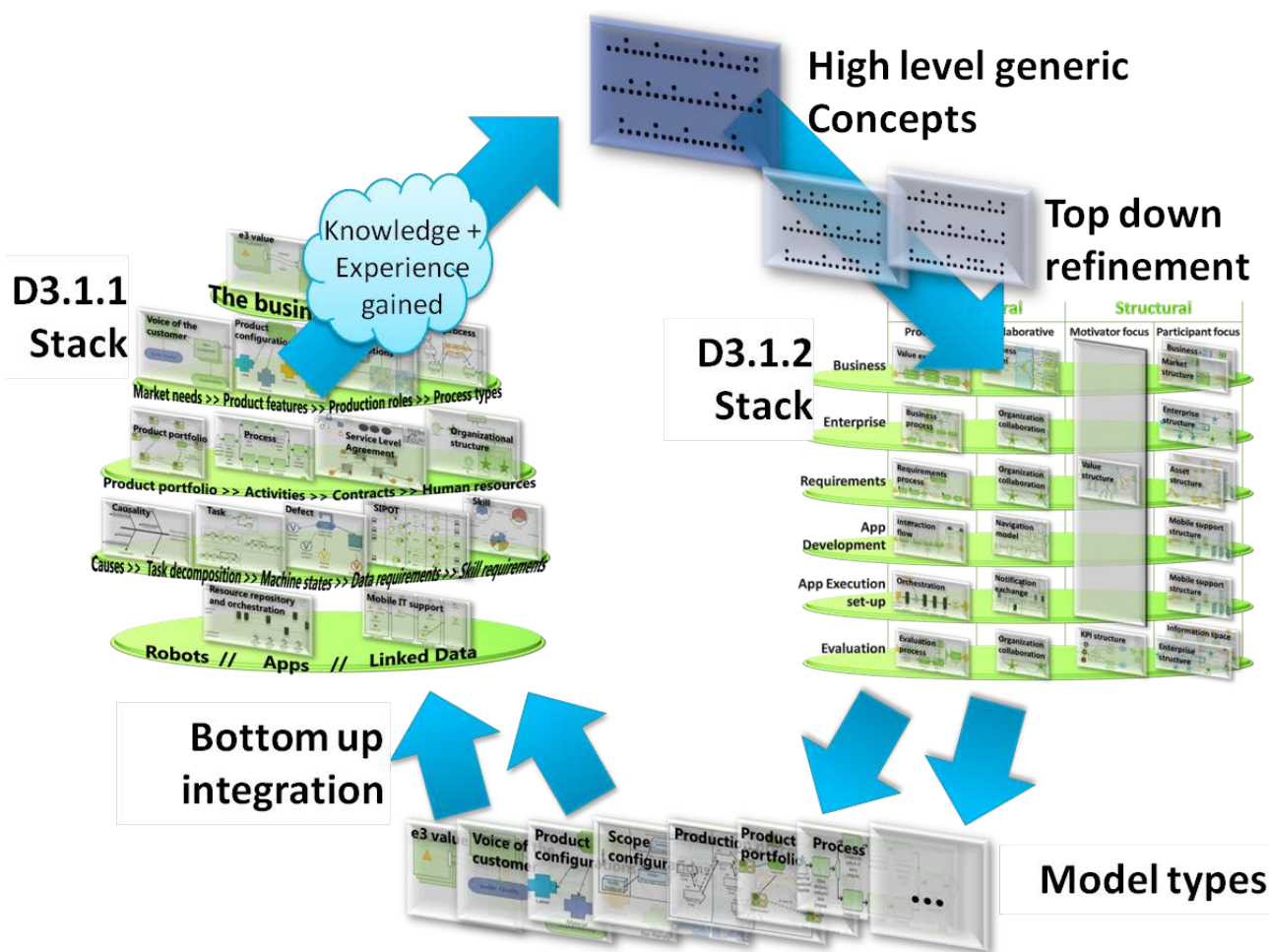


Figure 2: Process of refinement from previous specification

2.1.1 Envisioned Modelling Procedure

The modelling procedure drives the new modelling stack structure. The aim for the current iteration is to fulfil the description of work requirement of providing modelling means for key aspects such as mobile apps, Linked Data and access policies. Process evaluation and collaboration must also be described. The envisioned procedure is as follows:

1. The modeller describes the business context (including the business model mapped on the targeted market structure and the values to be provided);
2. The modeller describes the operational business processes that must be performed in order to sustain a regular enactment of the value exchanges envisioned by the business model;
3. The modeller describes app requirements along the operational business processes;
 - In a collaborative approach, app designers refine these requirements to early mockup proposals in order to validate their interpretation on requirements;
4. The modeller describes data requirements along the operational business processes and in relation with the app requirements;
 - In a collaborative approach, app designers refine these requirements in order to design a required information space;

5. The modeller describes permission requirements for the required information;
 - In a collaborative approach, information owners collect the access requirements and extend the models with granted permissions based on decisions of the data owners;
6. The modeller evaluates the efficiency of the modelled control flows with respect to: a) various abstractions of the general notions of “cost/value” (estimated time, money, different types of wastes or value consumptions); b) semantically modelled KPIs for which data sources have been identified and evaluation processes have been modelled (independent of the general flow of this procedure).

2.1.2 Assumptions about the Meta²-model

As (Kern et al, 2011) indicates, most meta-metamodels rely on the dichotomy between concepts and properties (relations, attributes) or can be reduced to such a dichotomy. We mention however several assumptions made about the meta-metamodel in order heighten the level of generality (hence reuse) of the method compared to its initial iteration (which followed some constraints of the ADOxx metamodeling platform):

- Relations are binary; relations of higher arity (with higher number of participants) can be created by adding an intermediate concept linking through binary relations to all required participants;
- Relations are concepts, so they can be endpoints for other relations; a workaround similar to the previous point can be employed in implementations that do not support this;
- Unlike the approach of D3.1.1, there is no more prescribed delimitation of models and model types. The partitioning of the conceptualisation outcome in models should be driven by implementation (mostly usability) decisions. The concept groups to be described in the next sections for various scopes and specialisations can be interpreted for implementation purposes as “model types”, but this is not mandatory from a specification point of view (they can be merged or further split);
- Concept names should be unique (hence reusing a concept name means reusing a concept). The modelling objects should get unique or reused identifiers relative to their potential usage when models are exported in the Linked Data cloud (in order to support model linking);

The assumptions are also made in relation to the proposal of having models exported in the Linked Data cloud, with each modelling object having a global identity (URI), possibly with properties relative to a contextual graph representing the point of view of the model author and a base URI further indicating model provenance.

2.2 Structure of Modelling Method

Through the refinement, several changes have been made to the structure of the modelling method. One of those changes concerns the range of the described concepts. It now covers very generic concepts like template, decomposition or implication, which are then further specialised towards more application oriented concepts like activities or sequence relations. This gradual specialisation is described throughout sections 2.3 (very generic concepts), 2.4 (aspect-specific concepts) and 2.5 (specialised aspect-specific concepts).

Another change altered the modelling stack, by structuring its elements according to *Scopes* and *Aspects*. *Scopes* depict different application domains like *Enterprise* or *App development*. They are meant to be selected and applied as required by the use case. This also means that additional *Scopes* can be added if necessary. The *ComVantage Modelling Method* covers the following *Scopes*:

- **Business** – description of high level of business and revenue, describing business scenarios and the partners involved in those.

- **Enterprise** – description of products or services and how they are accomplished through business processes as well as the involved participants.
- **Requirements** – capturing requirements on participants for business processes and describing how they should be employed.
- **App development** – description of apps, their structure and the requirements posed on them.
- **App execution set-up** – description of how certain apps should be orchestrated.
- **Evaluation** – description of (performance) indicators and how they are evaluated.

The mapping of the *Scopes* on the procedure steps is as follows:

- Step 1. uses the *Business Scope*
- Step 2. uses the *Enterprise Scope*
- Step 3. uses the *Requirements, App development* and *App execution set-up Scopes*
- Step 4. uses the *Requirements, App development* and *App execution set-up Scopes*
- Step 5. uses mostly the *Requirements Scope* and other *Scopes* as required
- Step 6. uses mostly the *Evaluation Scope* and other *Scopes* as required

Aspects on the other hand are fixed and provide a certain view on a *Scope*. The *Aspects* considered by the modelling method are:

- **Behavioural** – this *Aspect* focuses on the description of task dependent elements, mostly through implications of sequence or dependency.
 - **Procedural** – this *Aspect* focuses on actions that have to be performed and their relation to one another according to their execution sequence in time. It can be considered a facet of the *Behavioural* aspect.
 - **Collaborative** – this *Aspect* focuses on the collaboration between *Structural elements* according to one or several *Procedural* descriptions. It can be considered a facet of the *Behavioural* aspect.
- **Structural** – this *Aspect* focuses on the description of element structures through decomposition.
 - **Motivator** – this *Aspect* focuses on the motivators for the *Procedural* aspect, e.g. the shifting of values like money or specific products during a process execution, and their structure. It can be considered a facet of the *Structural* aspect.
 - **Participant** – this *Aspect* focuses on the participants of a *Procedural* aspect, e.g. the liable entities and assets employed in the execution of a process, and their structure. It can be considered a facet of the *Structural* aspect.

The elements of the different *Aspects* are linked with one another and in the case of the *Structural Aspect* an element can be a *Participant* in one *Scope* and a *Motivator* in another *Scope*. However, since the modelling method follows a (business) process centric approach, the *Behavioural Aspect* is usually in the centre of those links. Table 1 shows the *Scopes* and their specialisations of the *Behavioural Aspects* as well as the recommended focus on specialisations of the *Structural Aspect*. It focuses on the parts necessary to cover the envisioned procedure of the *ComVantage Modelling Method* described in section 2.1.1.

Scope \ Aspect	Behavioural		Structural	
	Procedural	Collaborative	Motivator focus	Participant focus
Business	Value exchange flow	Business model	Value structure	Market and Business structure
Enterprise	Business process	Participant collaboration		Enterprise structure
Requirements	Requirements process	Participant collaboration		Asset structure

Aspect Scope	Behavioural		Structural	
	Procedural	Collaborative	Motivator focus	Participant focus
App development	Interaction flow	Navigation model		Mobile support structure
App execution setup	Orchestration	Notification exchange		Mobile support structure
Evaluation	Evaluation process	Participant collaboration	KPI structure	Enterprise structure and Information space

Table 1: Overview of model specialisations for different Scopes

Metamodels for the different abstraction levels, as well as describing the concept hierarchy can be found in the appendix (section 6.2, Figure 27 through Figure 32).

2.3 General Modelling Method Concepts

In this section the general concepts, which are independent of any *Aspect* or *Scope*, are described. Since they are recurring on multiple levels of abstractions they will be presented once in this section. They provide a common ground and are therefore often specialised by more specific concepts.

2.3.1 Aspect-independent Concepts

The here presented general concepts are called *Aspect*-independent, because they do not belong to one specific *Aspect* and instead are used in several or sometimes even all *Aspects*.

2.3.1.1 Templates, Instances and Instantiation

Throughout this document, we are referring to modelling constructs of type “instance” and “template”, which might cause confusion with other two popular uses of the term “instance”:

- In context of the “model-reality instance” relation, which is often implied in business process modelling, where the instance is a reality enactment of a model - see (Weske, 2007);
- In context of the “metamodel-instance model”, which is often implied in metamodeling.

For our approach, templates represent sets of instances, meaning that instances can fit in the template. Templates can generally be considered a variation space specified and limited through requirements or capabilities. Therefore instances of a template also fulfil the capabilities posed by it. In some cases templates can be so specific that only one instance can fit, making it difficult to properly distinguish between the two. Also templates can represent both existing (as-is) and possible (to-be) instances. During an actual reality enactment, templates have to be replaced by instances, either by finding available ones or by creating new instances.

Therefore, for our purposes, instances represent things that are on the lowest level of specialisation that is of modelling interest. Hence they can represent different things than instances found in the real system under study. For example, while a “performer” modelling instance would have a 1-to-1 mapping to a real person, an information modelling instance could represent multiple records different from one execution to another, or an application modelling instance could represent multiple software licenses running on multiple computers that can be involved in different executions. Therefore instances (in the sense used here) can be further instantiated in the reality enactment; but this is not in the scope of the current conceptualisation.

To summarise, throughout this document “instance” will be used relative to “template”, and both refer to specific types of modelling objects. Instantiation simply denotes the explicit descriptive relation between instances and templates, indicating into which templates an instance fits. The relation is named “instance of” and the inverse is “has instance”.

2.3.1.2 Inclusion types and Control elements

Inclusion types allow describing variability of dynamic parts where some choices have to be made. They only apply on a certain set of relations on which they pose some conditions. Three inclusion types are considered:

- **AND** - indicating an “All of” inclusion type, meaning that all of the relations have to be chosen. Therefore it dictates how many and which specific relations should be used.
- **OR** - indicating a “Some of” inclusion type, meaning that one or more relations have to be chosen. It forces neither a specific amount nor a selection of relations to be used.
- **XOR** - indicating a “One of” inclusion type, meaning that exactly one relation has to be chosen. Therefore it dictates how many, but not which specific relations should be used.

An example for employing inclusion types is in a process model where the path splits. Depending on the selected inclusion types either one, some, or all of the outgoing paths should be taken. Using an inclusion type other than “AND” leads to variability. Therefore those are only applied for templates, since those can represent more than one instance. By default, if an inclusion type is missing, assume the “AND” inclusion.

Since the inclusion types apply to a set of relations, it has to be possible to somehow group them together. However, because the here presented approach only uses directed relations between two elements, the grouping has to happen through a different element. A solution is achieved by using a *Control element* with the desired inclusion type for a certain relation type. This control can be part of an already available concept, or be a spare concept if necessary. They can be one of two types: *Split* or *Merge*¹. The *Split* indicates that the inclusion type should be applied on the outgoing relations, while the *Merge* does the same only for incoming relations. Therefore they heavily depend on the direction of modelling. For example in an organisation diagram an *Organisational unit* would be denoted as an AND-Split on decomposition relations.

2.3.1.3 Decomposition

Decomposition is a relation describing of what smaller parts a larger part consists and is here called “contains”, while the inverse is called “contained by”. Because of decomposition, the larger part can be considered a (decomposition) set of the smaller parts linked through decomposition and some additional unknown part as well. Similar to instantiation, a larger part can contain several smaller parts and a smaller part can also be part of several larger things. There can however be exceptions to this, depending on where the decomposition is applied. Also, the structure created through decomposition should form a directed acyclic graph, meaning that loops in decomposition are not allowed.

Decomposition is generally described using the AND-inclusion type. However, variability can be achieved in some cases by also using different ones. For example, when describing products, the OR- and XOR-inclusion types can be used to denote the potential for customisability. In addition to the inclusion types the decomposition relation can also be marked as separable or inseparable. An inseparable decomposition indicates that the larger must contain the smaller, while with a separable relation the smaller part is considered optional. Having a separable contains relation to a smaller thing is the same as a contains relation to a decomposition set of XOR-inclusion type that itself contains the smaller thing and an empty set (see Figure 3).

2.3.1.4 Implication

The implication, simply called “implication” here, is a relation that indicates what conclusion can be drawn by which premises (or simply: when/if A, then B). This means that the premises are the source and the conclusion is the target of the relation. Using the previously described inclusion types it is possible to build

¹ Everything that is denoted as a *Split* or *Merge* is also a *Control element* for a certain type of relation. A concept can also be a *Control element* for several different relations at once.

complex premises and conclusions for implications. Also in most cases the implication is not used directly, but specialised first, because it is too vague by itself. Additionally there is a special type of implication called *Prohibition*. It is simply the implication of the negated conclusion. The inverse is simply called “inverse implication”

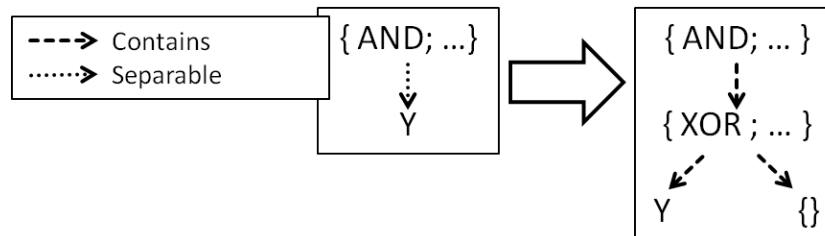


Figure 3: Understanding separable decomposition

2.3.1.5 Specialisation

Specialisation is a relation which indicates that something is more specific than something else and is here called “specialisation of”, while the inverse is called “generalisation of”. It is applied to *Templates*, where the specialised template has more restrictive capabilities or boundaries than the general one. This also means that in terms of space it cannot go outside the boundary set by the general template and because of this all specialisation relations of an element are of AND-inclusion type. Also, because of the bottom-up direction for modelling specialisation, the specialisations are *Merges*. So generally a *Template* can be the specialisation of several other *Templates*, indicating an overlapping. There can however be exceptions to this.

2.3.2 Generally used Properties of Concepts

Several general properties are recommended for most concepts. These properties are:

- **Name / Label** – Something for the human to identify the elements.
- **Global identifier** – An identifier for a global scope, which allows identifying elements anywhere. It is recommended to use URI’s, in order to further link the elements to RDF descriptions and Linked Data.
- **Property collector** – A collection of additional statements about the element. It is meant to allow adding information about elements which is not covered through the properties prescribed by the modelling method (e.g. using own custom types for the element) and should be machine usable. Therefore it is recommended to use a structure from which RDF statements can be derived.
- **Description** – To describe the element further for a human and provide information for the human, like why things are a certain way.

They should be applied where they are meaningful or useful. For example all of those make sense to be available in *Actions*. For *Decisions* on the other hand a property denoting the question to be answered would be more meaningful than a name or label. *Constants* directly represent a value and therefore it makes little sense to attach a name to them. *Implications* usually do not need any of the above properties besides maybe a description. It is left to the readers’ discretion to decide which of those properties to use in which concepts.

2.4 Aspect-specific Modelling Method Concepts

The *Aspect*-specific concepts are more specific than the previously presented general (*Aspect*-independent) concepts. They represent a granularity which is better fit for specific purposes, while still being general enough to allow for use and further specialisation into *Scope*-specific concepts (see section 2.5). In order to bridge the descriptions from the different aspects, special relation concepts are provided, and following the process centric approach the behavioural aspect is involved in all of them. A short description about each

Aspect can be found starting on page 13. The descriptions of the concepts can be found in Tables 2 through 9.

The descriptions follow a more structured approach in this section through tables which provide the names of the concepts, which other concepts they specialise (can be general or *Aspect*-specific concepts), their description and in case of relations which types of elements they can connect and the name of the inverse relation in brackets. Note that the hierarchy created by the general and *Aspect*-specific concepts does apply everywhere. In order to prevent the collision of names used here for concepts and normal words the names will be written in italics to distinguish them from the normal words (e.g. “*Process*” for the thing depicted by the name given here and “process” for the general concept). Also some simple self-explanatory names for sets of concepts are used (e.g. “*Procedural element*” for any element of the *Procedural Aspect*).

2.4.1 Concepts in Motivator Aspect

The following concepts are used to describe the motivators of an enterprise:

Concept	Specialisation of	Description
Motivating value	<i>Split-Decomposition set</i> (unspecified inclusion type); <i>Template</i>	It is a value that represents some form of motivator (like a product, money or a KPI), so someone (e.g. person, organisation) holds some value in it. It is usually changed by something (e.g. an activity), so values are often “created” or “consumed”.
Description function	<i>Split-Decomposition set</i> (unspecified inclusion type);	It is a function that is executed on the things it contains, like composition, addition, subtraction etc. It should be used to detail complex <i>Motivating values</i> .

Table 2: Concepts of Motivator Aspect

In addition the following relation concepts are used:

Concept	Specialisation of	Source(s) → Target(s)	Description
mandates (mandated by)	<i>implication</i>	<i>Motivator value</i> → <i>Motivator value</i> ; <i>Description function</i>	It indicates that one <i>Motivating value</i> also mandates another <i>Motivating value</i> or <i>Description function</i> . For example the embroidery of a shirt (<i>Motivating value</i>) can mandate or prohibit certain shirt colours.
specialised value of (general value of)	<i>specialisation of</i>	<i>Motivating value</i> → <i>Motivating value</i>	It describes the specialisation of <i>Motivating values</i> . One limitation here is that unlike the general specialisation, a <i>Motivating value</i> can be the specialisation of only one other <i>Motivating value</i> .

Table 3: Relation concepts of Motivator Aspect

While the specialised concepts should be preferred, the general concepts should also be available in this *Aspect*, most notably:

- **contains (contained by)** – It should be used to decompose Values and Value sets. When instantiating such relations in the *Motivator Aspect* it should also be specified if they are separable or inseparable.

2.4.2 Concepts in Participant Aspect

In the Participant aspect the following concepts are used:

Concept	Specialisation of	Description
Participant template	<i>Split-Decomposition set (AND-inclusion type); Template</i>	It is a template for participants (like a role). The actual instance/execution of a process will however use <i>Participant instances</i> .
Capability	<i>Participant template</i>	It is a more detailed description of a <i>Participant template</i> , meant to provide grounds for comparison of templates and instances (like a skill). Depending on the point of view a <i>Capability</i> can be fulfilled or required. Examples for human capabilities would be skills or knowledge.
Access means	<i>Capability</i>	Represents a means of how a template or instance can be accessed (e.g. through a query, a phone number etc.). The <i>Access means</i> can also be separated into two subtypes: “Secured access” or “Opened access”.
Participant instance	<i>Split-Decomposition set (AND-inclusion type); Instance</i>	It represents an instance (i.e. something on the lowest level of specialisation that is of interest) of a participant (like a person or an app). The instance can be used during the execution of a process, if it fits the requirements, which can be stated through the templates or capabilities.
Liabe entity	---	It is a participant that can hold responsibility, similar to a “legal person” from law (like a person). It should be used as a second type for templates, capabilities and instances.
Asset	---	It is the complement of <i>Liabe entity</i> , i.e. a participant that cannot hold responsibility (like a machine). An <i>Asset</i> can however be the cause of something. It should be used as a second type for templates, capabilities and instances.

Table 4: Concepts of Participant Aspect

In addition the following relation concepts are used:

Concept	Specialisation of	Source(s) → Target(s)	Description
has capability (capability of)	<i>contains (inseparable)</i>	<i>Participant template</i> → <i>Capability</i>	Special decomposition which explicitly decomposes a template into its capabilities.
fulfils (fulfilled by)	<i>instance of</i>	<i>Participant instance</i> → <i>Participant template</i>	It indicates which templates and capabilities an instance fulfils. The capability set of an instance are all the <i>Capabilities</i> linked through this relation.
owned by (owns)	<i>implication</i>	<i>Participant</i> → <i>Liabe entity</i>	It is the implication that involving the <i>Asset</i> also requires somehow the involvement of the owner (e.g. their permission). In general it denotes ownership.

Table 5: Relation concepts of Participant Aspect

While the specialised concepts should be preferred, the general concepts should also be available in this *Aspect*, most notably:

- **contains (contained by)** – It should be used to decompose *Templates* and *Instances*. However, it should not be used to decompose *Templates* into *Instances* or vice versa. When instantiating such relations in the *Participant Aspect* it should also be specified if they are separable or inseparable.
- **specialisation of (generalisation of)** – It should be used to describe specialisation of *Templates*, in most cases only between *Templates* of the same type.

In this *Aspect* also the control of access to the *Assets* is handled through “Resource usage policies”. Those cover

- the Subject - the participant that can get access
- the Action - the action that can be performed by the *Subject*
- the Resource - the participant that is accessed, i.e. upon which the *Action* is performed

The idea is to describe those policies through permission rules, which state what is allowed. This means that *Subjects* trying to perform an *Action* on a *Resource* for which no fitting permission rule is available should be denied. Also a link could be established between *Access means* and permission rules, to indicate upon which rules the means are based on (e.g. this means of access is available because of this permission rule).

2.4.3 Concepts in Procedural Aspect

In order to capture procedural descriptions the following concepts are used:

Concept	Specialisation of	Description
Process	<i>Split-Decomposition set</i> (unspecified inclusion type); <i>Template</i>	It is a set of <i>Procedural elements</i> , describing an <i>Action</i> from a higher level of abstraction from a certain point of view. Its main purpose is to provide an interface between descriptions from different <i>Scopes</i> for the same <i>Action</i> without enforcing a specific granularity.
Action	<i>Split-Decomposition set</i> (unspecified inclusion type); <i>Template</i>	An <i>Action</i> represents something that is performed in order to change a certain state (i.e. “you Act”). They are triggered by an event and performing them leads to an event, since otherwise the <i>Action</i> would have been meaningless. Performing an <i>Action</i> usually requires some form of resource or value, like time, money, personnel, computer etc. Generally the stages “Started” and “Finished” can be distinguished, with “Finished” implying “Started”, and an <i>Action</i> being performed if it has “Started” but not “Finished”.
Action type	<i>Merge-Specialisation set</i> (AND-inclusion type); <i>Template</i>	An <i>Action type</i> represents the possible types of actions that can be performed (like a read or a write action type). They can further be described through specialisation.
Event	<i>Split-Decomposition set</i> (unspecified inclusion type); <i>Template</i>	Represents the occurrence of certain changes in the state which are of some form of interest (e.g. “Order arrived”, “One hour passed since last execution”, “Machine analysed” etc.). <i>Events</i> are also both the causes for starting <i>Actions</i> and the result of finishing other <i>Actions</i> .
Initiation event	<i>Event</i>	It represents an <i>Event</i> that starts a whole <i>Process</i> . Since it is from the point of view of a specific process it is valid only in its context. This means that the <i>Initiation event</i> of one process can be the <i>Termination event</i> of another. Therefore, while the general <i>Event</i> itself can be reused in another process, it does not mean that it is also an <i>Initiation event</i> there.

Concept	Specialisation of	Description
Termination event	<i>Event</i>	It represents an <i>Event</i> that ends a path in a whole <i>Process</i> , which can be successful or not. Since it is from the point of view of a specific process it is valid only in its context. This means that the <i>Termination event</i> of one process can be the <i>Initiation event</i> of another. Therefore, while the general <i>Event</i> itself can be reused in another process, it does not mean that it is also a <i>Termination event</i> there.
Control	<i>Implication set</i> (unspecified inclusion type)	It allows controlling the sequence in a process through the available inclusion and set types (i.e. AND, OR, XOR; Split, Merge). This control is specific to a process. Therefore the reuse of the same <i>Control</i> elements throughout several processes is limited, only reasonable in processes of the same <i>Process</i> decomposition hierarchy.

Table 6: Concepts of the Procedural Aspect

In addition the following relation concepts are used:

Concept	Specialisation of	Source(s) → Target(s)	Description
followed by (preceded by)	<i>implication</i>	<i>Action; Event; Control</i> → <i>Action; Event, Control</i>	It is the implication of the target starting when the source has finished. Instantaneous elements are considered to be finished the moment they start. It is used to provide a sequence of <i>Actions</i> and other elements in the order they should be executed. Additionally the relation should also allow specifying additional conditions (i.e. premises of the implication) beside the one provided by it.
impacts (affected by)	<i>implication</i>	<i>Action</i> → <i>Action; Event</i>	It is the implication of performing the source resulting in an impact on the target. The impact can be positive (e.g. enables, supports, notifies etc.) or negative (e.g. suspends, terminates).
detailed by (describes)	<i>contains</i> (inseparable)	<i>Action</i> → <i>Process</i>	The decomposition relation for <i>Actions</i> , to describe them from different points of views by different <i>Processes</i> (e.g. business process view, requirements view, orchestration view etc.). However, all <i>Processes</i> should still describe the same <i>Action</i> . The targeted <i>Process</i> can be considered a sub-process of the <i>Action</i> .
has part (part of)	<i>contains</i> (inseparable)	<i>Process</i> → <i>Action; Event; Control; Process</i>	The decomposition relation from <i>Processes</i> to all other <i>Procedural elements</i> .

Table 7: Relation concepts of the Procedural Aspect

While the specialised concepts should be preferred, the general concepts should also be available in this *Aspect*, most notably:

- **specialisation of (generalisation of)** – It should be used to describe specialisation of *Action types*. For example there can be a global “Any action” type, which is further specialised into “Read” and “Write” *Action types*.
- **instance of (has instance)** – It should be used to indicate the *Action types* for an *Action*.

From the descriptions of Table 6 and Table 7 one can see that in the *Procedural Aspect*:

1. A chain of *followed by* relations should alternate between *Actions* and *Events*. It is recommended to start with an (*Initiation*) *Event* and end with a (*Termination*) *Event*, because they indicate the cause and the result.
2. An *Action* should only be directly decomposed into *Processes*. *Processes* however can be decomposed into *Actions* or other *Processes* to structure them. It is recommended to have an *Action* as the root of the decomposition, because *Processes* only describe *Actions* from a certain point of view.

Since every *Action* should be followed by an *Event*, which in turn should be followed by another *Action*, it is possible to omit one of those types during modelling and assume that something without further description is there (see Figure 4). This allows process model descriptions similar to the ones found in D3.1.1² to still fit into the here presented approach.

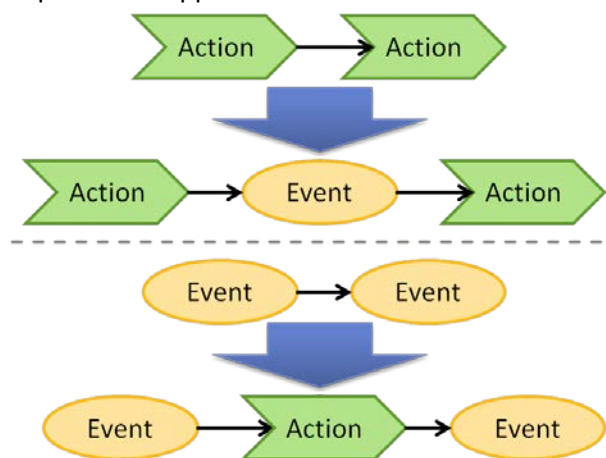


Figure 4: Possible assumption based on proposed Action/Event alternation for followed by relations

Also, in most cases the AND-inclusion type is used by default. This means that if an element other than a merging *Control* would have two incoming *followed by* relations, both preceding *Actions* would have to finish in order to continue. However for processes it is more intuitive to treat such a case with an XOR-inclusion type as shown in Figure 5.

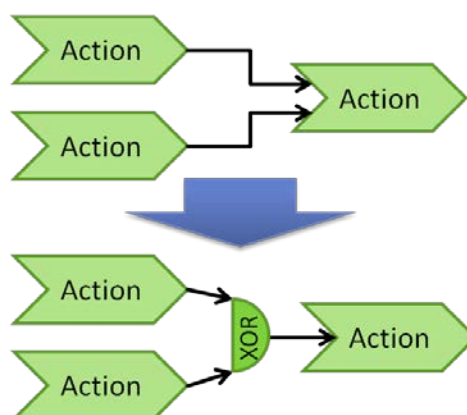


Figure 5: More intuitive interpretation of multiple incoming followed by relations

The decomposition allows the same *Action* to be reused in several *Processes* (see section 2.3.1.3). There is however a problem with the direct reuse of *Actions* and *Events*: certain things only apply in a specific *Process* or other certain circumstances, which will here simply be called “context”. Most notably are the

² Consider “Activities” to be *Actions* and “Process start” and “Process end” to be *Initiation/Termination events*.

flowed by relations and the *Initiation* or *Termination* event. This means that the context has to be assigned to the parts to create valid models. A simple example can be seen in Figure 6, where the process context is assigned in square brackets to the *followed by* relations.

