ENDORSE

Legal Technical Framework for Privacy Preserving Data Management

ENDORSE

Deliverable D3.1

Functional Architecture

Editor:	Stefania Marrara, Pierfranco Ferronato, Fabrizio Gheller
Deliverable nature:	Report
Dissemination level: (Confidentiality)	Public
Contractual delivery date:	M12
Actual delivery date:	
Suggested readers:	Consortium
Version:	00.02.00
Total number of pages:	39
Keywords:	Functional Requirements, Non-functional Requirements, Functional Architecture, Use Cases

Abstract

This document is the outcome of the work carried out in the T3.1 Functional Requirements of the project Endorse. This deliverable is divided into two main parts: the platform requirements that consists in functional and non functional, and the functional architecture itself.

This deliverable is an important step in the development of ENDORSE Functional Architecture which is object of T3.1 Functional Requirements which does not end at the time of the delivery of this work but will continue until April 2012.

Therefore the aim of this work is to define the ENDORSE Functional Architecture by identifying the main components that will be detailed in D3.3 Technical Architecture and the main functionalities that each component will be responsible of.

Disclaimer

Impressum

Legal Technical Framework for Privacy Preserving Data Management

ENDORSE

WP3.1 Functional Requirements

D3.1 Functional Architecture

Editor: Stefania Marrara, Soluta.Net

Work-package leader: Pierfranco Ferronato, Soluta.Net

Executive summary

This document is the outcome of the work carried out in the T3.1 Functional Requirements of the project Endorse. This deliverable strictly follows the deliverable D2.3 Trial Scenarios and is divided into two main parts: the functional and non functional platform requirements, and the functional architecture itself.

This deliverable is an important step in the development of ENDORSE Functional Architecture which is object of T3.1 Functional Requirements which does not end at the time of the delivery of this work but will continue until April 2012.

Therefore the aim of this work is to design the ENDORSE Functional architecture by identifying the main components that will be detailed in D3.3 Technical Architecture and the main functionalities that each component is responsible of.

The definition of the interfaces, that will be target of T3.1 Functional Requirements since September 2011 to April 2012 will be included as part of D3.3 Technical Architecture since no further deliverable is planned at the end of the task itself.

Regarding the core of ENDORSE, PRDL language and corresponding Rule engine, further work will be performed in both WP3 and WP4 and presented in other deliverables.

The development of the PRDL will result in the final specification of the language in D.3.4 which is due April, 2012. As the language should provide a usable tool; there has to be a lot of work done in implementing example rules and test them against requests. With the input of the legal partners in the ENDORSE consortium the language will be primed to face the challenge of representing data privacy statements accurately. Key will also be the editor that should enable the stakeholders to create the rules as easy and comfortable as possible.

As it was already stated in D3.2 the first step towards a working solution will be Drools which was presented before. The upcoming step will be a feasibility testing in order to proof that all the significant constructs needed for the full implementation of the PRDL can be concealed by Drools. In addition to that an evaluation will be carried out to identify and check other possible candidates regard the use as rule engine for the PRDL. The results of this test phase will be presented in D4.3.

List of authors

Participant	Author
Soluta.Net	Stefania Marrara
Soluta.Net	Pierfranco Ferronato
Soluta.Net	Fabrizio Gheller
TILT	Ronald Leenes
Create.Net	Giovanni Russello
Create.Net	Rizwan Asghar
SUAS	Thomas Kurz
WIT	Mark Mc Laughlin
WIT	Paul Malone

Table of Contents

List of Figures

1. Introduction

This document is the outcome of the work carried out in the T3.1 Functional Requirements of the project Endorse. This deliverable strictly follows the deliverable D2.3 Trial Scenarios and is divided into two main parts: the platform requirements that consists in the functional and non functional, and the functional architecture itself.

The second part of the document is the UML formalization of the functional specification of the project ENDORSE. The Functional Architecture is the architectural viewpoint concerned with the functional aspects of the system; that is, with the declarative and behavioural specification of a system that has to satisfy the functional requirements. Usually this viewpoint is completely agnostic with respect to the solutions of the problem, it deals with “what” the system will do, rather then “how” the system will act to provide the required functionalities.

The Use Case modelling has been used to define the interactions between users and system. The set of Use Cases has been aggregated in diagrams and these in turn into packages in order to represent the different components that will realize them. The main contributes to the functional architecture and described in this work are:

1. the functional and non-functional requirements
2. the behavioural model described in Chapter 5.

1.1. The ENDORSE approach

In the last few years privacy and data protection have become one of the new challenges of European Companies due to the increased consciousness of the target customers. End-users, indeed, require assurance that their personal data are fairly and correctly gathered and managed only for the purposed for which they have given their consent. Therefore Companies managing personal data are required to ensure that the policies they adopt to handle these data are in compliance with the legal requirements and not available to misuse by their employees or third parties. But, as a matter of fact, data protection activities introduce an overhead, both financial and operational, which can be result to be expensive to implement especially for SMEs.

The idea at the core of ENDORSE is to create an open source environment able to support SME, large organizations and end-users in having all personal data management policies compliant with both National and European legislation.

Following the guidelines of the Model Driven Architecture, the functional architecture of ENDORSE has been designed with the aim to be completely independent from the technological solution that will be adopted during the development phase. Thanks to this paradigm, the functional architecture described in this deliverable will just focus on the application's business functionality and behavior, leaving any technical decision to D3.3 Technical Architecture and to WP4 Implementation.

1.2. The Functional Requirements extraction process

In the last decade the need to achieve high quality in products under stringent resources and budgets, has raised the need of a careful elicitation of requirements. Requirements elicitation and analysis are one of the most crucial steps in products design and development process. Bad requirements are may cause the resulting products development to fail and they are the most difficult phase to rectify in a late stage of the product development process. In any case elicitation and analysis of requirements are much more difficult than they appear to be.

Unfortunately, most of the software developers assume that what customers give as their requirements are the effective requirements of the product, but even if customers do try their best to communicate, since they

are not specialists, their best efforts without guidance may produce only a shadow of what the true requirements are. From the point of view of designers, they can design and implement products which satisfy the customer needs only after the true customer requirements are caught. Besides, the requirements need to be classified and reorganized so that designers can clearly see the focus of the customer requirements. Designers also need to prioritize requirements so that they can make decisions on which requirements to fulfil when different requirements cannot be fulfilled at the same time. In order to envision a complete and coherent set of functional and non-functional requirements for Endorse, the project methodology has followed the guidelines below:

- creation of a systematic process based on use-cases extraction;
- creation of a complete set by mapping requirements and scenarios presented in D2.3;
- functional and non-functional requirements have been classified and priorized in agreement with the two user Companies Europe Assistance and Seecomms;
- the communication barrier between analysts and user Companies have been overcome by face-to-face meeting, con-calls and document sharing using Enterprise Architect diagrams.

The full process is detailed in the following section after a brief state of the art of the main approaches available in literature that have been taken into account in the process of tailoring an ad-hoc requirements elicitation process for Endorse.

2. State of the Art review

In this section we briefly summarize some approaches that were useful in the design process of the requirements extraction process of Endorse.

Part of the following review has been summarized from the state of the art presented in the "Deliverable 2.2 Scenario Description, Use Cases and Technical Requirements specification" of the FP7 EU project SOCIETIES.

2.1. FURPS and FURPS+ classifications of requirements

FURPS[FRP], and later FURPS+ [FRP+], provides both a way of categorizing software requirements and a quality model that reflects the quality of various characteristics of the software product. FURPS uses the following classification system for capturing requirements from:

1. Functionality: such as feature sets, capabilities, and security.
2. Usability: such as human factors, aesthetics, consistency in the user interface, online and context sensitive help, wizards and agents, user documentation, and training materials.
3. Reliability: such as frequency and severity of failure, recoverability, predictability, accuracy, and mean time between failures (MTBF).
4. Performance: it imposes conditions on functional requirements such as speed, efficiency, availability, accuracy, throughput, response time, recovery time, and resource usage.
5. Supportability: such as testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, installability, and localizability.

Additional categories were added to FURPS by IBM [FRPIBM], which altogether comprise FURPS+. These additional categories are Design requirements, Implementation requirements, Interface requirements and Physical requirements. A comparative analysis of FURPS with other quality models (McCall, Boehm, Dromey and ISO 9126) is given in [BOEM].

2.2. Requirements and Use-cases with the Unified Process

The Unified Software Development Process is an incremental and iterative process, split into 4 phases, Inception, Elaboration, Construction and Transition [RUP].

In the unified process, Use-case are created in order to define the functional requirements. These use-cases focus on the user interactions with the system and external system. They treat the system as a black box and do not elaborate on technology used, look and feel, or other design or implementation issues.

The Unified process uses well defined milestones to define the end of each phase, helping to ensure that the process is clearly progressing towards its goals without overlooking any major prerequisite tasks.

As the Unified process is an iterative and incremental process, not all requirements need to be defined before implementation can proceed. In later iterations the use-case can be extended and elaborated with further requirements. The use-cases should be 10-20% complete in the Inception phase and 80% complete in Elaboration, according to the Rational best practices guidelines.

An iterative approach has lower risk of complete failure than a simple waterfall model, where all requirements are defined at the beginning. An incremental approach is more flexible, accommodating new discoveries or ideas that frequently arise during the development of prototypes and demonstrations.

2.3. Systems Engineering Management and Requirements

Systems engineering processes apply systems engineering techniques to the development of any systems. Systems engineering processes are related to the stages in a system life cycle. In System Engineering the definition of requirements is one of the most important steps of the life cycle, since a big change in a late stage of the system design or development can affect completely the entire work done so far. Each requirement should be periodically examined for validity, consistency, desirability and attainability. Once the system design has been developed, a requirements analysis is the first necessary step to provide the functional design. This functional design is a description of the product aimed to be input of the functional architecture. This architecture describes the behaviour of the system-to-be and of its main components. The most popular methodology used in requirements engineering is the use-case approach. Using this method all stakeholders, also non computer skilled, are able to understand and interact with the descriptions of the operational action sequences than will be part of the target system. At the same time, scenarios are useful for determining and validating requirements, and for making them concrete, agreed-on, and consistent.

2.4. Identification of Actors, stakeholders, non-functional organizational and trust-related requirements

i* modelling framework

The i* modelling framework [Yu98] includes the concepts of social and organisational actors, goals and actor dependencies, useful to model both software systems and organizational settings.

Tropos methodology

Tropos [Castro00] is an agent-oriented methodology which uses extended i* notation with actor, goal, softgoal, resource, and dependency as basic modelling constructs. Actor diagrams and rationale diagrams are used to model the organizational environment. Actor diagrams show actor dependencies, discovered analysing goals. Rationale diagrams describe which goals of a specific actor are analysed and their dependencies with other actors. These dependencies define functional and non-functional requirements for the system-to-be.

AUML

AUML (Agent UML) [Odell00], is aimed at modelling agents, their internal behaviour and interactions within an organization. AUML can e described as an agent-oriented UML. It includes agent-oriented sequence diagrams and agent class diagrams. Sequence diagrams in multi-agent systems express the exchange of messages through protocols, while agent class diagrams are used to illustrate the static design view of a system, showing a set of classes, interfaces and collaborations and their relationships.

KAOS Language and Methodology

KAOS [BaBe03] is a goal-oriented approach to model functional and non-functional system requirements. KAOS methodology provides a specification language for determining actors and behaviours and goal-driven elaboration. The language provides a rich ontology for capturing requirements in terms of goals, constraints, objects, actions, agents, events, etc. Links between requirements are represented as well to capture refinements, conflicts, responsibility assignments, etc.

