ICT Seventh Framework Programme (ICT FP7)

Grant Agreement No: 288513

Policy Formulation and Validation through non Moderated Crowdsourcing

D6.2First Integrated Prototype

Deliverable Form	
Project Reference No.	ICT FP7 288828
Deliverable No.	D6.2
Relevant Workpackage:	WP6: Design and Implementation of NOMAD tools
Nature:	P
Dissemination Level:	PU
Document version:	2.0
Date:	08/08/2013
Authors:	V. Tountopoulos, T. Dalianis (ATC), George, Petasis, George Giannakopoulos, George Kiomourtzis and Vangelis Karkaletsis (NCSR 'D')
Document description:	This deliverable describes the first working prototype of the integrated view for the NOMAD tools.

Document History

Version	Date	Author (Partner)	Remarks
Draft v0.1	08/04/2013	V. Tountopoulos, T. Dalianis (ATC)	Table of Contents
Draft v0.2	02/05/2013	V. Tountopoulos, T. Dalianis (ATC)	First Draft
Draft v0.3	13/05/2013	G. Petasis (NCSR'D')	Input to Section 4
Draft v0.4	14/05/2013	G. Giannakopoulos (NCSR'D')	Input to Section 4
Draft v0.5	21/05/2013	V. Tountopoulos, T. Dalianis (ATC)	Second Draft
Draft v0.6	01/07/2013	V. Tountopoulos, T. Dalianis (ATC)	Updates on the use cases
Draft v0.7	17/07/2013	V. Tountopoulos, T. Dalianis (ATC)	Third draft
Draft v0.8	23/07/2013	V. Tountopoulos, T. Dalianis (ATC)	Prefinal version
Draft v0.9	25/07/2013	Tobias Ruppert (Fraunhofer IGD), Costas Koutras (AEGEAN)	Peer Review
Draft v0.95	29/07/2013	V. Tountopoulos, T. Dalianis (ATC)	Final version
Draft v0.96	06/08/2013	V. Tountopoulos (ATC)	Small corrections to the final version,
Final v1.0	08/08/2013	Aggeliki Androutsopoulou, Yannis Charalabidids, Euripidis Loukis (AEGEAN)	Document Finalisation

EXECUTIVE SUMMARY

NOMAD aims to deliver a novel methodology for treating the social web as a valuable resource of information for policy making, through non-moderated crowdsourcing. Thus, it will provide a complete set of tools for searching, analysing and visualising arguments, opinions and sentiments regarding a policy domain. These tools will be integrated in the context of an open, modular platform for policy support,applied in specific policy making cases in threecountries.

In order to achieve this, NOMAD specifies a novel approach in the policy making field and conducts innovative research aiming to advance the state-of-the-art in the areas of linguistic processing, argumentation modelling and analysis, opinion mining and visualisation. The research effort has led to the release of draft prototype versions of the respective software modules, which have been integrated into a common platform, the NOMAD prototype.The NOMAD platform enables target stakeholders, such as policy makers, policy advisors and communication experts or even journalists and members of public associations, “listen” the public opinion, as it is mainly expressed in the social media, offering valuable assessment towards the behaviour of the public to a specific policy.

In this context, this document describes the first integrated prototype of the NOMAD platform. This version is the first fully working version of the platform and will act as the test bed for the NOMAD stakeholders to experience with the NOMAD provisions and assess the concepts and knowledge conveyed by the project.

The deliverable is examining the platform from two different aspects; from the internal point of view, in which the functional and architectural structure is described, and from the end-user point of view, in which the external appearance, look-and-feel and the overall high-level functionalities and interaction potentials are explained. Moreover, the document delves into the detailed presentation of the internal NOMAD workflow, the formats for the service-based interfaces and all features necessary to describe the platform, the current status and the envisaged outcome.

In order to achieve the project objectives, the NOMAD platform enables the policy formulation is a five steps approach. Initially, the end users identify the policy that they want to analyse. The policy lies on a specific domain of application, thus the NOMAD platform offers flexible and easy to use functionalities for modelling the domain, consisting of domain terms (entities). The NOMAD stakeholders can then analyse the policy and model it to identify relevant policy statement and associate them with domain entities and positive/negative arguments. Through modern visualisation techniques, the NOMAD platform enables the target end users visually assess the social media and the Web reaction to the specific policy model, through different perspectives, in terms on trends, time and demographics. On top of that, it supports discovering argumentation on the policy statements, through which the models can be refined and enriched, in order to progressively build a detailed and mature policy model, which can reach the maximum of the public stance in social media.