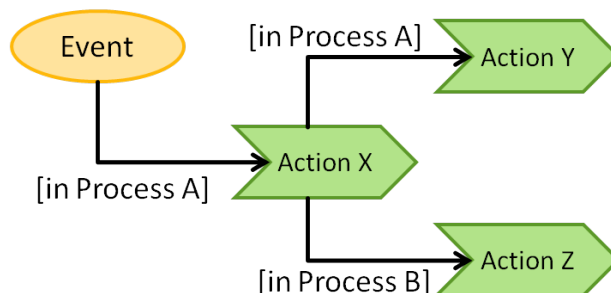


Figure 6: Global Procedural elements with context assigned to followed by relations

However this is not very intuitive and should be improved upon. The proposed solution here is to use *Representative elements*. They represent a certain element like *Action* or *Event* (which is stored somewhere in a pool or repository) with some additional context attached, most notably the process to which they belong through decomposition and their sequence in the process. The *Representative elements* are not meant to be reused and not be part of several *Processes*. They should be considered *Instances*, which are instances of the *Template* they represent (e.g. specific *Action*, *Event* ...), since they describe the lowest level of specialisation that is of modelling interest. An example based on Figure 6 can be seen in Figure 7, where the dashed elements are *Representative elements* representing things from the “Reuse pool”. This also allows reusing the same *Action* more than once in the same *Process*. An alternative solution would be to prevent the reuse of *Procedural elements*.

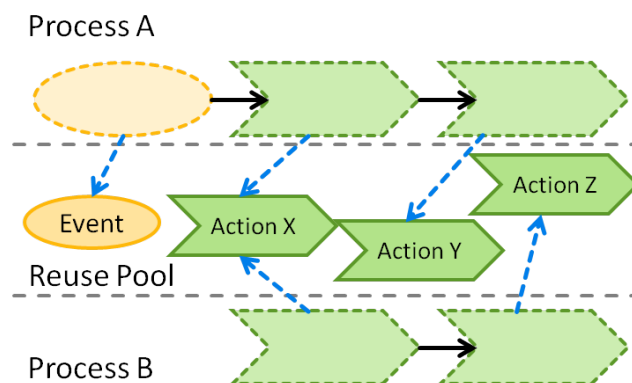


Figure 7: Proposed solution for element reuse using Representative elements

2.4.4 Concepts in Collaborative Aspect

The collaborative descriptions heavily depend on the reuse of concepts from the *Procedural* Aspect and the *Structural* Aspects (*Motivator/Participant*) to provide a collaborative overview. Therefore the concepts presented in the other Aspects can be used here in addition to the following ones:

Concept	Specialisation of	Description
Collaboration	<i>Split-Decomposition set (unspecified inclusion type); Template</i>	It depicts the collaboration of <i>Structural elements</i> for one or several <i>Actions/Processes</i> . It generally focuses on a certain set of <i>Structural elements</i> , like <i>Liable entities</i> .
Participant involvement	<i>Instance</i>	A <i>Participant involvement</i> depicts a certain contribution of a participant. It is not simply the participant, but also their involvement.

Table 8: Concepts of the Collaborative Aspect

While the specialised concepts should be preferred, the general concepts should also be available in this *Aspect*, most notably:

- **contains (contained by)** – It should be used to describe decomposition of a *Collaboration* into the elements it contains.
- **implication (inverse implication)** – It should be used to describe the additional things depicted in the *Collaborative Aspect*. However, because of its very general nature, it should first be specialised further.

The same problem from the *Procedural Aspect* for the decomposition and reuse of elements can be found here as well. However, the same solutions with either using *Representative elements* or preventing the reuse also apply. In many cases the *Participant involvement* can be used as the *Representative element*.

2.4.5 Inter-Aspect Concepts

In the previous sections the concepts for each *Aspect* have been described, without any relations between the *Aspects*. Such relations are however necessary in order to describe the different elements in an integrated manner. Therefore some additional *Inter-Aspect* relation concepts are introduced:

Concept	Specialisation of	Source(s) → Target(s)	Description
influences (influenced by)	<i>implication</i>	<i>Procedural element</i> → <i>Motivating value</i>	It implies that a <i>Procedural element</i> leads to the change of a <i>Motivator value</i> . This change can be positive or negative (e.g. create or consume). Additionally it should also allow stating a quantity for the change. Through this the costs and duration of an <i>Action</i> can be denoted. An example would be “Action X negatively influences 500 (quantity) € (value)” to describe that performing an <i>Action</i> costs 500€.
requires (required by)	<i>implication</i>	<i>Procedural element</i> → <i>Participant template, Participant instance</i>	It implies the use of a <i>Participant</i> in the execution of a <i>Procedural element</i> . Typically this also means that the used <i>Participant instance</i> is blocked for the time being. Under circumstances it can still be possible to perform the <i>Procedural element</i> even when the <i>Participant</i> is not available (e.g. when it was only necessary to improve efficiency). If the same <i>Participant instance</i> is assigned to different procedural elements, then it means that the same reality instance (e.g. the same human, the same data set) should be reused throughout one enactment of those elements. This is not guaranteed when using <i>Participant templates</i> , as they represent a variability space and different reality instances fitting a template could be involved during the same enactment. Additionally the <i>requires</i> relation can be described similarly to the permission rules (see end of section 2.4.2). Through this access requirements can be stated on which

Concept	Specialisation of	Source(s) → Target(s)	Description
has responsible (responsible for)	<i>requires</i>	<i>Procedural element → Liable entity template, Liable entity instance</i>	<p>the permission rules should be based.</p> <p>A specialisation of <i>requires</i>, indicating the <i>Liable entity</i> that is responsible for a <i>Procedural element</i>. The approval of the responsible is required, which can be for example verbally, in writing or by a general rule.</p> <p>A <i>Procedural element</i> should have only one assigned responsible <i>Liable entity</i>.</p>
has performer (performs)	<i>requires</i>	<i>Procedural element → Participant template, Participant instance</i>	<p>A specialisation of <i>requires</i>, indicating the participant that actually performs a <i>Procedural element</i>, typically a <i>Liable entity</i>. If only one <i>Liable entity</i> performer is assigned to an element, and no responsible, then it will be assumed that the performer is responsible. If more than one performer is assigned then it is considered collaboration. If the assigned performers are from different organisations, then it is an inter-organisation collaboration. The exact flow of collaboration can be further detailed through a <i>Process</i>.</p> <p>For covering access control: for each <i>Participant template</i> assigned that is not a performer or responsible, there should be at least one performer that has the right to access it.</p>
is for (has collaboration)	<i>implication</i>	<i>Collaboration → Action</i>	<p>It implies that the <i>Collaboration</i> can be derived from the <i>Actions</i> it is linked to. This also means that the <i>Collaboration</i> is based upon those.</p>
involved participant (involved in)	<i>implication</i>	<i>Participant involvement → Participant template, Participant instance</i>	<p>It implies the use of a <i>Participant</i>. It denotes which <i>Participant</i> is involved in a <i>Participant involvement</i>. In one <i>Collaboration</i> only a limited set of possible <i>Participant template</i> subtypes should be used (e.g. <i>Role</i>, <i>Liable entity</i> or <i>Interaction component</i> and <i>Point of interaction</i>).</p>

Table 9: Relation concepts between the different Aspects

2.5 Scope-specific Modelling Method Concepts

The Scope-specific concepts represent the last level of concept specialisation and it is closest to the granularity used in D3.1.1. They are specialisations of the *Aspect*-specific concepts. While it is necessary to have special relations to link models from the different *Aspects*, the already available relations can be used to bridge model parts from different *Scopes* (e.g. through decomposition).

The here presented descriptions follow a more structured approach where each concept is described by its own table containing a description, the concepts it specialises, the considered sub-types as well as properties that should be available (omitting properties described in section 2.3.2). The considered sub-

types indicate what additional categorisations for the concepts should be available. The concepts are later on (see section 2.6) grouped, which can be instantiated into useful descriptions. Again the hierarchy created by the general and *Aspect*-specific concepts does apply (e.g. properties have to be also available in the specialised versions) and references to the names will be written in *italic*.

Again, while the specialised concepts should be preferred, the more general concepts (*Aspect*-independent and –specific) should also be available in every *Aspect* and *Scope*. For example the general decomposition relation *contains* should be used to decompose *Business entities* into *Organisation units* and further decomposed into *Performers*.

2.5.1 Specialised Concepts in Structural Aspect

2.5.1.1 Concepts mostly focused in Motivator Aspect

Value	
Specialisation of: <ul style="list-style-type: none"> • <i>Motivating value</i> (AND-inclusion type) 	<p>The <i>Value</i> is a general concept that is used to describe products and services which are provided or consumed by an enterprise, as well as more general or abstract values like money, time, warranty or eco-friendliness. It should always represent one object (e.g. “Shirt”, not “10 Shirts”).</p> <p>It can be decomposed into other <i>Values</i> to describe what it is made up of and customisability can be captured through the use of <i>Value sets</i>. Generally the price and cost of a <i>Value</i> should be described in models of the <i>Procedural Aspect</i>. However, both can be captured to a certain degree through decomposition. The cost of a <i>Value</i> can be captured through the prices of the values it is decomposed into, while the price would be the cost plus the directly assigned money.</p>
Subtypes	
Value type	The general type of the <i>Value</i> . Among the envisioned types are “Material”, “Component”, “Service” and “Abstract”, although others are possible as well. It should be only one of those types.
Excitement type	The excitement type is a simplification of the <i>Customisation feature</i> classification from D3.1.1. Two types are considered: “Basic” and “Competitive”. “Basic” values are considered inherent by the value provider, while “Competitive” values are the parts on which the provider focuses its competing strategies.
Axiological type	It can be “Value” (desirable) or “Anti-Value” (non-desirable, or “waste” in the sense of the Lean paradigm (Bicheno, 2009)). While it is better to have more of a <i>Value</i> , the opposite is true for “Anti-Values” (i.e. preferred to have less). Examples would be “Health” and “Illness” or “Delivery speed” and “Time to delivery”. In some cases it is more intuitive to use <i>Anti-Values</i> to describe things (e.g. <i>Time to delivery</i> , <i>Execution cost</i>). It is recommended to choose only one approach and use it through all descriptions in a company, e.g. either use “Money” and negatively influence it or use “Execution costs” and positively influence it.

Value set	
Specialisation of: <ul style="list-style-type: none"> • <i>Description function</i> • <i>Split-Decomposition set</i> 	<p>The <i>Value set</i> is used to describe possible options or customisability in the structure of a <i>Value</i>. Using them a general structure for a product can be created, for which special configuration can be provided by using the <i>configuration of</i> and <i>implies</i> or <i>prohibits</i> relations.</p>

Subtypes	
Inclusion type	Based on the three available inclusion types, the subtypes can be “of all” (AND), “of some” (OR) or “of one” (XOR). It should be only one of those types.

has value (value of)	
Specialisation of: <ul style="list-style-type: none"> contains 	This relation concept should be used to describe the decomposition of <i>Values</i> and <i>Value sets</i> .
Allowed source concepts: <ul style="list-style-type: none"> Value Value set 	Allowed target concepts: <ul style="list-style-type: none"> Value Value set
Subtypes	
Separable type	Special type indicating if the target element can be separated from source element. The possible types are therefore “inseparable” and “separable”.
Properties	
Quantity	The quantity of the target that is contained in the source.

mandates value (mandated by value)	
Specialisation of: <ul style="list-style-type: none"> mandates 	This relation indicates what target <i>Values</i> or <i>Value sets</i> have to be also available/used (implied) or absent/not-used (prohibited), when the source <i>Value</i> is available/used. For example this relation can be used to describe the use or prohibition of certain colours of a shirt when it is also to be embroidered.
Allowed source concepts: <ul style="list-style-type: none"> Value 	Allowed target concepts: <ul style="list-style-type: none"> Value Value set
Subtypes	
Implication type	Indicates if the target is implied (“implies”) or if the negation is implied (“prohibits”).

configuration of (configured into)	
Specialisation of: <ul style="list-style-type: none"> specialised value of 	This special relation concept should be used to describe specialisation of <i>Values</i> . A <i>Value</i> should be the specialisation of only one other <i>Value</i> (i.e. no multiple inheritances).
Allowed source concepts: <ul style="list-style-type: none"> Value 	Allowed target concepts: <ul style="list-style-type: none"> Value

annihilates (annihilates)	
Specialisation of: ---	A special symmetric relation associating a normal <i>Value</i> to its <i>Anti-Value</i> and vice versa. Allows creating bridges between the two if necessary for computations.
Allowed source concepts: <ul style="list-style-type: none"> Value 	Allowed target concepts: <ul style="list-style-type: none"> Value

Constant	
Specialisation of: <ul style="list-style-type: none"> • <i>Motivating value</i> 	A <i>Constant</i> represents a specific numerical or boolean value, like 2, 5, 14, {23, 42}, {1, 2, 3, 4, 5}, “true” (1) or “false” (0). Those are used to describe functions or calculations as well as the target that performance indicators should achieve.
Subtypes	
Value type	The values can be separated in “Numeric” or “Boolean”. However, even boolean values can be represented as numbers (1 or 0) and vice versa (not 0 or 0).
Set type	Two set types are considered for numerical values: “Single”, representing just one value and “(ordered) Set”, representing several values in a specific order.
Properties	
Value	The value it represents.

Variable	
Specialisation of: <ul style="list-style-type: none"> • <i>Motivating value</i> • <i>Description function</i> 	A <i>Variable</i> represents a value that has to be determined first, usually through calculation or retrieval. This can be achieved through the execution of a function.
Subtypes	
Value type	The values can be separated in “Numeric” or “Boolean”. However, even boolean values can be represented as numbers (1 or 0) and vice versa (not 0 or 0).
Set type	Two value types are considered for numerical values: “Single”, representing just one value and “(ordered) Set”, representing several values in a specific order.
Variable type	In this context <i>Variables</i> can be “Named”, in which case it should be possible to reuse them, or “Unnamed/Anonymous”.
Properties	
Function	The function that is executed in order to determine the value of the <i>Variable</i> (e.g. addition, subtraction, standard deviation, load values from table etc.). Some recommended functions that should be available can be found in section 2.6.6.

KPI	
Specialisation of: <ul style="list-style-type: none"> • <i>Variable (Named)</i> 	The <i>KPI</i> , which is short for Key Performance Indicator, is a measure used to evaluate performance and has a desired target value that should be achieved.

Level	
Specialisation of: <ul style="list-style-type: none"> • <i>Motivating value</i> 	The <i>Level</i> is something that can be achieved by a KPI, if a desired target value is reached. Typically they are colour coded (e.g. green, blue, yellow, orange, red)

has operand (operand for)	
Specialisation of: <ul style="list-style-type: none"> contains 	Through this relation, the operands or parameters for the functions used by <i>Variables</i> and <i>KPIs</i> are specified. Using this approach and the laws of mathematics it is possible to calculate the values.
Allowed source concepts: <ul style="list-style-type: none"> Variable 	Allowed target concepts: <ul style="list-style-type: none"> Variable Constant
Properties	
Order number	The order number should be used to indicate the sequence of operands for functions where it is necessary, for example for subtraction or division. Therefore, before calculating a value the operands should first be arranged based on this property in ascending order and passed to the function in the resulting sequence.

covers also (covered by)	
Specialisation of: <ul style="list-style-type: none"> specialised value of 	This specialisation type is used to describe precedence between the available <i>Levels</i> . It indicates that the target value of this relations source <i>Level</i> is more restrictive than the one of the relations target <i>Level</i> .
Allowed source concepts: <ul style="list-style-type: none"> Level 	Allowed target concepts: <ul style="list-style-type: none"> Level

achieves (achieved by)	
Specialisation of: <ul style="list-style-type: none"> mandates 	This relation denotes the implication that the <i>KPI</i> can achieve the targeted <i>Level</i> . The premise for when and how exactly this <i>Level</i> is achieved is covered by the <i>condition for</i> relation.
Allowed source concepts: <ul style="list-style-type: none"> KPI 	Allowed target concepts: <ul style="list-style-type: none"> Level

condition for (under condition)	
Specialisation of: <ul style="list-style-type: none"> mandates 	This relation represents the premise for a <i>KPI</i> achieving a certain <i>Level</i> . It means that when the condition (the source) is met, that also the conclusion (the targeted implication) should be carried out.
Allowed source concepts: <ul style="list-style-type: none"> Variable (Single, Boolean) 	Allowed target concepts: <ul style="list-style-type: none"> achieves

Table 10: Scope-specific concepts and relation concepts of the Motivator Aspect

2.5.1.2 Concepts mostly focused in Participant Aspect

Location	
Specialisation of: <ul style="list-style-type: none"> Capability Asset 	A <i>Location</i> represents a place that can be visited. It is generally described by an area of unspecified size and can be as big as the universe or as small as a dot.
Subtypes	
Type	Two general types are considered: “Physical”, denoting physical locations (e.g. country, city etc.) in which case the <i>Location</i> is considered to be a template of all possible locations there. The other type is “Digital”, which represents a location in the digital world and is often represented by a URL.

	However, a digital location can also represent locations not denoted by a URL, like the internal database of a phone.
Dependency type	It indicates if the <i>Location</i> is dependent on a point of reference, meaning that a location can be “Absolute” (e.g. Vienna) or “Relative” (e.g. next to machine). For a relative location the point of reference should be provided as well.
Properties	
Area	The area that is represented by a <i>Location</i> . For absolute physical locations it can be for example “Austria”, “Vienna, Austria”, a specific address or latitude and longitude. In this case it is recommended to use a generally understood format (e.g. Google Maps). For digital locations it is recommended to use something that a browser can understand. For relative locations some hint on the point of reference should be provided, for example through natural language.

Market segment	
Specialisation of: <ul style="list-style-type: none">• <i>Participant template</i>• <i>Liable entity</i>	A <i>Market segment</i> represents a part of the market, which is classified by one or several characteristics, to identify groups. Usually, <i>Market segments</i> are used to depict groups of customers.
Subtypes	
Market interest	The <i>Market segments</i> can further be typed through the interest in them, resulting in “Targeted” or “Not targeted” segments.
Properties	
Share	The share a company holds in the <i>Market segment</i> .

Characteristic	
Specialisation of:	A <i>Characteristic</i> represents an attribute by which a <i>Market segment</i> is described and if applicable it also represents a value for that attribute. Examples for <i>Characteristics</i> would be certain age ranges, traits or attitudes like “Sustainability”
<ul style="list-style-type: none"> • <i>Capability</i> • <i>Liable entity</i> 	

justifies (justified by)	
Specialisation of:	This is a special relation between different facets of an <i>Aspect</i> . It is the implication that a value must be available to satisfy a certain characteristic of a segment. Therefore, it can be used to describe the necessity of certain <i>Motivating values</i> based on the <i>Characteristics</i> of a <i>Market segment</i> .
<ul style="list-style-type: none"> • <i>implication</i> 	
Allowed source concepts:	Allowed target concepts:
<ul style="list-style-type: none"> • <i>Characteristic</i> 	<ul style="list-style-type: none"> • <i>Motivating value</i>

Business entity	
Specialisation of:	A <i>Business entity</i> represents an entity that is conducting relevant business. It can be the own company, another company, a part of a company or a specific customer.
<ul style="list-style-type: none"> • <i>Participant instance</i> • <i>Liable entity</i> 	

Business role	
Specialisation of: <ul style="list-style-type: none"> • <i>Participant template</i> • <i>Liabile entity</i> 	A <i>Business role</i> represents a template for <i>Business entities</i> . For example “material supplier”, “customer” or “competitor” would be <i>Business roles</i> .

Business capability	
Specialisation of: <ul style="list-style-type: none"> • <i>Capability</i> • <i>Liabile entity</i> 	A <i>Business capability</i> represents the ability of a business to provide some value for some compensation.

Business entity access	
Specialisation of: <ul style="list-style-type: none"> • <i>Access means</i> • <i>Liabile entity</i> 	A <i>Business entity access</i> denotes how a <i>Business entity</i> can be accessed. It represents its location and how it can be contacted.
Properties	
Contact information	Some information about how to contact a <i>Business entity</i> .

Organisation unit	
Specialisation of: <ul style="list-style-type: none"> • <i>Participant instance</i> • <i>Liabile entity</i> 	An <i>Organisation unit</i> represents a part of an organisation or <i>Business entity</i> . It allows describing a structure through decomposition.
Subtypes	
Unit type	There are different types of <i>Organisation units</i> possible, like “Division”, “Department” or “Team”.
Properties	
Preferred occupation	Through this a preferred number of positions in an <i>Organisation unit</i> can be specified.
Function	The general function an <i>Organisation unit</i> performs.

Performer	
Specialisation of: <ul style="list-style-type: none"> • <i>Participant instance</i> • <i>Liabile entity</i> 	A <i>Performer</i> represents a human that works in a company and performs some form of task or job there. The <i>Performer</i> occupies a position at the company. Often they are described through their <i>Skills</i> and <i>Knowledge</i> .
Properties	
Availability	The availability of a <i>Performer</i> .

Role	
Specialisation of: <ul style="list-style-type: none"> • <i>Participant template</i> • <i>Liabile entity</i> 	A <i>Role</i> represents a template for <i>Performers</i> . Examples for Roles are “Expert”, “CEO”, “Programmer” or “Administrative personnel”. Can be described through both specialisation and through <i>Skills</i> and <i>Knowledge</i> .
Properties	
Preferred occupation	Through this a preferred number of <i>Performers</i> for a <i>Role</i> can be specified.

Skill

Specialisation of:

- *Capability*
- *Liable entity*

A *Skill* represents capabilities that a person can be proficient in. It is driven by performing a task. The level of skill proficiency is attached to the relation between the *Skill* and the participant.

Knowledge

Specialisation of:

- *Capability*
- *Liable entity*

A *Knowledge* represents knowledge that a person can have. It is driven by having some information. The level of knowledge is attached to the relation between the *Knowledge* and the participant.

has chief (chief of)

Specialisation of:

- *contains* (inseparable)

The *has chief* relation indicates the person that is the chief of a *Business entity*. It also indicates that the person has some overall responsibility for all tasks performed by their employees.

Allowed source concepts:

- *Business entity*

Allowed target concepts:

- *Performer*

provided value (can be provided by)

Specialisation of:

- *has capability*

This relation denotes what value is provided by a *Business capability*. This should also be used to denote the time to delivery. For this there needs to be a “Time to delivery” *Anti-Value*, where the quantity denotes the duration.³

Allowed source concepts:

- *Business capability*

Allowed target concepts:

- *Motivating value*

Properties

Quantity

The amount of the value that can be provided at once.

necessary compensation (compensation for)

Specialisation of:

- *has capability*

This relation denotes what values should be given as compensation in return for providing a value.

Allowed source concepts:

- *Business capability*

Allowed target concepts:

- *Motivating value*

Properties

Quantity

The amount of the value used for compensation.

provided at (can find business capability)

Specialisation of:

- *has capability*

This relation denotes where the value denoted by a *Business capability* can be provided geographically.

Allowed source concepts:

- *Business capability*

Allowed target concepts:

- *Location*

³ Actually the *Value* provided is “Delivery speed” (which is X divided by time). However, in many cases the human just thinks of the delivery time. Therefore the possibility of describing and using the *Anti-Value* “Time to delivery” is provided.

access business through (accesses business)

Specialisation of: • <i>has capability</i>	This relation denotes what access means should be used to contact a <i>Business entity</i> .
Allowed source concepts: • <i>Business entity</i>	Allowed target concepts: • <i>Business entity access</i>

has business location (business location of)

Specialisation of: • <i>contains</i> (inseparable)	This relation is used to assign a <i>Location</i> to a <i>Business entity access</i> , and therefore also denoting the location of a <i>Business entity</i> .
Allowed source concepts: • <i>Business entity access</i>	Allowed target concepts: • <i>Location</i>

has manager (manager of)

Specialisation of: • <i>contains</i> (inseparable)	The <i>has manager</i> relation indicates the person that is the manager of an <i>Organisation unit</i> . It also indicates that the person has some overall responsibility for all tasks performed by their employees.
Allowed source concepts: • <i>Organisation unit</i>	Allowed target concepts: • <i>Performer</i>

fulfils skill/knowledge (skill/knowledge fulfilled by)

Specialisation of: • <i>fulfils</i>	This relation is used to denote some skill proficiency or knowledge about a domain for elements of type <i>Instance</i> (e.g. <i>Performer</i>). It also indicates the level of aptitude they have.
Allowed source concepts: • <i>Participant instance</i>	Allowed target concepts: • <i>Skill</i> • <i>Knowledge</i>

Properties

Level of aptitude	In addition the relation should also convey to what level a <i>Skill</i> or <i>Knowledge</i> is covered by the <i>Instance</i> . The recommended types in ascending order of aptitude are “Novice”, “Advanced”, “Competent”, “Proficient” and “Expert”. “None” should generally be denoted by not using this relation. In terms of numbers it can be represented from 0 (None) to 5 (Expert).
-------------------	---

has skill/knowledge (skill/knowledge of)

Specialisation of: • <i>has capability</i>	This relation is used to denote some skill proficiency or knowledge about a domain for elements of type <i>Template</i> (e.g. <i>Role</i>). It also indicates the level of aptitude they have.
Allowed source concepts: • <i>Participant template</i>	Allowed target concepts: • <i>Skill</i> • <i>Knowledge</i>

Properties

Level of aptitude	In addition the relation should also convey to what level a <i>Skill</i> or <i>Knowledge</i> is covered by the <i>Instance</i> . The recommended types in ascending order of aptitude are “Novice”, “Advanced”, “Competent”, “Proficient” and “Expert”. “None” should generally be denoted by not using this relation. In terms of numbers it can be represented from 0 (None) to 5 (Expert).
-------------------	---

Mobile app	
Specialisation of: <ul style="list-style-type: none"> • <i>Participant instance</i> • <i>Asset</i> 	A <i>Mobile app</i> represents an application that can be run on a mobile device. In this case the <i>Mobile app</i> should be used to depict the abstract understanding of an application (e.g. OpenOffice Writer), not a certain license, deployment/installation or setup of that application (not e.g. this installation of OpenOffice Writer on this computer).
Properties	
Download link	If available a link to where an app can be downloaded should be provided.

Mobile app template	
Specialisation of: <ul style="list-style-type: none"> • <i>Participant template</i> • <i>Asset</i> 	A <i>Mobile app template</i> represents a template for <i>Mobile apps</i> . Examples for such templates are “Communication apps”, “VoIP apps” and “Data visualisation apps”.

Mobile app capability	
Specialisation of: <ul style="list-style-type: none"> • <i>Capability</i> • <i>Asset</i> 	A <i>Mobile app capability</i> represents a capability that can be provided by a <i>Mobile app</i> or <i>Mobile app template</i> . Examples for such capabilities are “Audio communication”, “Scan barcodes” and “Take pictures”.

Point of interaction	
Specialisation of: <ul style="list-style-type: none"> • <i>Capability</i> • <i>Asset</i> 	A <i>Point of interaction</i> represents a simple interaction part of an app or a template between the user and the mobile device. A <i>Point of interaction</i> is considered independent of a specific modality. This means that here the <i>Point of interaction</i> for entering some text either through a keyboard or through a microphone is considered the same.
Subtypes	
Interaction type	The interaction is considered directed. This means that it can happen from device to user, indicated by a “Readable” <i>Point of interaction</i> , from the user to the device, indicated by an “Interactive” <i>Point of interaction</i> , or both, meaning that an element is used to both interact with the user and receive information from them (e.g. a text field where the user can change his address which also shows their currently known address). The “Readable” interactions should only be used when information is provided to the user (e.g. a sensor value, the name of a product etc.), not for simple reference points (e.g. labels, empty text areas etc.).
Awareness type	A <i>Point of interaction</i> can either be presented to the user (e.g. a separate button) or be generally hidden to him (e.g. as part of another <i>Point of interaction</i>) and the user knows about them from an outside source (e.g. manual) or through expectations. Therefore they are categorised in “Presentable” and “Non-presentable” <i>Points of interaction</i> . When considering for example visual modality, then “Presentable” <i>Points of interaction</i> occupy screen space, while “Non-presentable” do not.
Properties	
Data type	The data type should be used to describe what type of data is used for the interaction between user and device. It refers to the type of content rather than the channel of acquiring the content. For a list of recommended values for this property see section 2.5.1.3.

Interaction component	
Specialisation of: <ul style="list-style-type: none"> • Capability • Asset 	An <i>Interaction component</i> represents a complex interaction part of an app or a template. It can be simply understood as an aggregation of other interaction parts. Therefore it can be further described through decomposition into other <i>Interaction components</i> or <i>Points of interaction</i> . An <i>Interaction component</i> is considered independent of a specific modality. <i>Interaction components</i> can also be used to represent the patterns from D5.2.2, especially when making use of the global identifier (see section 2.3.2). Additionally, more special versions of those patterns can be described through <i>Interaction components</i> by using specialisation.
Subtypes	
Awareness type	The boundary of an <i>Interaction component</i> can either be presented to the user or be generally hidden to him, meaning that a user can be made aware of the aggregation of its contents or just know about the contents, which again depend on the awareness type of the contained elements. Therefore <i>Interaction components</i> are categorised in “Presentable” and “Non-presentable”.
Content multiplication type	In some cases the content structure of a <i>Interaction component</i> should repeat itself based on runtime data and cannot be fixed during modelling, for example in the case of lists that should hold sensors of different machines or tables that have no fixed amount of rows. Therefore <i>Interaction components</i> are considered either “Repeatable” or “Non-repeatable”.
Properties	
Intended for device	This property should be used to specify for what devices an <i>Interaction component</i> is intended for.

Information instance	
Specialisation of: <ul style="list-style-type: none"> • Participant instance • Asset 	An <i>Information instance</i> represents a certain information, that can be general (e.g. “Order”) or very specific (e.g. “Contract 53132”). Using the same <i>Information instance element</i> in several <i>Actions</i> means that during executions of those <i>Actions</i> the same piece of information should be used. If it is not necessary to describe such behaviours then <i>Information templates</i> should be used instead.

Information template	
Specialisation of: <ul style="list-style-type: none"> • Participant template • Asset 	An <i>Information template</i> represents a template for some information. Information is considered in a general sense, so an example for a template would be “Orders”, “Standard contract” or “Machine manual”, no matter if it is available as a tangible document, as a digital text file or through RDF.
Subtypes	
Information type	The type of information can be one of: “First hand information”, “Refined information” or “Aggregated information”. “First hand information” depicts information that is raw and mostly unprocessed, while “Refined information” is information that has been processed and enriched. “Aggregated information” indicates that it is information assembled from different sources and it is assumed to also be “Refined” (i.e. process of aggregation is considered a refinement).