The method roughly consists of the following steps: identifying and refining goals until they are assignable each agent; identifying objects and actions progressively from goals; deriving requirements on the objects and actions to meet the goals; and assigning the goals, objects and actions to the agents composing the system.

2.5. Capturing non-functional security and dependability related requirements

Abuse Cases

McDermott and Fox [DeFox99] propose an object-oriented modelling technique which make use of use-cases to capture and analyse security requirements. In this approach the concept of abuse case is presented.

An abuse case is a description of an interaction between the system and one or more actors, where the result of such interaction is harmful to the system or one of the actors in the system. Essentially, the focus is on analysing how an internal malicious actor can cause harm to the system or actors of the system. Using this approach security requirements are modelled separately from functional requirements. The drawback of this approach is that it is not easy to verify the consistence between functional and security requirements represented by an actor in a use-case diagram.

Misuse Case

The approach based on the concept of *misuse case* [PaXu05] captures security requirements by defining a misuse case as the inverse of a use-case. This concept allows to represent actions that the system should prevent together with those actions which it should support. In this approach, use and misuse are depicted in the same diagram where use and misuse are closely connected.

Anti-goals

The KAOS methodology [BaBe03] is composed by a specification language, an elaboration method, and a tool support for requirements discovery. The extension of KAOS framework provided by Darimont and Van Lamsweerde introduces a way to define a set of undesirable behaviours for the system. Goal obstruction definition yields sufficient obstacles for the goal to be violated. The prevention of such obstacles yields necessary preconditions for achieving a target goal. Obstacles analysis consists in analysing all possible issues that can prevent a goal fulfilment. It aims at identifying as many ways of breaking goals as possible in order to resolve each such situation. The framework offers different resolution techniques, namely goal substitution, agent substitution, goal weakening, goal restoration, obstacle prevention and obstacle mitigation.

3. ENDORSE Methodology

Part of the work carried out by Task 3.1 Functional Requirements was to evaluate the most important state of the art methodology approaches to drive the construction of a requirements elicitation process suitable for Endorse. The approaches reviewed are presented in Section 2.

Based on this evaluation and on the experience made by some of the partners in another EU project, SOCIETIES, the final decision was to classify requirements following an approach similar to the one suggested by the FURPS model, which has been briefly described in Section 2.

The process of collecting requirements was based on a 5 stage process to collect and specify both functional and technical requirements. The entire process was the result of a strict cooperation between WP2, Requirements, WP3.1, Functional Requirements and WP 3.2 Technical Architecture.

The stages of the requirements elicitation process are:

1. Scenario brainstorming;
2. Gathering of initial requirements;
3. Scenario evaluation, analysis, ranking, filtering and refinement;
4. Refinement of functional and non-functional requirements and extraction of use-cases;
5. Harmonisation, prioritisation and ranking of requirements;

In the following we provide a detailed description of the five steps listed above:

1. In the process of scenario brainstorming, WP2.4 partners produced several stories trying to capture and extend the features mentioned in the description of work (DoW). The scenarios were created on the basis of meetings and colloquia with people from Europe Assistance and Seecomms who tried to gather their desiderata w.r.t. the system-to-be in their companies. At this stage, the focus was to capture the behavioural aspects of the system and to identify the main stakeholders involved.
2. In the second stage, an initial set of functional and technical requirements was extracted from the scenarios produced in stage 1. This set of preliminary requirements was documented using a requirements description template. The main purpose of this set of requirements was to guide the further refinement, evaluation and selection of the initial scenarios.
3. In the third stage, the initial scenarios were evaluated, prioritised, and, subsequently, compiled into new refined scenarios following the feedbacks obtained by EuropAssistance and Seecomms. These scenarios were built so as to include multiple scenes and map all the companies desiderata features. Furthermore, the scenarios were harmonised to demonstrate the same level of detail. At the end of this stage, the refined set of scenarios was ranked, sorted and filtered. A selected set of final scenarios was eventually produced. The final set of scenarios were presented in Deliverable 2.3 Trail Scenarios specification.
4. In the fourth stage, the initial requirements collected in stage two were refined. Initially, the stage two requirements were homogenised, merged, eliminated, extended and classified to produce a first refined set of requirements. Then, the final set of scenarios delivered by WP2.4 was thoroughly studied to extract additional technical requirements. Both functional and non-functional requirements were identified at this stage. All requirements were mapped to specific segments of scenario scenes. Once all technical requirements were specified in detail, high level use-cases were extracted. Each extracted use-case was mapped to the requirement(s) it implements, as well as the actor(s) that can make use of it.
5. In the fifth and final stage the elicited requirements were prioritised. At this stage, the requirements were checked for consistency and any potential inconsistencies were harmonised.

The final set of functional requirements extracted is presented in Section 4.

The elicitation process is shown in Figure 1. In blue the part of the process that produced the set of scenarios presented in Deliverable 2.3, in yellow the part included into this work.

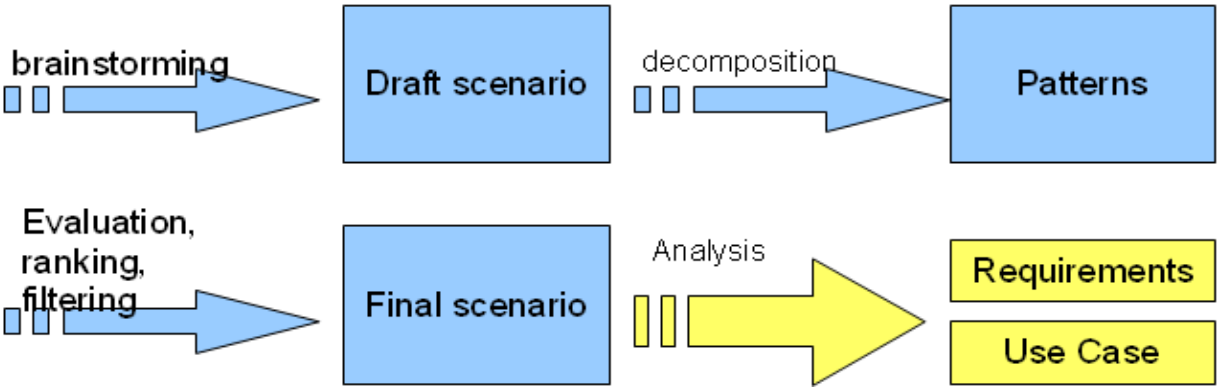

Figure 1: The requirements gathering process in ENDORSE

4. Endorse Requirements

The ENDORSE requirements will be defined by creating scenarios that were designed in deliverable 2.3. The initial scenarios make possible to define an initial set of requirements. These requirements were used to evaluate and to guide the revision of the initial scenarios that were included in the final version of Deliverable 2.3. The latest version of the scenarios allows to define the additional requirements. The revised requirements have been divided into functional and non-functional requirements and modelled into Enterprise Architect to guide the design of both Endorse Functional and Technical Architecture.

4.1. Functional Requirements

The diagram shown in Figure 2 presents the entire set of Functional Requirements modeled using Enterprise Architect. Functional Requirements describe the expected behavior of the Endorse Platform and the components that are included into.

Functional - *(Requirements diagram)*

Created By: The Administrator *on* 26/07/2011
Last Modified: 26/07/2011
Version: 1.0. *Locked:* False

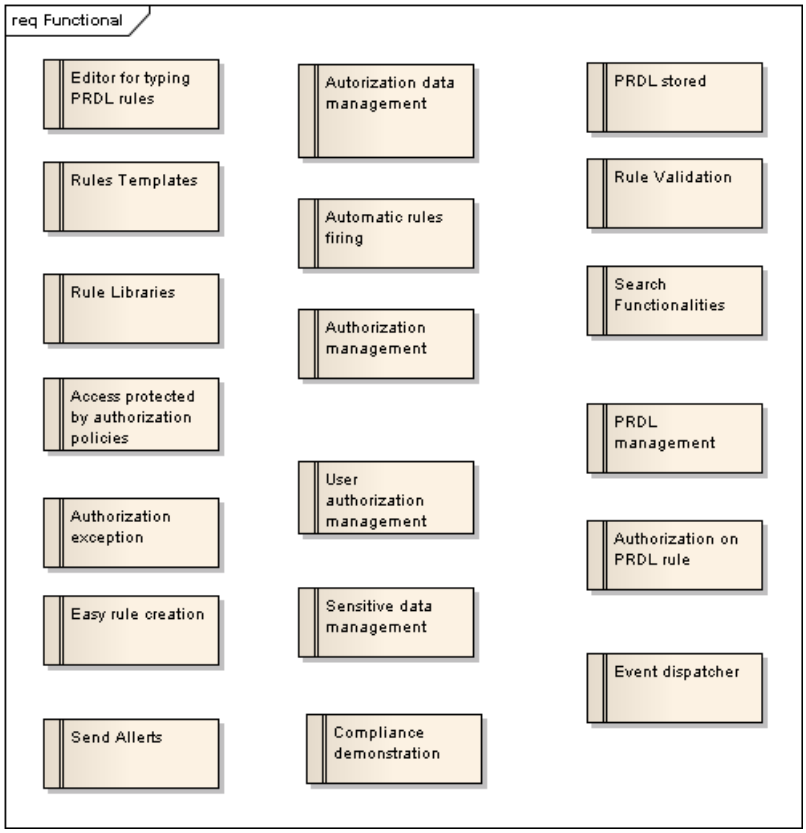

Figure 2: ENDORSE Functional Requirements set

The following list details the Endorse Functional requirements, they are the final outcome of the elicitation process detailed in Section 3.

1. *Req Title:* Access protected by authorization policies

   Type: **Requirement**
   Status: Proposed. Version 1.0. Phase 1.0.

Package: Functional
Description: Endorse access should be protected by authorization policies

2. *Req Title:*Authorization exception

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Functional
Description: Endorse should allow do add an exception for an user when granting an authorization to a group of users

3. *Req Title:*Authorization management

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Functional
Description: Endorse should provide and manage a list of authorization profiles

4. *Req Title:*Authorization on PRDL rule

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Functional
Description: Endorse should allow to manage groups of users in a PRDL rule

5. *Req Title:*Automatic rules firing

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Functional
Description: Endorse should be able to allow the setting of automatic firing of rules at a certain time or when an event occurs

6. *Req Title:*Easy rule creation

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Functional
Description: The Editor should provide drag&drop features in the creation of a rule

7. *Req Title:*Editor for typing PRDL rules

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Functional
Description: Endorse should contain an editor for typing in PRDL rules

8. *Req Title:* Event dispatcher

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Functional
Description: Endorse can order actions on back end systems or send alerts to an operator
regarding actions required on back end systems when a rule fires

9. *Req Title:* PRDL management

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Functional

Description: Endorse should provide functionality to write, update, delete and retrieve PRDL rules

10. *Req Title:* PRDL stored

    Type: **Requirement**
    Status: Proposed. Version 1.0. Phase 1.0.
    Package: Functional
    Description: Endorse should store PRDL rules

11. *Req Title:* Autorization data management

    Type: **Requirement**
    Status: Proposed. Version 1.0. Phase 1.0.
    Package: Functional
    Description: Endorse can allow/deny the access of certain data by a certain user/application/back end system

12. *Req Title:* Rule Libraries

    Type: **Requirement**
    Status: Proposed. Version 1.0. Phase 1.0.
    Package: Functional
    Description: Endorse should provide libraries of rules encoded in PRDL

13. *Req Title:* Rule Validation

    Type: **Requirement**
    Status: Proposed. Version 1.0. Phase 1.0.
    Package: Functional
    Description: Endorse should be able to validate a rule against a provided set of PRDL rules

14. *Req Title:* Rules Templates

    Type: **Requirement**
    Status: Proposed. Version 1.0. Phase 1.0.
    Package: Functional
    Description: The Endorse editor should provide templates for writing the rules

15. *Req Title:* Search Functionalities

    Type: **Requirement**
    Status: Proposed. Version 1.0. Phase 1.0.
    Package: Functional
    Description: Endorse should provide keyword based search functionality for the PRDL rules and return a list of matching results

16. *Req Title:* Send Alerts

    Type: **Requirement**
    Status: Proposed. Version 1.0. Phase 1.0.
    Package: Functional
    Description: Endorse can send alerts to an operator by email or pop up windows

17. *Req Title:* Sensitive data management

    Type: **Requirement**
    Status: Proposed. Version 1.0. Phase 1.0.
    Package: Functional
    Description: Endorse should include an ontology or a dictionary of the sensitive data the

company manages

18. *Req Title:* User authorization management

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Functional
Description: Endorse should be able to query and manage users consent documents

4.2. Non- Functional Requirements

The non-functional requirements describe constraints that apply to the technology, usually it deals with performance, security, SLA, technical platform (preferring PHP over Java per example) and others. The following set of requirements have been designed to drive the Technical Architecture that will be target of Deliverable 3.3.