The NOMAD platform, far from being a simple container for the individual modules, is a coherent application, where several different components reside and collaborate in harmony. The first prototype is a proof of concept to the NOMAD end users. It encapsulates most of the underlying technologies and gives a clear and easy to use graphical interface, exposing every available feature so far.

The benefit of the current architecture is that any additional functionality can be wrapped into a separate component and be added to the platform, provided that it abides by the basic communication standards exposed by the NOMAD architecture. The scope of this practice is to keep the NOMAD platform evolving and enable future extensions to arising functionalities, which may maximise the potentials for further exploitation and adoption of the platform beyond the project end.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF TERMS AND ABBREVIATIONS

API	Application Programming Interface
UI	User Interface
HTTP	Hypertext Transport Protocol
JSF	Java Server Faces Framework
JSON	JavaScript Object Notation
ORM	Object-relational mapping
REST	Representational State Transfer
SOA	Service Oriented Architecture
XML	eXtensible Markup Language

1. INTRODUCTION

1.1 Purpose and Scope

The purpose of this document is to describe the NOMAD platform both from an end user and a more technical point of view. It starts with an overview of the revised Architectural Design and the general NOMAD workflow, and it elaborates on the envisaged functionalities to support the requirements of the target end users through the NOMAD platform use cases.

1.2 Approach for Work Package and Relation to other Work Packages and Deliverables

WP6 aims at providing the architectural and implementation aspects for the delivery of the NOMAD tools and their integration in a unified platform. The design of the NOMAD tools is driven from the requirements definition in WP2 and the analysis of the envisaged functionalities into platform use cases, as realised in this WP.

Following the implementation of the individual modules in WP3, WP4 and WP5, this WP delivers an integrated view of the NOMAD platform to act as the test bed for building the NOMAD pilots in WP7 and evaluating them in real life scenarios. The decisions presented in this deliverable are subject to refinements and modifications, based on the progress of the technical work packages, as well as the validation and evaluation phases.

1.3 Methodology and Structure of the Deliverable

This document reports on the initial activities and effort placed in the integration of the various technologies and tools towards delivering a functional NOMAD platform. Following the NOMAD approach towards the policy making lifecycle, the integration effort is guided from the Agile Software Development methodology, aiming to progress the development work in parallel teams and regularly integrating their output, based on a well-defined design.

The scope of this document is to act as appendix to the current version of the NOMAD platform prototype and, as such, it is structured as follows:

- Section 2 provides an overview of the NOMAD architecture, highlighting the revisions made with respect to [1]. It describes the mapping between the logical view of the NOMAD design and the physical components being developed in the course of the project and updates the NOMAD workflow for supporting the policy making through crowdsourcing. It, then, elaborates on the NOMAD platform use cases, as the pilot to realise the NOMAD functionalities from the end user point of view.
- Section 3 describes the integrated prototype of the NOMAD platform by describing the integration activities for linking the processing, layer, the presentation layer and the storage layer components together. This section, also, provides a detailed description of the implemented services for accessing the NOMAD Database. Furthermore, the specific technologies for the implementation of the WP6 components and the integrated prototype are presented.
- Section 4provides the framework that is being exploited to define the evaluation metrics for testing the NOMAD integrated prototype. Some initial test results are, also, included in this section.
- Finally, Section 5 concludes this report and presents the next steps for the final integration of the NOMAD platform.

2. REVISION ON THE ARCHITECTURAL ASPECTS

2.1 Revised Design of the NOMAD Architecture

As it has been described in [1], the NOMAD architecture adopts the principles of the Service-Oriented paradigm to define the necessary software components to facilitate the set of functionalities required for the authoring of domain and policy models and the visualisation of specific policy making concepts, which drive the target stakeholders in identifying the evolution of citizens' trends in social media with respect to the defined policy arguments.

The NOMAD logical components are placed in a three-layeredArchitecture, as shown Figure 1.