Access modifier type	Categorisation according to a general type for access restriction. The three possible types are: “Public”, “Restricted” and “Private”. While “Public” information is generally available to everyone, “Restricted” information should be restricted to the user and a close circle around them, while “Private” information should only be available to the people using it. It is not a replacement for the proper description of access control for the information.
----------------------	--

Entity

Specialisation of: <ul style="list-style-type: none"> • <i>Capability</i> • <i>Asset</i> 	An <i>Entity</i> represents a capability of information and should be understood similarly to entities from an ER-Model. It can be decomposed into <i>Attributes</i> and “separable” and “inseparable” should be used to indicate which attributes have to be part of an entity and which are optional. However, the set of “inseparable” <i>Attributes</i> is not necessarily the identifier or primary key of the <i>Entity</i> , but it should be part of it.
---	--

Relation

Specialisation of: <ul style="list-style-type: none"> • <i>Capability</i> • <i>Asset</i> 	A <i>Relation</i> represents a capability of information and should be understood similarly to relations from an ER-Model.
---	--

Attribute

Specialisation of: <ul style="list-style-type: none"> • <i>Capability</i> • <i>Asset</i> 	An <i>Attribute</i> represents a capability of information and should be understood similarly to attributes from an ER-Model.
---	---

Properties

Data type	The type of data an attribute accepts. Unlike the data type of a <i>Point of interaction</i> , this type should be closer to the programming and implementation domain.
-----------	---

Information access

Specialisation of: <ul style="list-style-type: none"> • <i>Access means</i> • <i>Asset</i> 	An <i>Information access</i> describes how some information can be accessed.
---	--

Subtypes

Medium type	The medium type indicates how the information is available. Examples are “Paper-based” or “Paperless” as well as “Linked Data”.
Data source type	This type indicates the source for accessing the information. It can be “Server”, “Cache”, “Local” or “Peripheral”.

Properties

Performed operations	Information can be accessed in order to read it or to alter it. This property allows denoting what operations can be performed through this <i>Information access</i> . It is recommended to use one or several of: “Create”, “Read”, “Update” and “Delete”.
----------------------	--

Location control	
Specialisation of: <ul style="list-style-type: none"> • <i>Split-Implication set</i> 	A <i>Location control</i> is used to describe possible options for accessing information from different <i>Locations</i> . Using them it is possible to provide alternative sources for retrieving some information, or to describe federated queries.
Subtypes	
Inclusion type	Based on the available inclusion types, the subtypes can be “all of” (AND), or “one of” (XOR). It should be only one of those types.

relates (related by)	
Specialisation of: <ul style="list-style-type: none"> • <i>contains</i> (inseparable) 	This relation is used to indicate which <i>Entities</i> are parts of a <i>Relation</i> .
Allowed source concepts: <ul style="list-style-type: none"> • <i>Relation</i> 	Allowed target concepts: <ul style="list-style-type: none"> • <i>Entity</i>
Properties	
Role	Through this property a role for the linked <i>Entity</i> can be provided, giving some context for it in the <i>Relation</i> .
Cardinality	The cardinality for the linked <i>Entity</i> in this relation. Typical values are “1”, “0...1”, “1...*” and “0...*”.

accessed through (access for)	
Specialisation of: <ul style="list-style-type: none"> • <i>has capability</i> 	The <i>accessed through</i> relation should be used to indicate how information can be accessed. It means that all of those access possibilities are available for the information, however usually only one of those is necessary for a certain purpose.
Allowed source concepts: <ul style="list-style-type: none"> • <i>Information template</i> 	Allowed target concepts: <ul style="list-style-type: none"> • <i>Information access</i>

execute on (endpoint for)	
Specialisation of: <ul style="list-style-type: none"> • <i>implication</i> 	This relation links an <i>Information access</i> to the <i>Locations</i> where it can be performed, in which case the target can be considered an endpoint. It is the implication that accessing information requires accessing the location. More detailed descriptions of federated queries or alternative endpoints can be described by using <i>Location control</i> .
Allowed source concepts: <ul style="list-style-type: none"> • <i>Information access</i> • <i>Location control</i> 	Allowed target concepts: <ul style="list-style-type: none"> • <i>Location control</i> • <i>Location</i>
Properties	
Query	This is an optional property that can be used to detail how the information is accessed. It can be for example normal text for a human (for “Paper-based” access) or a SPARQL query (for “Linked Data” access).

Permission rule	
Specialisation of: <ul style="list-style-type: none"> • <i>Participant template</i> • <i>Asset</i> 	The <i>Permission rule</i> represents a single rule which allows performing a subject some action on a resource. Because its decomposition is of the AND-inclusion type, it means that the relations of the same type (i.e. <i>for subject</i> ,

	for action type and for resource) should build an intersection for the <i>Permission rule</i> . For example if one <i>Permission rule</i> links to two subjects, then only participants that belong to both of those subjects have permission to perform an action on the intersection of the resource types.
--	---

for subject (permitted subject of)

Specialisation of: <ul style="list-style-type: none"> contains (inseparable) 	Denotes what subjects should gain permission by the rule. If more than one subject is linked to the same rule, then only participants of the intersection created by those links should have access. A description can be used to further depict restrictions on the subject.
Allowed source concepts: <ul style="list-style-type: none"> Permission rule 	Allowed target concepts: <ul style="list-style-type: none"> Participant template Participant instance

for action type (permitted action type of)

Specialisation of: <ul style="list-style-type: none"> contains (inseparable) 	Denotes what types of actions that can be performed because of the rule. If more than one type of action is linked to the same rule, then only actions with a type created by the intersection of those links should be possible. A description can be used to further depict restrictions on the action.
Allowed source concepts: <ul style="list-style-type: none"> Permission rule 	Allowed target concepts: <ul style="list-style-type: none"> Action type

for resource (permitted resource of)

Specialisation of: <ul style="list-style-type: none"> contains (inseparable) 	Denotes what resources or types of resources can be acted upon because of the rule. If more than one is linked to the same rule, then only resources with a type created by the intersection of those links should be possible. A description can be used to further depict restrictions on the resource.
Allowed source concepts: <ul style="list-style-type: none"> Permission rule 	Allowed target concepts: <ul style="list-style-type: none"> Asset

basis for (based on)

Specialisation of: <ul style="list-style-type: none"> implication 	The implication of needing a <i>Permission rule</i> based on a given requirement from a <i>Procedural element</i> , denoted through <i>requires</i> relations.
Allowed source concepts: <ul style="list-style-type: none"> requires 	Allowed target concepts: <ul style="list-style-type: none"> Permission rule

implemented by (implements)

Specialisation of: <ul style="list-style-type: none"> implication 	The implication that a <i>Permission rule</i> is covered to some degree by an <i>Access means</i> .
Allowed source concepts: <ul style="list-style-type: none"> Permission rule 	Allowed target concepts: <ul style="list-style-type: none"> Access means

Table 11: Scope-specific concepts and relation concepts of the Participant Aspect

2.5.1.3 Recommended data types for Points of interactions

As previously described, a data type should be specified for a *Point of interaction*. This section covers a hierarchy of data types that are recommended. They are structured in a hierarchy, in order to prevent enforcing a specific granularity and to give some freedom to the modeller. Still the modeller should choose the most specific type that they deem fit. The hierarchy is:

- Data
 - Value
 - Text
 - Location
 - Email
 - Date and time
 - Phone
 - URI
 - Number
 - Integer
 - Decimal
 - Percentage
 - Currency
 - Boolean
 - Object
 - File
 - Image
 - Video
 - Sound
 - Document
- Event

2.5.2 Specialised Concepts in Behavioural Aspect

2.5.2.1 Concepts mostly focused in Procedural Aspect

Value exchange flow

Specialisation of:	A <i>Value exchange flow</i> represents a process, focusing on the exchange of <i>Values</i> .
• <i>Process</i>	

Value exchange

Specialisation of:	A <i>Value exchange</i> represents an <i>Action</i> which focuses on the exchange of <i>Values</i> between two partners. While the exchange of a single <i>Value</i> is always considered between two partners, a single <i>Value exchange</i> can describe more than one such exchange, but never less. It should be described from the point of view of the initiator and only exchanges in which the initiator participates should be part of it.
• <i>Action</i>	The <i>Values</i> that are exchanged are indicated by the influences relation, and the direction of exchange for that <i>Value</i> is indicated by the sign of the quantity according to the initiator. It is of <i>Action type</i> “Value exchange”.

Start stimulus

Specialisation of:	A <i>Start stimulus</i> indicates the point of origin which leads to a series of <i>Value exchanges</i> .
• <i>Initiation event</i>	
Properties	
Occurrences	The estimated amount of occurrences for the stimulus in a certain time frame. Usually the time frame of one year should be used.

End stimulus

Specialisation of:	An <i>End stimulus</i> indicates the end of a series of <i>Value exchanges</i> .
• <i>Termination event</i>	

Exchange control

Specialisation of: <ul style="list-style-type: none">• <i>Control</i>	An <i>Exchange control</i> is used to control the flow of <i>Value exchanges</i> .
Subtypes	
Inclusion type	Based on the available inclusion types, the subtypes can be one of “all” (AND) or “one” (XOR), as well as one of “Split” or “Merge”.

with partner (partner of)

Specialisation of:	This relation indicates the partners participating in a <i>Value exchange</i> . <i>Value exchanges</i> should be described on such a level that only two partners are participating and one of them should be the initiator.	
• <i>requires</i>		
Allowed source concepts:		Allowed target concepts:
• <i>Value exchange</i>		• <i>Liabe entity</i>

has initiator (initiator of)

Specialisation of:	This relation indicates the initiator for a <i>Value exchange</i> , which also participates in the exchange itself. It also states from which point of view the <i>influences</i> relations are being described for the <i>Value exchange</i> .	
• <i>with partner</i> • <i>has responsible</i>		
Allowed source concepts:		Allowed target concepts:
• <i>Value exchange</i>		• <i>Liabe entity</i>

Business process

Specialisation of:	A <i>Business process</i> represents a more detailed description of an <i>Action</i> that describes procedurally how a certain goal should be achieved. It should concentrate on what the human has to do and avoid technical details about the execution (e.g. visualise data).
• <i>Process</i>	

Activity

Specialisation of:	An <i>Activity</i> describes a general <i>Action</i> that is performed by somebody to achieve a certain goal. The waste that is produced by an <i>Activity</i> can be described by positively influencing <i>Anti-Values</i> . For example an <i>Activity</i> can that has a 3% waste through defects would link to a “Defects” <i>Anti-value</i> using the <i>influences</i> relation with a quantity of 0.03.
• <i>Action</i>	

Properties	
Instructions	This property should describe or link to some instructions on how to carry out the <i>Activity</i> .
Auditing requirements	This property describes the requirements and what is necessary to prove the proper execution of the <i>Activity</i> . In addition an <i>Evaluation process</i> that <i>evaluates</i> this <i>Activity</i> can be used to describe additional details.

Start	
Specialisation of: • <i>Initiation event</i>	A special type of the <i>Initiation event</i> that is used in processes which have general actions.
Subtypes	
Intention type	The <i>Start</i> of a process can be intended (something that is expected and generally welcomed, like receiving an order) or it can be as a reaction to something (something that can happen and has to be taken care of, like cancelling an order). Therefore the possible types are “Intended” and “Incident”.

End	
Specialisation of: • <i>Termination event</i>	A special type of the <i>Termination event</i> that is used in processes which have general actions.
Subtypes	
Intention type	The <i>End</i> in a process can be as desired (main) or it can be an alternative end. When the execution of a process finishes only in alternative ends, then the results can be considered less satisfactory. The two corresponding types are “Main” or “Alternative”.

Decision	
Specialisation of: • <i>Action</i> • <i>Split-Control</i>	A <i>Decision</i> represents the action of deciding something in a process and by that influencing how the following sequence should continue.
Subtypes	
Inclusion type	Based on the available inclusion types, the subtypes can be “for some of” (OR) or “for one of” (XOR). It should be only one of those types.
Properties	
Question	The question that emphasises the decision that should be made.

Parallelism	
Specialisation of: • <i>Split-Control</i> (AND-inclusion type)	A <i>Parallelism</i> indicates that the outgoing paths in a process can be executed simultaneously instead of requiring a specific sequence.

Synchronisation	
Specialisation of: • <i>Merge-Control</i>	A <i>Synchronisation</i> indicates the merging of several paths back into one. Its inclusion type indicates when the elements following the <i>Synchronisation</i> should be started (i.e. if one, if some or if all of the incoming paths are

	finished). The <i>Synchronisation</i> itself does not terminate any remaining <i>Actions</i> of the incoming paths ⁴ .
Subtypes	
Inclusion type	Based on the three available inclusion types, the subtypes can be “of all” (AND), “of some” (OR) or “of one” (XOR). It should be only one of those types.

Requirements process

Specialisation of: <ul style="list-style-type: none"> • <i>Process</i> 	A <i>Requirements process</i> represents a more detailed description of an <i>Action</i> that focuses on depicting the things that are required by it and how they are used. It can therefore look closer like a workflow than a <i>Business process</i> , and is often created by splitting and merging <i>Actions</i> from a <i>Business process</i> .
--	--

strictly requires (strictly required by)

Specialisation of: <ul style="list-style-type: none"> • <i>requires</i> 	This specialisation of <i>requires</i> indicates that the target is necessary to perform something. It is recommended to use this level of detail in the <i>Requirements process</i> .
Allowed source concepts: <ul style="list-style-type: none"> • Elements contained by a <i>Requirements process</i> 	Allowed target concepts: <ul style="list-style-type: none"> • <i>Participant template</i>

supported by (supports)

Specialisation of: <ul style="list-style-type: none"> • <i>requires</i> 	This specialisation of <i>requires</i> indicates that the target supports performing something, meaning it is good to have it, but even if it is missing you can perform the action. It is recommended to use this level of detail in the <i>Requirements process</i> .
Allowed source concepts: <ul style="list-style-type: none"> • Elements contained by a <i>Requirements process</i> 	Allowed target concepts: <ul style="list-style-type: none"> • <i>Participant template</i>

Interaction flow

Specialisation of: <ul style="list-style-type: none"> • <i>Process</i> 	An <i>Interaction flow</i> represents a more detailed description of an <i>Action</i> that centres on depicting the interactions between a device and its user as well as what functions a device is executing. Mostly mobile devices and the interaction with their mobile applications are in the focus.
--	--

Interaction

Specialisation of: <ul style="list-style-type: none"> • <i>Action</i> 	An <i>Interaction</i> represents an <i>Action</i> that is performed by a single user on a device. They should link to a <i>Point of interaction</i> or an <i>Interaction component</i> , to indicate through which parts the interaction is happening. If an <i>Interaction</i> is followed by a <i>Function execution</i> , then it means that interacting with the linked <i>Point of interaction</i> or <i>Interaction component</i> should trigger something.
---	---

⁴ This is important for the “OR” and “XOR” type, since then some *Actions* of the incoming paths might still be executing while the *Synchronisation* is passed.

Function execution

Specialisation of: • <i>Action</i>	A <i>Function execution</i> represents an <i>Action</i> performed by the device. It makes therefore little sense to link a human performer to it. It is however possible to link a responsible person.
Subtypes	
Source type	The source type indicates where the function is executed. Examples are “in App”, “in Device”, “in Environment” and “Remote”. It is also possible to have functions that are executed in several sources.

Path split

Specialisation of: • <i>Control</i>	A <i>Path split</i> indicates a splitting of the possible paths into one, several or all outgoing paths. The necessary preparations and finding a decision should happen beforehand, in the case that not all outgoing paths should be chosen (i.e. a <i>Path split</i> is not the action of deciding).
Subtypes	
Inclusion type	Based on the three available inclusion types, the subtypes can be “to all” (AND), “to some” (OR) or “to one” (XOR). It should be only one of those types.

Orchestration

Specialisation of: • <i>Process</i>	An <i>Orchestration</i> represents a more detailed description of an <i>Action</i> that centres on describing a technical workflow. It focuses on specifying a sequence in which mobile apps should be executed.
---	--

App execution

Specialisation of: • <i>Action</i>	An <i>App execution</i> represents an <i>Action</i> that depicts the execution of a mobile app or an <i>Orchestration</i> on a mobile device. During the execution of the app there is usually interaction between the user and the device. This interaction can be further described for example by an <i>Interaction flow</i> .
--	---

Notification received

Specialisation of: • <i>Event</i>	A <i>Notification received</i> represents the event that is triggered when receiving a notification on a device.
---	--

Entry

Specialisation of: • <i>Initiation event</i>	An <i>Entry</i> indicates a point where an <i>Orchestration</i> can be started from.
Subtypes	
Start type	The impulse for starting an <i>Orchestration</i> can be either the receiving of a notification or it can be from a human deciding to start it. So it can be “Delegated” (when receiving a notification), “Autonomous” (by a humans will) or both.

Halt

Specialisation of:	A <i>Halt</i> represents the end of a thread of path in an <i>Orchestration</i> .
• <i>Termination event</i>	

executes (executed in)

Specialisation of:	A specialisation of <i>requires</i> that indicates what app should be executed by an <i>App execution</i> .	
• <i>requires</i>		
Allowed source concepts:		Allowed target concepts:
• <i>App execution</i>		• <i>Mobile app</i>
		• <i>Mobile app template</i>
		• <i>Mobile app feature</i>

notifies (notified by)

Specialisation of:	A specialisation of <i>impacts</i> that indicates which other <i>Orchestrations</i> should receive a notification in order to continue.	
• <i>impacts</i> (positive)		
Allowed source concepts:		Allowed target concepts:
• <i>App execution</i>		• <i>Notification received</i>
		• <i>Entry</i>

has message (message of)

Specialisation of:	This relation indicates what information is provided in a notification. Note that this does not necessarily mean that all of the data is sent. It is also possible that only a key is sent which can be used to retrieve the information. It is the implication of sending the attached message when a notification happens.	
• <i>implication</i>		
Allowed source concepts:		Allowed target concepts:
• <i>notifies</i>		• <i>Information template</i>
		• <i>Information instance</i>

designed for (has orchestration)

Specialisation of:	This relation indicates for which <i>Liabile entities</i> an <i>Orchestration</i> has been designed. In other words, it states who should use a specific orchestration.	
• <i>has performer</i>		
Allowed source concepts:		Allowed target concepts:
• <i>Orchestration</i>		• <i>Liabile entity template</i>

Evaluation process

Specialisation of:	An <i>Evaluation process</i> represents a more detailed description of an <i>Action</i> that centres on describing the evaluation of some <i>Action</i> or <i>Motivating value</i> . It focuses on depicting how evaluation is handled and who participates in it.	
• <i>Process</i>		

Evaluation action

Specialisation of:	An <i>Evaluation action</i> represents an <i>Action</i> that is carried out for the sake of assessing a value or action.	
• <i>Action</i>		

Subtypes	
Action type	Different types of actions can be considered here, focusing on how the data is handled. Examples are “Data collection”, “Data transmission” and “Data processing”. However, it is recommended to use the <i>Action type</i> from the Aspect-specific concepts to denote those.

evaluates (evaluated by)	
Specialisation of: ---	This relation indicates what is evaluated by an <i>Evaluation action</i> .
Allowed source concepts: • <i>Evaluation action</i>	Allowed target concepts: • <i>Action</i> • <i>Motivating value</i>

performed on (in performance of)	
Specialisation of: ---	This relation indicates on what data or information an <i>Evaluation action</i> is performed on. Using the previously mentioned action types (see <i>Evaluation action</i>) it indicates “what data is collected”, “what data is transmitted” or “what data is processed”.
Allowed source concepts: • <i>Evaluation action</i>	Allowed target concepts: • <i>Variable</i> • <i>KPI</i> • <i>Information template</i> • <i>Information instance</i>

Table 12: Scope-specific concepts and relation concepts of the Procedural Aspect

2.5.2.2 Concepts mostly focused in Collaborative Aspect

Business model	
Specialisation of: • <i>Collaboration</i>	A <i>Business model</i> represents a scenario in which business partners perform transactions or <i>Value exchanges</i> , which depend on one another.

Value interface	
Specialisation of: • <i>Participant involvement</i>	A <i>Value interface</i> represents a part of a liable entities boundary, through which values are exchanged. It can be considered a part of a <i>Liable entity</i> .

Dependency control	
Specialisation of: • <i>Control</i>	A <i>Dependency control</i> allows splitting and merging dependency paths. Together with the fractions of <i>depends on</i> relations it allows to control the occurrences based on its inclusion type. Details can be found in Table 20.
Subtypes	
Inclusion type	Based on the available inclusion types, the subtypes can be one of “all” (AND) or “one” (XOR), as well as one of “Split” or “Merge”. With an “AND” inclusion a fraction of one side (i.e. either the outgoing or the incoming relations) is related to the fractions of the other side. With an “XOR” inclusion a fraction of one side is related to the sum of fractions of that side.

interface part of (has value interface)

Specialisation of: • <i>involved participant</i>	A specialisation of <i>involved participant</i> , which indicates whose <i>Value interface</i> it is. The <i>Value interface</i> can be also considered to be part of the target <i>Liabe entity</i> .	
Allowed source concepts: • <i>Value interface</i>	Allowed target concepts: • <i>Liabe entity</i>	

exchanges value with (receives value from)

Specialisation of: • <i>implication</i>	This relation indicates between which two <i>Value interfaces</i> a <i>Value</i> is exchanged. It is the implication that using the <i>Value interface</i> of the initiator (the source) also implies using the <i>Value interface</i> of the linked partner.	
Allowed source concepts: • <i>Value interface</i>	Allowed target concepts: • <i>Value interface</i>	
Properties		
Outgoing valuation	A number representing the worth of the exchanged value from the point of view of the partner giving up the value. It should be the same as the <i>Incoming valuation</i> , if the value “Money” is exchanged.	
Incoming valuation	A number representing the worth of the exchanged value from the point of view of the partner receiving the value. It should be the same as the <i>Outgoing valuation</i> , if the value “Money” is exchanged.	

exchanged value (exchanged in)

Specialisation of: • <i>implication</i>	This relation indicates what <i>Value</i> is exchanged. It is the implication that exchanging a value also requires that value.	
Allowed source concepts: • <i>exchanges value with</i>	Allowed target concepts: • <i>Value</i>	
Properties		
Quantity	The quantity of the <i>Value</i> that is exchanged.	

depends on (dependency of)

Specialisation of: • <i>implication</i>	This relation indicates dependency between the elements in a <i>Business model</i> . It is the implication that using the source also requires using the target.	
Allowed source concepts: • <i>Start stimulus</i> • <i>Value interface</i> • <i>Dependency control</i>	Allowed target concepts: • <i>End stimulus</i> • <i>Value interface</i> • <i>Dependency control</i>	
Properties		
Fraction	The fraction indicates a rate for change in occurrences when using <i>Dependency control</i> . It should be seen in relation to the other <i>depends on</i> relations of the linked <i>Dependency control</i> . Details can be found in Table 20.	

Participant collaboration

Specialisation of: • <i>Collaboration</i>	A <i>Participant collaboration</i> describes the <i>Participants</i> and their collaboration for an <i>Action</i> . One <i>Participant collaboration</i> should focus on a limited set of participant types (e.g. <i>Roles</i> , <i>Liabe entities</i> , <i>Information templates</i> etc.). The	
---	--	--

	collaboration is described through the <i>switches to</i> relations.
--	--

switches to (switched from)	
Specialisation of: <ul style="list-style-type: none"> • <i>implication</i> 	This relation indicates what other participants are involved during or after the use of a participant in a <i>Collaboration</i> . The involvement of the source implies the involvement of the target in a process and possibly also an interface between them.
Allowed source concepts: <ul style="list-style-type: none"> • <i>Start</i> • <i>Participant involvement</i> 	Allowed target concepts: <ul style="list-style-type: none"> • <i>Participant involvement</i> • <i>End</i>

Navigation model	
Specialisation of: <ul style="list-style-type: none"> • <i>Collaboration</i> 	A <i>Navigation model</i> depicts the possible navigations between components in an app and should be based on one or several interaction flows.

triggers (triggered by)	
Specialisation of: <ul style="list-style-type: none"> • <i>implication</i> 	This relation indicates that interacting with a <i>Point of interaction</i> or <i>Interaction component</i> of an app triggers a change of the elements available to the app user, resulting in navigating through the app (e.g. switching a screen, showing a popup etc.). It is the implication that interaction with the source triggers the target.
Allowed source concepts: <ul style="list-style-type: none"> • <i>Start</i> • <i>Participant involvement (Interaction component or Point of interaction)</i> 	Allowed target concepts: <ul style="list-style-type: none"> • <i>Participant involvement (Interaction component)</i> • <i>End</i>
Properties	
Conditions	Additional conditions besides interacting with the source can be specified through this property.
Available navigation patterns	The navigation patterns allowed or employed can be specified here. Some navigation patterns have been described in D5.2.2.

Notification exchange	
Specialisation of: <ul style="list-style-type: none"> • <i>Collaboration</i> 	A <i>Notification exchange</i> depicts the notifications that are exchanged between different <i>Orchestrations</i> .

Table 13: Scope-specific concepts and relation concepts of the Collaborative Aspect

2.6 Assignment of Concepts to Aspect Specialisations

In this section the previously presented concepts will be assigned to groups similar to the *Scope* and *Aspect* specific cells presented in Table 1. Those groups represent a set of elements that are recommended to be used together and can be considered the “model types”. Some of the here used relation concepts create connections between the groups as described in the previous sections (e.g. decomposition between *Business structure* and *Enterprise structure* groups). The groups and additional details can be found in Tables 14 to 34 and each group will be presented the first time it is introduced, while later on previous groups that are reused will be referenced. Examples on how those groups could look in an implementation can be found in section 3.2.

2.6.1 Business Scope

The main focus of this *Scope* is on the high level of business, where several business entities are participating in the creation of a value (e.g. product or service) for a customer with the goal of receiving some compensation in return (e.g. money). It aims to capture the exchange of values between the different participants. The major groups used here are:

Value structure group	
The <i>Value structure group</i> describes the values, their structure and their variability that are of interest in the current application domain. It is designed to cover products and services through values.	
Concept	Comment
Value	The core for describing the value structure.
Value set	To allow describing variability in the structure.
has value (value of)	The core for structuring the values.
mandates value (mandated by value)	To allow limiting the variability in the structure.
configuration of (configured into)	
annihilates (annihilates)	In order to link values to their anti-values.

Table 14: Concepts in Value structure group

Market structure group	
The <i>Market structure group</i> describes a company's view on the market and the segments it targets.	
Concept	Comment
Market segment	The core for describing the market structure.
Characteristic	Used to describe market segments.
justifies (justified by)	Indicates why certain values of a value structure are necessary, based on the market structure.
contains (contained by)	The general concept, to decompose the market structure.
specialisation of (generalisation of)	The general concept, to describe specialisations in the market structure.
has capability (capability of)	The aspect-specific concept, to assign characteristics to segments.

Table 15: Concepts in Market structure group

Business structure group	
The <i>Business structure group</i> describes companies and their distribution over several locations.	
Concept	Comment
Business entity	The core instance for describing business structure.
Business role	The core template for describing business structure.
Business capability	The core capability for describing business structure.
Business entity access	How to access business entities.
has chief (chief of)	
provided value (can be provided by)	The values provided by a capability. Typically a product or service as well as the duration for delivery.
necessary compensation (compensation for)	The values asked in return in a capability.
provided at (can find business capability)	Where the capability can be made use of.
access business through (accesses business)	

has business location (business location of)	
contains (contained by)	The general concept, to decompose the business structure.
specialisation of (generalisation of)	The general concept, to describe specialisations in the business structure.
has capability (capability of)	The aspect-specific concept, to assign capabilities to roles.
fulfils (fulfilled by)	The aspect-specific concept, to describe instantiations.
owned by (owns)	The aspect-specific concept, to state the owners of a business.

Table 16: Concepts in Business structure group

Location structure group	
The <i>Location structure group</i> describes certain locations (physical or digital) in relation to other locations.	
Concept	Comment
Location	The core for describing location structure.
contains (contained by)	The general concept, to decompose the location structure.

Table 17: Concepts in Location structure group

Value exchange flow group	
The <i>Value exchange group</i> describes the exchange of values between two or more businesses. It focuses more on the sequence in which values have to be exchanged than the <i>Business model</i> .	
Concept	Comment
Value exchange flow	Use different flows for different scenarios.
Value exchange	The core element for describing a flow. It can be considered an action of type “Value exchange”.
Start stimulus	
End stimulus	
Exchange control	
with partner (partner of)	Should only be two for the same value exchange and one of them should be the initiator.
has initiator (initiator of)	Should only be one for the one value exchange, who is also a partner of the exchange.
followed by (preceded by)	The aspect-specific concept, to specify the sequence and conditions for the flow.
impacts (affected by)	The aspect-specific concept, to allow describing the impact of a value exchange on other actions.
detailed by (describes)	The aspect-specific concept, to decompose actions into a finer granularity.
has part (part of)	The aspect-specific concept, to state what is contained by a value exchange flow.
influences (influenced by)	The inter-aspect concept, to specify what values are exchanged.

Table 18: Concepts in Value exchange flow group

Business model group	
The <i>Business model group</i> describes the exchange of values between two or more businesses. It focuses more on the general structure of exchanges and the dependency between those than the <i>Value exchange</i> .	
Concept	Comment
Business model	Use different models for different scenarios.

Value interface	The involvement of a participant (market segment, business entity or business role) in exchanges of value. Should be considered a part of the participant.
Start stimulus	
End stimulus	
Dependency control	Because of the occurrence formulas in Table 20 it should be possible to also use Dependency controls with only one incoming and one outgoing <i>depends on</i> relation.
interface part of (has value interface)	
exchanges value with (receives value from)	
exchanged value (exchanged in)	
depends on (dependency of)	
is for (has collaboration)	The inter-aspect concept, to indicate on which actions the collaboration is based on.

Table 19: Concepts in Business model group

In this *Scope* the products or services that are provided to customers are identified not in the *Value structure*, but in the *Business model*. In the *Business model* the products or services sold by a *Business entity* are determined through the outgoing *exchanges value with* relations that also have an incoming monetary value in the same *Value interface*.

The *Business model group* is based on the e3value model⁵. Therefore the described business model deals with several occurrences at once, which are specified in the start stimulus. The fractions on the *depends on* relations and the *Dependency control* can be used to control the flow, multiplication and reduction of those occurrences. The following formulas should be used to calculate the occurrences for different *Dependency control* cases, where n is the amount of paths and x indicates the path in question if there are several:

Case	Formula
Split AND-inclusion type	$outOccurrence_x = incOccurrence * \frac{outFraction_x}{incFraction}$
Split XOR-inclusion type⁶	$outOccurrence_x = incOccurrence * \frac{outFraction_x}{\sum_{i=1}^n (outFraction_i)}$
Merge AND-inclusion type	$outOccurrence = incOccurrence_1 * \frac{outFraction}{incFraction_1}$
Merge XOR-inclusion type	$outOccurrence = \sum_{i=1}^n incOccurrence_i$

Table 20: Formulas for calculating occurrences in Business models

In the case of a dependency merge of AND-inclusion type, all of the incoming occurrences in relation to the fractions should be the same (i.e. using the above formula, if a different incoming relation instead of “1” is used it should still lead to the same result). This is necessary to properly merge a previous dependency split of AND-inclusion type. This is depicted in Figure 8, where both x and y should be the same amount of

⁵ For more information see (Gordijn and Akkermans, 2001) and <http://e3value.few.vu.nl/> (accessed 17.01.2014)

⁶ Because of its nature, it can be used with only one outgoing dependency to change the fraction without changing the occurrences.

occurrences, if the left most and right most relations have the same fraction and independent of what the other fractions in-between are.