The set is shown in the following Figure 3 while the details are presented in the list below.

Non-Functional - *(Requirements diagram)*

Created By: The Administrator *on* 26/07/2011
Last Modified: 26/07/2011

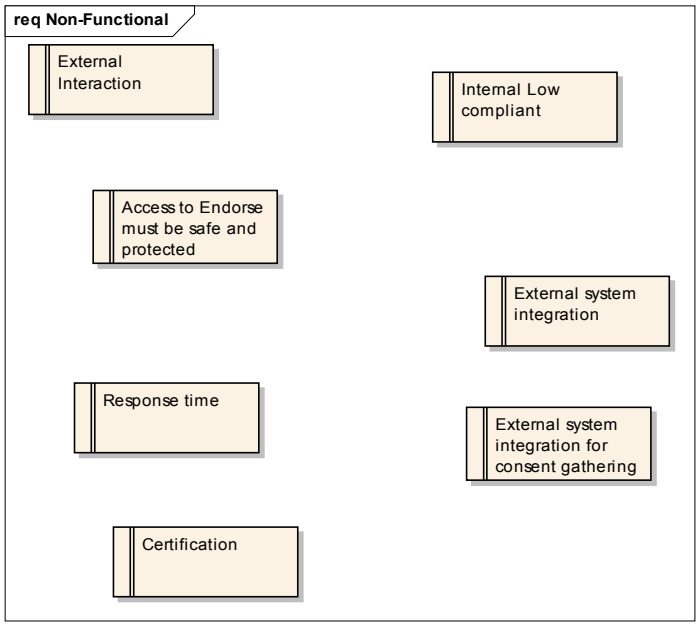

Figure 3: Non Functional Requirements set

1. *Req Title:*Access to Endorse must be safe and protected

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Non-Functional
Description: Access to Endorse must be safe and protected

2. *Req Title:*Certification

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Non-Functional
Description: Endorse must provide a certification mechanism

3. *Req Title:* External Interaction

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Non-Functional
Description: Endorse can interact with any number of Company backend systems, sending alerts and operation commands

4. *Req Title:* External system integration

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Non-Functional
Description: The interaction with back end systems must require the minimum development effort in the back end system themselves

5. *Req Title:* External system integration for consent gathering

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Non-Functional
Description: Endorse must interact with any consent gathering module adopted by the Company

6. *Req Title:* Internally Law compliant

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Non-Functional
Description: Endorse has to be safe, all rules internally written must be law compliant

7. *Req Title:* Response time

Type: **Requirement**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Non-Functional
Description: Endorse must work runtime or almost runtime

5. The ENDORSE Functional Architecture

This chapter provides the description of the main components of the ENDORSE functional architecture and provides a milestone for T3.1 Functional Requirements whose work will be completed in April 2012.

5.1. General Picture of the Architecture

The idea at the basis of ENDORSE is to provide a platform able to guarantee that a company will gather, manage and use the personal data it needs for its business in a way that is complainant with the law. For this reasons ENDORSE should be able to be integrated in the informative system of the Company in a non obtrusive way, ensuring that the normal activity of the back end systems will be not affected, but in case enhanced, by its introduction.

Following this idea the architecture of ENDORSE has been envisioned as a set of components loosely coupled that communicate among each other and with the back end systems through an Enterprise Service Bus (ESB) that also implements a publish and subscribe messaging system.

The use-case diagram shown in Figure 4 describes the main functionalities performed by the ENDORSE platform.

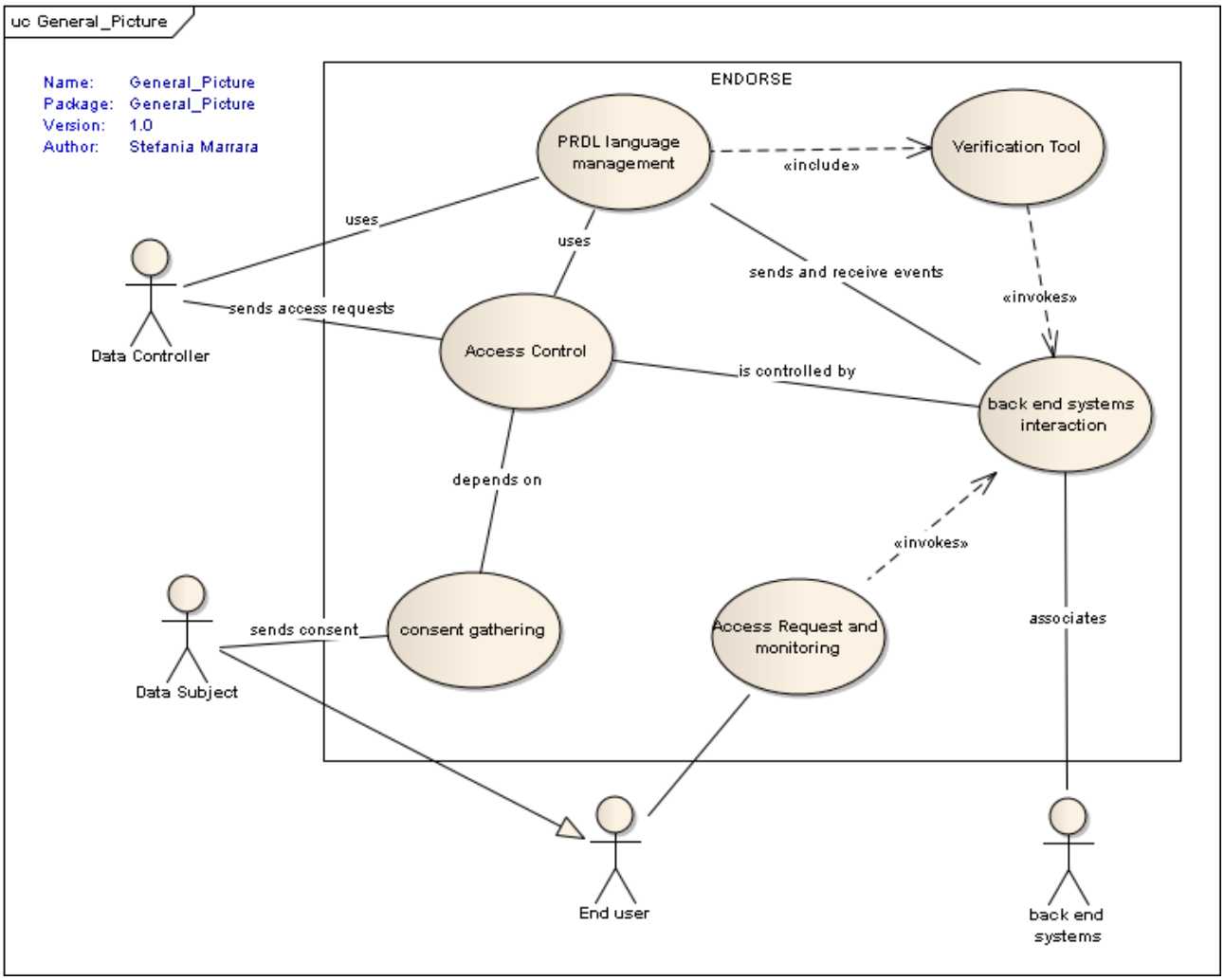

Figure 4: The functional architecture of Endorse

In ENDORSE there are three main actors who interact with the platform:

- the Company Data Controller
- the Data Subject
- the back end systems

Going into further details:

Data Controller
Type: **Actor**
Package: General_Picture

The word **Data Controller** shall mean the natural or legal person, public authority, agency or any other body which alone or jointly with others determines the purposes and means of the processing of personal data; where the purposes and means of processing are determined by national or Community laws or regulations, the controller or the specific criteria for his nomination may be designated by national or Community law.

When using Endorse, the main tasks of the Data Controller are:

- creation of the PRDL rule set describing the data privacy policy of the Company
- updating the rules
- querying the rules

Data Subject
Type: **Actor**
Package: General_Picture

The **Data Subject** The data subject is the natural person to whom personal data (a given piece or item of information in whatever form) refers.

Back end systems
Type: **Actor**
Package: General_Picture

All systems in the Company that need to manage personal data to perform their activities. They can be:

- call center operators GUI;
- databases;
- -report management systems;
- others.The main use-cases that describe the behavior of the ENDORSE platform are:

1. Access Control;
2. PRDL rules management;
3. Consent Gathering;
4. Back end systems interaction.

Access Control
Type: **UseCase**
Status: Proposed. Version 1.0. Phase 1.0.
Package: General_Picture

This use-case is in charge to filter the access of personal data by people (company employees, third parties) or systems (for instance a report management system) and includes:

- privacy protected data access control;
- users and groups of users management.

PRDL language management
Type: **UseCase**
Status: Proposed. Version 1.0. Phase 1.0.
Package: General_Picture

One of the core activities of the ENDORSE Platform is the PRDL language management, which includes:

- creation of PRDL rules;
- update, deletion of PRDL rules;
- query of PRDL rules;
- validation of a PRDL rule w.r.t a predefined set (a library) of PRDL rules.

Back end systems interaction
Type: **UseCase**
Status: Proposed. Version 1.0. Phase 1.0.
Package: General_Picture

This use-case includes:

- gathering of the personal data access requests by the back end systems;
- declaration of operations (data update, cancel, store) to be performed by the back end systems when certain conditions occur.

Consent gathering
Type: **UseCase**
Status: Proposed. Version 1.0. Phase 1.0.
Package: General_Picture

This use-case is responsible of gathering, storing and updating the consent, explicit or implicit, given by the user w.r.t a certain treatment of his/her personal data.

The architecture can be essentially described as shown in Figure 5.