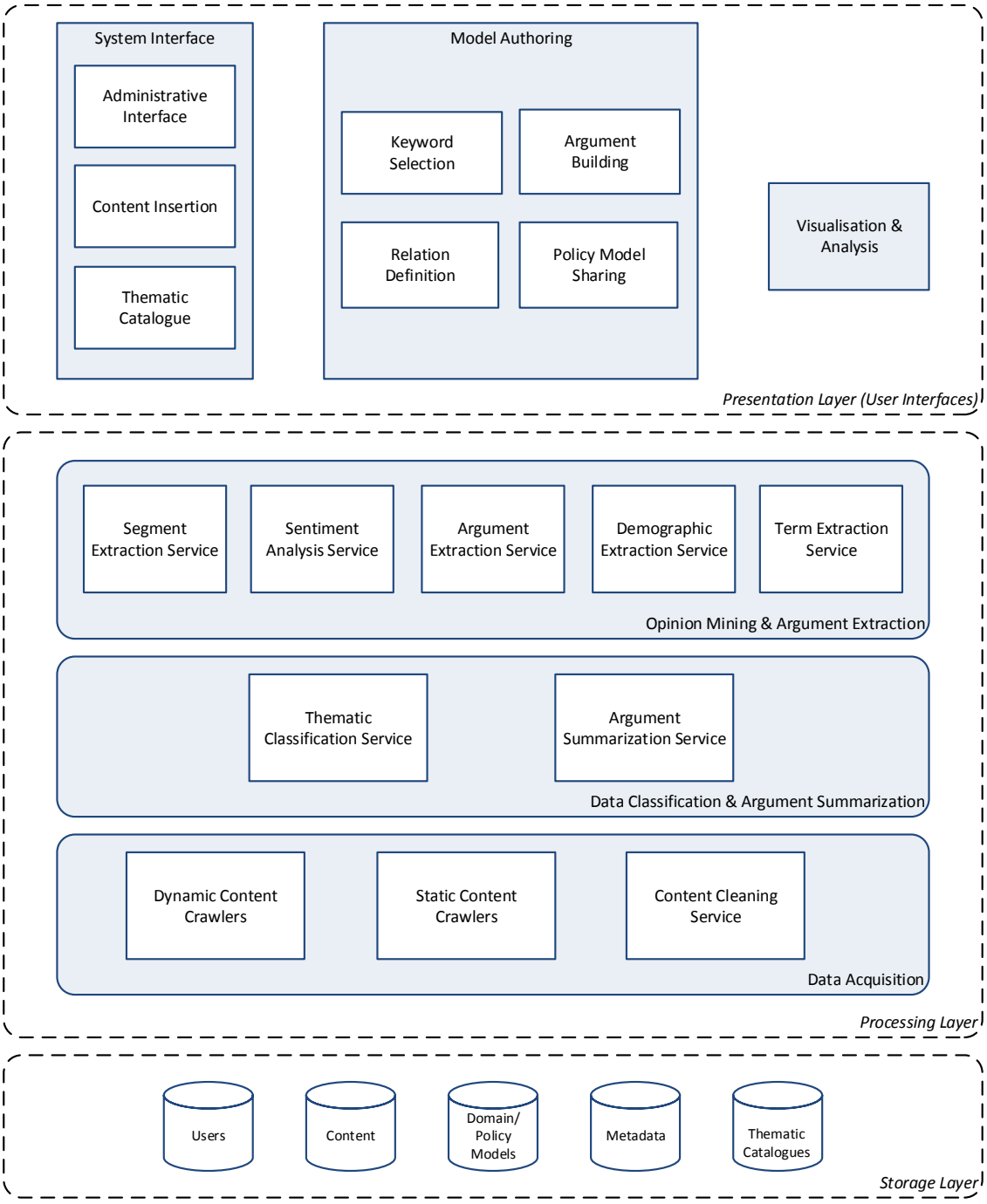

Figure 1: The NOMAD Logical Layered Architecture

Going to the real world, the NOMAD Architecture decouples the processing from the presentation layer, as depicted in Figure 2. The logical components and their associated functionalities are mapped to physical components, which, in turn, govern the way that the NOMAD data flow is performed.