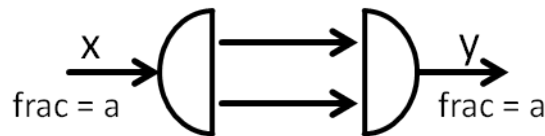


Figure 8: Dependency control of AND type, where occurrences x and y should be the same

2.6.2 Enterprise Scope

The main focus of this *Scope* is on describing what activities are performed by humans in a certain company in order to achieve certain goals. Oftentimes the goal is to process values (e.g. materials) to create something of more value that is then provided to a customer for compensation. However, supporting activities (e.g. financing, procurement etc.) can also be described in this *Scope*. The major groups used here are:

Enterprise structure	
The <i>Enterprise structure group</i> describes the inner structure of a company in terms of units and employees.	
Concept	Comment
Organisation unit	The core instance to describe units in the enterprise structure.
Performer	The core instance to describe employees in the enterprise structure.
Role	The core template to describe employees in the enterprise structure.
Skill	One of the capabilities for employees.
Knowledge	The other capability for employees.
has manager (manager of)	To indicate the manager of an organisation unit.
fulfils skill/knowledge (skill/knowledge fulfilled by)	Used with performers.
has skill/knowledge (skill/knowledge of)	Used with roles.
contains (contained by)	The general concept, to decompose the enterprise structure.
specialisation of (generalisation of)	The general concept, to describe specialisations of roles in the enterprise structure.
fulfils (fulfilled by)	The aspect-specific concept, to describe instantiations.

Table 21: Concepts in Enterprise structure group

Business process group	
The <i>Business process group</i> describes the actions that are performed by humans to achieve a certain goal.	
Concept	Comment
Business process	Preferably should have only one start and one main end.
Activity	The core element for describing a process.
Event	Can be used and reused in other processes to provide points of reference between different processes.
Start	
End	
Decision	Used to split a path. Its action can be considered of type “decide”.
Parallelism	Used to split a path.
Synchronisation	Used to merge paths.
Action type	The aspect-specific concept, to allow categorising activities into different types.

specialisation of (generalisation of)	The general concept, to specialise action types.
instance of (has instance)	The general concept, to indicate the action type of an activity.
followed by (preceded by)	The aspect-specific concept, to specify the sequence and conditions for the process.
impacts (affected by)	The aspect-specific concept, to allow describing the impact of an activity on other actions.
detailed by (describes)	The aspect-specific concept, to decompose actions into a finer granularity.
has part (part of)	The aspect-specific concept, to state what is contained by a business process.
influences (influenced by)	The inter-aspect concept, to specify what values are consumed or produced.
requires (required by)	The inter-aspect concept, to indicate the use or adaptation of participants.
has responsible (responsible for)	The inter-aspect concept, to state the responsible for an activity.
has performer (performs)	The inter-aspect concept, to state the performer for an activity.

Table 22: Concepts in Business process group

Participant collaboration group	
The <i>Participant collaboration group</i> describes the collaboration of several participants of the same or similar type (e.g. liable entities).	
Concept	Comment
Participant collaboration	Use different collaborations for different scenarios.
Participant involvement	Any type of participant, but should be limited to similar types in one collaboration (e.g. roles and performers or liable entities or information templates).
Start	
End	
switches to (switched from)	
contains (contained by)	The general concept, to decompose a collaboration into its contents.
is for (has collaboration)	The inter-aspect concept, to indicate on which actions the collaboration is based on.
involved participant (involved in)	The inter-aspect concept, to indicate what specific template of participants is involved. Target can be any type of participant, but should be limited to similar types in one collaboration (e.g. roles and performers or liable entities or information templates).

Table 23: Concepts in Participant collaboration group

The following groups should also be available in this *Scope*:

- **Value structure group** (Table 14) – Should be used to describe what a business process achieves and what it consumes/wastes.

2.6.3 Requirements Scope

The main focus of this *Scope* is on describing what resources are necessary and how they should be used to achieve a certain goal. This description follows a similar process based structure to the one used in the *Enterprise Scope*, to allow facilitate the alignment between the two. The major groups used here are:

Permission pool	
The <i>Permission pool group</i> describes structures on which access permission is based. Requests fitting into one of the described structures should be granted, all others rejected.	
Concept	Comment
Permission rule	The core of the permission pool, on which permissions for a request are granted. If a request does not fit to the structure of any permission rule then access should be prohibited.
for subject (permitted subject of)	If several subjects are specified for one rule then the intersection of those is denoted.
for action type (permitted action type of)	If several actions are specified for one rule then the intersection of those is denoted.
for resource (permitted resource of)	If several resources are specified for one rule then the intersection of those is denoted.
basis for (based on)	To keep track which requirements are covered.
implemented by (implements)	To keep track of which permission rules are covered through access means.

Table 24: Concepts in Permission pool group

Mobile support structure	
The <i>Mobile support structure group</i> describes mobile applications and their capabilities through components and interaction functionalities. In this <i>Scope</i> mostly the available apps are of interest.	
Concept	Comment
Mobile app	The core instance to describe mobile support structure. The point where to switch from template to instance in the description has to be decided by the user. A recommendation is to see finished applications as instances.
Mobile app template	The core template to describe mobile support structure.
Mobile app capability	A general capability for describing mobile apps.
Point of interaction	A simple interaction part of a mobile app.
Interaction component	A composite of interactions that can be further described by <i>Points of interaction</i> .
contains (contained by)	The general concept, to decompose the mobile support structure (both apps and capabilities).
specialisation of (generalisation of)	The general concept, to describe specialisations in the mobile support structure.
has capability (capability of)	The aspect-specific concept, to assign capabilities to app templates.
fulfils (fulfilled by)	The aspect-specific concept, to describe instantiations.
owned by (owns)	The aspect-specific concept, to state the owners of an app.

Table 25: Concepts in Mobile support structure group

Information space	
The <i>Information space group</i> describes what information is available from where and details information by describing its general data.	
Concept	Comment
Information instance	The core instance to describe an information space. In this case an instance is not necessarily the data for information. It is recommended to use instances to indicate that the same data object

	should be used by different processes/actions during the execution for one case.
Information template	The core template to describe an information space. Reusing the information template does not necessarily mean reusing the same data object, only using data complying with the information template.
Entity	A capability to describe the data structure of information.
Relation	A capability to describe the data structure of information.
Attribute	A capability to describe the data structure of information.
Information access	
Location control	
relates (related by)	
accessed through (access for)	
execute on (endpoint for)	
contains (contained by)	The general concept, to decompose information.
specialisation of (generalisation of)	The general concept, to describe specialisations of information.
has capability (capability of)	The aspect-specific concept, to assign capabilities to information templates.
fulfils (fulfilled by)	The aspect-specific concept, to describe instantiations.
owned by (owns)	The aspect-specific concept, to state the owners of information and data.

Table 26: Concepts in Information space group

Requirements process group	
The <i>Requirements process group</i> describes the actions that are performed on resources to achieve a certain goal.	
Concept	Comment
Requirements process	Several can be used to describe the same action and each one could focus on different types of resources. Should preferably have only one start and one main end.
Activity	The core element for describing a process. Focusing in this group more on the used participants.
Event	Can be used and reused in other processes to provide points of reference between different processes.
Start	
End	
Decision	Used to split a path. Its action can be considered of type “decide”.
Parallelism	Used to split a path.
Synchronisation	Used to merge paths.
Action type	The aspect-specific concept, to allow categorising activities into different types.
strictly requires (strictly required by)	Indicates necessary participants.
supported by (supports)	Indicates optional participants.
specialisation of (generalisation of)	The general concept, to specialise action types.
instance of (has instance)	The general concept, to indicate the action type of an activity.

followed by (preceded by)	The aspect-specific concept, to specify the sequence and conditions for the process.
impacts (affected by)	The aspect-specific concept, to allow describing the impact of an activity on other actions.
detailed by (describes)	The aspect-specific concept, to decompose actions into a finer granularity.
has part (part of)	The aspect-specific concept, to state what is contained by a requirements process.
influences (influenced by)	The inter-aspect concept, to specify what values are consumed or produced.
has responsible (responsible for)	The inter-aspect concept, to state the responsible for an activity.
has performer (performs)	The inter-aspect concept, to state the performer for an activity.

Table 27: Concepts in Requirements process group

The following groups should also be available in this *Scope*:

- **Value structure group** (Table 14) – Should be used to describe what a process achieves and what it consumes/wastes.
- **Enterprise structure group** (Table 21) – To indicate who is performing and who is responsible for an action.
- **Participant collaboration group** (Table 23) – Similar to enterprise scope.

2.6.4 App development Scope

The main focus of this *Scope* is on the capturing and reviewing of requirements for the development of mobile apps. Those requirements are captured through the interactions with the app, both in a structural and a behavioural manner, which can also be used to communicate the intended way of involving the app in a business process. The major groups used here are:

Interaction flow group	
The <i>Interaction flow group</i> describes the interactions that are performed between a device and its user to achieve a certain goal.	
Concept	Comment
Interaction flow	Should preferably have only one start and one main end.
Interaction	A core element for describing an interaction flow. It can be considered an action of type “Interaction”.
Function execution	A core element for describing an interaction flow. It can be considered an action of type “Function execution”.
Event	Can be used and reused in other processes to provide points of reference between different processes.
Start	
End	
Path split	Used to split a path.
Synchronisation	Used to merge paths.
Action type	The aspect-specific concept, to allow categorising actions into additional types.
specialisation of (generalisation of)	The general concept, to specialise action types.
instance of (has instance)	The general concept, to indicate the action type of an interaction or a function execution.
followed by (preceded by)	The aspect-specific concept, to specify the sequence and conditions for the flow.

impacts (affected by)	The aspect-specific concept, to allow describing the impact of an action on other actions.
detailed by (describes)	The aspect-specific concept, to decompose actions into a finer granularity.
has part (part of)	The aspect-specific concept, to state what is contained by an interaction flow.
influences (influenced by)	The inter-aspect concept, to specify what values are consumed or produced.
requires (required by)	The inter-aspect concept, to indicate the use or adaptation of participants. In this case it should focus on the <i>Points of interaction</i> and <i>Interaction components</i> that are used.

Table 28: Concepts in the Interaction flow group

Navigation model group	
The <i>Navigation model group</i> describes what navigation is possible through a mobile app.	
Concept	Comment
Navigation model	One navigation model can capture several interaction flows, but it should not depict several mobile apps.
Participant involvement	Should represent either an <i>Interaction component</i> or a <i>Point of interaction</i> .
Start	
End	
triggers (triggered by)	
contains (contained by)	The general concept, to decompose a collaboration into its contents.
is for (has collaboration)	The inter-aspect concept, to indicate on which actions the collaboration is based on.
involved participant (involved in)	The inter-aspect concept, to indicate what specific template of participants is involved. Targets here should be either <i>Interaction components</i> or <i>Points of interaction</i> .

Table 29: Concepts in the Navigation model group

The following groups should also be available in this *Scope*:

- **Value structure group** (Table 14) – Should be used to describe what a flow achieves and what it consumes/wastes.
- **Mobile support structure group** (Table 25) – In this *Scope* the mobile support structure should be used to describe the capabilities of an app in terms of *Points of interaction* and *Interaction components*.

Triggers of a mobile app are indicated in the *Interaction flow*. When an interaction is followed by a function execution, then it means that one of the assigned points of interaction or interaction components is the source for triggering something.

The Interaction flow can also be used to differentiate between downloading and streaming. Downloading means that the device first receives all the data, loads it into a part of the application which is then read by the user. Streaming on the other hand means that the downloading by the device and reading by the user is happening in parallel.

There are two possible ways of describing that logging in is necessary:

1. Use the condition of the *followed by* relation in the interaction flow to state that the user has to be logged in. A process handling the case that the user tries to access parts without being logged in could then describe the process of logging in.

2. Describe the action of logging in (just an action or further describe it through a process) and use a positive *impacts* relation (e.g. “Enables” or “Necessary to enable”) to all other actions that require the login.

The second approach is recommended, because it creates a relation for dependency which can be queried.

2.6.5 App execution set-up Scope

The main focus of this *Scope* is on the description of app orchestrations, by specifying the sequences of app executions and the possible paths in an orchestration similar to a process. This description follows a similar process based structure to the one used in other *Scopes*, to allow facilitate the alignment with them. Since it describes an orchestration on a mobile device (“local” orchestration) and mobile devices are usually not switching the user (everybody uses their own mobile device), a collaboration of several such “local” orchestrations might be necessary to properly implement a business or requirements process. The major groups used here are:

Orchestration group	
The <i>Orchestration group</i> describes the execution of apps to achieve a certain goal.	
Concept	Comment
Orchestration	Additional constraints of an execution engine on what is allowed should be considered here. They however depend on the used execution engine.
App execution	A core element for describing an orchestration. It can be considered an action of type “execution”. Can also depict other orchestrations.
Event	Can be used and reused in other processes to provide points of reference between different processes.
Notification received	
Entry	
Halt	
Path split	Used to split a path. Consider splitting one path into several (i.e. not an XOR split) to be similar to creating multiple threads in programming.
Synchronisation	Used to merge paths.
executes (executed in)	Replaces the <i>requires</i> relation to only focus on the app that is executed.
notifies (notified by)	Replaces the <i>impacts</i> relation to only focus on notifications between orchestrations.
has message (message of)	
designed for (has orchestration)	
followed by (preceded by)	The aspect-specific concept, to specify the sequence and conditions for the flow.
detailed by (describes)	The aspect-specific concept, to decompose actions into a finer granularity.
has part (part of)	The aspect-specific concept, to state what is contained by an interaction flow.
influences (influenced by)	The inter-aspect concept, to specify what values are consumed or produced.

Table 30: Concepts in Orchestration group

Notification exchange group	
The <i>Notification exchange group</i> describes what notifications are exchanged between which orchestrations. The involved participants are handled through the procedural elements used here.	
Concept	Comment
Notification exchange	It describes the notifications between several orchestrations, so it is also for several actions.
Orchestration	
App execution	
Notification received	
Entry	
executes (executed in)	
notifies (notified by)	
has message (message of)	
contains (contained by)	The general concept, to decompose a collaboration into its contents.
is for (has collaboration)	The inter-aspect concept, to indicate on which actions the collaboration is based on.

Table 31: Concepts in Notification exchange group

The following groups should also be available in this *Scope*:

- **Value structure group** (Table 14) – Should be used to describe what an orchestration achieves and what it consumes/wastes.
- **Mobile support structure group** (Table 25) – In this *Scope* the mobile support structure is mostly used to state which apps are executed in an orchestration.
- **Information space group** (Table 26) – Can be used here to describe the messages/notifications that are sent in an orchestration.

One of the goals of this *Scope* is to provide input, with some additional adaptations outside of the specification, to workflow or orchestration execution engines. However, some execution engines might not be able to handle all of the possible cases that can be modelled (e.g. splitting of one thread into several, OR-splits, more than one entry in an orchestration etc.). Therefore the abilities of the chosen engine should be considered when creating orchestration models.

2.6.6 Evaluation Scope

The main focus of this *Scope* is on the evaluation performance indicators, measures and activities. In addition to describing the process of how the evaluation is carried out and what resources are used, mathematical descriptions (to a certain degree) can be used to formalise performance indicators and measures that can later be calculated. The major groups used here are:

KPI structure group	
The <i>KPI structure group</i> describes the key performance indicators and how they are calculated. The later part is denoted through the structure of the KPIs.	
Concept	Comment
Constant	
Variable	Together with the function and operands allows specifying formulas.
KPI	The core for describing the KPI structure.
Level	
has operand (operand for)	To allow building formulas through variables.
covers also (covered by)	

achieves (achieved by)
condition for (under condition)

Table 32: Concepts in KPI structure group

Evaluation process group	
The <i>Evaluation process group</i> describes the actions that are performed by humans to evaluate an action or a value through KPIs.	
Concept	Comment
Evaluation process	
Evaluation action	The core element for describing a process. It can be considered an action of type “evaluation”.
Event	Can be used and reused in other processes to provide points of reference between different processes.
Start	
End	
Decision	Used to split a path. Its action can be considered of type “decide”.
Parallelism	Used to split a path.
Synchronisation	Used to merge paths.
Action type	The aspect-specific concept, to allow categorising actions into additional types. Can be used to indicate the data collection, data transition and data processing subtypes of an evaluation action.
evaluates (evaluated by)	Use this to indicate what is evaluated.
performed on (in performance of)	Use this to indicate on what data an evaluation action works.
specialisation of (generalisation of)	The general concept, to specialise action types.
instance of (has instance)	The general concept, to indicate the action type of an action.
followed by (preceded by)	The aspect-specific concept, to specify the sequence and conditions for the process.
impacts (affected by)	The aspect-specific concept, to allow describing the impact of an action on other actions.
detailed by (describes)	The aspect-specific concept, to decompose actions into a finer granularity.
has part (part of)	The aspect-specific concept, to state what is contained by an evaluation process.
influences (influenced by)	The inter-aspect concept, to specify what values are consumed or produced.
requires (required by)	The inter-aspect concept, to indicate the use or adaptation of participants.
has responsible (responsible for)	The inter-aspect concept, to state the responsible for an activity.
has performer (performs)	The inter-aspect concept, to state the performer for an activity.

Table 33: Concepts in Evaluation process group

The following groups should also be available in this *Scope*:

- **Enterprise structure group** (Table 21) – The enterprise structure is used in this *Scope* to specify who is participating in the evaluation.
- **Participant collaboration group** (Table 23) – Similar to enterprise scope.

Through the use of *has operand* relations, *Constants* and *Variables* it is possible to describe formulas for the calculation of KPIs in a structured way. The here considered types are numbers, booleans and sets (not ordered) of both (examples with sets can be found below).

For some functions the order of the provided operands is relevant, meaning that a different operand order also creates a different result. To determine the order of operands the *Order number* property of the *has operand* relation should be used. Therefore, before calculating a value the operands should first be arranged based on this property in ascending order and passed to the function in the resulting sequence. For usability it is recommended that operands with the same order number are ordered arbitrary next to one another and if the *Order number* is missing it is assumed to be infinite (e.g. the following represents a sequence of properly arranged order numbers: “1 3 3 3 6 ∞ ∞ ”).

Many functions are binary functions (i.e. operate on two operands or arguments). If more than two operands are provided for a binary function, then several function executions have to be performed in a chain according to the order of the operands (if the order is relevant, otherwise in any order)⁷. The first execution should take the first two operands, while all other executions should take the result of the previous execution as the first argument and the next not used operand in the chain as the second argument. For example a subtraction using the operands “10”, “7”, “4” and “2” should first calculate “10 – 7 = 3”, then “3 – 4 = -1” and then “-1 – 2 = -3” with “-3” as the end result.

Also the here described binary functions accept several numbers, but can also use one set as an operand. In this case the binary operation should be performed on each value of that set and the result should be again a set. For example a subtraction using the operands “10”, “[1, 2, 3]” and “4” should first calculate “[10 – 1, 10 – 2, 10 – 3 = [9, 8, 7]]” and then “[9 – 4, 8 – 4, 7 – 4] = [5, 4, 3]” with “[5, 4, 3]” as the end result. It should be noted that when the order is relevant, there is also a difference between “Set X Number” and “Number X Set” (where X is the function/operation). Using more than one set as an operand is not considered here to prevent errors due to different lengths.

Additionally there are aggregation functions (aggreg. func.), which work on a set of values and create as a result a single value. When such a function is provided with several single values and sets as operands, then all of those numbers should first be assembled to a single set on which the function is then performed. For example when using sum on the operands “10”, “[1, 2, 3]” and “4”, then the sum function should be performed on one set of “[10, 1, 2, 3, 4]” and result in “20”.

Furthermore there are comparative functions (comp. func.), which are also binary functions and compare one operand to the other. If more than two operands are used then the same rules as described for the binary functions apply. This means building a chain of function executions based on the order (if the order is relevant) and in case sets are used to perform the function for each value of the set. For example when using the “smaller than” function with the operands “2”, “[4, 6, 8]” and “10”, each value of the set has to be larger than the lower ordered values (in this case larger than 2) and smaller than the higher ordered values (in this case smaller than 10). Unlike the other binary functions however, more than one set can be used. Using the “smaller than” function again for an example on the operands “[4, 6, 7]” and “[5, 8, 9]”, each of the numbers of the first set has to be smaller than any of the numbers of the second set. In this example the result should be “false” because “6” (as well as “7”) is not smaller than “5”.

It is recommended to provide the following functions:

⁷ In other words: Break it down to several binary operations.

Function	Input ⁸	Output	Order	Behaviour
Addition (Binary func.)	Several numbers, max. one set	One number or set	Irrelev.	Simple addition of all operands. When breaking it down to a chain of binary executions the order is irrelevant. Note that this is not the sum function for sets.
Subtraction (Binary func.)	Several numbers, max. one set	One number or set	Relev.	Simple subtraction where binary subtractions are performed according to the operand order.
Multiplication (Binary func.)	Several numbers, max one set	One number or set	Irrelev.	Simple multiplication of all operands. When breaking it down to a chain of binary executions the order is irrelevant. Note that this is not the product function for sets.
Division (Binary func.)	Several numbers, max one set	One number or set	Relev.	Simple division where binary divisions are performed according to the operand order.
Power (Binary func.)	Several numbers, max one set	One number or set	Relev.	It is the power function performed according to the operand order. This function can also be used to take any root.
Modulo (Binary func.)	Several numbers, max one set	One number or set	Relev.	The modulo function returns the remainder of a division and should be performed according to the operand order. For example “8 modulo 3” is “2”.
Negation (Unary func.)	One number, boolean or set	One number, boolean or set	Irrelev.	This is a unary function (i.e. only one operand) of negation. For boolean it means turning true to false and vice versa. For numbers it means turning positive numbers negative and vice versa. For sets it is executed on each value of the set.
Sum (Aggreg. func.)	Several numbers and/or sets	One number	Irrelev.	The sum (addition) of all the values.
Product (Aggreg. func.)	Several numbers and/or sets	One number	Irrelev.	The product (multiplication) of all the values.
Minimum (Aggreg. func.)	Several numbers and/or sets	One number	Irrelev.	The smallest number of all the values.
Maximum (Aggreg. func.)	Several numbers and/or sets	One number	Irrelev.	The largest number of all the values.
Count (Aggreg. func.)	Several numbers and/or sets	One number	Irrelev.	The amount of values from the operands.

⁸ If the input can be a set, then the set should only contain values of the other allowed types, i.e. if numbers and set, then only sets with numbers; if boolean and set then only sets with boolean; if numbers and boolean and sets, then the set can contain either numbers or boolean

Function	Input ⁸	Output	Order	Behaviour
Average (Aggreg. func.)	Several numbers and/or sets	One number	Irrelev.	The average of all the operands. Can also be calculated by using the sum and count functions: sum / count.
Variance (Aggreg. func.)	Several numbers and/or sets	One number	Irrelev.	The variance from mathematical probability theory and statistics. It measures how far the input values are spread out. Can also be calculated using the other functions: sum(power[subtraction(X, average[X]), 2]) / count.
Equal to (Binary comp. func.)	Several numbers, booleans and/or sets	One boolean	Irrelev.	This function should return true if all operands are equal. It differs from the “And” function in that if all values are false, this function should return true.
Different than (Binary comp. func.)	Several numbers, booleans and/or sets (numeric)	One boolean	Irrelev.	This function should return true if all operands are different from one another. When using boolean values it doesn’t make sense to have more than two values. Therefore only using numeric sets makes sense.
Smaller than (Binary comp. func.)	Several numbers and/or sets	One boolean	Relev.	This function should return true if the lower ordered operands are smaller than the higher ordered operands.
Smaller or equal (Binary comp. func.)	Several numbers and/or sets	One boolean	Relev.	This function should return true if the lower ordered operands are smaller or equal to the higher ordered operands.
Larger than (Binary comp. func.)	Several numbers and/or sets	One boolean	Relev.	This function should return true if the lower ordered operands are larger than the higher ordered operands.
Larger or equal (Binary comp. func.)	Several numbers and/or sets	One boolean	Relev.	This function should return true if the lower ordered operands are larger or equal to the higher ordered operands.
And (Aggreg. func.)	Several boolean and/or sets	One boolean	Irrelev.	This function should return true if all the values of the operands are true.
Or (Aggreg. func.)	Several boolean and/or sets	One boolean	Irrelev.	This function should return true if one of the values of the operands is true.
Load value (Interface func.)	---	One number or boolean	---	This function should allow loading a value from outside of the modelling tool (e.g. from a spreadsheet, a database, Linked Data etc.). Therefore the input and the order of operands are unspecified here.
Load set (Interface func.)	---	One set (numeric or boolean)	---	This function should allow loading values as a set from outside of the modelling tool (e.g. from a spreadsheet, a database, Linked Data etc.). Therefore the input and the order of operands re unspecified here.

Function	Input ⁸	Output	Order	Behaviour
Load property value (Interface func.)	---	One number or boolean	---	This function should allow loading a value from inside of the modelling tool (e.g. from an attribute of an object or relation). Therefore the input and the order of operands are unspecified here.
Load property set (Interface func.)	---	One number or boolean	---	This function should allow loading values as a set from inside of the modelling tool (e.g. from an attribute of an object or relation). Therefore the input and the order of operands are unspecified here.

Table 34: Recommended functions for Variables and KPIs

2.7 Mechanisms and Algorithms

This section proposes some mechanisms or algorithms to support the previously described procedures. It will focus on describing what the goal of a mechanism or algorithm is and what it should be capable of doing. Additional assumptions and prerequisites can be posed by the implementation. Details about how the input is acquired and how the output should be presented are omitted (unless absolutely necessary), since they are highly dependent on the possibilities of the used implementation approach.

2.7.1 Determine Instances/Templates for required Capabilities

The goal of this feature is to find matching instance and/or templates for a set of required capabilities. As previously described, instances can fulfil and templates can have capabilities. These can be checked against a set of required capabilities to determine which instances/templates fulfil the requirement. The set of required capabilities can be acquired through different means, for example it can be directly provided by a user or it can be based on the capabilities attached to a template. The latter case can be used to search for instances that fulfil the role according to its capabilities. This can be applied for example with *Business entities* and their provided *Business capabilities*. A requirement can be posed in the form of providing a certain value, for a certain price at a certain location. The capabilities provided by the available *Business entities* can then be checked against that requirement to find a set of possible business partners. The main application of this feature is in steps 1 and 2 of the procedure (see section 2.1.1).

This feature should be capable to:

- Receive one or several required *Capabilities* as input.
- Access available instances and/or templates and their fulfilled *Capabilities*.
- Compare the required *Capabilities* against the *Capabilities* fulfilled by an instance/template.
 - The comparison depends on the used *Capabilities*. For example comparing the provided value of a *Business capability* requires that either the value or a more general value is provided. When comparing skill levels then the provided *Skill* has to be better than the required one (higher levels of skill are better). When comparing costs or compensation then the available number should be smaller than the required one (lower costs are better).
- Return the available instances/templates that adhere to the required *Capabilities*.
 - The result can further contain some information about how much a fitting instance/template deviates from the required *Capabilities*. This can be useful for example when looking for *Performers* to prevent choosing over-qualified personnel.

2.7.2 Derivation of Participant collaboration

The goal of this feature is to derive a *Participant collaboration* out of already available processes to reduce the workload on the modeller. Since the Collaboration is dependent on an action, the Participant collaboration can be automatically derived from a process describing the action to some degree. Manual

adaptation might however still be necessary to complete a collaboration that properly represents reality. This feature can also be used in combination with a comparison (see for example section 2.7.11). Two *Participant collaborations* can be created for distinct processes, which can also be from different *Scopes*, and then structurally compared to one another. If both processes should have the same collaboration, then there should be no differences between the two *Participant collaborations*. The main application of this feature is in steps 2, 3 and 4 of the procedure (see section 2.1.1).

This feature should be capable to:

- Receive a process and the description of the desired participants as input.
- Create a *Participant collaboration*, which can be based on different assumptions.
 - The involvement of a participant in an activity is denoted by the *requires* relation (and its sub-relations). A switch between participants is indicated by two actions that require different participants.
 - One possible assumption would be that “if more than one participant element of the same type is required by the same action, then there should be *switches to* relations between all of them in both directions”. However different implementations can pose different assumptions.

2.7.3 Interaction stepper

The goal of this feature is to provide the user with a possibility to “walk through” or “step through” an interaction process in order to test and evaluate a mobile app concept as well as detect gaps in the requirements. It iteratively shows details about the actions and especially interactions with mobile devices executed by humans in order to showcase how a mobile app could be used to achieve a certain goal. For this the *Interaction flows* and *Participant* descriptions created in the *App-development Scope* should be used. However, it is also possible to start with a higher level process (e.g. a business process) that is decomposed into interaction flows. The main application of this feature is in step 3 of the procedure (see section 2.1.1).

This feature should be capable to:

- Receive a process, the necessary sub-processes (i.e. processes further detailing actions) and the description of the mobile app participants as input.
- Step through the actions according to their sequence, starting from an *Initiation event*. This should happen at a speed with which the user can keep up with.
 - When a path is split (e.g. because of a decision or a parallelism), then the feature should adhere to the splits semantics (AND, OR, XOR). If not all of the paths should be selected, it is recommended to ask the user instead of randomly choosing a path.
- Show details for each action that is visited if available (description of the activity, which *Interaction components* / *Points of interaction* are used etc.). For interactions it is recommended to show a mockup of the app and highlight the parts the user should interact with (if available).
- Directly step into another process, when it is further detailing the action that is visited. This functionality is optional, but recommended.

2.7.4 Derivation of Orchestration

The goal of this feature is to derive an *Orchestration* out of already available processes to reduce the workload on the modeller. Since both the source and the target of the derivation are processes, it is possible to automatically derive parts of an *Orchestration* process to some degree (e.g. *App executions* based on actions that require mobile apps, *Entries* and *Halts* based on *Initiation/Termination events* etc.). However, manual adaptations to the result might still be necessary. The derivation can also create different results based on certain assumptions for different cases. For example instead of creating one orchestration for one process, several orchestrations can be created, each one for a certain role, based on the assumption that each role will execute its own orchestration. The main application of this feature is in step 3 of the procedure (see section 2.1.1).