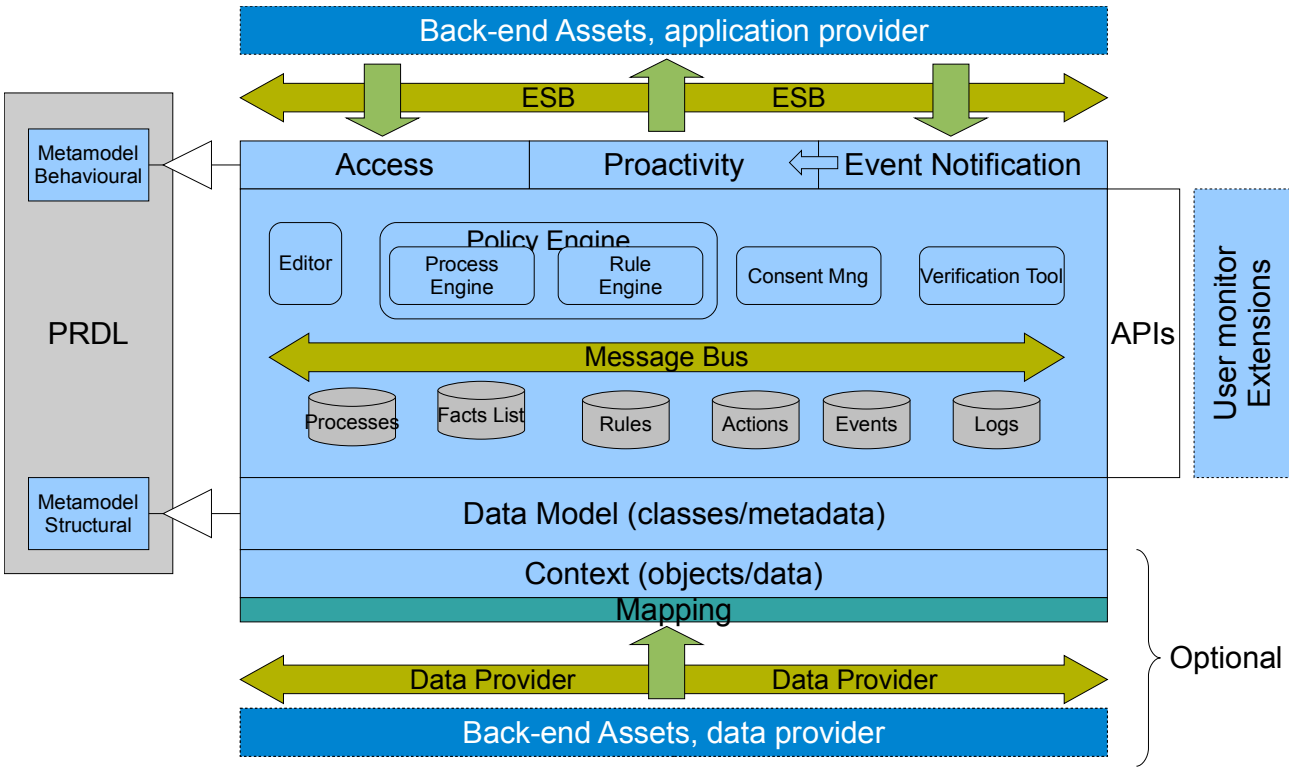

Figure 5: ENDORSE Architecture

The ENDORSE platform presents four main dependencies:

- the PRDL language, which is the reference language defined in D2.4 Language Requirements Specification and D3.2 Privacy Rule Definition Language - Preliminary Specification. This module, shown on the left side of the diagram, contains the Behavioral Metamodel and the Structural Metamodel that determine the way ENDORSE behaves. Section 5.3 will go into further details of this important module;
- the API, a programmatic interface that can be developed on top of a rich web interface. User can use this web interface to interact with the platform in order to monitor its behaviour, to configure it, to register users, to import or export information, to monitor the management of personal data, or affect the way the Company data privacy process behave by changing the thresholds that can appear in some rules (see the right side of Figure 5.). The web interface, or ENDORSE editor, is described in Section 5.2 and is also target of D4.1 Rule Editor Prototype;
- the Event driven message bus used to allow the communication between the ENDORSE platform and the legacy systems. Through this bus privacy related events are triggered, captured and delivered both ways across the legacy applications (i.e., Company back end systems) and ENDORSE (see top side of Figure 5). This part is further detailed in Section 5.8;
- a communication framework used to access the Company data repositories to allow the PRDL rule verification (bottom side of Figure 5).

5.2. The ENDORSE Editor

The ENDORSE editor has been built as a rich web interface on top of the platform, the scope of this component is to allow the Rule Maneger to fornally define the rules that reflect the way personal data need to be processed and managed. One of the main characteristics of the editor is that it allows several ENDORSE platforms to interact, by exchanging rule sets allowing access to their internal rule repositories.

The main use-cases which describe the behaviour or the editor are shown in Figure 6

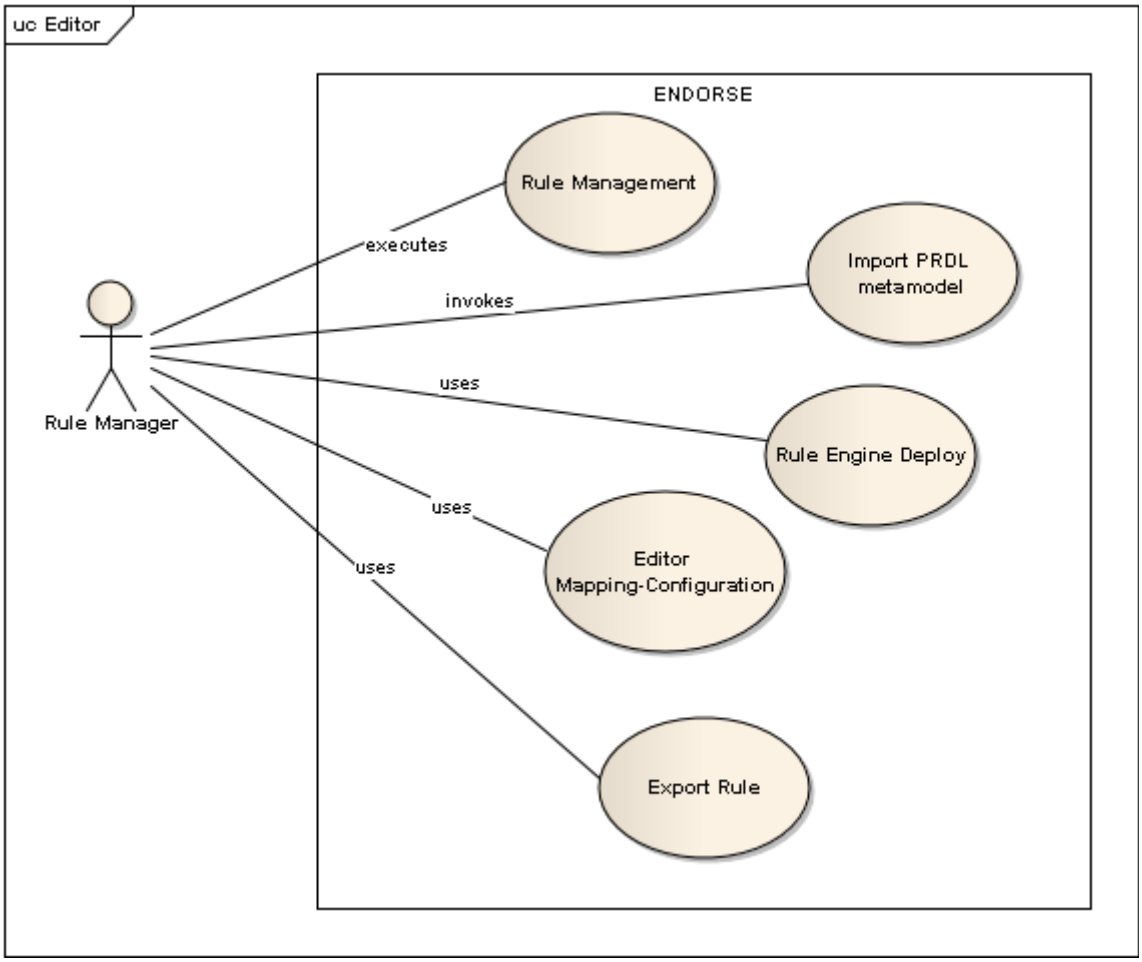

Figure 6: The ENDORSE editor use-case

The actor shown in the figure is the Rule Manager. The Rule manager is the Data Controller who is in charge to create or modify a PRDL rule or to create the necessary items to create a new rule model. This actor uses the editor to first create a rule model (i.e., the template of a generic rule with subject, object and constraint), then to create the instance of the rule based on the PRDL metamodel, and finally to define the process of this rule and to create the corresponding event.

The main use-cases that describe this component are:

Editor Mapping-Configuration

Type: **UseCase**

Status: Proposed. Version 1.0.

This use-case allows the actor to customise the editor and to add or modify the existing relationships between the PRDL metamodel and the editor metamodel.

Export Rule

Type: **UseCase**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Edito

This use-case describes the possibility for the editor to export the rule persisted in the model repository in a different form:

- as PRDL metamodel
- or as customized form

Import PRDL metamodel
Type: **UseCase**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Editor

The editor allows to import the PRDL metamodel.
In this use-case the editor imports the metamodel and auto configures the internal behavior.
During the import phase the editor maps each internal metamodel element with the corresponding element of the PRDL metamodel, this mapping is the mechanism that allows to transform a generic editor into a specific PRDL editor.

Rule Enginee Deploy
Type: **UseCase**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Editor

This use-case allows to export the PRDL rule expression into the rule engine. During the deploy phase the rule will be transformed from the PRDL model into the rule engine execution model.

Rule Management
Type: **UseCase**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Editor

This use-case allows the Data Controller (named here Rule Manager) to:
- Create a rule,
- Read a rule,
- Update a rule
- Verify if a rule exist
- Delete a specific rule
- Search using the query by example method a specific set of rule

Providing legal information
Type: **UseCase**
Status: Proposed. Version 1.0. Phase 1.0.
Package: Editor

In the editor environment the data controller is to be provided with legal information. The editor should give both general legal information and popup screens when the dc has to take a certain action (e.g. send a notification to the data protection authority)

A more complete description of this component is included into D4.1 Rule Editor Prototype Manual.

5.3. The PRDL metamodel

The development of PRDL is conducted within the ENDORSE project and aims at bridging the gap between the policy makers, the data handling organisations and the data subjects. The acronym PRDL stands for Privacy Rule Definition Language (PRDL) which will allow organisations and personnel responsible for data privacy to migrate their existing privacy policies into computationally executable rules. These rules can be enforced within the ENDORSE framework.

To be able to represent privacy policies PRDL has to be designed to have enough expressivity to cover the most important concepts of data privacy and should avoid a large vocabulary which probably makes the

usage too complicated and unhandy. With the usage of PRDL the two main objectives of the ENDORSE project will be tackled. These are more transparency for the data subject (the one whose data are processed) and legal compliance with relevant European directives and their national legalie implementations for data controllers (responsible personnel in the organizations). PRDL has been designed to find the right balance between expressive power and efficiency. Expressive power in terms of being able to formulate large parts of the relevant data privacy regulations and privacy policies. Efficiency meaning the usability and applicability for small and medium-sized enterprises (SME) and consequently for all types of users.

PRDL in general has a very broad field of application which spans from the end user over the organisation user to the policy creators. Moreover, the language has to meet all the user skills concerning usability, understand ability and extensibility. As the focus of the language lies mainly on designing rules for data handling of privacy relevant data, all the relevant constructs within data privacy have to be considered. The rules should be capable of expressing the usage of data within a business context. In addition the relevant legal aspects of such a data handling should be expressible. The language has to support purpose within the data handling context, as data collection has to be purpose driven by law.