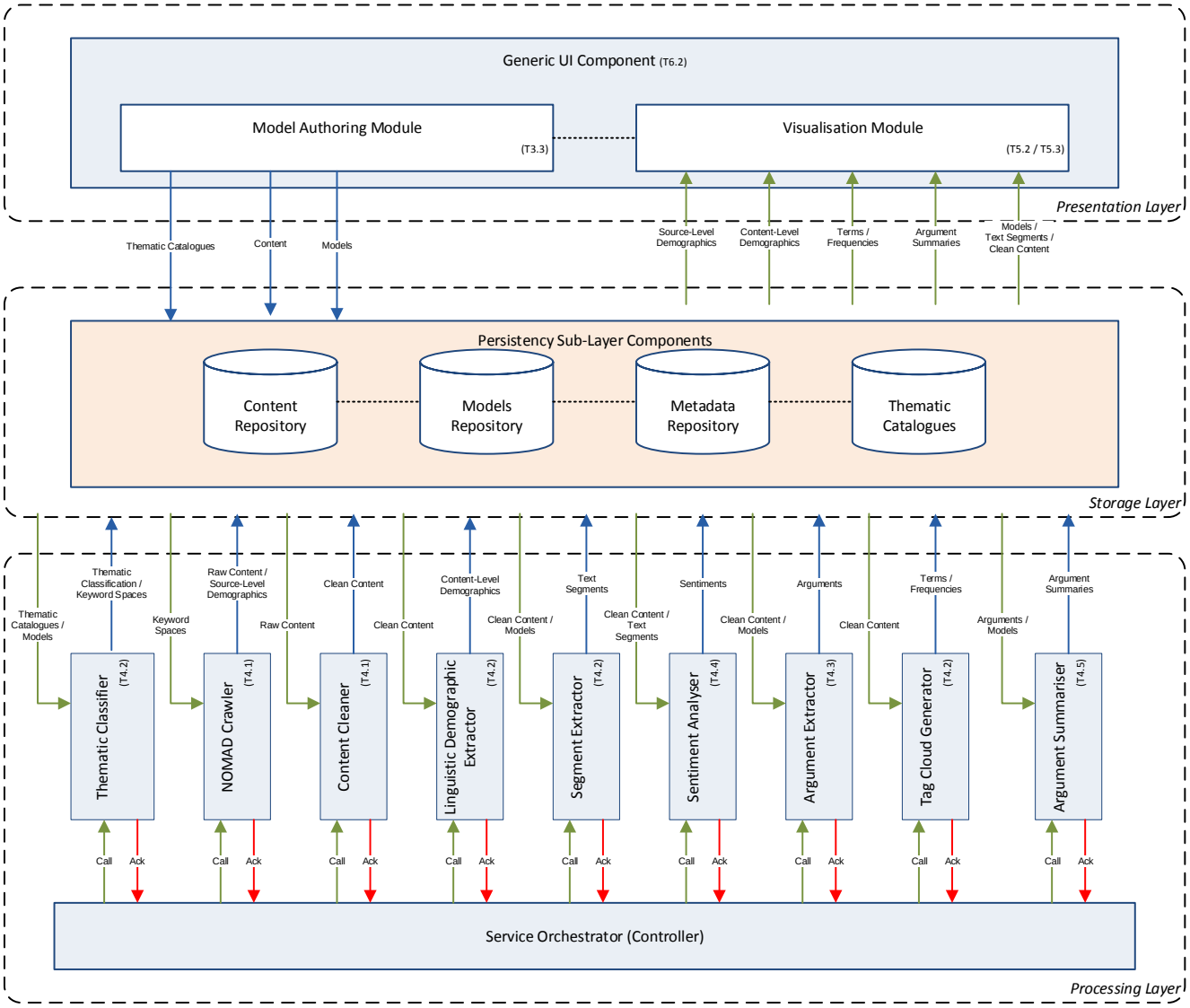

Figure 2: The Software Components of the NOMAD Physical Architecture

The presentation layer consists of two main components, which enable the interaction of the target stakeholders with the analysed NOMAD specific content. These components facilitate different needs and aim to assist stakeholders in authoring the models for describing domains and policies of interest, on the one hand, and visualising the structuring of the social media content based on these models on the other hand.

The processing layer components support the presentation layer functionalities by running on the background and analysing the crawled Internet sources and associating them with the defined domain and policy models.