This feature should be capable to:

- Receive a process and the description of the mobile app participants as input.
- Create an *Orchestration* based on certain assumptions.
 - Those can be assumptions like “each *Action* that requires a *Mobile app* or *Mobile app template* should be an *App execution* in the *Orchestration*” or “each *Role* in the source process should have a separate *Orchestration*”.

2.7.5 Gathering access requirements

The goal of this feature is to gather access requirements described in processes and support the user in creating *Permission Rules*. When the requirement of participants in a process (e.g. stated through the *requires* relations from *Actions*) follows a similar structure to the description of permission rules (as it has been recommended), then it is easy to automatically create initial permission rules based on the requirements. These automatically created rules can then be checked and enhanced by an expert to create a security or access policy. The main application of this feature is in step 5 of the procedure (see section 2.1.1).

This feature should be capable to:

- Receive actions and/or processes as input. The actions should use the *requires* relation (or one of its sub-relations) to state the requirement for accessing a participant.
- Create *Permission rules* based on the stated access requirements.
 - It is recommended to flag the permission rules created this way to be checked by an expert.
 - Also it is recommended to skip requirements for which a *Permission rule* already exists.

2.7.6 Access requirement coverage check

The goal of this feature is to check if the access requirements based on a process are covered by the currently described permission rules. When the requirement of participants in a process (e.g. stated through the *requires* relations from *Actions*) follows a similar structure to the description of permission rules (as it has been recommended), then the two can be compared against one another. With this comparison the structure of the subjects, actions and resources has to be considered. For example, when *Role X* requires access to a certain resource, *Role Y* is permitted to get access to that resource and *Role X* is a specialisation of *Role Y*, then the access requirement is covered. The result should indicate which access requirements are not sufficiently covered, which can be used as input to solve the problem (e.g. either the requirement is not allowed due to legal constraints and the process has to be changed or the set of permission rules has to be changed). The main application of this feature is in step 5 of the procedure (see section 2.1.1).

This feature should be capable to:

- Receive permission rules and actions and/or processes as input. The actions should use the *requires* relation (or one of its sub-relations) to state the requirement for accessing a participant.
- Check each access requirement against the permission rules. The question that is asked here is “does a *Permission rule* exist that permits access for an access requirement?”
- Return the access requirements that are not covered by permission rules.
 - Additional information can be attached to this, for example permission rules that might be similar or access requirements that have to be manually checked against certain permission rules, because human readable constraints have been used.

2.7.7 Calculation of KPIs/Variables

The goal of this feature is to calculate definite values for KPIs and variables, based on their described structure. The automatic calculation of KPIs can support the evaluation, since KPIs are used there as measures. The details about the structure and description of KPIs and variables have been presented in previous sections (see sections 2.5.1.1 and 2.6.6). In addition to calculating the values, they can also be

used to automatically determine the *Level* that a *KPI* has achieved. The main application of this feature is in step 6 of the procedure (see section 2.1.1).

This feature should be capable to:

- Receive *KPIs*, their structure and their links to levels as input as well as any external data that is necessary.
- Calculate the values for each *Variable* and *KPI*, starting from the parts that have no further operands and working up towards the elements that are not used as operands.
- Also determine what *Level* a *KPI* achieves, if they are available and the *KPIs* are put in relation to them (*achieves* relation).
- Return the result of the calculation.
 - The result can additionally be stored as a property of the *KPI*.

2.7.8 Simulation of Procedural models

The goal of this feature is to simulate the execution of procedural models (e.g. *Business process*) in order to gather data about it without the necessity of carrying it out. Often the simulation of processes is used to estimate execution times and costs, but it can also be used to identify the paths in a process and their probabilities. Such data can then be used to look for optimisation potential in the process. Different approaches can be used to achieve the simulation and they have different assumptions and pose different restrictions on the processes (e.g. no loops, no jumps between different paths etc.). While those restrictions have to be considered when creating the models, they are based on the chosen approach and implementation, which is not covered in this section. The main application of this feature is in step 6 of the procedure (see section 2.1.1), but can also be used to generally simulate processes.

This feature should be capable to:

- Receive a *Process*, its *Initiation event* as a starting point and the participants and values necessary for the simulation as input (e.g. when simulating costs then the “Money” value and the influences relations towards it).
- Perform the simulation, during which the desired data is gathered.
 - It is recommended to let the user decide what data should be gathered. Typical data that is gathered:
 - Costs (e.g. quantity of negative influences relations towards “Money” value)
 - Running times (e.g. quantity of negative influences relations towards “Time” value or positive influences relations towards “Duration” anti-value)
 - Resource usage (e.g. requires relations towards any or a specific type of participant)
 - Value creation and consumption (e.g. quantity of influences relations towards *Values*)
 - Additionally the data is usually gathered for the different paths in a process (if more than one exists) and aggregated into a minimum, maximum and average value over all paths. When probabilities for splits in a process sequence are available, then the probability of each path can also be determined.
 - Also different strategies for collecting the data along a path are necessary. For example when confronted with parallel running actions (or action sequences), costs can simply be summed up while for the time the largest of the parallel running sequences has to be used.
- Return the simulation result.

2.7.9 Business model evaluation

The goal of this feature is to calculate the estimated revenue and expenses for a certain business model. Such estimates can then be used to compare different business models based on their profit. Both

expenses and revenues for each participant in a business model are described through the *exchanges value with* relations. Using those and the occurrences from the start stimuli a projection of the values exchanged and their valuation for each participant can be determined. The main application of this feature is in step 6 the procedure (see section 2.1.1), but can also be used to generally evaluate business models.

This feature should be capable to:

- Receive a *Business model* and the participants and values used in it as input.
- Determine the occurrences for each exchange of values based on the available constructs (start stimuli, depends on relations, their fractions and dependency controls).
- Aggregate for participants the value exchanges and their valuations based on the direction of the value exchanges.
- Return the aggregated result.

2.7.10 Serialisation of models as Linked Data

The goal of this feature is to serialise models or parts of models in a uniform format, to allow. The Resource Description Framework (RDF) and the concepts from Linked Data should be used for the serialisation format. Details which update the description from D3.1.1 can be found in section 3.3.1. The result can then be for example uploaded to a Linked Data server, where it can be further connected to other data and queried using SPARQL. This is a general feature that can be used throughout the procedure.

This feature should be capable to:

- Receive one or several models as input.
- Create the Linked Data based on the input.
 - It is strongly recommended to use the *Global identifier* (see section 2.3.2) as the URI for the RDF resources (if applicable) and to directly transform the parts from the *Property collector* (also section 2.3.2) directly into triples, in order to allow linking the data to other Linked Data sources.

2.7.11 Comparison of model serialisations in Linked Data

The goal of this feature is to determine and communicate the difference between two models (e.g. an older version compared to a newer) that have been serialised as Linked Data. It can be used to identify differences between different versions or for synchronisation of parts from the collaborative and procedural *Aspects*. The latter can be achieved when deriving the new *Collaboration* based on an updated *Process* (describing an *Action*) and checking the new *Collaboration* against the previous one⁹. The output can either be presented to the user or can be provided in a machine-readable manner to be further processed. This is a general feature that can be used throughout the procedure.

This feature should be capable to:

- Receive two Linked Data serialisations as input, one considered the source and the other considered the target of the comparison.
- Identify and return the differences between the two serialisations.
 - Since both should be RDF, and therefore based on triples the differences can be categorised in Add (triple to source) and Remove (triple from source) in relation to the target.

2.7.12 Model querying

The goal of this feature is to query data from created models. Such data can then be used to check for correctness and completeness of models as well as to create reports. This is a general feature that can be used throughout the procedure.

⁹ Such an approach for synchronisation is presented, because the derivation of collaborations is not considered 100% automated and can contain manual adaptations, therefore also complicating 100% automatic synchronisation.

This feature should be capable to:

- Receive a query and the model data it should be executed on it as input.
 - It is recommended to allow querying for objects, relations and properties as well as to filter those based on certain objects, relations and properties (e.g. property equal to, object id equal to, in relation to object etc.).
- Perform the query.
- Return the result of the query.

3 IMPLEMENTATION SPECIFIC RECOMMENDATIONS

This section provides some recommendations on how the previously described specification can be realised to facilitate implementation. It is based on the experience and knowledge gained through the existing prototypes delivered in D3.4.1 and assumes for most of its part similar approaches to the implementation (e.g. graphical modelling tool, both visualised and attribute-like relations etc.). The descriptions here should be understood as recommendations and propositions, from which deviations are possible and even encouraged if it better fits or improves a specific case. Also it is possible to handle certain parts of the specification differently to improve certain cases. For example in many scenarios the approach for describing levels for KPIs and their achievement isn't necessary in such a detail and can be simplified to the "traffic light" approach (levels green, yellow and red) in order to improve user friendliness for the modeller. However, the implementer should be aware of the implications that such changes result in.

3.1 Recommended Classes, Relations and Attributes

Different concepts and sub-concepts have been presented through sections 2.3-2.5 and assigned to groups in section 2.6. When implementing those concepts in a modelling tool a balance has to be struck between types that should be handled by separate objects and types that are handled through attribute values. In some cases certain concepts can be presented as attributes (e.g. *Business entity access*) or simplified (e.g. Levels and their hierarchy simplified to commonly used "traffic light" with red, yellow and green).

Table 35 and Table 36 provide a recommendation of classes, relations and attributes for the implementation. The class/relation/attribute names correspond to the names of the concepts/properties and further details (relation targets, descriptions etc.) can be found in the previous chapters. Also the generally used properties are omitted in the table (see 2.3.2). The here described classes/relations/attributes should be considered as suggestions, which means that they can be changed or new ones can be added as necessary. The general idea is that classes are drawn on the modelling canvas, attributes can be accessed through a separate window (e.g. "Notebook") and relations can either be drawn, treated similar to attributes or both. A metamodel describing the recommended implementation can be found in the appendix (section 6.2, Figure 33).

Class name	Attributes	Comment
Action type	---	
Activity	<ul style="list-style-type: none"> • Instructions • Auditing requirements 	
App execution	---	
Attribute	<ul style="list-style-type: none"> • Data type • Permission rules 	<i>Permission rules</i> should be described through attributes. The <i>for resource</i> relation is not needed in this case, since it is implicitly covered by the attribute.
Business entity	<ul style="list-style-type: none"> • Business capabilities • Contact information 	<p><i>Business capabilities</i> are described as tables, since their direct reuse is very limited. A capability (i.e. row in the table) should be simplified to contain: a) the Value (<i>provided value</i> relation without quantity), b) the price per unit (replaces <i>necessary compensation</i>), c) the time to delivery (replaces <i>provided value</i> of delivery speed), d) the maximum quantity and e) the location where it can be provided (<i>provided at</i> relation)</p> <p>Also the <i>Business entity access</i> is described directly through its attributes in the <i>Business entity</i>.</p>

Class name	Attributes	Comment
Business model	---	Can be a model type. It is also recommended to contain the <i>Business entities</i> , <i>Business roles</i> and <i>Market segments</i> that are used through the <i>Value interfaces</i> in the <i>Business model</i> .
Business process	---	Can be a model type.
Business role	<ul style="list-style-type: none"> Business capabilities 	<i>Business capabilities</i> are described as tables, since their direct reuse is very limited. A capability (i.e. row in the table) should be simplified to contain: a) the Value (<i>provided value</i> relation without quantity), b) the price per unit (replaces <i>necessary compensation</i>), c) the time to delivery (replaces <i>provided value</i> of delivery speed), d) the maximum quantity and e) the location where it can be provided (<i>provided at</i> relation)
Characteristic	---	
Constant	<ul style="list-style-type: none"> Value 	The <i>Value type</i> and <i>Set type</i> can be determined based on the provided value.
Decision	<ul style="list-style-type: none"> Inclusion type Question 	
Dependency control	<ul style="list-style-type: none"> Inclusion type 	
End	<ul style="list-style-type: none"> Intention type 	
End stimulus	---	
Entity	<ul style="list-style-type: none"> Permission rules 	<i>Permission rules</i> should be described through attributes. The <i>for resource</i> relation is not needed in this case, since it is implicitly covered by the attribute.
Entry	<ul style="list-style-type: none"> Start type 	
Evaluation action	---	Use <i>Action types</i> and <i>instance of</i> relation instead of an attribute for the types.
Evaluation process	---	Can be a model type
Event	---	
Exchange control	<ul style="list-style-type: none"> Inclusion type 	
Function execution	<ul style="list-style-type: none"> Source type 	
Halt	---	
Information access	<ul style="list-style-type: none"> Medium type Data source type Performed operations 	
Information instance	<ul style="list-style-type: none"> Permission rules 	<i>Permission rules</i> should be described through attributes. The <i>for resource</i> relation is not needed in this case, since it is implicitly covered by the attribute.
Information template	<ul style="list-style-type: none"> Permission rules Information type Access modifiers 	<i>Permission rules</i> should be described through attributes. The <i>for resource</i> relation is not needed in this case, since it is implicitly covered by the attribute.
Interaction	---	
Interaction component	<ul style="list-style-type: none"> Awareness type Content multiplication 	

Class name	Attributes	Comment
	type <ul style="list-style-type: none"> Intended for device 	
Interaction flow	---	Can be a model type.
Knowledge	---	
KPI	<ul style="list-style-type: none"> Function Last values 	The <i>Value type</i> and <i>Set type</i> can be determined based on the used function and operands. <i>Last values</i> should be used to store the last value/s hat has/have been calculated.
Location	<ul style="list-style-type: none"> Type Dependency type Area 	
Location control	<ul style="list-style-type: none"> Inclusion type 	
Market segment	<ul style="list-style-type: none"> Targeted Share 	
Mobile app	<ul style="list-style-type: none"> Permission rules Download link 	<i>Permission rules</i> should be described through attributes. The <i>for resource</i> relation is not needed in this case, since it is implicitly covered by the attribute.
Mobile app capability	---	
Mobile app template	<ul style="list-style-type: none"> Permission rules 	<i>Permission rules</i> should be described through attributes. The <i>for resource</i> relation is not needed in this case, since it is implicitly covered by the attribute.
Navigation model	---	Can be a model type.
Note	---	A class that can be used to leave comments or graphics on the modelling canvas.
Notification exchange	---	Can be a model type.
Notification received	---	
Orchestration	---	Can be a model type.
Organisation unit	<ul style="list-style-type: none"> Type Function Preferred occupation 	
Parallelism	---	
Path split	<ul style="list-style-type: none"> Inclusion type 	
Participant collaboration	---	Can be a model type.
Performer	<ul style="list-style-type: none"> Availability 	
Point of interaction	<ul style="list-style-type: none"> Interaction type Awareness type Data type 	
Relation	<ul style="list-style-type: none"> Permission rules 	<i>Permission rules</i> should be described through attributes. The <i>for resource</i> relation is not needed in this case, since it is implicitly covered by the attribute.
Requirements process	---	Can be a model type.
Role	<ul style="list-style-type: none"> Preferred occupation 	
Skill	---	
Start	<ul style="list-style-type: none"> Intention type 	

Class name	Attributes	Comment
Start stimulus	• Occurrences	
Swimlane	---	A class that can be used to visually structure models for human readers. It should not be used beyond that.
Synchronisation	• Inclusion type	
Value	• Value type • Excitement type • Axiologic type	
Value exchange	---	
Value exchange flow	---	Can be a model type.
Value interface	---	
Value set	• Inclusion type	
Variable	• Function • Last values	The <i>Value type</i> and <i>Set type</i> can be determined based on the used function and operands. <i>Last values</i> should be used to store the last value/s that has/have been calculated.

Table 35: Recommended classes and attributes

Relation name	Attributes	Comment
accessed through	---	
achieves green if	---	The relation itself covers which level is achieved. The target is the condition for achieving the level and should be a <i>Variable</i> .
achieves yellow if	---	The relation itself covers which level is achieved. The target is the condition for achieving the level and should be a <i>Variable</i> .
annihilates	---	
basis for	---	
configuration of	---	
contains	• Separable	Note that using the <i>Separable</i> attribute might not always be applicable.
depends on	• Fraction	
designed for	---	
detailed by	---	
evaluates	---	
exchanged value	• Quantity	
exchanges value with	• Outgoing valuation • Incoming valuation	
execute on	• Query	
executes	---	
followed by	• Conditions • Probability	
for subject	---	The source should also allow <i>requires</i> relations, to allow stating access requirements. Description can be used to specify additional human readable constraints.

Relation name	Attributes	Comment
for action type	---	The source should also allow <i>requires</i> relations, to allow stating access requirements. Description can be used to specify additional human readable constraints.
fulfils	---	
fulfils skill/knowledge	• Level of aptitude	
has business location	---	Source should be <i>Business entity</i> instead of <i>Business entity access</i> .
has capability	---	
has chief	---	
has initiator	---	
has manager	---	
has message	---	
has note	---	Source can be any element and the target should be a <i>Note</i> .
has operand	• Order number	
has part	---	
has performer	---	
has responsible	---	
has skill/knowledge	• Level of aptitude	
has value	• Separable • Quantity	
impacts	• Impact type	
implemented by	---	
implies	---	It is the subtype of <i>mandates value</i>
influences	• Quantity	
instance of	---	
interface part of	---	
involved participant	---	
is for	---	
justifies	---	
notifies	---	
owned by	---	
performed on	---	
prohibits	---	It is the subtype of <i>mandates value</i>
relates	• Role • Cardinality	
requires	---	In terms of access requirements, a description can be used to specify additional human readable constraints on the resource. The <i>for resource</i> relation is not needed in this case, since it is implicitly covered by this one.
specialisation of	---	
specialised value of	---	
strictly requires	---	See comment of <i>requires</i>
supported by	---	See comment of <i>requires</i>
switches to	---	

Relation name	Attributes	Comment
triggers	<ul style="list-style-type: none"> • Conditions • Available navigation patterns 	
with partner	---	

Table 36: Recommended relations and attributes

Some things have been changed in the proposed classes/relations/attributes compared to the concepts described in sections 2.3 to 2.5. Those changes are introduced to reduce the modelling effort and improve the modelling experience. Specifically they are:

- Added a *Note* and *Swimlane*, which allow to comment and structure models for human readers, without changing the meaning of the models. Also a relation *has note* to attach the note to objects. They can be used anywhere.
- *Business capabilities* are simplified to concentrate on “what” and “how much of it” is provided “how fast” for “what cost” and available at “which locations” through an attributes of the *Business entity* and *Business role*.
- *Business entity access* is handled through attributes instead of dedicated objects.
- *Permission rules* are also handled through complex attributes instead of using objects that would be drawn on a canvas. This also allows to omit the *for resource* relation, since they are part of the resource they are for.
- Access requirements are described through the *requires* relation (and its subtypes), which follows closely the structure of the *Permission rule*.
- *Value type* and *Set type* from the concepts of the *KPI structure group* can be determined by using the functions and operands (or the value for constants).
- The *Levels* have been simplified to plain “Red”, “Yellow” and “Green”. The relations *achieves green if* and *achieves yellow if* are used to indicate which variables have to be true to achieve a certain level, with “Green” overriding “Yellow” (i.e. if both “Green” and “Yellow” are true, then “Green” is achieved). If none of the two relations is true then the KPI is assumed to be on “Red” *Level*.
- *Participant involvement* and *involved participant* are not an explicit class/relation. Instead they should be handled like *Representative elements* from the *Procedural Aspect* (i.e. hide the specifics of their creation and change from the user if possible).
- The type of an *Evaluation action* is denoted using the *instance of* relation and *Action types*, instead of using an attribute.

3.2 Proposed Notation Guidelines

It is recommended to use notations that are intuitive and follow a coherent style, to facilitate the creation of models by a user. Following are some general guidelines for notations that can be used, assuming a two dimensional space based on graphical notations (“modelling on a canvas”) is used:

1) Motivator

- Use the shapes and/or colours to indicate the types of values

2) Participants




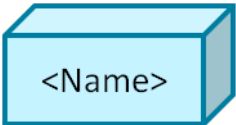

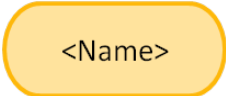

- All templates should use the same colour (e.g. green)
- All capabilities should use the same colour (e.g. orange)
- All instances should use the same colour (e.g. blue)
- The shape should indicate the types of templates and instances and should be the same for both with minor differences (e.g. a star for both *Performer [Instance]* and *Role [Template]*)
- Decomposition relations should use black lines
 - Solid if inseparable or unknown
 - Dashed if separable








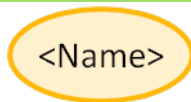



- f) Specialisation relations should be blue lines










3) Procedural


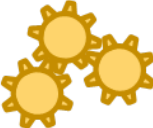

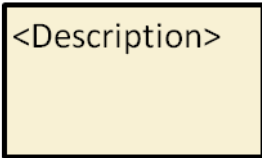







- a) *Actions* should use one colour and *Events* another (e.g. green for *Actions*, orange for *Events*)
- b) *followed by* relations should use solid black lines
- c) Decomposition between a *Process* and its contents should be denoted by drawing the contents inside the *Process*.
 - i) It is recommended to omit the visualisation of the decomposition between an *Action* and its *Processes* as arrows. It should be available through different means (e.g. attribute in a notebook)




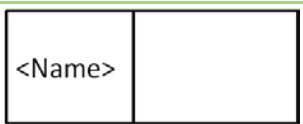









Table 37 shows some notations that can be used for the visualisation of the classes described in section 3.1. If a class/relation is missing, then it is recommended to not directly show it as an object/arrow on a canvas and instead handle its use through repositories, lists, queries, notebooks etc. Even the relations that have a notation specified should be manageable through means other than the modelling canvas if possible. Also many of the different decomposition relations (i.e. specialisations of *contains*) can be shown by the smaller part being in the larger part. It is additionally recommended to make use of the metamodeling platforms specific capabilities (e.g. hyperlinks, dynamic notations etc.) to enhance the user friendliness.

Class/Relation name	Proposed notation	Comments
Classes		
Activity	 <Name>	The responsible role could be shown in the middle. Other icons can be used to visualise additional relations or attributes (e.g. assigned resource types).
App execution	 <Name>	
Attribute	 <Name>	
Business entity	 <Name>	
Business role	 <Name>	
Characteristic	 <Name>	
Constant	 <Val>	The value can be shown in the middle.

Class/Relation name	Proposed notation	Comments
Decision	 <Question>	The middle can be used to show the inclusion type.
Dependency control		The shape should indicate the inclusion type. The notation shows two recommended shapes (round: AND, diamond: XOR).
End	 <Name>	
End stimulus	 <Name>	
Entity	 <Name>	
Entry	 <Name>	
Evaluation action	 <Name>	Colours can be used to indicate the type (e.g. data collection, transmission or processing)
Event	 <Name>	
Exchange control		The shape should indicate the inclusion type. The notation shows two recommended shapes (round: AND, diamond: XOR).
Function execution	 <Name>	
Halt	 <Name>	

Class/Relation name	Proposed notation	Comments
Information access	 <Name>	<p>The different types can be visualised in the middle.</p> <p>The performed operations can be shown at the top right of the element.</p>
Information instance	 <Name>	
Information template	 <Name>	
Interaction	 <Name>	
Interaction component	 <Name>	<p>Decomposition can be shown by putting the parts into the <i>Interaction component</i>.</p> <p>In some cases it can be useful to hide the name.</p>
KPI	 <Name>	<p>The function can be shown at the top right (as a simple icon/character like Σ for sum).</p> <p>The arrow in the middle (here green) can show the achieved level (green, yellow or red).</p>
Location	 <Name>	<p>The type of location can be visualised in the bubble.</p>
Location control		<p>The middle part should indicate the inclusion type.</p>
Market segment	 <Name>	<p>An icon can be used to indicate if it is targeted or not.</p>

Class/Relation name	Proposed notation	Comments
Mobile app	 <Name>	
Mobile app capability	 <Name>	
Mobile app template	 <Name>	
Note	 <Description>	
Notification received	 <Name>	
Organisation unit	 <Name>	Different colours (or other changes to the notation) can be used to visualise the type.
Parallelism		
Path split		The middle can be used to show the inclusion type.
Performer	 <Name>	
Point of interaction	 <Name>	The icons and colours can change depending on the different types. Decomposition can be shown by putting the <i>Point of interaction</i> into an <i>Interaction component</i> . In some cases it can be useful to hide the name.
Relation	 <Name>	

Class/Relation name	Proposed notation	Comments
Role	 <Name>	
Start	 <Name>	
Start stimulus	 <Name>	
Swimlane		The notation shows a horizontal version. A vertical version can also be provided for a different modelling direction.
Synchronisation		The middle should be used to show the inclusion type.
Value	 <Name>	Use colours to indicate the different types.
Value exchange	 <Name>	Several incoming and outgoing values can be shown in the middle (only one for each is shown in the example)
Value interface		The interface part of relation to the <i>Business entity / Business role / Market segment</i> can be denoted by putting the Value interface inside the entity or on its border.
Value set	 <Name>	The middle part should indicate the inclusion type.
Variable	 <Name>	The function can be shown at the top right (as a simple icon/character like Σ for sum). The <i>Value type</i> and <i>Set type</i> can be shown in the middle (where the "." is)
Relations		
accessed through		
configuration of		
contains		Use a dashed line for separable <i>contains</i> relations.







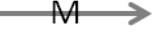









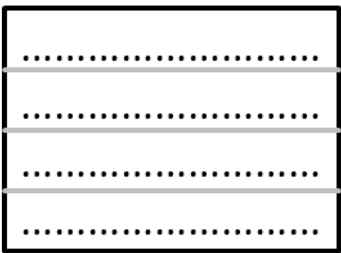
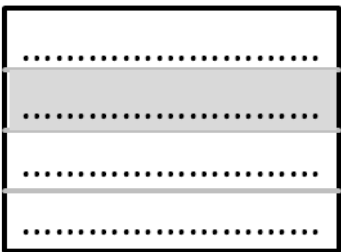
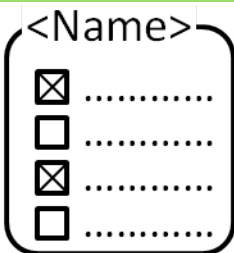
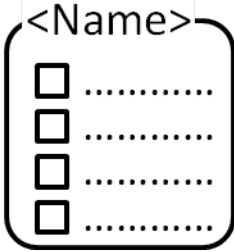

Class/Relation name	Proposed notation	Comments
depends on		The <i>Fraction</i> can be shown in the middle.
exchanges value with		The name of the exchanged <i>Value</i> can be shown in the middle.
execute on		
followed by		The <i>Conditions</i> can be shown in the middle.
fulfils		
has capability		
has manager		
has note		
has operand		The <i>Order number</i> can be shown in the middle.
has value		
notifies		The name of the message (<i>has message</i> relation) can be shown in the middle.
relates		
specialisation of		
switches to		
triggers		Allowed navigation patterns can be shown in the middle.

Table 37: Proposed notations for classes

In addition some classes can have new attributes for pictures or logos which can be visualised in the notation. For example a logo could be provided for a *Business entity*, which is then shown next to the proposed notation or completely replaces it, in order to improve identifying business entities. Similar approaches can be used for *Values*, *Market segments*, *Performers*, *Physical locations*, *Mobile apps*, *Points of interaction* and *Interaction components*.

Furthermore, alternative notations (here called concrete notations) for *Points of interactions* and *Interaction components* can be provided, which follow a visualisation style that is similar to how they would look on a mobile device. Those can then be used to create simple mockups in addition to describing the structure of a mobile app. Some examples for *Points of interactions* together with their mapping on the different types can be found in Table 38. In addition to considering the sub-types and data type of the *Point of interaction*, the content multiplication type of the enclosing *Interaction component* has to be regarded as well when choosing a concrete notation. If it is decided to provide such alternative depictions, then the implementer should further develop concrete notations depending on how visually suggestive the models should be. Depending on how extensive the visualisation capabilities should be additional attributes for controlling the notation might be needed.

Type	Proposed notation	Comments
Non-repeatable readable text	<Name>	
Non-repeatable interactive text	<Name> 	
Repeatable readable text		This is a list without any interaction.
Repeatable interactive text		This is a list where the user can select one or several items.
Non-repeatable readable boolean	<input checked="" type="checkbox"/> <Name>	
Non-repeatable interactive boolean	<input type="checkbox"/> <Name>	
Repeatable readable boolean		These are several selected elements without any user interaction.
Repeatable interactive boolean		These are several elements which the user can select to turn on or off.
Readable picture		The notation can show an example picture, preferably one that the user can set.

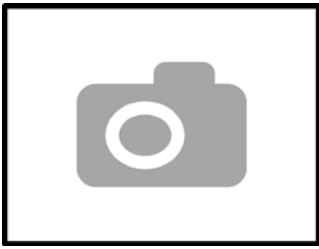

Type	Proposed notation	Comments
Interactive picture		The notation can show an example picture, preferably one that the user can set.
Interactive event		Technically a button.

Table 38: Examples of alternative notations for Points of interaction

The following figures show example mockups depicting in a notational manner the groups described in section 2.6. A mockup for the *Requirements process group* is omitted, since from a notational perspective it would look rather similar to the one of the *Business process group*, the envisioned differences being in granularity and semantics (activities are characterised by their relations to various types of participants). A notational distinction can be made for usability purposes, by including in the activity pictogram visual cues indicating what types of assets have been assigned and/or indicating hyperlinks for navigating to the related participant models.

Figure 9 depicts an example for a value structure where a shirt product is decomposed and shows its variability options (which can potentially become customisation options exposed to the end-customer). In the example, the product can have an optional embroidery component, and has sleeve length and colour as customisation options. The product is specialised in two configurations: a) a plain shirt, which has all the features of the root shirt, but explicitly prohibits the optional embroidery; b) an embroidered shirt that explicitly prohibits the black colour option but requires the optional embroidery. Warranty and eco-friendliness are shown as abstract features on which the modeller decides to compete on the market.

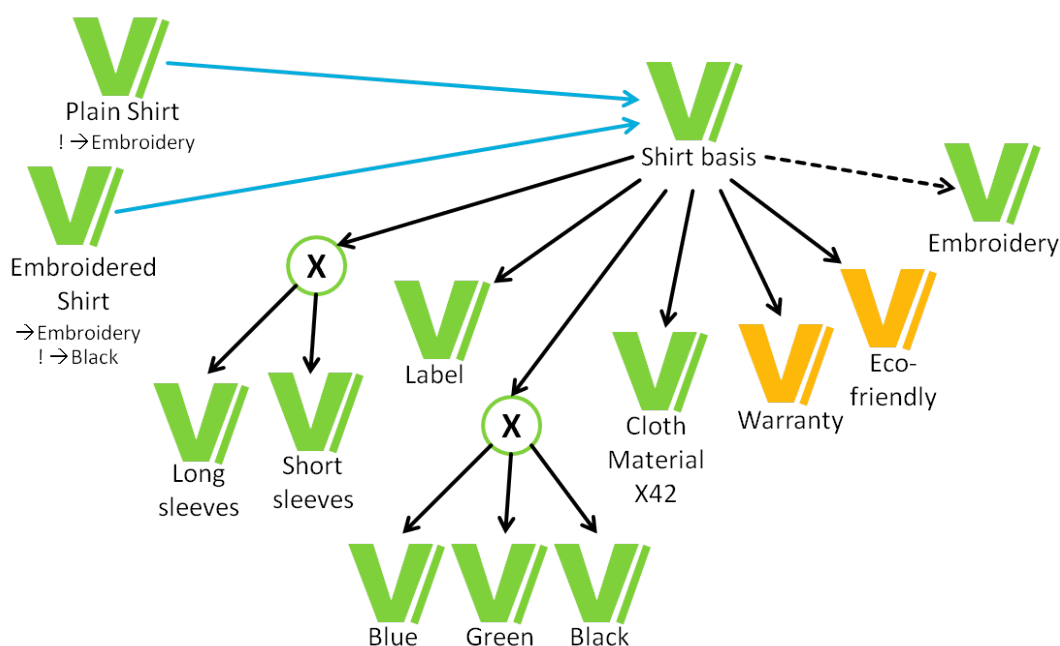


Figure 9: Example mockup for the Value structure group

Figure 10 depicts a decomposition of the 50+ (aged) market segment in a price-sensitive sub-segment and a 100+ sub-segment. For each segment, its key characteristics are assigned. These have to be considered when defining the features of the value structure.