The design of PRDL is accomplished in two meta model which are the Meta Access Model and the Meta Rule Model. On the one hand, the meta access model explains and illustrates the creation of a request for data to the ENDORSE system and the relevant data roles in this requesting process. On the other hand, the meta rule model depicts the layout of the PRDL rule and all the language constructs needed to express a data privacy related rule. These two meta models are described in the following sections followed by a section about the future work and challenges faced.

PRDL Meta Access Model

In a nutshell, the meta access model, shown in Figure 7, describes how a request to the ENDORSE system has to composed in order to enable the rules developed with PRDL evaluate the request. The model should fit into the architecture developed within this deliverable and provide help with the understanding of the rule enforcement. The model will be used for showing the interactions between the PRDL rules and a request within the demonstrator which is the topic in D.4.3. To give an overview the model is described shortly in the following as an explanation to the given figure. The elements of the model are marked italic to provide an easier comprehension of the text.

The root element of the meta access model is the *Request*. A request contains the four elements which are the *RequestPurpose*, the *RequestProcessing_ground,* the requested *Data,* and the reference to the *DataController* who executes the request. The rule engine will have to handle this request and either grant or deny the access by evaluating the request against the stored PRDL rules. The model includes three elements that play a role in this type of access control decision. First there is the *Processing_ground* that states the legal ground on the basis of which data was collected (e.g. consent, or compliance with a legal obligation). The engine has to verify whether the *RequestProcessing_ground* is compatible with the *Processing_ground* under which the *Data* was collected (for instance, 'legal obligation' in principle overrules 'consent'). The second element is *Purpose,* which represents the original purpose for which the data is collected (for instance, 'marketing' in the case of 'consent' based processing, or 'legal compliance' in case of 'legal obligation'). Third, the *RequestPurpose* which represents the purpose of the current access request. The engine has to establish whether *RequestPurpose* is compatible with the *Purpose* associated to the *Data.*

A request naturally has to include a *Data* element. This element represents the data elements which the *DataController* wants to use for a specific purpose. It is composed out of a *Consent*, *Constraint, Purpose*, *ProcessingGround* and a list of *DataObjects*. *Data* is additionally a container for a subset of *DataObjects* that describe the *Data* in greater detail. A Data element must have at least one *DataObject* and a *Consent* and can have one or more *Constraints*.

The *DataObject* object is the atomic representation of a part of personal data and it represents the smallest part of information within a *Data* element. (e.g. telephone number, first name, last name, etc.). In case of special circumstances, that prevent certain access or actions on the *Data* element, a *Constraint* element is defined. Processing grounds give the legal basis for processing, hence the *ProcessingGround* element is introduced. Where the processing ground is consent, the *Consent* element in every *Data* element grants that

the *DataSubject* has given his consent. *Purpose* specifies the reason why data is collected and processed and why a certain action on data is performed,such as direct marketing, etc.

There are three actor elements in meta access model. These are the *DataSubject*, *DataProcessor* and *DataController*. They are all related to data and therefore the *DataRole* super element was created to interlink these elements. The *DataSubject* has one or many *DataElements* assigned to it. As a central actor, the *DataController* creates a request for *Data*. Additionally every *DataController* has a *DataProcessor* assigned which is responsible for the executed actions on the *Data*. The meta access model is under active development and might be change to tackle future challenges within the project.

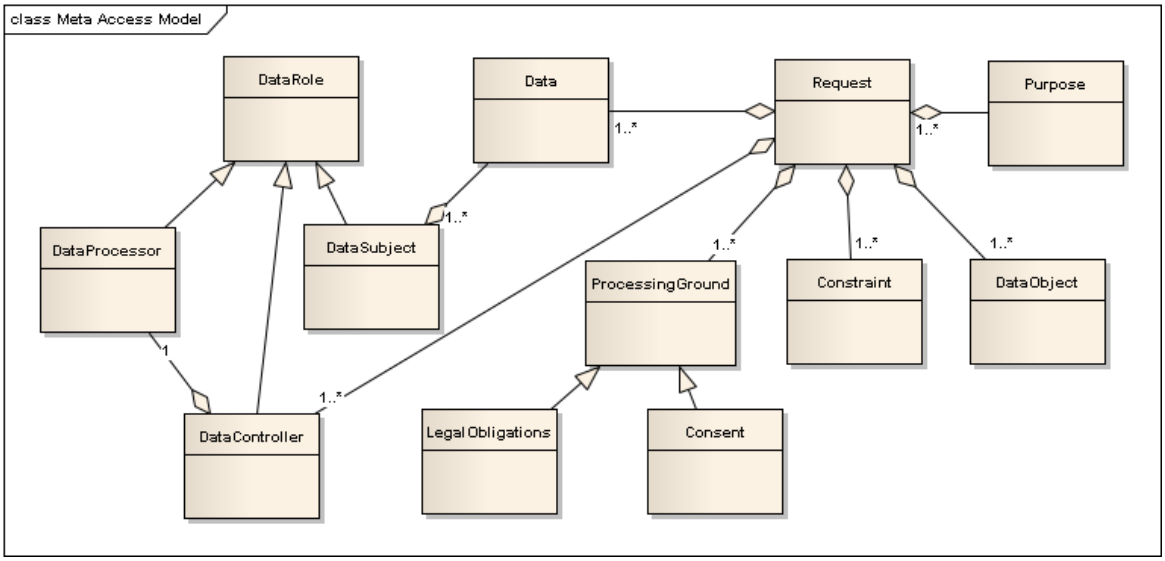

Figure 7: PRDL Meta Access Model

PRDL Meta Rule Model

The meta rule model is the current status of the PRDL development and is illustrated in Figure 8: the PRDL metamodel is not yet released. It shows all the relevant elements needed for representing a rule that can handle requests for personal data in the context of a data processing company. During the development process it was decided to create a rule language based on syntax templates and further a generic XML language that is able to represent these rules and additionally is parseable into an existing rule language (e.g. Drools[1], shown in section Rule Engine). It has to be noted that the meta rule model is preliminary and there certainly adoptions will occur to handle requirements that appear in the future. The following paragraphs describe only the main elements in the rule meta model. The complete description is given in the deliverable D3.2.

The root element of the rule model is *PRDLPolicySet* is responsible for providing a container to the PRDL policies. A *PRDLPolicySet* can include one or more *PRDLPolicies*. The *PRDLPolicy* element includes the actual PRDL rules and represents the second container within the PRDL language. The *PRDLRule* element is the central element of the rule model. All the rule relevant elements are stored within the *PRDLRule* which are the *StaticRuleAttributes* and the *DynamicRuleAttributes*. The rule element does not have any other elements contained as the attribute container encapsulates them.

The *StaticRuleAttributes* are the container element for all the rule elements that are not involved in a dynamic process. These attributes are the *DataController*, *DataObject* which belongs to a *DataSubject*, *Modality*, *Purpose*, *Instrument*, *Location* and *Exception*. Most of the elements are self-explanatory or are described in the glossary section of D3.2. The elements which require a short introduction are the *Instrument*, *Location* and the *Exception*. The *Instrument* element represents the possibility of specifying which instrument is used by the *DataController* to perform an action on the data. When the data access is restricted to the location of the *DataController* the corresponding element is used for implementation. Finally the

1 http://www.jboss.org/drools is a Business Logic integration Platform, developed inside the JBoss project

Exception element defines if there are specific exceptions concerning the processing of data such as special authentication needed or the prohibition of processing for certain people.

To introduce a difference in the type of attributes a rule consists of the *DynamicRuleAttribute* container was developed. It includes all attributes that can be related to a dynamic condition or a process. The element container is composed of the *Condition*, *State*, *Effect* and *CronAttribute* elements. Additionally the *Action* element was moved from the static attributes. First the *Condition element* is used to model conditions that influence the rule in a dynamic way. These conditions can include one or more *ConditionStatements*. Third, the *Effect* element introduces an ability of PRDL to change the effect that a rule actually has on a system. It can be used to influence other processes or data within the system besides clarifying whether performing actions on data is allowed or not. Fourth, the *CronAttribute* had to be embedded to meet a special requirement that originates from data protection legislation. The rule has to be able to model the fact that data, for example, has to be deleted after 10 years. As a Cron job is a event scheduler under Unix systems, the *CronAttribute* has similar functionalities triggering a rule to check such time based actions. The *DynamicRuleAttributes* are in a preliminary state at the time of writing and will be specified in greater detail in the D3.4 final specification of PRDL.

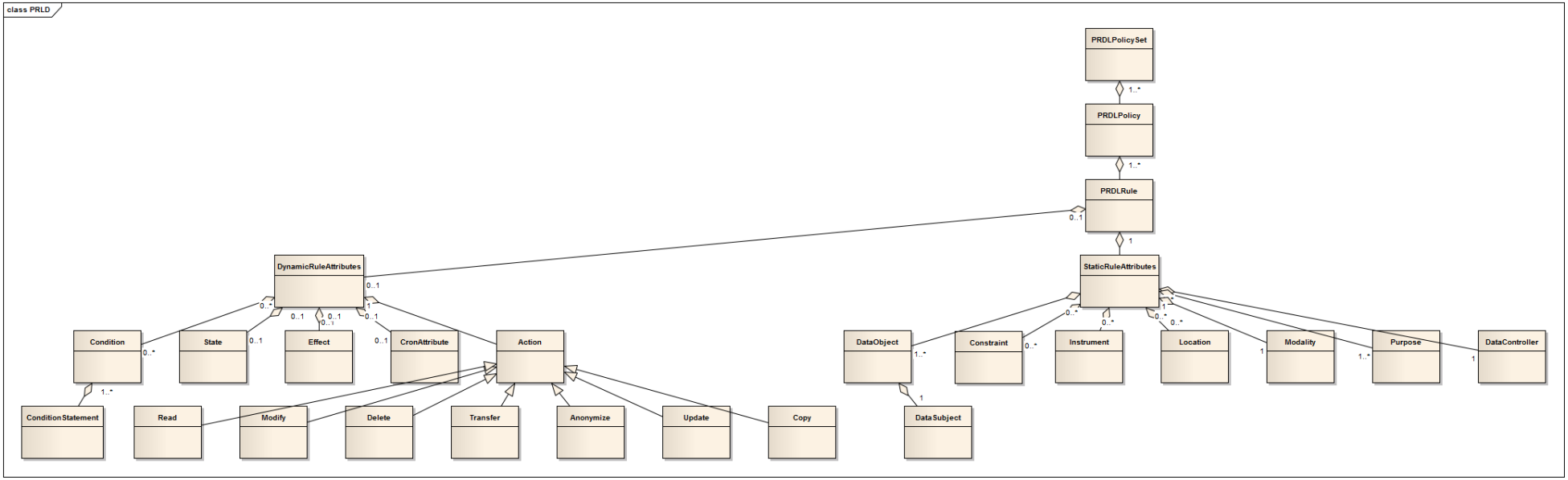

Figure 8: PRDL Meta Rule Model

5.4. Rule Engine in the ENDORSE framework

The enforcement of policies within organizations is an important issue due to the fact that these policies are used to protect and restrict access to confidential information. There exist a lot of applications in various areas starting from real-time solutions for privacy policies in sensor networks, over location-aware environments up to law enforcement related policies like in the Endorse project. The used access policies can be described as a rule or a set of rules used to provide control over shared resources within an organization for both owners and administrators. Every-time a user wants to access a resource the rule engine evaluates the stored policies (rule sets) and then decides if access is granted or not and also if further action are necessary.