As per Figure 2, the following **physical components** have been identified for the NOMAD platform, which map to the logical functionalities of the NOMAD platform in Figure 1:

- On the Presentation Layer:
  - The Model Authoring Module, which integrates the Keyword Selection interface, the Relation Definition interface, the Policy Model Sharing interface and Argument Building interface
  - The Visualisation Module, which implementsthe Visualisation & Analysis interface
  - The Generic UI Component, which wraps the Thematic Catalogue and the administration functionalities of the NOMAD platform
- On the Processing Layer:
  - The Thematic Classifier, which integrates the functionality of the Keyword Inflation and Thematic Classification Service in the Data Classification Sub-layer

- The NOMAD Crawler, which implements the relevant Crawling Service functionalities of the Static and Dynamic Content Crawler in the Data Acquisition Sub-layer.
- The Content Cleaner, which implements the relevant Content Cleaning Service functionalities in the Data Acquisition Sub-layer.
- The Linguistic Demographic Extractor, which implements the functionalities of the Demographic Extraction Service from the Opinion Mining &Argument Extraction Sub-layer
- The Segment Extractor, which implements the functionality of the Segment Extraction Service in the Opinion Mining &Argument Extraction Sub-layer
- The Sentiment Analyser, which implements the functionality of the Sentiment Analysis Service in the Opinion Mining &Argument Extraction Sub-layer
- The Argument Extractor, which implements the functionality of the Argument Extraction Service in the Opinion Mining &Argument Extraction Sub-layer
- The Tag Cloud Generator, which implements the functionality of the Term Extraction Service in the Opinion Mining &Argument Extraction Sub-layer
- The Argument Summariser, which implements the functionality of the Argument Summarisation Service in the Argument Summarisation Sub-layer
- The Service Orchestrator, which is responsible for controlling the delegation of tasks in the processing pipeline and the synchronisation of their outcomes.

- On the Storage Layer:
  - The Content Repository
  - The Model Repository
  - The Metadata Repository
  - The Thematic Catalogues
  - The Persistency Sub-Layer Components

Thus, compared to the initial design defined in [1], the NOMAD Architecture has now considered for an intermediate persistency sub-layer to enable access to the data stored into the Storage Layer logical components. By doing so, the complexity of the data source models is encapsulated into the functionalities of the persistency component, offering a simplified façade to be used directly by the consumers of the NOMAD components.

2.2 Physical Deployment

Based on the logical design of the NOMAD architecture in the previous section 2.1, this section elaborates on the physical deployment of the NOMAD platform and the configuration of the actual NOMAD components. More specifically, it describes the way that the various platform components communicate with the content repositories (for hosting the crawled information, the models, the thematic catalogues and the metadata analysis) in order to retrieve and store data produced during the operation of the whole platform.

The NOMAD implementation has followed the Service-Oriented paradigm and all the physical components offer system interfaces to enable interaction with remote software objects through RESTful Web Services. This means that a loosely coupled approach can be adopted for the deployment of the NOMAD platform and the support of multiple policy formulation scenarios, in which all the components are installed on separate physical servers. The interaction among the components is not direct, but it takes place via the communication with the platform repositories.

At this point, the physical deployment of the NOMAD platform is shown in Figure 3. As it can be seen, the current configuration splits the NOMAD functionalities into three main physical locations, which in turn host the following:

- The NOMAD Server, which distinguishes between fourdifferent virtual machines (VM): one for the crawling service to acquire content from selected Internet social media sources, a second one for the Argument Extraction & Opinion Mining, which hosts the respective components from the Processing Layer, a third one for hosting the persistency component and a last one for the User Interface (UI) Components. The latter incorporates the Model Authoring

Module and the administration panel for managing the NOMAD platform services (which is now on called NOMAD Generic UI). Furthermore, the persistency VM enables access to the NOMAD DB, which integrates the Storage Layer logical components (i.e. the Content Repository, the Model Repository, the Metadata Repository and the Thematic Catalogues).

- The Visualisation Server, which hosts the respective Visual Analytics Module.