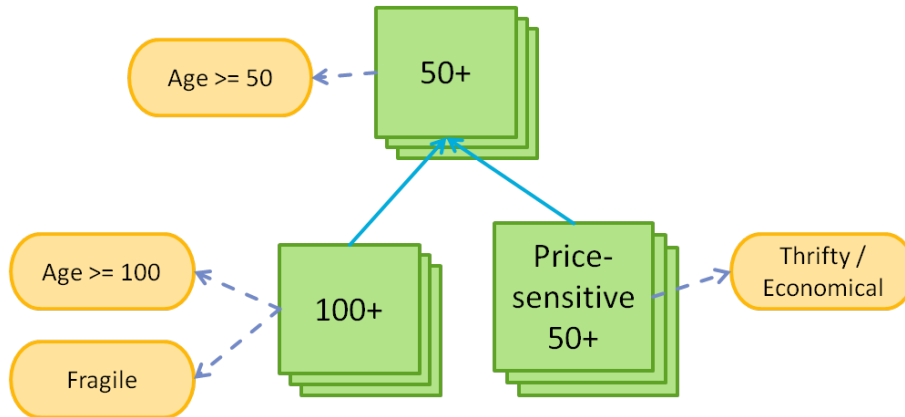


Figure 10: Example mockup for the Market structure group

Figure 11 describes in a colour coded manner business entities (e.g. “SewInc.”) and business roles that they can fulfil (e.g. Sewing).

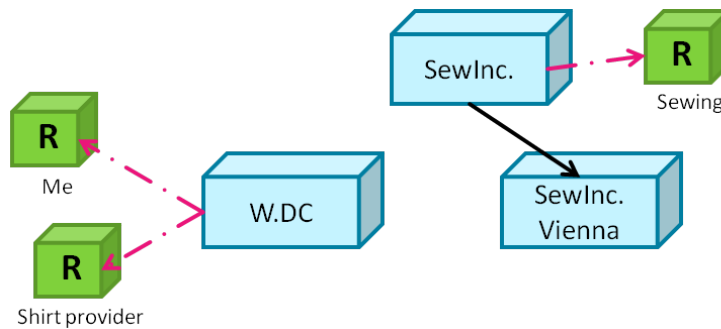


Figure 11: Example mockup for the Business structure group

Figure 12 depicts a decomposition of physical locations (characterising, for example, business entities) and digital locations (webpage URLs or endpoints for data access).

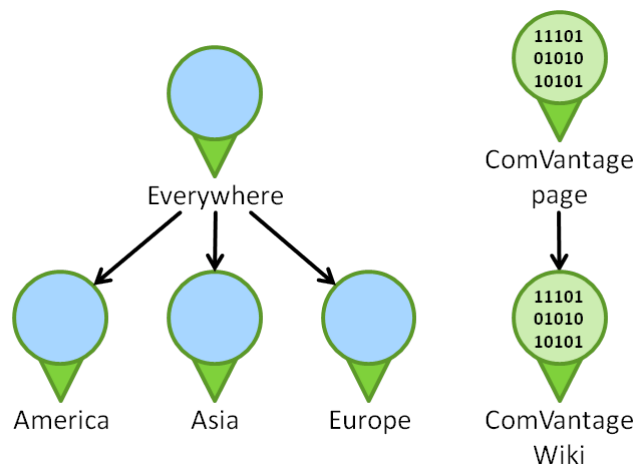


Figure 12: Example mockup for the Location structure

Figure 13 depicts a value exchange process that complements the value exchanges described from a collaboration viewpoint in Figure 14 (where order is not captured, with the exception of initiation and finalisation). The example has in its centre a company that targets the Wealthy market segment (linked to a

model like Figure 10) with Shirt products whose creation involves a Textile producer (taking money as input) and a Shirt sewer (taking money and textile as input) – both linked to business roles or entities depicting Figure 11.



Figure 13 Example mockup for the Value exchange flow group

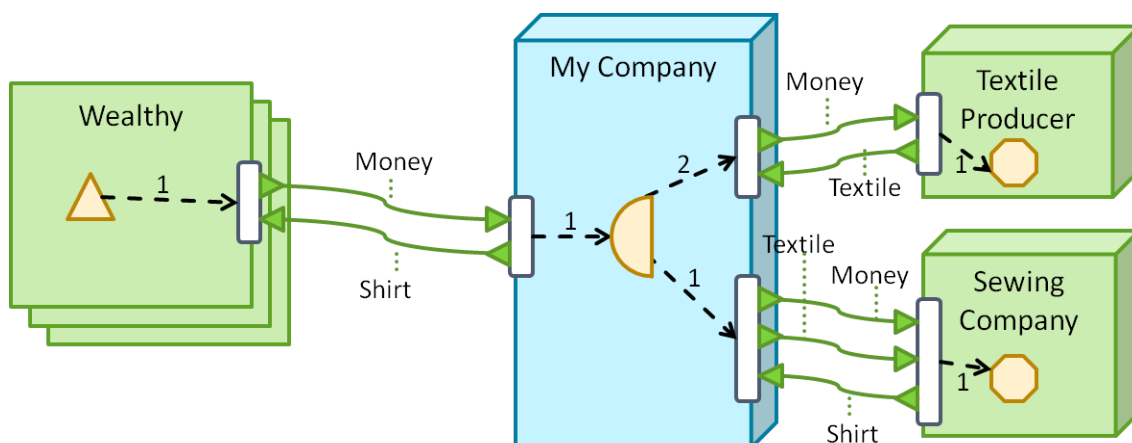


Figure 14: Example mockup for the Business model group

Figure 15 is an extension of an eco-system (like the one depicted in Figure 11) at enterprise level, hence describing organisation decomposition in departments and performers (including visual cues for managers). Again, colour coding enables the distinction between concrete instances (to be used, for example, in an as-is model) and roles (from a role hierarchy), with the fulfilment relation linking them.

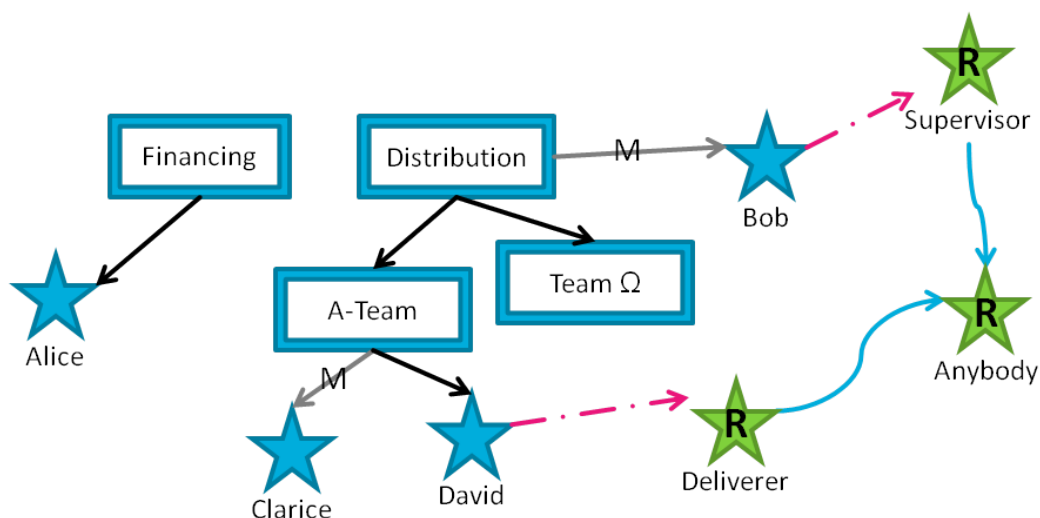


Figure 15: Example mockup for the Enterprise structure group

Figure 16 depicts a cycle time incident handling process with role assignments (from a model like the one depicted in Figure 15), where the transition conditions outgoing from the decision “Cycle time OK?” are represented as outcome events with the goal of benefiting from reuse of events. For example, if two different process model describe the same process with different granularity (e.g. a business process model

and a requirements-oriented process model), applying the same events for both of them would create common “checkpoints” highlighting the commonality between the two models and enabling additional model queries. Event reuse is also aimed to be applied in event handling, when the process start in one model (the incident handling process) is the same object as an event from another model (main process).

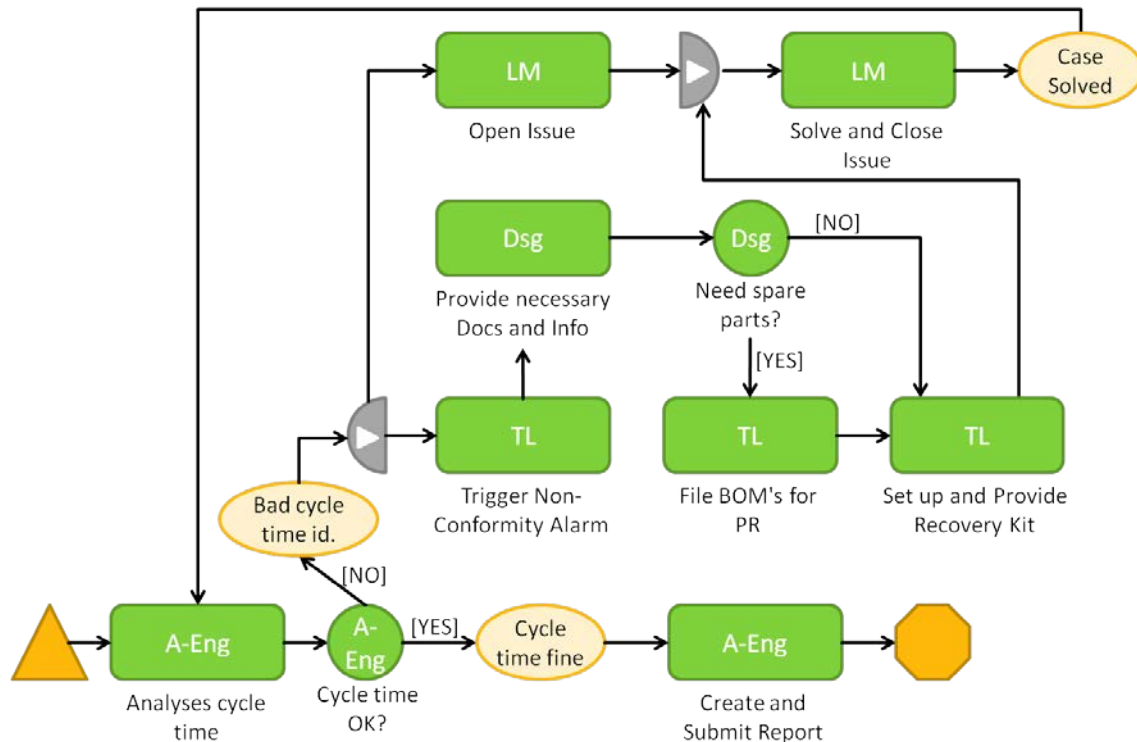


Figure 16: Example mockup for the Business process group

Figure 17 complements the process from Figure 16 with a collaborative view on the involved roles. Notice the relative similarity to how the business model (Figure 14) and the value exchange process (Figure 14) complement each other in giving a complete picture of both the procedural and participant interaction facets.

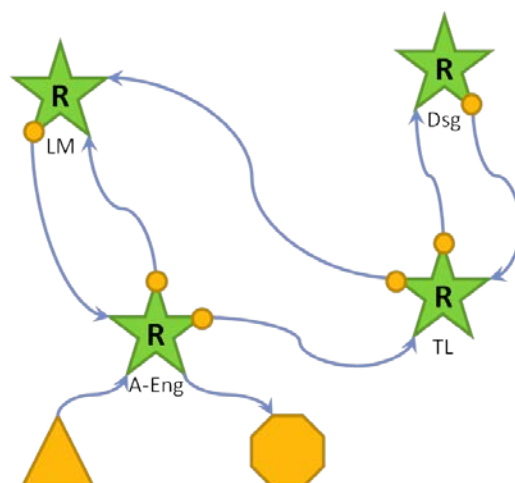


Figure 17: Example mockup for the Participant collaboration group

The next figures take participants descriptions down to the level of assets. Figure 18 covers the mobile app requirements on several levels of detail: in the middle, an app template hierarchy with mapped capabilities and an identified instance (Skype) is depicted. On the left side, a particular app template is structured in its

interaction components according to the POI taxonomy proposed by the work at hand. On the right side, an orchestration is explicitly described by comprising several apps.

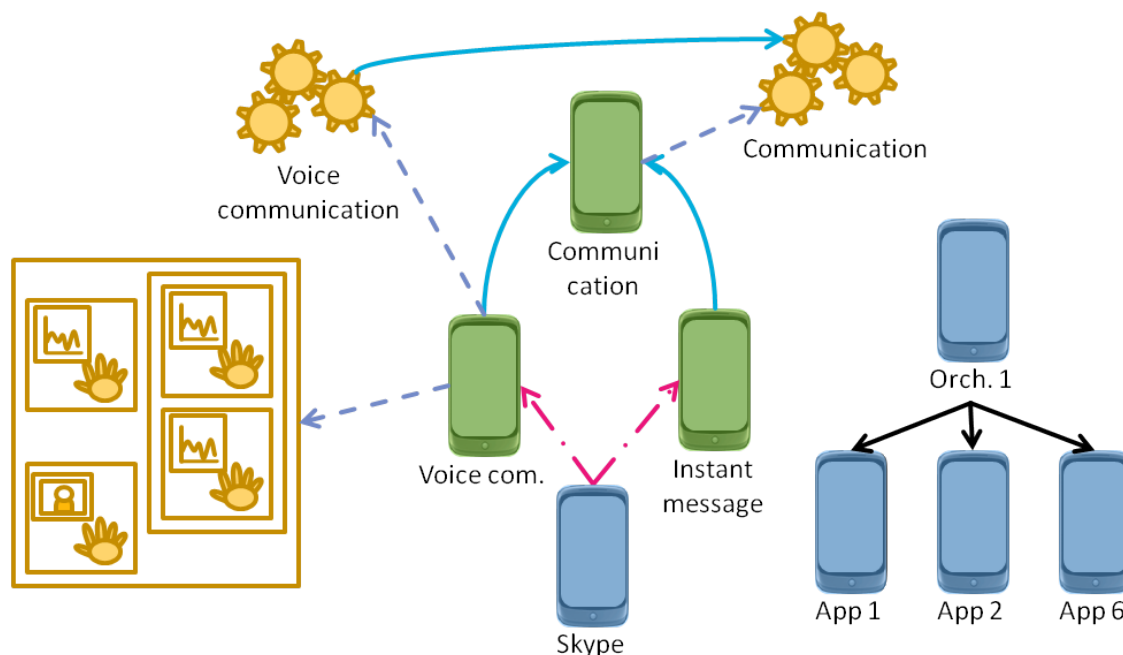


Figure 18: Example mockup for the Mobile support structure group

Figure 19 describes a required information space, with several types of modelling objects: information resources (e.g. Orders), elements of an entity-relationship diagram detailing what data entities and which of their properties are covered by an information resource (e.g. the Person entity, the Name attribute) and what access means are provided for each information resource (further linked to endpoint locations from models like the one depicted in Figure 12). An XOR location control indicator shows that "Access 5" is provided at multiple locations (e.g. a query supported by multiple endpoints).

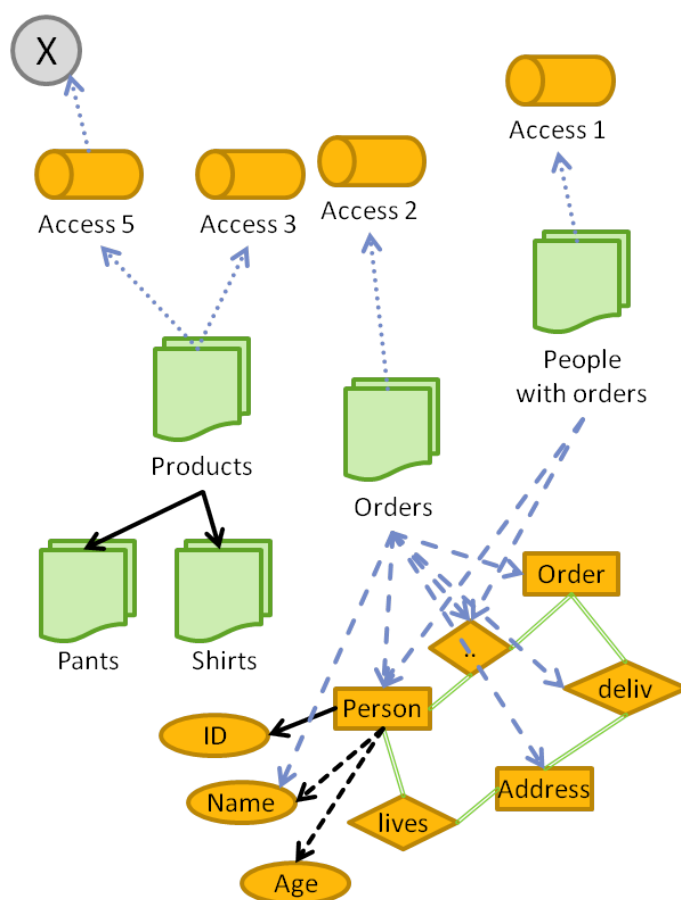


Figure 19: Example mockup for the Information space group

Figure 20 depicts an interaction process whose steps are either human-app interactions or app functions reacting to interaction steps. Interaction elements (like those on the left-side of Figure 18) and information assets (like those from Figure 19) can be linked to the elements of such a process.

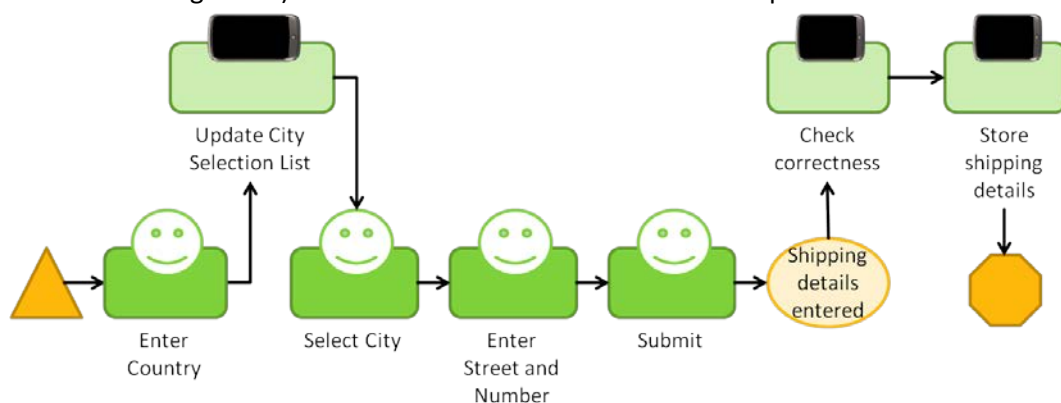


Figure 20: Example mockup for the Interaction flow group

Figure 21 describes a navigational map considering the app components and their triggers involved in one or several interaction flows.

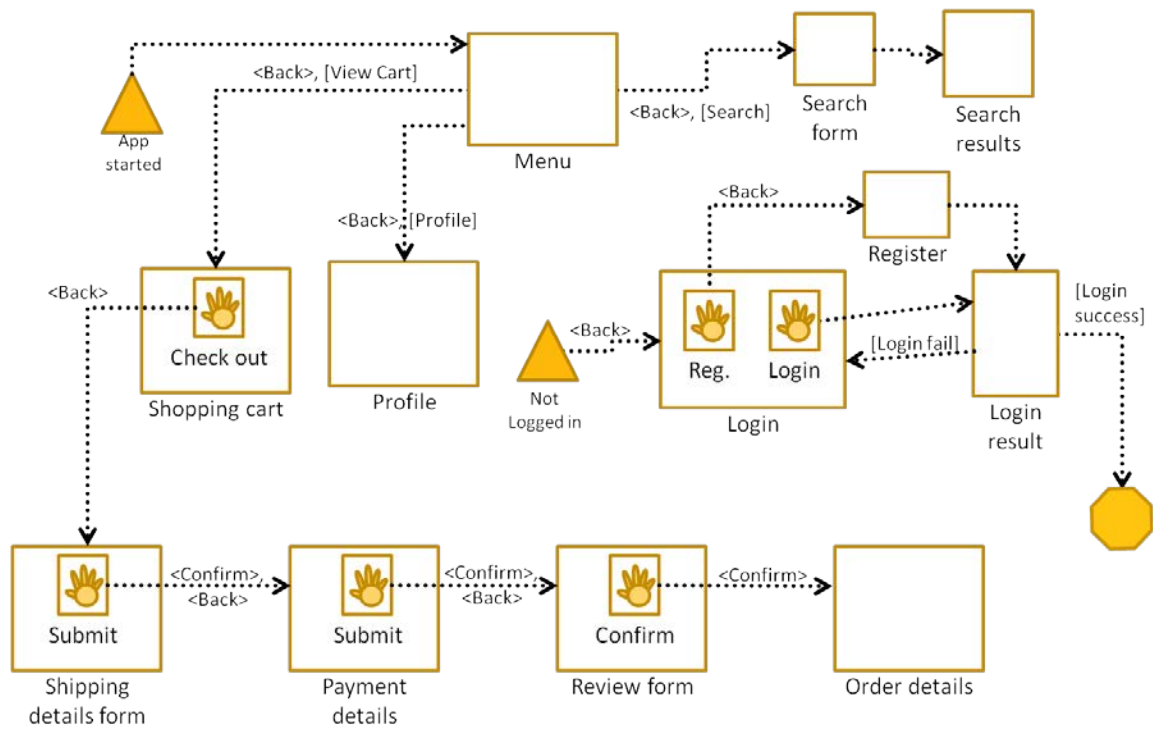


Figure 21: Example mockup for the Navigation map group

Figure 22 describes an orchestration model indicating the app usage flow derived from a business process with app requirements. It has visual cues for the initiation of a notification (last use of Issue management) and for a dependency on a notification (Repairedocs received).

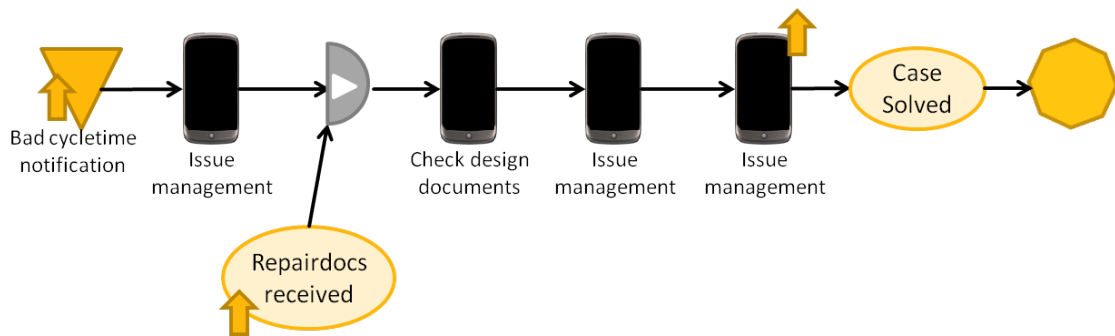


Figure 22: Example mockup for the Orchestration group

Figure 23 depicts notification dependencies between different app ensembles, hence describing app interactions in a similar fashion to the role interactions from Figure 17.

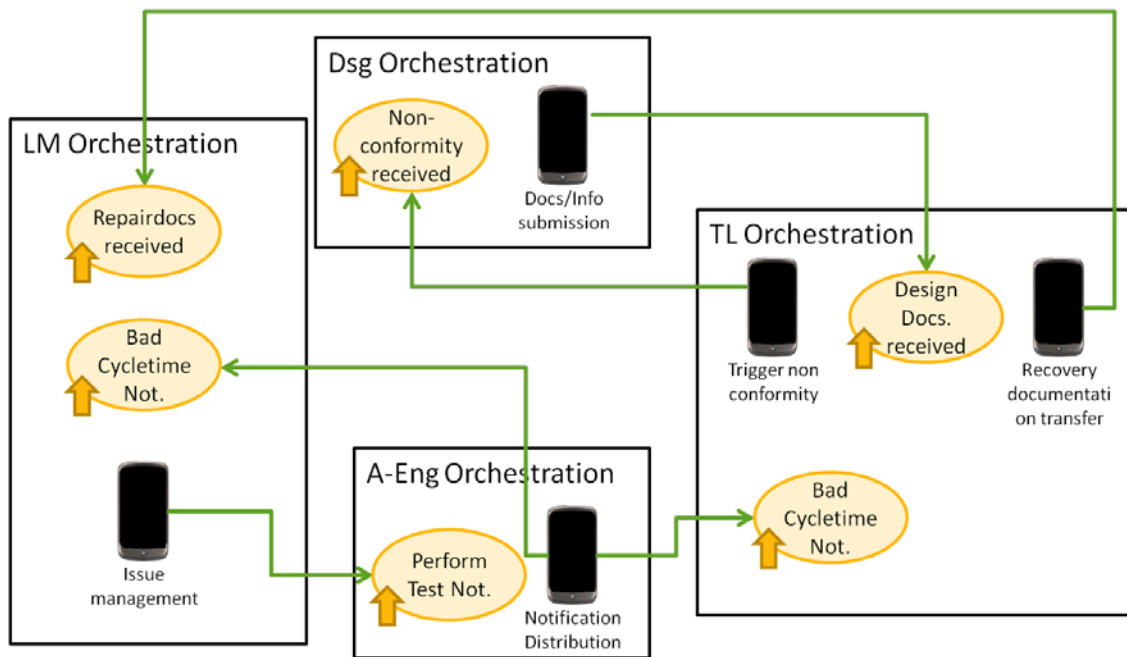


Figure 23: Example mockup for the Notification exchange group

Figure 24 depicts the structure of two KPIs. On the right side, the *Mean time to repair* KPI is computed as the division between a sum and a count of repair times (with visual cues indicating the operation and the digit 1 indicating the first operand in the division). On the left side, two conditions are defined for this KPI: the green condition obtained if the KPI has a value lower than 3, the yellow condition if the value is lower than 5.

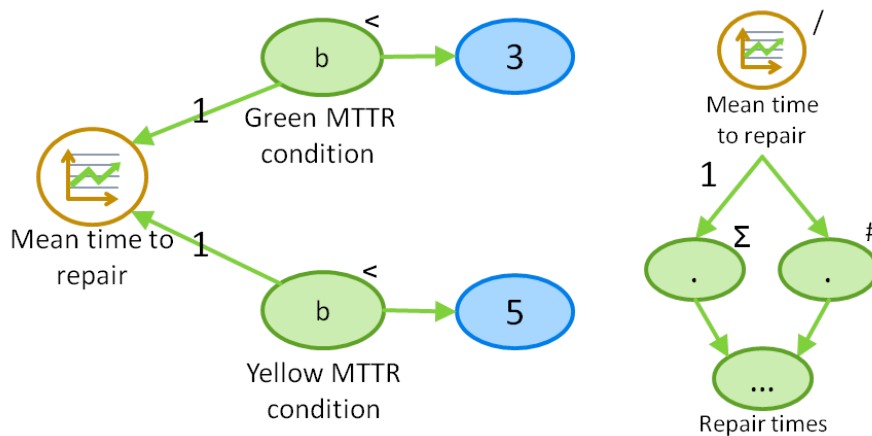


Figure 24: Example mockup for the KPI structure group

Figure 25 describes an evaluation process with visual cues suggesting the typing of the process tasks (data collection steps, data transmission steps and calculation steps linked to models like those depicted in Figure 24).

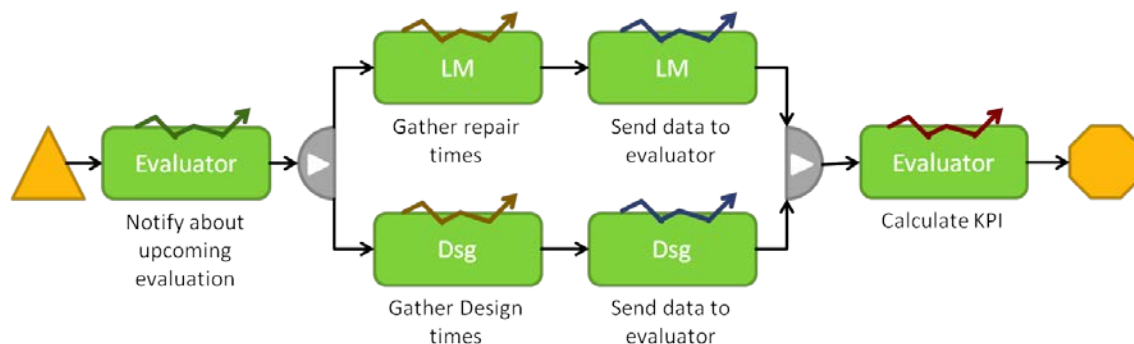


Figure 25: Example mockup for the Evaluation process group

3.3 Recommended approaches for Mechanism and Algorithm implementations

This chapter provides some recommendations on how the functionalities described in section 2.7 can be implemented. Some of them however pose additional restrictions on the input that is used (e.g. the simulation does not allow the use of the OR-inclusion type on any path). Again the descriptions here are propositions and can be improved upon (e.g. add caching to increase performance) or other approaches and designs can be chosen by the implementers (because of e.g. restrictions of the chosen architecture, to lift some of the restrictions of the proposed approaches etc.).

3.3.1 Recommended approach for Determine Instances/Templates for required Capabilities

This functionality determines instances or templates for a set of required capabilities. Differently put it can be thought of as finding a fitting instance or template to substitute a set of requirements. Here the general approach on how to find the desired elements is in the focus. The details of the comparison of capabilities, the data structure that should be used as well as any details on the user interface and presentation of inputs and outputs are up to the implementer. The here presented approach poses some requirements that must be fulfilled:

- Access to the necessary model data of the participants is available.
- Only capabilities are used that the implementation knows how to interpret and compare.