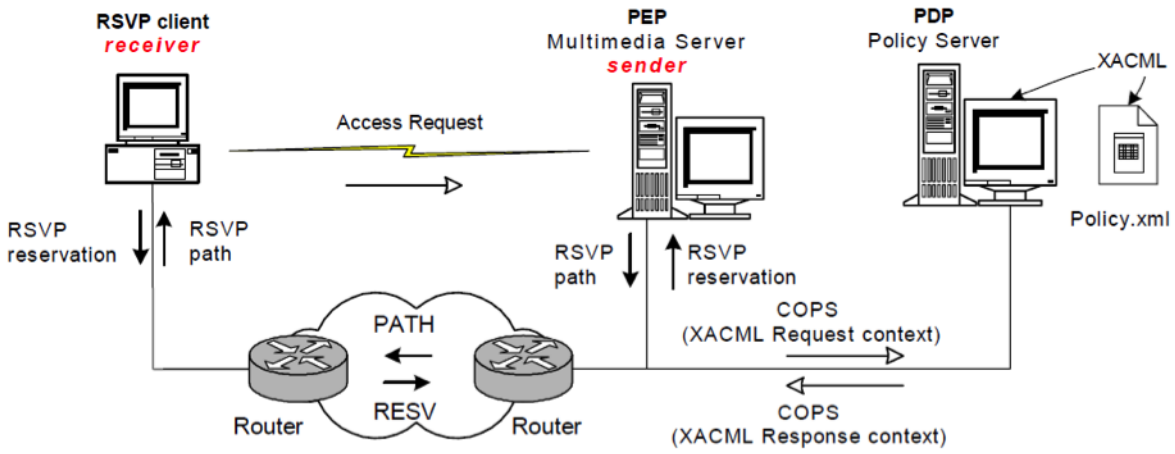

Figure 9: Request and Response with XACML

In case of XACML the rule engine or policy enforcement engine is based on two main components namely the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP), see Figure 9. First is handling request by translating them into XACML and forwards them to the PDP which in return is checking the request according to stored policies and decides if the request is granted or not. The decision process itself can be split up into two actions. The first action is the selection process of the fitting policy to be used and the second action the selection of the actual rule to be applied out of the policy (rule set).

Drools

In the first trial phase, the consortium is going to investigate Drools as the tool to execute the PRDL rules. It is a rule engine in Java which utilizes the Rete algorithm [Rete82] in order to check the rules. The rules are declared in a declarative way and expressed in a non-XML language. In addition to that it is possible to integrate Java code in-line inside the rules. The Drools rule engine is open-source and free of charge, driven forward by an active community of developers. In the current version of Drools a lot of improvements were implemented for instance an over hauled version of the language for rule expression featuring less coding effort and improving human-readability at the same time. Furthermore, there exists a plug-in for the Eclipse IDE to provide comfortable programming surroundings by checking the code for syntax errors, providing suggestions via auto-completion and the possibility to debug the coded rule.

Example

The code in this section shows an example for a rule coded in Drools. A specific description of this example can be found in D3.2 Privacy rule definition language preliminary specification.

EurA MAY read name, address, mobile number, FOR customer identification

rule "Access to name, address and mobile number"

when

$r : Request (dataprocessor.getAuth == "EurA"

&& modality.get() == "may"

&& requesteddata.isEqual("name,address,mobilenr")

&& purpose.getPurpose() == "customer identification")

then

$r.grantAccess();

end

5.5. The Model Repository

The model repository is the container for the PRDL expressions. It will contain both rule templates (i.e. *models*) and rule instances.

The main use-cases that represent this component behaviour are *crudel* and *search* (see Figure 10)

In the *crudel* use-case the model repository will allow to perform typical crude operations (Create, Read, Update, Delete Exist and List) on the PRDL rules contained into the repository.

In the *search* use-case retrieves a PRDL expression into the model repository.

ENDORSE will allow to search a rule using a query by example syntax which follows the PRDL syntax defined by D3.2 Privacy Rule Definition Language - Preliminary Specification.

The repository is then in charge of:

- Retrieve Rules, Processes and Events
- Publish Rules, Processes and Events
- Import/Export consistent PRDL expressions across Model Repositories belonging to other ENDORSE platforms.

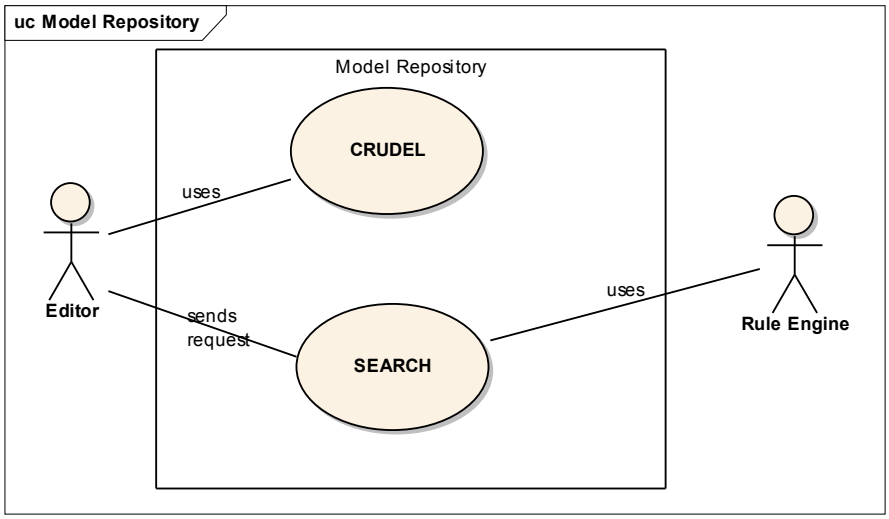

Figure 10: The Model Repository use-cases

The actor *Editor* shown in Figure 10 is the Endorse component responsible to manage the PRDL expressions and detailed in Section 5.2.

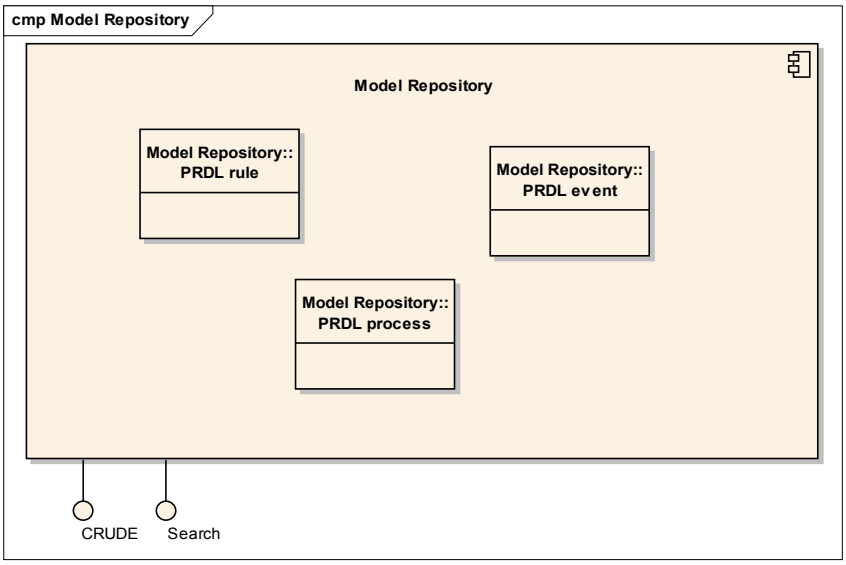

Figure 11: The Model Repository component structure

Figure 11 shows the model repository component which provides the storage of the PRDL expressions.

The component is used by the editor to persist, retrieve and modify the rule expression, it is used by the rule engine to retrieve a particular rule to execute.

The component exposes well defined Interface APIs to the Editor component.

The CRUDEL interface allows to manage the PRDL rule expression stored into the Model Repository.

The functionalities offered by the CRUDEL interface are:

- Create: it is possible to store a PRDL expression into the model repository;
- Read: it is possible to read a PRDL expression;
- Update: it is possible to update a rule persisted into the model repository;
- Delete: it is possible to delete a specific rule stored into the repository;
- Exist: it is possible to verify if a PRDL expression exists in the repository;
- List: it is possible to create a list of PRDL expressions;

The SEARCH interface allows to retrieve a PRDL expression contained into the model repository.

It is possible to search for a certain rule using a query by example syntax, the expose interface will accept a set of parameters as input value and will search the rule containing all the target parameters.

5.6. The Workflow Engine

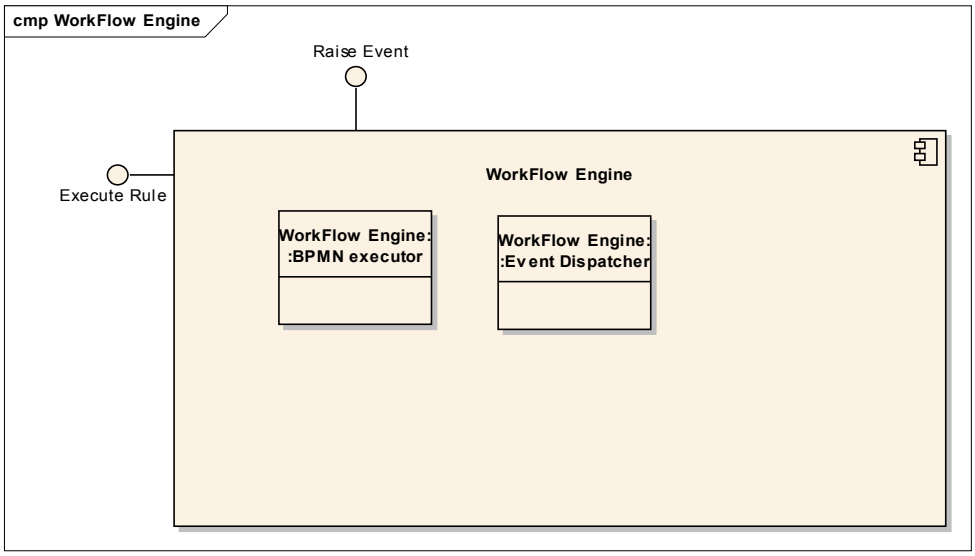

Figure 12: The Workflow Engine component

One of the properties of ENDORSE is the possibility to execute complex rules as defined by D3.2 Privacy rule definition language preliminary specification.

A complex rule includes constraints and choices and therefore requires the execution of a process to retrieve a result. Often the execution of a complex rule will require the definition of a chain of states for the ENDORSE rule execution process.

The component responsible to execute such a complex process is the Workflow Engine, which performs its activity in strict relationship with the Rule Engine detailed in Section 5.4.

The Workflow Engine runs the PRDL expression process developed by the editor and persisted into the model repository.

The component is able to run a Business Process Management and Notation (BPMN) model and interact with the editor, model repository, the event repository and the rule engine through the ESB. A possible notation that could be chosen here is BPEL which the standard language supported by the OMG Business Process Management Institute.

5.7. Data Integration Layer

The data integration layer is responsible of the communication between ENDORSE and the back-end systems of the Company which are in charge of processes that involve personal data protected by the law.

The data integration layer is a common solution adopted in SOA to abstract business logic from the underlying data structures. A data integration layer understands and maintains the location, structure, format, synchronization patterns and cross-reference relationships of the underlying data. It uses this information as a foundation to provide data integration services to reliably access and update the data, greatly increasing productivity and reducing the cost and complexity of building and maintaining business services and thus of implementing an SOA. Furthermore, a data integration layer provides the information and processes to unify data into a single logical view, which can then be stored in an operational data store or data warehouse or accessed via business activity monitoring (BAM) tools for real-time reporting.

Thanks to this layer ENDORSE can have access to data stored in the back end systems necessary to execute or validate the PRDL rules that exemplify a certain privacy protection policy.