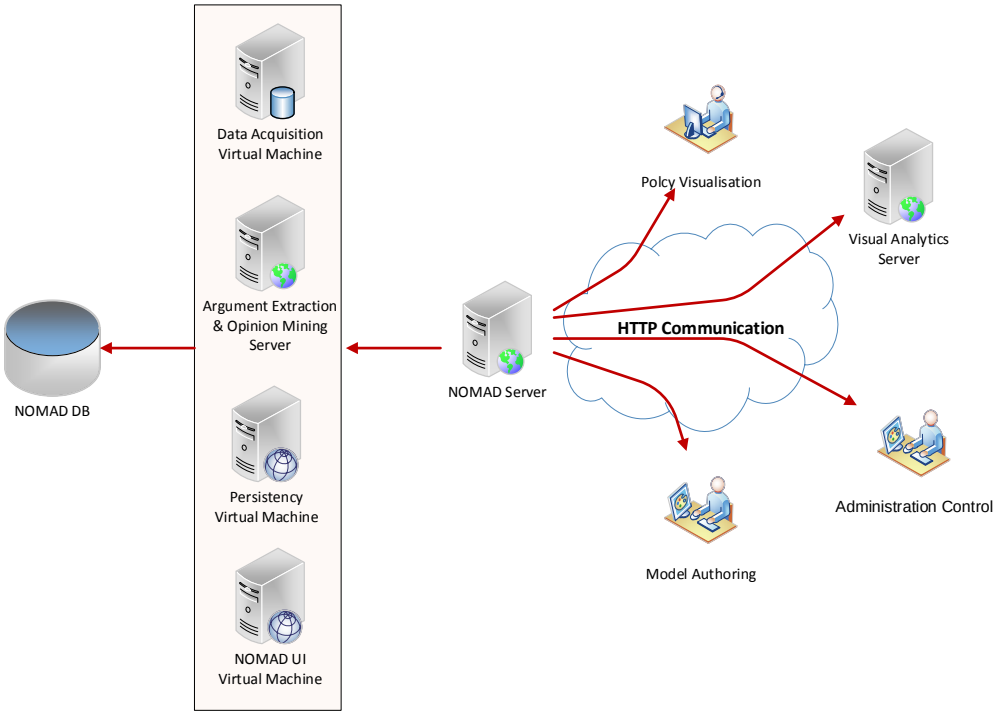

Figure 3: NOMAD Physical Architecture

2.3 Updated Information Data Flow

Figure 4 makes an update of the NOMAD Data Flow, as it has been initially introduced in [1]. This data flow describes the main steps for the accomplishment of the NOMAD functionalities, which are governed from the Service Orchestrator and are summarised in the following:

- The NOMAD Crawler collects raw material from various Internet sources, which are then cleaned (i.e. to remove html tags) through the Content Cleaner component. The crawling sources are user defined.
- Based on a defined domain model, the processing pipeline performs linguistic analysis, based on the thematic areas of the model and identifies the language dependent demographics for this model.
- The linguistic analysis is extended to the classification of the crawled content to the thematic areas of the model and the extraction of content level segments referring to the defined policy model.
- The processing involves matching the model arguments with the content segments.
- The segments are exploited to perform sentiment analysis with respect to the polarity of the segment to certain statement.
- The content is further analysed to derive key terms, which are used to produce the tag cloud of the domain and/or policy model.
- The segments associated to model specific arguments are summarised to produce a concise, abbreviated report that preserves the main points of all the arguments.