The procedure for finding the instances and templates takes as an input both the required capabilities (called Required Capabilities) and the desired type of element (called Desired Type, e.g. *Performer*, *Role*, *Business entity* etc.).

1. Get all eligible elements that are of Desired Type as Possible Candidates
2. For each of those elements as Candidate
 - 2.1. For each of Required Capabilities as Requirement
 - 2.1.1. If Candidate does not fulfil Requirement
 - 2.1.1.1. Remove Candidate from Possible Candidates
3. Return Possible Candidates

The most complicated part of this procedure is checking if the candidate fulfils the requirement. For this one of the available capabilities has to cover the required one. This check however depends on the type of used capability. For examples *Characteristics* can simply be checked using their identifications, while *Skills* and *Knowledge* should be checked for both their identification (i.e. that both talk about the same skill/knowledge) and also the level (i.e. the level of the available skill has to be the same or higher as the required level). For more complex capabilities like *Business capabilities*, which contain the provided value, the maximum quantity and the price per unit among other things, it is recommended to split them up and compare the smaller parts accordingly (e.g. available price is smaller or equal than required price, available quantity is larger or equal than required quantity etc.).

Additionally, since both capabilities as well as the parts that they can contain can be described through a hierarchy, this hierarchy has to be considered when comparing them. For such a comparison, the available

capability has to be either the same as either the required one or a more general capability of the required one. A simple way of determining this is to put the required capability and all the capabilities it specialises (including transitively) in a list and then check if the available capability is in that list. For efficiency those lists could be determined and stored before step 1. A more sophisticated approach would be to consider the structure of specialisation as a graph and see if the available capability (target) can be reached from the required capability (source) using one of many different algorithms (depth-first search, Dijkstra etc.)

An alternative approach for finding elements for a set of required capabilities is to use queries. An example for a query that finds *Performers* for a *Role* (i.e. the Role describes the required capabilities) and is executed on model data serialised as Linked Data can be found in the appendix (section 6.1). However, for different cases the query has to be adapted depending on the types of elements and capabilities as well as the structure of the serialisation. Also it does not consider the capability hierarchy.

3.3.2 Recommended approach for Derivation of Participant collaboration

This functionality supports the user with the creation of a *Participant collaboration* out of a procedural model (simply called process). The approach chosen here derives an initial *Participant collaboration* out of a process, which can further be enhanced by a user. The data structure that should be used as well as any details on the user interface and presentation of inputs and outputs are up to the implementer. The approach poses some requirements that must be fulfilled:

- The procedural model uses the *requires* (or one of its specialisations) relation to indicate what participant is participating.
- Access to the model data about the necessary elements is available.
- Further decomposition of elements is ignored.

The approach for deriving the participant collaboration uses a process as an input and also the desired participant type for the collaboration. Additionally it uses a set Participants to temporarily store several participants.

1. Create a participant collaboration and create the *is for* relation to the corresponding action
2. For each *requires* relation where the target is of the desired participant type
 - 2.1. Add the target of the requires relation to the participant collaboration
3. For each element as Procedural that is the source of a *requires* relation to an element of the desired participant type
 - 3.1. Get all *requires* relations where the target is of the desired participant type and the source is Procedural
 - 3.2. For each of those relations
 - 3.2.1. Add the target of the relation to Participants
 - 3.3. Create *switches to* relations between all elements in Participants (in both directions)
 - 3.4. Follow the path(s) going out of Procedural (use *followed by* relations) until
 - a) reach a procedural element that is the source of a *requires* relation, where the target of the relation is of the desired participant type; the procedural element will be called Followed Element
 - b) cannot go any further (because an end is reached or a loop is detected).
 - 3.5. For each case a) (can happen multiple times if several paths are found)
 - 3.5.1. Get all *requires* relations where the target is of the desired participant type and the source is Followed Element
 - 3.5.1.1. For each of those relations as Requires
 - 3.5.1.1.1. Create *switches to* relations from all of the elements in Participant to the target of Requires
4. For each *Initiation event* in the process as Initiation
 - 4.1. Create a *Start* in the participant collaboration
 - 4.2. Follow the path(s) going out of Initiation until a procedural element that is the source of a *requires* relation, where the target of the relation is of the desired participant type is reached
 - 4.3. For each of those elements found as Followed Element

4.3.1. Create a *switches to* relation from the created *Start* to Followed Element

5. For each *Termination event* in the process as Termination
 - 5.1. Create an *End* in the participant collaboration
 - 5.2. Follow the path(s) in reverse coming in Termination until a procedural element that is the source of a *requires* relation, where the target of the relation is of the desired participant type is reached
 - 5.3. For each of those elements found as Followed Element
 - 5.3.1. Create a *switches to* relation from Followed Element to the created *End*

When creating a relation between two elements, creating duplicates should be avoided (i.e. don't create a relation of a certain type between the source and the target if another relation of the same type, with the same source and the same target already exists). Also avoid creating relations where the source and the target are the same element. In general the 5 main steps from above can be simplified as:

1. Create empty participant collaboration
2. Put participant involvements in participant collaboration
3. Connect participant involvements based on process
4. and 5. Add possible *Start* and *End* elements based on process and connect them

Any step that uses "Follow the path(s)" is meant to find the next element in the process sequence that has some information that is necessary. This can lead to several elements, since control elements (e.g. *Decisions*, *Parallelisms* etc.) can be along the path. It can be realised through a depth-first search with the corresponding termination rules.

3.3.3 Recommended approach for Interaction stepper

This functionality showcases the interactions with a mobile device for a certain process. The approach described here focuses on how procedural models can be used for that, leaving details about the data structure, the user interface and the presentation of inputs and outputs to the implementer. However, an example for how the interaction stepper could look is shown in Figure 26. The upper part is showing the process or processes that are currently stepped through, with the current *Action* highlighted, while the bottom part shows some of the properties of the current *Action*. The right part shows the mockup of the screen that would be seen on the mobile device with the parts highlighted that are used at the current *Action*. The screen mockup should be based on the description through the *Points of interaction* and *Interaction components*.

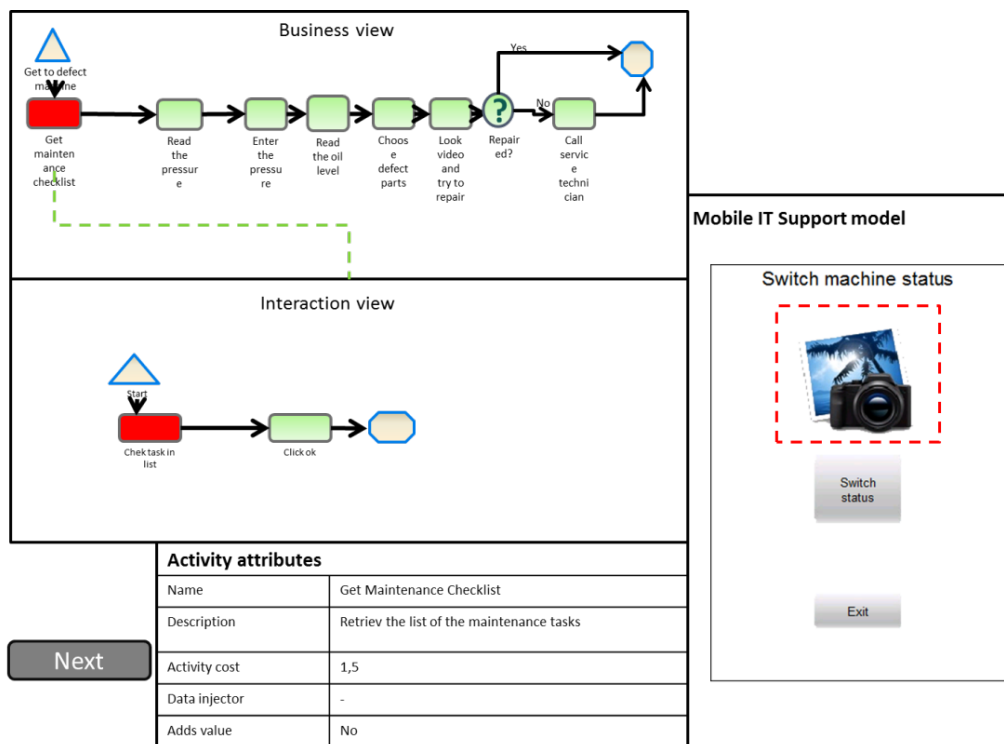


Figure 26: Example mockup for the Interaction stepper

The here presented approach poses some requirements that must be fulfilled:

- Only detailing *Actions* through *Interaction flows* (i.e. decomposition) is considered. Other decomposition of *Actions* into processes is ignored.
- Only *Split-Control* elements can have multiple outgoing *followed by* relations.
- Only *Control* elements of AND-/XOR-inclusion type are allowed.
- Every *Split-Control* element of AND-inclusion type must have one corresponding *Merge-Control* element of AND-inclusion type. Also all paths out of the *AND-Split* must be led together in the same *AND-Merge*.
- All outgoing *followed by* relations of a *Split-Control* element of XOR-inclusion type must have a transition condition specified.

Note that the here described approach can be modified to also cover different cases. For example instead of letting a user choose the path of a decision (*XOR-Split*), a predefined list of choices can be used, forcing the viewer on a specific path. Such and other extensions are however left open to the implementer.

The focus of this part lays in the approach of stepping through a process.. For this a function is used, that takes as an input the current node, steps to the next node, displays information about this node and updates the user interface appropriately. So the function should be called every time a “step” from the current Action to the next should be taken¹⁰. The next nodes that should be visited are stored in an ordered set (array) called “nodes to be parsed”. Therefore when starting the first time, the Initiation event should be in that array. A user interface similar to the one shown in Figure 26 is assumed in this description.

- Set the Current Node to the first element from “nodes to be parsed”
- Mark the Current Node in GUI as visited
- Remove first element from “nodes to be parsed” (i.e. remove the Current Node)
- If the Current Node is an *AND-Split*

¹⁰ This can be triggered for example by the user pressing a button or in timed intervals to automatically “play” through the process.

- 4.1. Add nodes after the Current Node to beginning of “nodes to be parsed”
- 4.2. Add number of nodes after the Current Node to beginning of “opened paths”
5. If the Current Node is an *XOR-Split*
 - 5.1. Ask which outgoing *followed by* relation to take (where the Current Node is the source of it)
 - 5.2. Set the Current Node to target of the relation
 - 5.3. If the Current Node is not an *AND-Merge*
 - 5.3.1. Add the Current Node to beginning of “nodes to be parsed”
6. Else
 - 6.1. Get the outgoing *followed by* relation (where the Current Node is the source of it)
 - 6.2. Set the Current Node to the target of the relation
7. If “nodes to be parsed” does not contain the Current Node
 - 7.1. If the Current Node is an *AND-Merge*
 - 7.1.1. Get the first element from “opened paths” as index
 - 7.1.2. Remove the first element from “opened paths” (i.e. remove index)
 - 7.1.3. Add the Current Node to “nodes to be parsed” at position of index
 - 7.2. Else
 - 7.2.1. Add the Current Node to beginning of “nodes to be parsed”
8. Set Current Node to the first element from “nodes to be parsed”
9. Mark the Current Node in GUI as active
10. Update display of attributes
11. If the Current Node requires a *Point of interaction* or *Interaction component*
 - 11.1. Highlight the *Point of interaction* / *Interaction component* in the mockup view
12. If the Current Node is further detailed by an Interaction flow
 - 12.1. Add the *Initiation event* of the Interaction flow to beginning of “nodes to be parsed”
 - 12.2. Change showed processes in GUI accordingly.

3.3.4 Recommended approach for Derivation of Orchestration

This functionality supports the user with the creation of an *Orchestration* out of a procedural model (simply called process). The approach chosen here derives an initial *Orchestration* out of a process, which should further be enhanced by a user. The data structure that should be used as well as any details on the user interface and presentation of inputs and outputs are up to the implementer. The approach poses some requirements that must be fulfilled:

- The procedural model uses the *requires* relation to indicate what resource should be used in the orchestration.
- Each *Action* should have only one performer and require one element of the desired resource types (e.g. only one *Mobile app* or *Mobile app template*).
- Access to the model data about the necessary elements is available.
- Further decomposition of elements is ignored.

The approach for deriving the orchestration uses a process as an input. Additionally to a set called “Mapping” that stores a pair containing the procedural element from the process and its corresponding resource¹¹ and a set called “Processed” to automatically add *Entries* in the orchestration.

1. For each distinct performer used in the process
 - 1.1. Create an orchestration and assign the performer (using the *designed for* relation)
2. For each element in the process that has a *requires* relation to an element of the desired resource type
 - 2.1. Get the performer of the element
 - 2.2. Get the orchestration for the performer
 - 2.3. Create an *App execution* in the orchestration and link it through *executes* to the resource
 - 2.4. Put the procedural element and the created *App execution* in Mapping

¹¹ This is used to store which resource in the orchestration represents which action from the process and vice versa.

3. For each orchestration as Orchestration1
 - 3.1. For each *App execution* in the orchestration as Execution1
 - 3.1.1. Get the corresponding procedural element from Mapping as Procedural1
 - 3.1.2. Follow the path(s) going out of Procedural1 (use *followed by* relations) until
 - a) reach a procedural element that is contained in Mapping or
 - b) cannot go any further (because an end is reached or a loop is detected).
 - 3.1.3. For each case a) (can happen multiple times if several paths are found)
 - 3.1.3.1. Get the *App execution* corresponding to the found procedural element from Mapping as Execution2
 - 3.1.3.2. If Execution2 is part of Orchestration1
 - 3.1.3.2.1. Create a *followed by* relations from Execution1 to Execution2
 - 3.1.4. For each case b) (can happen multiple times if several paths are found)
 - 3.1.4.1. Create a *Halt* in Orchestration1
 - 3.1.4.2. Create a *followed by* relation from Execution1 to the *Halt*
4. For each *Initiation event* in the process
 - 4.1. Follow the path(s) going out of it and for each element on that path as Procedural
 - 4.1.1. If Procedural is in Mapping
 - 4.1.1.1. Get the performer of Procedural
 - 4.1.1.2. If Processed does not contain the performer
 - 4.1.1.2.1. Put the performer in Processed
 - 4.1.1.2.2. Get for Procedural1 the corresponding *App execution* as Execution
 - 4.1.1.2.3. Get the orchestration which contains Execution
 - 4.1.1.2.4. Add an *Entry* to the orchestration
 - 4.1.1.2.5. Create a *followed by* relation from the *Entry* to Execution

When creating a relation between two elements, creating duplicates should be avoided (i.e. don't create a relation of a certain type between the source and the target if another relation of the same type, with the same source and the same target already exists). Also avoid creating relations where the source and the target are the same element. In general the 4 main steps from above can be simplified as:

1. Create empty orchestrations
2. Put *App executions* in orchestrations and store mappings
3. Connect *App executions* in orchestrations based on process
4. Add possible *Entry* elements based on process and connect them

This approach creates one orchestration for each distinct performer that is used in a process. If one complete orchestration independent of the performers should be created, then instead of performing step 1, simply create one orchestration and use it for every performer. Also step 4 can then be simplified to look on a path until the first element that is contained in Mapping is found. Any step that uses "Follow the path(s)" is meant to find the next element in the process sequence that has some information that is necessary. This can lead to several elements, since control elements (e.g. *Decisions*, *Parallelisms* etc.) can be along the path. It can be realised through a depth-first search with the corresponding termination rules (in step 4.1 a branch should continue until it can't go no further or a loop is detected). The requirement for allowing only one resource can be removed, by sequencing the assigned resources of one *Action* in an arbitrary but steady order¹² and correctly connecting them to the predecessors and successors of the *Action*.

An alternative approach based on transformation rules can be taken from the description in the Appendix of D3.1.1. It has however different requirements and covers less than the approach presented above.

¹² "Arbitrary but steady order" means that the order is generally not important, but creating it twice for the same *Action* it should be both times the same. This can be achieved for example by sorting the resources based on their internal id.

3.3.5 Recommended approach for Gathering access requirements

This functionality supports the user with the creation of permission rules. The here presented approach is a very simple one, where the access requirements are simply copied as permission rules. The data structure that should be used as well as any details on the user interface and presentation of inputs and outputs are up to the implementer. It does however pose some requirements that must be fulfilled:

- The general structure of the described access requirements and permission rules is the same:
 - They should both cover the subject, the action and the resource.
- A *requires* relation has at most one *for subject* relation.
- Access to the permission rules, the *requires* relations, the participant structure and the action type structure (i.e. the necessary model data) is available.

The approach here assumes that the input is the process and its access requirements. The procedure for this is as follows:

1. Get all the *requires* relations that have a source in the selected procedural model.
2. For each such relations referred to as Requirement
 - 2.1. If Requirement is not already covered by a rule
 - 2.1.1. Get the assigned subjects of Requirement as Requirement Performers
 - 2.1.2. Get the assigned action types of the Requirement as Requirement Actions
 - 2.1.3. Get the target of Requirement as Resource
 - 2.1.4. For each performer in Requirement Performers as Subject
 - 2.1.4.1. Create a new permission rule for Resource where the subject is Subject and the actions are all of Requirement Actions
 - 2.1.4.2. If possible mark the created permission rule as “to be revised”

In order to check if a requirement is covered by a permission rule, two approaches are possible:

1. Use the basis for relation to see if a rule already is based on a requirement
2. Perform a check if a rule exists that already covers the requirement. This can be performed by following the steps 2.1 to 2.6 from section 3.3.6.

If neither of those approaches is feasible, then consider that all requirements are not covered by any rule. For step 2.1.1, the subjects of a *requires* relation can be determined either through the *for subject* relation or if one is not available then the performers assigned to the source of the *requires* relation should be used.

After the permission rules have been created based on the access requirements, a cleanup can be performed. Such a cleanup should look if any of the permission rules is already covered by another one. This check can look similar to the steps 2.1 to 2.6 from section 3.3.6. However, note that it has to be performed in both directions (i.e. check the first rule against the second and vice-versa).

3.3.6 Recommended approach for Access requirement coverage check

This functionality focuses on finding and checking access requirements against the available permission rules. The description here focuses on how this can be achieved and what has to be watched out for. The data structure that should be used as well as any details on the user interface and presentation of inputs and outputs are up to the implementer. The here presented approach however poses some requirements that must be fulfilled:

- The general structure of the described access requirements and permission rules is the same:
 - They should both cover the subject, the action and the resource.
- A *requires* relation has at most one *for subject* relation.
- Access to the permission rules, the *requires* relations, the participant structure and the action type structure (i.e. the necessary model data) is available.

The approach considers being performed for one procedural model, it can however be repeated several times if the check should be performed for several such models. It returns a list of *requires* relations, which describe the access requirements, that do not fit into any of the available permission rules.

1. Get all the *requires* relations that have a source in the selected procedural model.
2. For each such relations referred to as Requirement
 - 2.1. Get the assigned subjects of Requirement
 - 2.2. Gather all the types and super-types of the performers (using *instance of* and *specialisation of* relations) and store them in Requirement Subjects
 - 2.3. Get the assigned action types of Requirement
 - 2.4. Gather all the types and super-types of the action (using *instance of* and *specialisation of* relations) and store them in Requirement Actions
 - 2.5. Get the assigned target (the resource) of the Requirement
 - 2.6. For each permission rule of the target as Rule
 - 2.6.1. If the subject of Rule is contained in Requirement Subjects and the action of Rule is contained in Requirement Actions
 - 2.6.1.1. Note that the access requirement is fulfilled
 - 2.6.1.2. Break out of the 2.6 for loop
 - 2.7. If the access requirement is not noted as fulfilled
 - 2.7.1. Add it to the result
3. Return the result

With this approach, if more than one performer is assigned to an *Action*, it is enough if one of the assigned performers has access to the required resource. For step 4.1, the subjects of a *requires* relation can be determined either through the *for subject* relation or if one is not available then the performers assigned to the source of the *requires* relation. The steps 2.2 and 2.4 both should return the templates (if an instance is used, then it is determined through the *instance of* relation) and further find all the more generic templates (by following the direction of the *specialisation of* relations). The latter part can be achieved by using a depth-first or breadth-first search. The result contains *requires* relations, because they cover the most interesting information of access requirements that are not fulfilled. Their source indicates during which *Procedural element* the access requirement is not fulfilled and their target indicates what resource cannot be properly accessed.

3.3.7 Recommended approach for Calculation of KPIs/Variables

This functionality deals with calculating values for *KPIs* and *Variables* based on their description in the models. Most of the details are handled by the mathematical rules, however an additional description can be found in section 2.6.6 and Table 34. The data structure that should be used as well as any details on the user interface and presentation of inputs and outputs are up to the implementer. The only additional requirement is that a proper structure with functions that can be understood by the implementations is used and available. A possible approach to calculate a certain variable is to use a function that takes as an input an element (i.e. a *KPI*, *Variable* or *Constant*) either returns its value, if it is known or loaded from an outside source, or dynamically calculates its value based on the used operation (addition, subtraction, sum etc.) and the operands. The function itself should again be used to get the value of the operands, leading to a recursive calculation of the final value. To find out the level of a KPI the relations *achieves green if* and *achieves yellow if* should be used, which target is a *Variable*, that should represent a boolean value indicating if said level is achieved.

3.3.8 Recommended approach for Simulation of Procedural models

This functionality focuses on “running” described procedural models in order to simulate them. Therefore the approach will focus on how procedural models on the level of *Aspect-specific* concepts can be simulated. The data structure that should be used as well as any details on the user interface and presentation of inputs and outputs are up to the implementer. The here presented approach however poses some requirements on the process that must be fulfilled:

- Further descriptions of *Actions* through other processes are ignored during simulation. Only the selected *Process* is simulated.
- Only *Split-Control* elements can have multiple outgoing *followed by* relations.
- Only *Control* elements of AND-/XOR-inclusion type are allowed.
- Every *Split-Control* element of AND-inclusion type must have one corresponding *Merge-Control* element of AND-inclusion type. Also all paths out of the *AND-Split* must be led together in the same *AND-Merge*.
- All outgoing *followed by* relations of a *Split-Control* element of XOR-inclusion type must have a transition probability specified.

Note that the here described approach can be modified to also cover different cases. For example instead of choosing a path after a decision (*XOR-Split*) at random, a predefined list of choices can be used to utilise the simulation to aggregate certain vales on a specific path. Also different functions can be used for different attribute types. Such extensions are however left open to the implementer.

In this part the approach of walking through one simulation run is in the centre. For this a recursive function that takes as input a node (current node), the current path and the current probability is used. One additional variable (i.e. parallelism end) is outside of the function, because it is needed throughout different recursion runs.

1. Until the end is reached (the current node has no more outgoing *followed by* relations)
 - 1.1. Add the Current Node to the Path
 - 1.2. Add desired attributes and their values to the Path
 - 1.3. If the Current Node is an *XOR-Split*
 - 1.3.1. Choose one outgoing *followed by* relation (where the Current Node is the source of it) at random based on their transition probabilities
 - 1.3.2. Multiply the Path probability with the transition probability of the chosen relation
 - 1.3.3. Set the Current Node to the target of the chosen relation
 - 1.4. Else, if the Current Node is not an *AND-Split*
 - 1.4.1. Get the outgoing *followed by* relation (where the Current Node is the source of it)
 - 1.4.2. Set the Current Node to the target of the relation
 - 1.5. If the Current Node is an *AND-Merge*
 - 1.5.1. Set global Parallelism End to Current Node (so that the parent recursion knows where to continue)
 - 1.5.2. Return
 - 1.6. Else, if the Current Node is an *AND-Split*
 - 1.6.1. Add the Current Node to the Path
 - 1.6.2. For each outgoing followed by relation (where the Current Node is the source of it)
 - 1.6.2.1. Recursively call this function with: Current Node = the target of the relation, Path = Path and Probability = Probability
 - 1.6.3. Set the Current Node to global Parallelism End
2. Add the Current Node to the Path
3. Return the Path

The first time the function is started with the start node of the process, a probability of 1 and an empty path object. The result is the path object that describes what sequence has been taken during the simulation run, its probability based on the transition probabilities and the attribute values summed up along the path. Such a sum can indicate for example the total costs for a certain path or the total time that performers have to spend executing the path. Since the costs, wastes and durations are described through *influences* relations to *Values* that also contain a quantity, those relations with the quantity should be used in step 1.2 when adding the attributes and their values to the path.

This simulation function can be run several times to find different paths (depending on probabilities and number of runs), similar paths (according to sequence of elements) can then be merged and individual path results as well as aggregated results for the whole process can be provided. Possible results for a process based on several simulations are:

- The lowest value of a certain attribute in a path and the probability of that path
- The largest value of a certain attribute in a path and the probability of that path
- The average value of a certain attribute (based on path probabilities) and the standard deviation

3.3.9 Recommended approach for Business model evaluation

This functionality is similar to the previously described simulation. However instead of simulating one “run” through a procedural model, it instead calculates the expected revenues and expenses of a *Business model*. The data structure that should be used as well as any details on the user interface and presentation of inputs and outputs are up to the implementer. The here presented approach focuses on describing how the revenues and expenses can be determined. However, it poses some requirements that must be fulfilled:

- Properly described Business model, with all the elements for a certain business case.
- No infinite loops are present in the Business model.

In order to evaluate the Business model, an algorithm similar to a depth-first search¹³ can be used, with the addition of keeping track and updating certain aspects that make out the result. The general idea is to start such a “search” from each *Start stimulus* (i.e. for each Start stimulus a “search” is performed) and continue it until no further processing is possible. Additionally, loops should be explored again instead of stopping a branch. It should however stop searching a branch when the occurrences reach a value of 0. During such a “search” the implementation should keep track of which *exchanges value with* relations are passed as well as how often based on the occurrences. The formulas on how to calculate the occurrences after a *Split-Dependency control* have been presented in Table 20. However, when encountering a *Merge-Dependency control* a different approach should be taken:

- *Merge* with AND-inclusion type – Since it is a merge of several paths into one, it should be passed only once. This means that the first time it is encountered, the exploration of the branch should continue, with the occurrences adapted based on the fractions (see Table 20). If it is reached again then the exploration of the branch should be stopped. This can be achieved by marking the *Merge* elements once they have been passed the first time.
- *Merge* with XOR-inclusion type – Here the exploration of the branch should simply continue. The summation of all incoming occurrences is not necessary, because of the way a deep-first search is executed (i.e. each branching that leads to the Merge will pass it anyway resulting in a summation at the end).

In the end the result of all the value exchanges with their occurrences can be aggregated based on the *Business entities* and their *Value interfaces*. It could be presented as a table for each *Business Entity* containing all the exchanged values with their amounts and valuations.

3.3.10 Recommended approach for Serialisation of models as Linked Data

This functionality can be considered an “Export” of data out of the tool. It can be achieved by either directly serialising into Linked Data or using an already available export format that contains all the necessary data and transform it. The here described approach omits the specific transformation details and focuses on recommending a vocabulary in order to achieve interoperability between different modelling tools and different application as well as to allow reuse of queries.

Table 39 presents this vocabulary and Table 40 provides a recommended mapping to the Linked Data concepts on both the metamodel and model level. Since the prototypical implementation uses attributes of type table to handle some cases (e.g. inter-model relations with attributes), but the concept does not consider tables (e.g. it considers “rows” to be objects), some special approaches for handling those are

¹³ Most importantly processing different branches and backtracking once a dead-end has been reached in a branch.

described based on different cases. The details on how to handle those tables can be found in Table 40 and the following cases and possible naming conventions to automatically recognise them¹⁴ have been identified:

1. *Direct transformation into triples* – This is the case with the *Property collector*, where the table denotes triples that should be serialised as such. In such a case each individual row represents a different triple. (Naming convention: call the attribute “Property collector” and use the column names “Subject”, “Predicate” and “Object” to identify the parts of a triple.)
2. *Inter-model reference with attributes* – Some implementations do not allow having attributes on inter-model relations. A workaround for this is to create a table where a row represents a relation and one of the columns denotes the target of the relation. The table represents the type of relation and the other columns represent the attributes attached to it. (Naming convention: end the attribute name with “(iref)” and the column containing the target “cv:to”)
3. *Primitive attributes with cardinality bigger than 1 (unordered)* – In some cases several primitive values (strings, numbers etc.) can be provided for the same object. Tables represent a way of depicting those, where the table represents the attribute and each row represents one primitive value for it. Such a table should only have one column, otherwise it would not be a primitive attribute. (Naming convention: end the attribute name with “(multiple uo)”)
4. *Primitive attributes with cardinality bigger than 1 (ordered)* – Same as case 3, with the difference that the order of the values (and therefore the rows) is relevant. (Naming convention: end the attribute name with “(multiple o)”)
5. *Simplification for omitting modelling objects (unordered)* – Sometimes it is easier or more user friendly to model objects similar to attributes instead of objects on the modelling canvas. In such a case a table can be used, where the table represents the relation towards the “hidden” objects and each row represents an object. This also means that such objects cannot exist without the relation and the object containing the table. The columns represent the attributes that the (row) object can have. (Naming convention: end the attribute name with “(object uo)”)
6. *Simplification for omitting modelling objects (ordered)* – Same as case 5, with the difference that the order of the relations towards the objects (and therefore the rows) is relevant. (Naming convention: end the attribute name with “(object o)”)

Specific Linked Data constructs

Some general meta constructs to be used as types. The here presented concepts should be considered to be on the meta²-model level. The cv: prefix should stand for “http://www.comvantage.eu/mm#”

Construct	Description
cv:Model	(The set of all possible models) A class containing models, meaning that a resource of this type represents a model. For example the model type <i>Business process</i> is a subclass of this.
cv:Modelling_object	(The set of all possible concept instances) A class containing elements used in models, meaning that resources of this type represent an instance/object. For example the concept class <i>Activity</i> is a subclass of this.
cv:Modelling_relation_a	(The set of all possible relation instances with attributes) A class containing relations which have properties (attributes), meaning that resources of this type represent a relation with attribute values. The resources should also use cv:from and cv:to to indicate the source and target of the relation. For

¹⁴ An alternative is to use special types in the Linked Data describing the metamodel to denote those cases. Such details are however left open to the specific implementations.

Specific Linked Data constructs	
Some general constructs to be used as types. The here presented concepts should be considered to be on the meta ² -model level. The cv: prefix should stand for “http://www.comvantage.eu/mm#”	
Construct	Description
	example the relation class <i>followed by</i> is a subclass of this, since it should have a property that allows stating additional conditions.
cv:modelling_relation_na	(The set of all possible relations without attributes) A class of properties containing relations without properties (attributes) between resources, meaning that properties of this type have no attributes. For example the <i>configuration of</i> relation, which does not have any properties, is an instance of this.
cv:attribute	(The set of all attributes) A class of properties containing concept properties (attributes), meaning that properties of this type represent an attribute and the object is its value. It is a subclass of rdf:Property. For example <i>Instructions of an Activity</i> is an instance of this.
cv:contains	A property stating that a thing (e.g. element, relation) is contained by a larger thing. It is the general <i>contains</i> relation from the decomposition.
cv:inseparable	A sub-property of cv:contains, depicting the inseparable type.
cv:separable	A sub-property of cv:contains, depicting the separable type.
cv:index	A concept to denote the index or order for certain relations.
cv:described_in	A property stating that additional information about a thing (e.g. element, relation) can be found in a different graph.
cv:from	A property providing the source of a relation with properties (i.e. of cv:Modelling_relation_a type). The subject is the relation and the object is the source. It is not possible to use rdf:domain to denote the source of a relation, because the domain is specified on the schema level.
cv:to	A property providing the target of a relation with properties (i.e. of cv:Modelling_relation_a type). The subject is the relation and the object is the target. It is not possible to use rdf:range to denote the target of a relation, because the range is specified on the schema level.