From an architectural point of view the data integration layer is here used to access data and information in the back-end systems outside of the Endorse Platform. This mechanism is not for writing information but

only as read only. If there is any need to modify information, as a consequence of a rule triggering, then a remote service need to be invoked via the ESB layer described above.

There are many good reasons to choose an architecture for ENDORSE which uses a data integration layer to communicate with the other parts of the Company informative system.

1. Leverage existing data assets – this mechanism has been widely used in SOA to decouple data and applications. This approach has many advantages since it reduces the need of deep changes in the product when new functionalities are required thus reduces management costs. Its adoption in ENDORSE will decouple data and privacy compliant data management policies allowing to fulfill the main goal of the project as defined in the DOW.
2. Lower development and maintenance costs - Through the reuse of existing assets, the deployment of new functionalities can be performed with low effort and costs. The use of a common semantic data model allows for independent development of each single part of the product.
3. Flexibility to change - The separation between data and functionality allows for privacy management logic in ENDORSE to be rapidly modified as law needs dictate, making the IT organization much more adaptive and flexible to change.

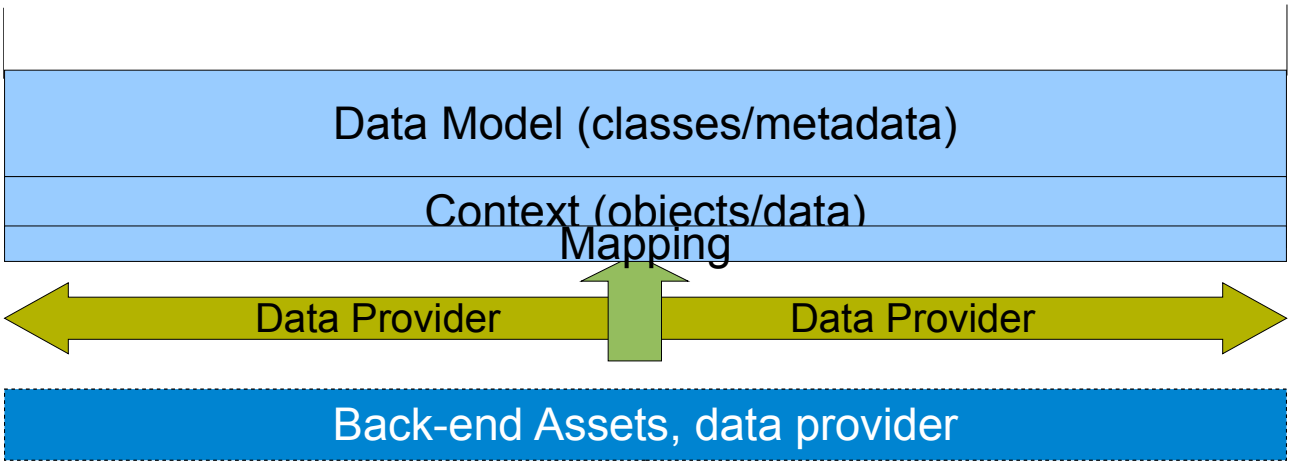

Figure 13: The Data Integration Layer architecture

5.8. Message Oriented Layer

ENDORSE Architecture has been designed with the aim to be loosely coupled respect to the companies' applications ecosystems (ERP, HR, CRM, accounting…). In this way the Consortium gain two important benefits:

1. all modules can be designed, developed and deployed independently;
2. no module or functionality requires a complete redesign of the ENDORSE platform in case of adjustments or changes.

ENDORSE components interact with other and with the external environment by using the Enterprise Service Bus (ESB) technologies (see Figure 14 and 15)

An ESB transports the design concept of modern operating systems to networks of disparate and independent computers and can be adopted to provide cooperation among almost independent components into a loosely coupled platform. Like current operating systems an ESB caters for commonly needed commodity services in addition to adoption, translation and routing of a client request to the appropriate answering service. The prime duties of an ESB are:

- Monitor and control routing of message exchange between services;
- Resolve contention between communicating service components;
- Delegator to guarantee delivery in case the destination is temporary unavailable;
- Control deployment and versioning of services;

- Marshalling data;
- Cater for commonly needed commodity services like event management, transformation, queuing, security or exception handling ESB and Atomic Services.

An ESB generally provides an abstraction layer on top of an implementation of an enterprise messaging system. In order for an integration broker to be considered a true ESB, it would need to have its base functions broken up into their constituent and atomic parts. The atomic components would then be capable of being separately deployed across the bus while working together in harmony as necessary. The details of the ENDORSE ESB will be provided into D3.3 Technical Architecture.

Outside Endorse

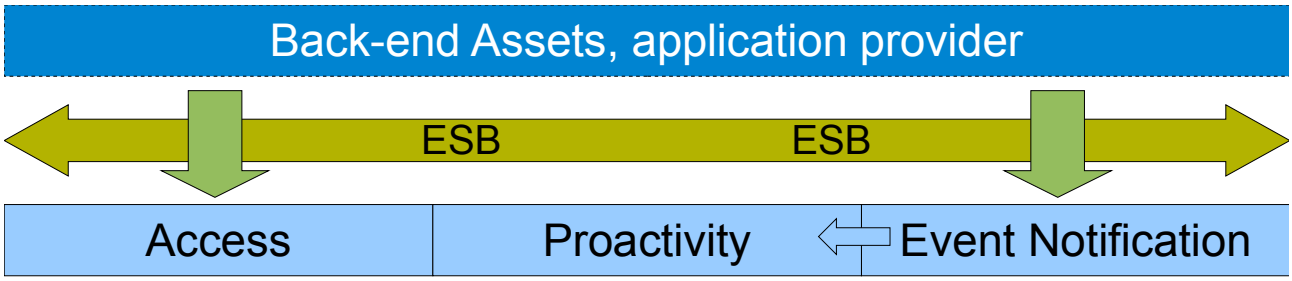

Figure 14: The ESB between ENDORSE and the back end systems

From a functional point of view here the ESB is a infrastructural technical component that binds the back-end system and the ENDORSE platform to comply to a unique transport protocols in order to send events either synchronously and asynchronously.

This decouples the dependency from technical infrastructures of the specific customer adopting ENDORSE allowing for better reuse across different installations.

Inter-Endorse

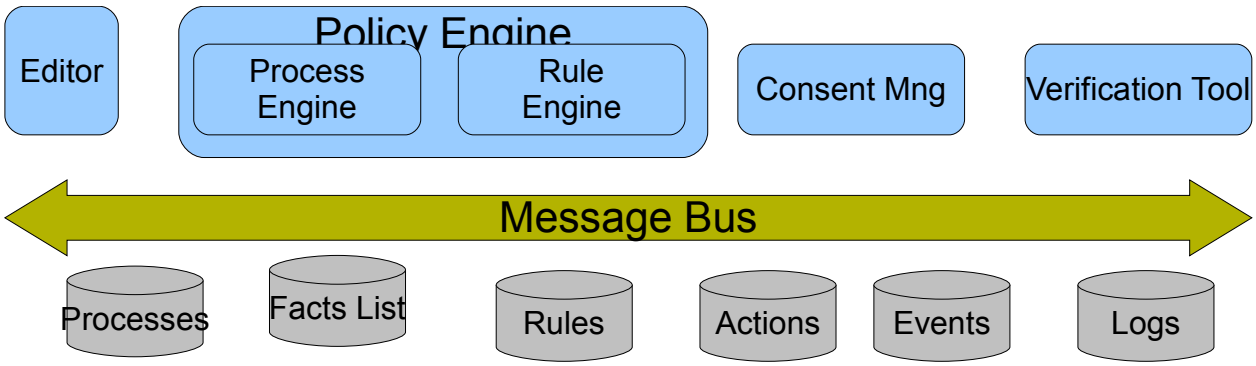

Figure 15: The ESB inside ENDORSE

When used inside the ENDORSE platform, the ESB is an internal specific feature that allow efficient method invocations across internal component. This allows hot-plugging component, isolation in case of a single component failure, velocity in memory communications, agile dependency management and a flexible development approach.

5.9. Logging Monitoring Facility

ENDORSE primary aim is to guarantee that the personal data management performed by the Company is compliant to the law. For this reason security of management processes and recording of relevant activities involving privacy protected data are of overcome importance.

The Logging Monitoring Facility is a framework aimed to having a uniform mechanism to record both logs and traces.

- Logs usually contain technical information recorded to inspect error or misbehaviour in the platform.
- Traces record the functional activities of the system, i.e. Functional outcomes of the rule and process engine.

5.10. Access Control

Access control is one of the main elements of ENDORSE with respect to privacy policy and legislative compliance. An Access Control component set in the ENDORSE architecture will present a generic functional interface through which access control can be either handled by a concrete implementation of ENDORSE via internal system implementation or as means to interfacing to back office legacy access control systems.

5.11. Authentication and Accountability

Authentication mechanisms in the ENDORSE framework will be defined via an interface which allows the concrete implementation of the system to implement such a mechanism through ENDORSE directly (potentially through the use of off the shelf components designed for authentication purposes or by interfacing with legacy back-end authentication systems in the enterprise.

Accountability provides an important facility for ENDORSE deployments in that it provides non-repudiable evidence of data handling. This component will make use of the logging facility together with the authentication facility and provide strong evidence in the event of data handling requests from end-users.

5.12. Consent Gathering

Consent gathering is an important component for any project related to privacy protection. The Data Subject consent is a required pre-requisite for any kind of data management by the Company.

By *consent* we usually mean the principle that a person must give their permission before they personal data are gathered of used for a specific purpose when processing takes place on the basis of consent of the data subject (art. 7a DPD).

The principle of consent is one of the cornerstones of privacy protection as declared by the European law.

But what do we mean with "*valid consent"*?

For consent to be valid, it must be voluntary and informed, and the person consenting must have the capacity to make the decision. These terms are explained below.

- **Voluntary:**the decision to consent or not consent to a certain usage of personal data must be made alone, and must not be due to pressure by the data controller, friends or family.
- **Informed:**the person must be given full information about what the data management policies involve, and how his/her data will be used for and for how much time.
- **Capacity:** the person must be capable of giving consent, which means that they understand the information given to them, and they can use it to make an informed decision.

The ENDORSE architecture include a module for consent gathering and management whose main functionalities are described below.

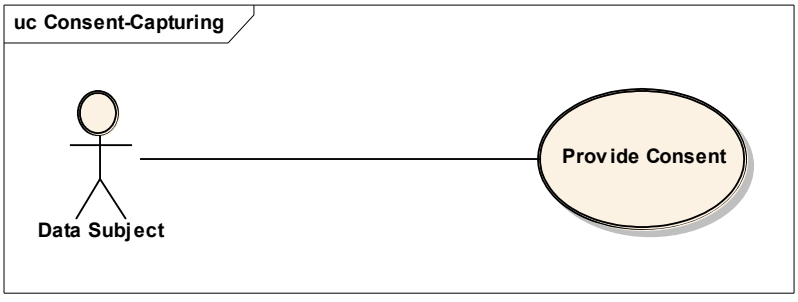

Figure 16: Use-case of providing consent of the data subject

Figure 16 illustrates the use-case where a Data Subject provides his/her consent. A Data Subject can provide his/her consent either *statically* or *dynamically*. In the former case, the consent is captured before making any request. While in the later case, the consent is captured dynamically after making the request. Each of these cases is described in the following sequence diagrams:

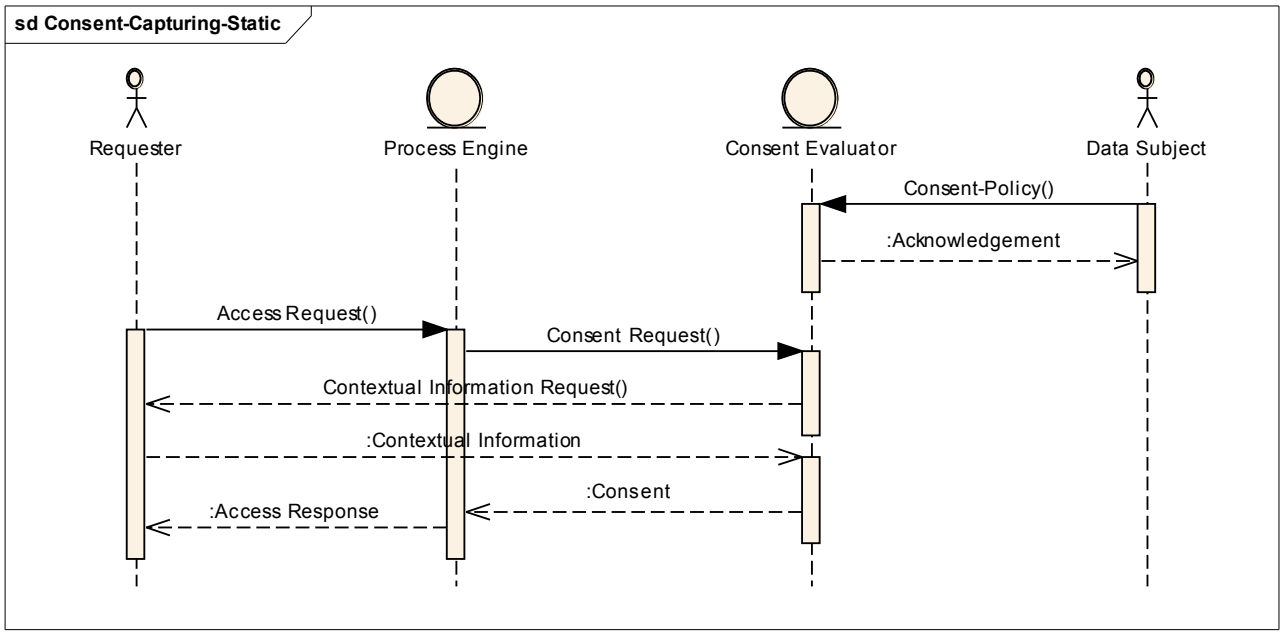

Figure 17: Sequence diagram when consent is captured statically (at the beginning)

Figure 17 illustrates the scenario when the consent is captured statically by storing the consent-policy at the beginning of any possible interaction between the new Data Subject and the Data Controller.. This is accomplished before making any request. The Consent Evaluator stores and manages the consent-policy. The Consent Evaluator is managed for each Data Subject. Here, the consent-policy is different from the consent. Actually, the consent is derived from the consent-policy after evaluating against the contextual information collected from the Requester. This contextual information may include, but not limited to, name, role, time, location, age, etc. Typically, the consent-policy is stored once and then the consent can be derived for the subsequent requests.

In case when the consent is captured statically, a Data Subject first sends a Consent-Policy to the Consent Evaluator as shown in Figure 17. Upon receiving the Consent-Policy, the Consent Evaluator sends an Acknowledgement back to the Data Subject to confirm that the Consent-Policy has been stored successfully. Once the Consent-Policy has been stored by the Consent Evaluator, a Requester can make an Access Request. The Process Engine receives the Access Request and identifies if the consent of the Data Subject is required. If it is required, the Process Engine sends a Consent Request to the Consent Evaluator. The Consent Evaluator finds and loads the consent-policy corresponding to the Consent Request. Optionally, the Consent Evaluator may send the Contextual Information Request to the Requester and wait for the Contextual Information. Afterwards, the Consent Evaluator evaluates the consent-policy against the Contextual Information and then sends the Consent to the Process Engine. The Process Engine takes the requested action based on the consent value and finally it sends an Access Response back to the Requester.

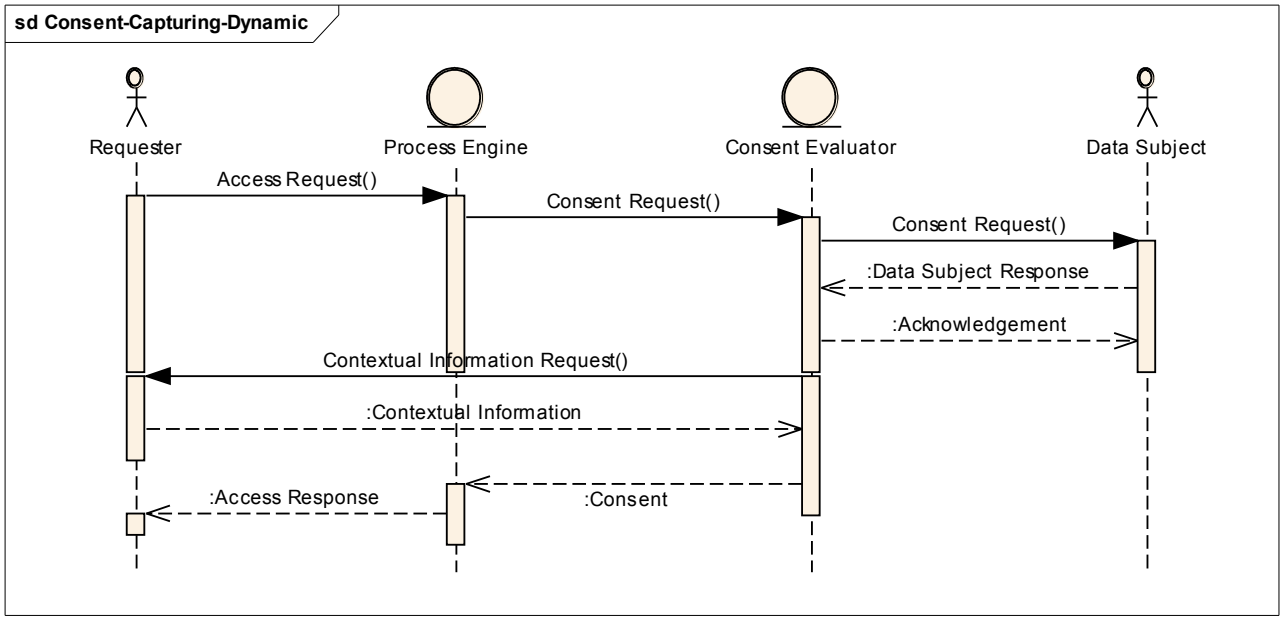

Figure 18: Sequence diagram when consent is captured dynamically (at the runtime)

Figure 18 illustrates the scenario when the consent is captured dynamically at the runtime when the Data Subject has already required a certain service instead of being stored at the beginning. In this case, a Requester starts making an Access Request which is sent to the Process Engine. The Process Engine identifies whether the consent is needed or not and if the consent is required then it sends the Consent Request to the Consent Evaluator. Since the Consent Evaluator does not have any kind of stored consent-policy, it forwards the Consent Request to the Data Subject. The Data Subject may decide either to send only the consent corresponding to the current request he/she receives or to send the consent-policy to the Consent Evaluator. Moreover, a Data Subject may decide to send both the consent for the current request and the consent-policy together in the Data Subject Response. In case if a Data Subject Response contains the consent-policy then an Acknowledgement is sent back to the Data Subject. If a Data Subject Response contains only the consent-policy then the Consent Evaluator sends Contextual Information Request to the Requester. The Requester provides the Contextual Information to the Consent Evaluator. After evaluating the consent-policy against the contextual information, the Consent Evaluator sends Consent to the Process Engine. Finally, the Process Engine sends Access Response to the Requester.

6. Conclusions and Future Work

This deliverable is an important step in the development of ENDORSE Functional Architecture which is object of T3.1 Functional Requirements which does not end at the time of the delivery of this work but will continue till April 2012.

Therefore aim of this work is to design the ENDORSE Functional architecture by identifying the main components that will be detailed in D3.3 Technical Architecture and the main functionalities that each component is responsible of.

The definition of the interfaces, that will be target of T3.1 Functional Requirements since September 2011 to April 2012 will be included as part of D3.3 Technical Architecture since no further deliverable is planned at the end of the task itself.

Regarding the core of ENDORSE, PRDL language and corresponding Rule engine, further work will be performed in both WP3 and WP4 and presented in other deliverables.

The development of the PRDL will result in the final specification of the language in D.3.4 which is due April, 2011. As the language should provide a usable tool, there has to be a lot of work done in implementing example rules and test them against requests. With the input of the legal partners in the ENDORSE consortium the language will be primed to face the challenge of representing data privacy statements accurately. Key will also be the editor that should enable the stake holders to create the rules as easy and comfortable as possible.

As it was already stated in D3.2 the first step towards a working solution will be Drools which was presented before. The upcoming step will be a feasibility testing in order to proof that all significant constructs needed for the full implementation of the PRDL can be concealed by Drools. In addition to that an evaluation will be carried out to identify and check other possible candidates regard the use as rule engine for the PRDL. The results of this test phase will be presented in D4.3.

Glossary

Term	Description
DSL	Domain Specific Language, a domain specific metamodel. It represents the semantics for specific domain area.
Metamodel	A model of a model, often referred to as a "abstract syntax".
Consent	A permission to do something
XACML	eXtensible Access Control Markup Language, http://www.oasis-open.org/committees/xacml/

References

[FRP] Functionality, Usability, Reliability, Performance, Supportability

[FRP+] FURPS adding Design requirements, Implementation requirements, Interface requirements, Physical requirements

[FRPIBM] http://www.ibm.com/developerworks/rational/library/4706.html#N100A7

[BOEM] B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. McLeod, and M. Merritt. Characteristics of software quality. 1978.

[RUP] Unified Software Development Process, The Unified Software Development Process, Ivar Jacobson, Grady Booch, James Rumbaugh, February 14, 1999, ISBN-10: 0201571692

[You98] Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering, Eric S. K. Yu, Toronto 1997, Ontario, Canada M5S 3G6

[Castro00] J. Castro; M. Kolp; J. Mylopoulos Tropos: Toward Agent-Oriented Information Systems Engineering. Second International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS2000) 2000

[Odell00] James J. Odell, H. Van Dyke Parunak, Bernhard Bauer, Agent-Oriented Software Engineering, Paolo Ciancarini and Michael Wooldridge eds., Springer, Berlin, pp. 121-140, 2001

[BaBe03] Requirements Engineering with GRAIL/KAOS: From Goal Analysis to Automatically Derived Requirements Documents Denis Ballant, Christophe Belpaire, Robert Darimont, Emmanuelle Delor, Denis Genard, Cédric Nève, Jean-Luc Roussel, Alain Vanbrabant. CEDITI, Avenue Georges Lemaître 21, B-6041 Charleroi, Belgium

[DeFox99] McDermott J, Fox C (1999) Using abuse case models for security requirements analysis. In: Proceedings of the 15th annual computer security applications conference (ACSAC'99), Phoenix, Arizona

[PaXu05] Pauli, J. J. and D. Xu (2005). Misuse case-based design and analysis of secure software architecture. International Conference on Information Technology: Coding and Computing (ITCC 2005), Las Vegas, IEEE.

[Rete82] Forgy, C. Rete: A fast algorithm for the many patterns/many objects match problem. Artif. Intell 19, 1 (1982), 17–37