Table 39: Linked Data constructs recommended for the serialisation

Metamodel level	
Modelling Concept	Linked Data mapping
Any Model type is ...	<ul style="list-style-type: none"> • Instance of rdf:Class • Subclass of cv:Model
Any Object class is ...	<ul style="list-style-type: none"> • Instance of rdf:Class • Subclass of cv:Modelling_object
Any Relation class with attributes is ...	<ul style="list-style-type: none"> • Instance of rdf:Class • Subclass of cv:Modelling_relation_a

Any Relation class without attributes is ...	<ul style="list-style-type: none"> Instance of cv:modelling_relation_na (and rdf:Property)
Any (not table) Attribute is ...	<ul style="list-style-type: none"> Instance of cv:attribute (and rdf:Property)
Any case 1 Table attribute is ...	<ul style="list-style-type: none"> Nothing on this level
Any case 2 Table attribute is ...	<p>Table / Attribute:</p> <ul style="list-style-type: none"> Treat like: <i>Any Relation class with attributes</i> <p>Column (not denoting target):</p> <ul style="list-style-type: none"> Depends on column type. Treat like: <ul style="list-style-type: none"> <i>Any Relation class without attributes</i>, or <i>Any Attribute</i>
Any case 3 Table attribute is ...	<p>Table / Attribute:</p> <ul style="list-style-type: none"> Treat like: <i>Any Attribute</i>
Any case 4 Table attribute is ...	<p>Table / Attribute</p> <ul style="list-style-type: none"> Treat like: <i>Any Relation class with attributes</i>
Any case 5 Table attribute is ...	<p>Table / Attribute:</p> <ul style="list-style-type: none"> Treat like: <i>Any Relation class without attributes</i> <p>Table type (if available):</p> <ul style="list-style-type: none"> Treat like: <i>Any Object class</i> <p>Column:</p> <ul style="list-style-type: none"> Depends on column type. Treat like: <ul style="list-style-type: none"> <i>Relation class without attributes</i>, or <i>Any Attribute</i>
Any case 6 Table attribute is ...	<p>Table / Attribute:</p> <ul style="list-style-type: none"> Treat like: <i>Any Relation class with attributes</i> <p>Table type (if available):</p> <ul style="list-style-type: none"> Treat like: <i>Any Object class</i> <p>Column:</p> <ul style="list-style-type: none"> Depends on column type. Treat like: <ul style="list-style-type: none"> <i>Relation class without attributes</i>, or <i>Any Attribute</i>
Model level	
Note: “corresponding X” should be understood in context to the Metamodel level. For example when an instance of type “Activity” is transformed, then “the corresponding <i>Object type class</i> ” means the Linked Data concept created for the “Activity” object class (e.g. cvmm:Activity).	
Modelling Concept	Linked Data mapping
A Repository with objects is ...	<ul style="list-style-type: none"> Instance of cv:Model. An RDF-Graph (called model graph).
Any Model is ...	<ul style="list-style-type: none"> Instance of the corresponding <i>Model type class</i>. An RDF-Graph (called model graph).
Any Object is ...	<ul style="list-style-type: none"> Instance of the corresponding <i>Object type class</i> in every model graph where it is used.
Any Relation with attributes is ...	<ul style="list-style-type: none"> Instance of the corresponding <i>Relation type class</i> in every model graph where it is used. It also has two properties indicating the source and target using cv:from and cv:to.
Any Relation without attributes is ...	<ul style="list-style-type: none"> A triple where the subject is the source element and the object is the target element. The predicate should

	<p>use the corresponding <i>Relation type</i> property. If the two elements are in different models then the statement should also be in both model graphs. The cv:described_in property should be used to state in both graphs where the other element can be found.</p>
Any (not table) Attribute value is ...	<ul style="list-style-type: none"> The object of a triple where the subject is the element and the predicate is the corresponding <i>Attribute</i> property.
A row for a case 1 Table is ...	<ul style="list-style-type: none"> A triple where the subject, predicate and object are based on the corresponding columns. If any of those columns is empty, then the object the table is in should be used as its value. For example if the subject column is empty in one row of object X, then X should be used as the subject for that row.
A row for a case 2 Table is ...	<ul style="list-style-type: none"> Instance of the corresponding <i>Relation type</i> class in every model graph where it is used. It also has two properties indicating the source and target using cv:from and cv:to. The cv:from points towards the object the table is in, and the cv:to towards the target of the inter-model reference (based on column denoting target). Every other column (not denoting target) should be treated like an attribute or relation (depending on column type) of the relation instance.
A row for a case 3 Table is ...	<ul style="list-style-type: none"> The value in the column (should be only one) should be the object of a triple where the subject is the element containing the table and the predicate is the corresponding <i>Attribute</i> property.
A row for a case 4 Table is ...	<ul style="list-style-type: none"> Instance of the corresponding <i>Relation type</i> class in every model graph where it is used. It also has two properties indicating the source and target using cv:from and cv:to. The cv:from points towards the object the table is in, and the cv:to property should have the value in the column (should be only one) as the object. The cv:index property should be used to indicate the order, where the subject is the relation and the object is the order number.
A row for a case 5 Table is ...	<ul style="list-style-type: none"> Instance of the corresponding <i>Object type</i> class (based on table class, if available) and of cv:Modelling_object in every model graph where it is used. It should be considered to be part of the same model graph as the element containing the table. A triple where the subject is the element containing the table and the object is the object created (item above). The predicate should use the corresponding <i>Relation type</i> property (based on Table/Attribute). Every column should be treated like an attribute or relation (depending on column type) of the object created (two items above).

<p>A row for a case 6 Table is ...</p>	<ul style="list-style-type: none"> • Instance of the corresponding <i>Object type</i> class (based on table class, if available) and of cv:Modelling_object in every model graph where it is used. It should be considered to be part of the same model graph as the element containing the table. • An Instance of the corresponding <i>Relation type</i> class in every model graph where it is used (based on Table/Attribute). • The relation has two properties indicating the source and target using cv:from and cv:to. The cv:from points towards the object the table is in, and the cv:to property should have the object created (two items above) as the object. Also the cv:index property should be used to indicate the order, where the subject is the relation and the object is the order number. • Every column should be treated like an attribute or relation (depending on column type) of the object created (three items above).
---	--

Table 40: Recommended transformation of modelling objects into Linked Data

3.3.11 Recommended approach for Comparison of model serialisations in Linked Data

This functionality should be used to determine differences between two models by using their Linked Data serialisations. The here described approach focuses on how they can be compared. For this it considers a directed comparison of a source and a target. The here presented approach poses some requirements that must be fulfilled:

- The serialisation has to be in conform to the Resource Description Framework (RDF)
- Different elements (i.e. resources) use different identifiers (i.e. URI) in both serialisations, while the same elements in both the source and the target use the same identifier.

When those requirements are met a simple comparison of the triples of both serialisations can be achieved by:

1. Load all the triples from the source and the target into separate lists (source list and target list)
2. For each triple in the source list, if it is also contained in the target list
 - 2.1. Remove the entry from both the source and the target list

In the end the source list will contain all the triples that have to be removed and the target list contains all the triples that have to be added in order to change the source to the target. In other words, both lists represent the difference between the source and the target. This is the result of the functionality and can be presented in different ways (e.g. graphical visualisation of differences, using a readable RDF syntax, a tree view based on the resources used as subjects/object etc.).

4 OUTLOOK AND CONCLUSION

This document describes a refinement of the *ComVantage* modelling method conceptualisation taking input from the initial specification (D3.1.1), its initial adaptations (D6.2.1, D7.2.1, D8.2.1), and the experience with its initial implementation (D3.4.1, D3.5.1). The document presents a top-down ontological approach in specialising highly abstract concepts and partitioning them in order to be fit for a modelling language. This specification is aimed to be independent on the implementation, and implementation decisions must be further taken towards facilitating user experience tailored for a specific implementation platform.

Several adaptations can be noted from the approach of the initial specification (D3.1.1):

- A top-down approach has been taken for a superior semantic cohesion, compared to the bottom-up initial approach;
- The metamodel has been significantly changed with additions (KPI modelling), merges (the value structure covering now the variability in product structure, service structure or a mix of them) and substitutions (KPI quantified causality instead of the purely visual Fishbone diagram);
- The focus has been placed on model machine-readability and on the richness of information that become query-able in the models (through their RDF serialisation). Therefore, several purely visual model types have been removed or replaced with semantically richer ones (e.g. the Fishbone diagram, the task decomposition view);
- The simulation approach initially suggested (based on system dynamics) has been replaced with a discrete event approach, due to the requirement for process-centricity and process-driven evaluation;
- A more generic overall approach has been taken, compared to the initially adopted supply chain management context; further developments towards the supply chain modelling direction will be subject of adaptation work in the application area where this has a high relevance (D7.2.2).

Further work will be invested in the adaptation deliverables D6.2.2, D7.2.2, D8.2.2 in the direction of assimilating stronger domain specificity from the application areas of *ComVantage* and also to integrate late agile developments of the OMI prototype (mostly in the sense of app modelling). Modelling guidelines will be developed in D3.5.2, based on the implementation outcome (D3.4.2).

5 REFERENCES

- Bicheno, J. and Holweg, M. (2009) *The Lean Toolbox*, PICSIE Books.
- Gordijn, J. and Akkermans H. (2001) *E3-value: Design and Evaluation of e-Business Models*. IEEE Intelligent Systems, Vol. 16(4): 11-17
- Kang K, Cohen S, Hess J, Novak W, Peterson A (1990) *Feature-oriented domain analysis (FODA) feasibility study*, Software Engineering Institute, Technical Report CMU/SEI-90-TR-021
- Kern H., Hummel A., Kuhne S (2011) *Towards a Comparative Analysis of Meta-Metamodels*, The 11th Workshop on Domain-Specific Modelling, Portland, USA.
<http://www.dsmforum.org/events/DSM11/Papers/kern.pdf>
- Weske, M. (2007) *Business Process Management: Concepts, Languages, Architectures*, Springer
- Zachman, J.A. (1987) *A Framework for Information systems Architecture*, IBM Systems Journal, 26(3):276-293
- D3.1.1 Specification of Modelling Method Including Conceptualisation Outline (first iteration)
- D3.4.x Prototype of *ComVantage* modelling tool
- D3.5.x Guidelines for the Secure Collaboration Model
- D5.2.2 UI Presentation and Workflow Models
- D6.2.x Adaptation of Secure Information Model Concept (WP6)
- D7.2.x Adaptation of Secure Information Model Concept (WP7)
- D8.2.x Adaptation of Secure Information Model Concept (WP8)

6 APPENDIX

6.1 Example Query for Capability Matching

In the following query use the one BIND statement to provide the point of origin that has the required Capabilities (e.g. Role ...)
Depending on the types of capabilities, this query has to be adapted.

```

SELECT ?role ?reqCount ?perf (COUNT(?avaSkill) AS ?avaCount)
WHERE {
  # Count how many skills are required by ?role
  {
    SELECT ?role (COUNT(?reqSkill) AS ?reqCount)
    WHERE {
      # NOTE # Use this bind to specify for which role you are looking
      BIND (<http://test.org#Role-365603-XYZ> AS ?role)
      ?role cv:SkillsKnowledge ?reqList .
      ?reqList rdf:rest*/rdf:first ?reqSkill .
    }
    GROUP BY ?role
  }

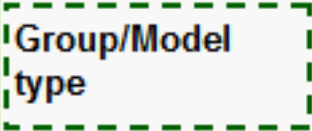
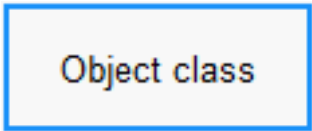
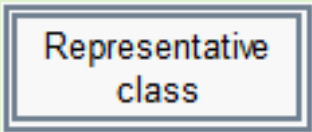
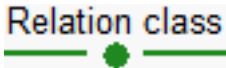
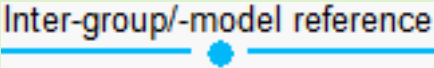

  # Here find out what specific skills ?role requires
  ?role cv:SkillsKnowledge ?reqList .
  ?reqList rdf:rest*/rdf:first ?reqSkill .
  ?reqSkill cv:Type ?reqSkillTyp .
  ?reqSkill cv:Label ?reqSkillLab .
  ?reqSkill cv:Level ?reqSkillLev .

  # Here the skills of the performers are determined. Only the ones that fulfil
  the required skill are considered "available"
  ?perf rdf:type cv:Performer .
  ?perf cv:SkillsKnowledge ?avaList .
  ?avaList rdf:rest*/rdf:first ?avaSkill .
  ?avaSkill cv:Type ?avaSkillTyp .
  ?avaSkill cv:Label ?avaSkillLab .
  ?avaSkill cv:Level ?avaSkillLev .
  # This is the part that compares the capabilities.
  FILTER(?avaSkillTyp = ?reqSkillTyp)
  FILTER(?avaSkillLab = ?reqSkillLab)
  FILTER(?avaSkillLev >= ?reqSkillLev)
}
GROUP BY ?perf ?role ?reqCount
HAVING (COUNT(?avaSkill) = ?reqCount)
# Only groups where the count of "availalbe" skills is the same as the count of
required

```

6.2 Metamodel diagrams

The here presented metamodels are a visual supplement and should not be seen as a replacement for the descriptions in the other chapters.

Representation	Description
	It represents a group (on conceptual level, ends with the word “group”) or model type (in the implementation recommendation). Models themselves can be seen as objects with attributes and participate in relations. The things it contains should be available in the group or be available to model in a model type.
	It represents the typical concept. If the border is drawn with a dotted line, then the class is there for readability reasons (e.g. to prevent relation lines going from one end of the model to the other etc.). In the hierarchy descriptions, if it contains a number and has a dashed line then it means that it is also used in a different place of the hierarchy (i.e. it is the sub-type of multiple other concepts).
	This represents a set of classes. It should be understood by its name, i.e. “Procedural element” means any element that can be used in the <i>Procedural Aspect</i> , “Liable entity” means any element that is also a liable entity etc.
	A normal relation between two concepts. The lines indicate what sources and targets are allowed.
	It is a relation that spans through different groups/model types.
	This represents generalisation or in another word inheritance. It is used to indicate sub- and super-types.

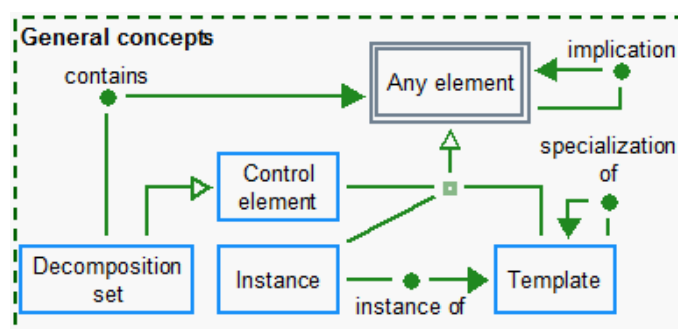


Figure 27: Metamodel for high abstraction level (General concepts)



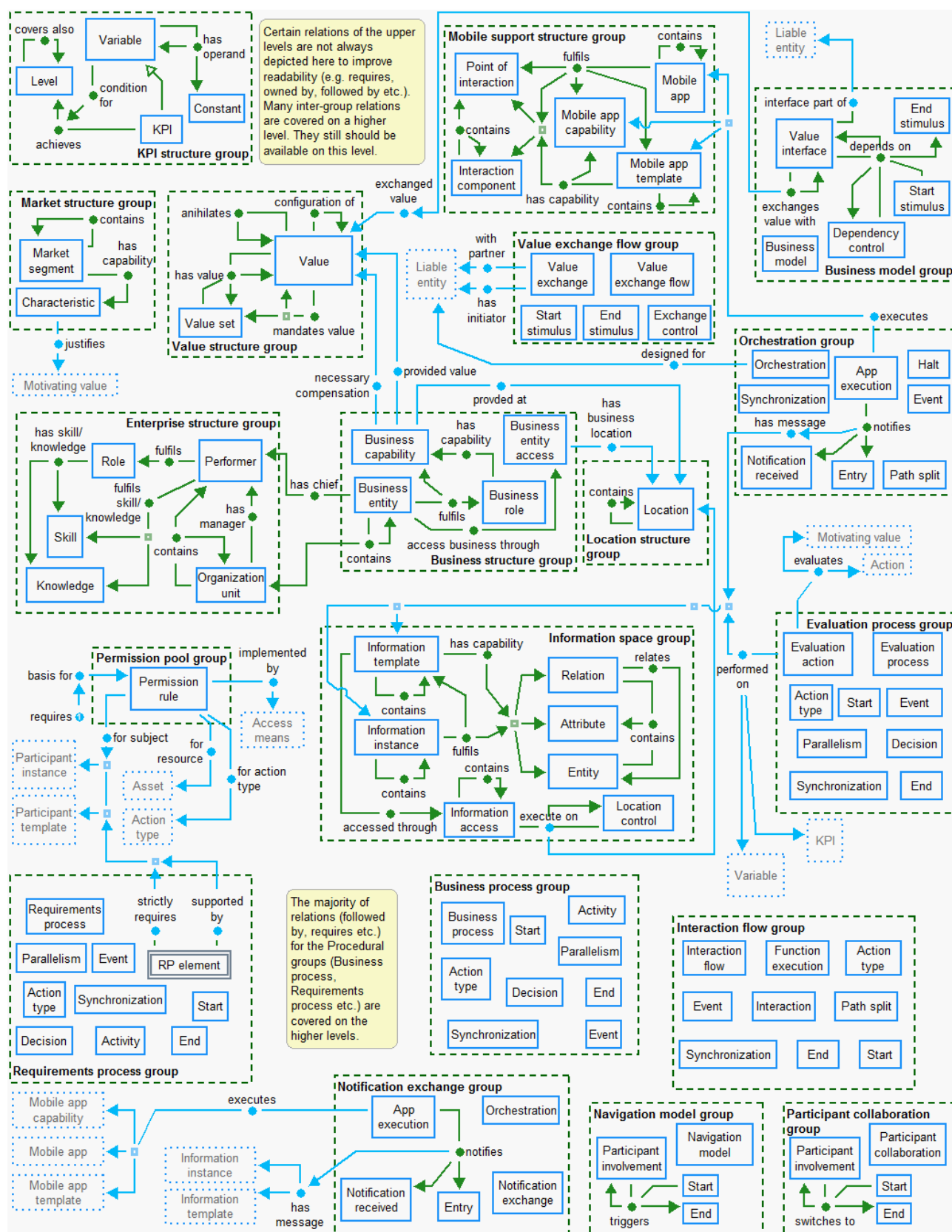


Figure 29: Metamodel for low abstraction level (Scope-specific concepts)

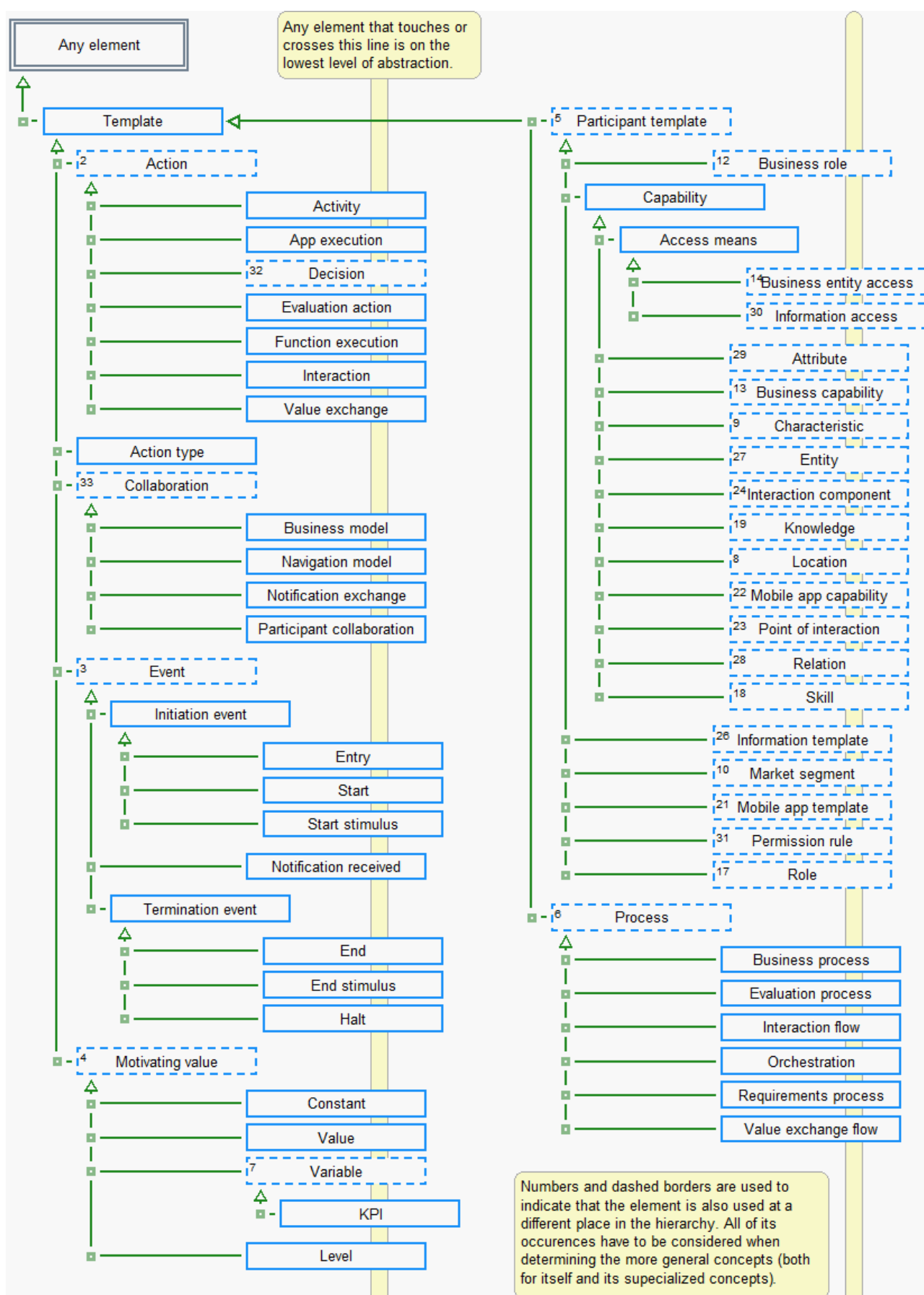


Figure 30: Metamodel concept hierarchy (Templates)

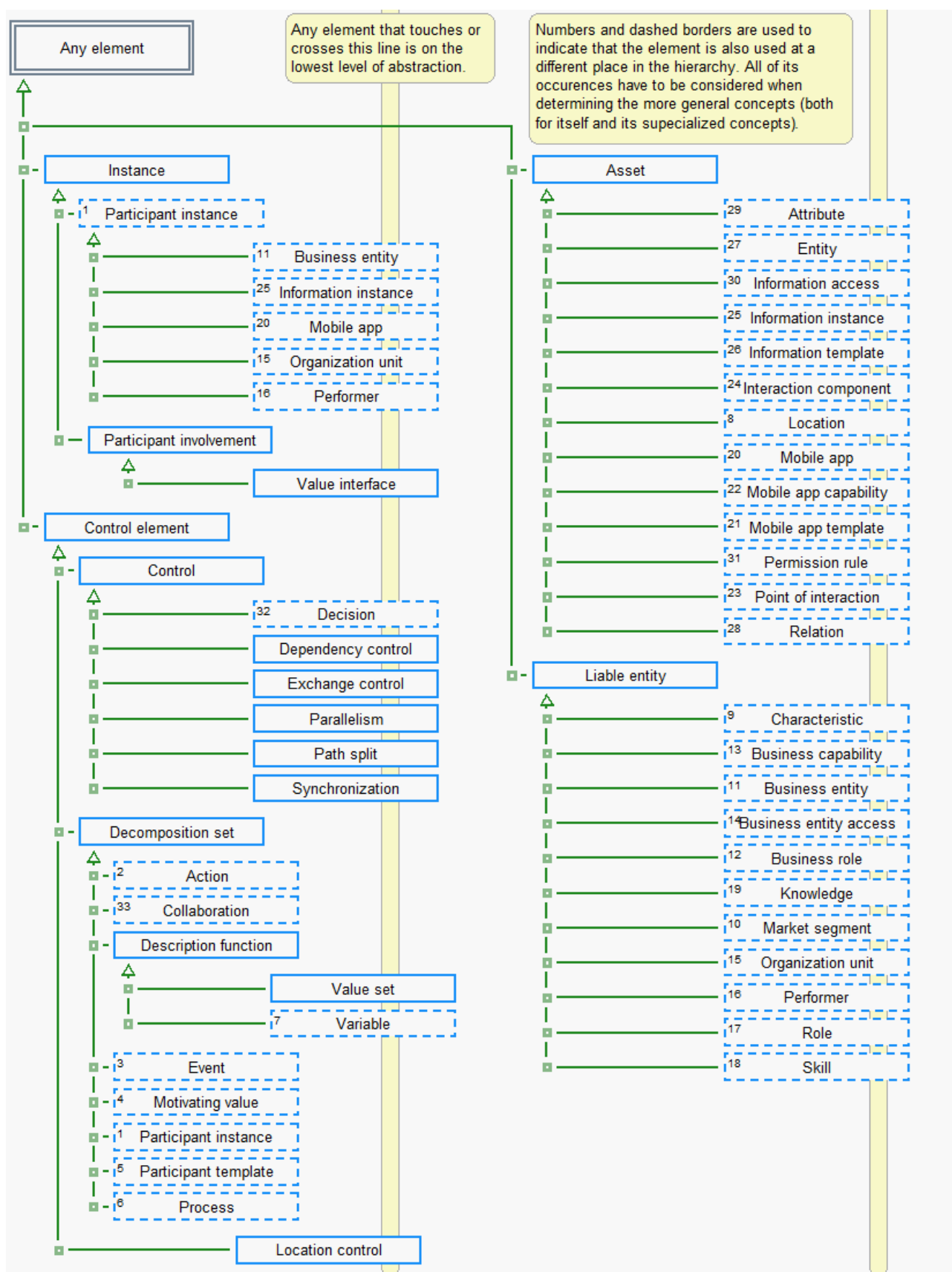


Figure 31: Metamodel concept hierarchy (other)

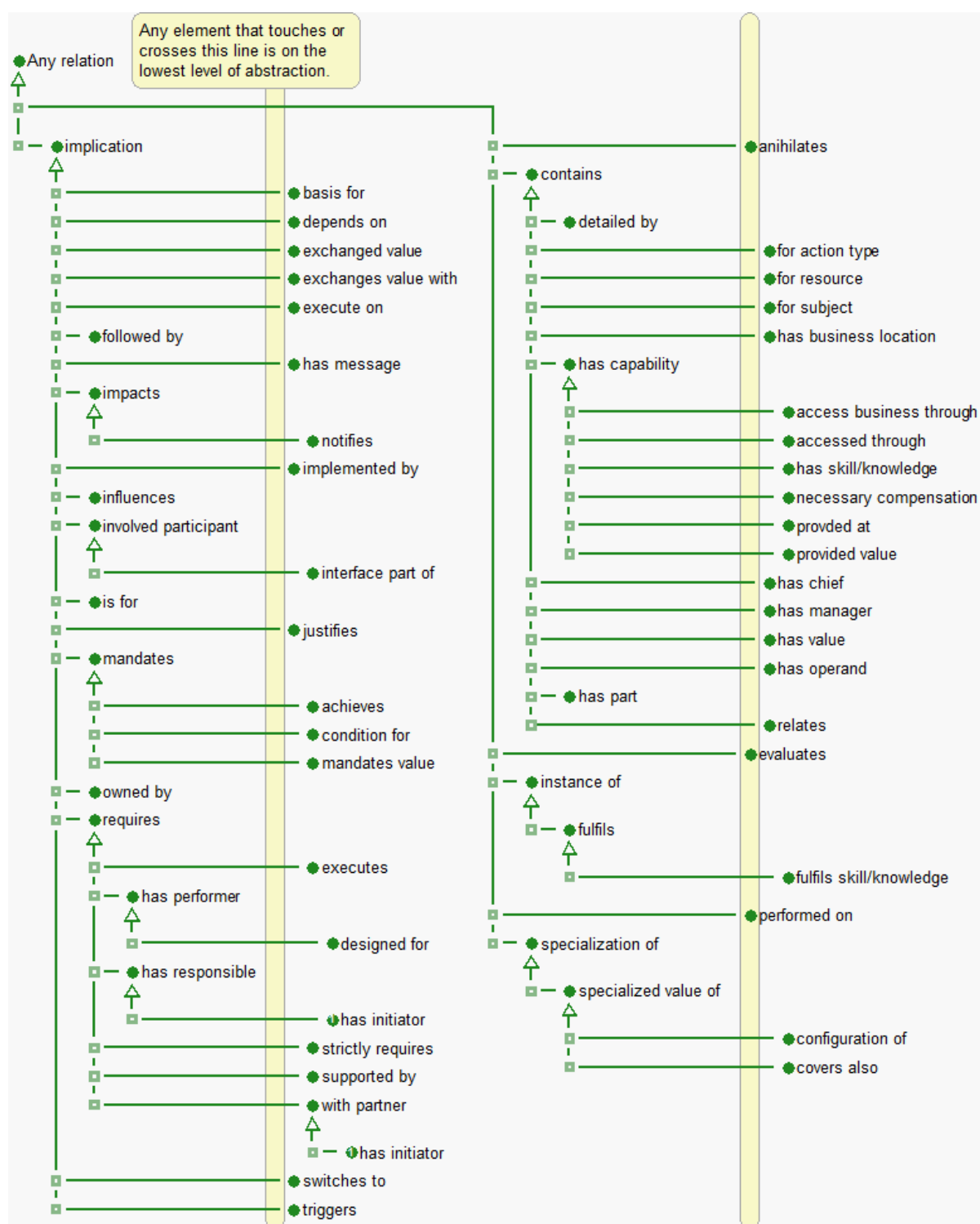


Figure 32: Metamodel relation hierarchy



DISCLAIMER

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law.

Copyright 2014 by SAP AG, Asociación de Empresas Tecnológicas Innovalia, Ben-Gurion University of the Negev, BOC Business Objectives Consulting S.L.U, Comau S.p.A., Technische Universität Dresden, Dresscode 21 GmbH, Evidian S.A., ISN Innovation Service Network d.o.o., Kölsch & Altmann GmbH, Nextel S.A., RST Industrie Automation GmbH, University of Vienna.