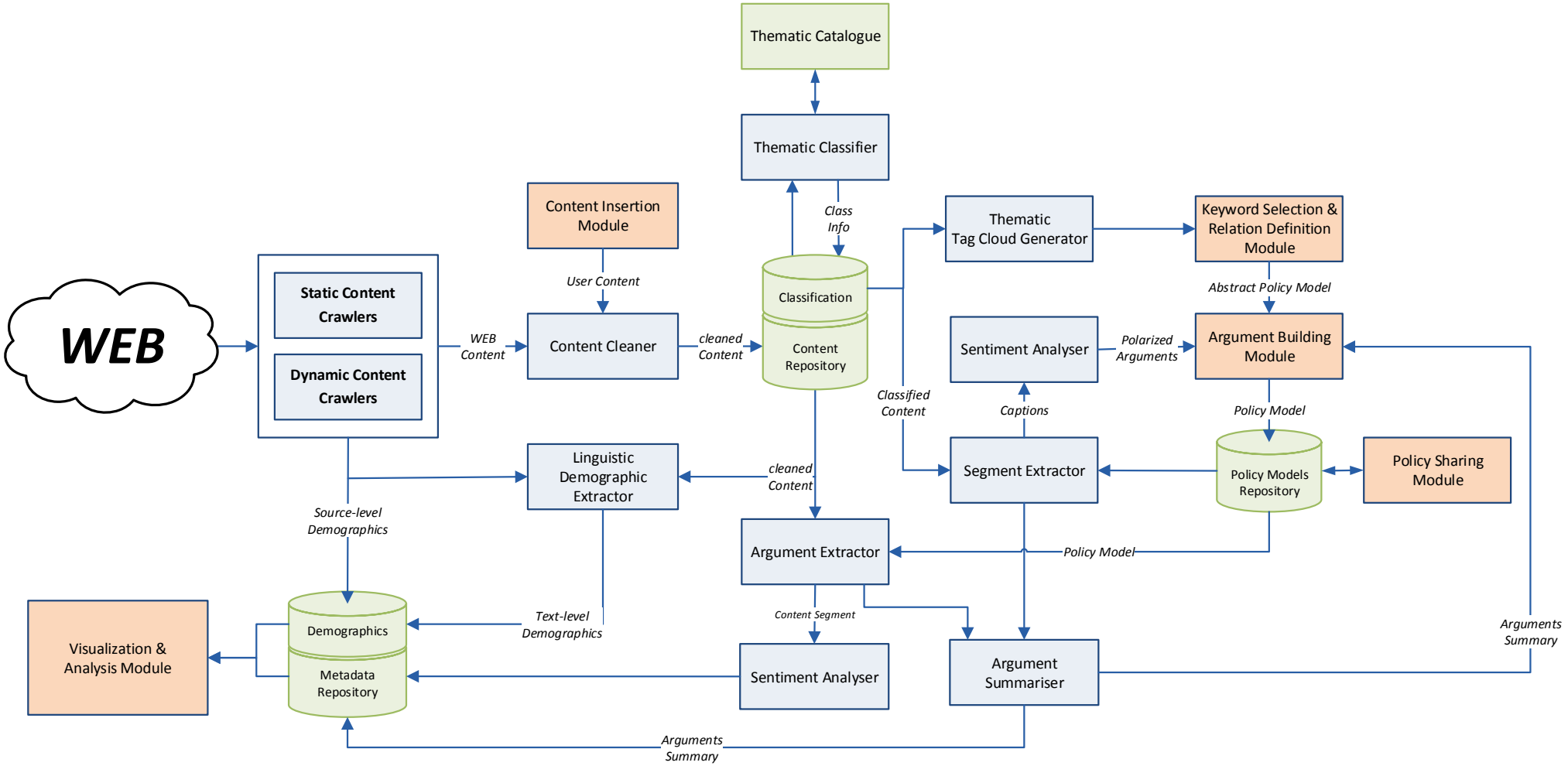

Figure 4: The NOMAD Data Flow

2.4 The NOMAD platform users

As it has been reported in [2], NOMAD targets policy makers, policy advisors and consultants, domain experts and policy communication analysts to provide them with a platform to monitor the trends of the citizens in social media with respect to specific policies and their arguments. To this end, the current implementation of the platform defines functionalities, which can be accessed from the following roles:

- Administrator: this role has full access to the whole platform functionalities, in order to maintain the operational view of the platform, manage the set of users accessing the platform and their access rights to the offered services and functionalities
- Domain Author: this role can create, modify and invoke the NOMAD analysis for a specific domain model, including the identification of the relevant keywords and the definition of relations among the key domain elements
- Policy Author: this role can create, modify and invoke the NOMAD analysis for a specific policy model, including the identification of basic policy statements and their related policy argumentations
- End User: this is a public access user, who can navigate through the results of the NOMAD analysis with respect to defined domain and policy models and their argumentations

Currently, the NOMAD platform does not make any differentiation between the Domain Author and the Policy Author, which are named as Author.It is apparent that the Administrator as a platform super user inherits all the access rights assigned to the Author, while the access rights to the End User are inherited by both the Administrator and the Author.

2.5 Realisation of the NOMAD Platform Use Cases

The analysis of user requirements and the relation to system functionalities has driven the identification of the system use cases, which can be invoked by the roles defined in section 2.4. These use cases can be realised by the interaction of these roles with the NOMAD Generic UI, the Model Authoring Module and the Visual Analytics Module. The use cases for the NOMAD Generic UI are derived from the requirements of the NOMAD platform to support the objectives of the project. The use cases for the Model Authoring Module are derived from the analysis of the work in WP3 and WP4 to offer the registered stakeholders with a tool to define the way that the attitude of the social media users towards policies can be structured. Finally, the use cases for the Visual Analytics Module are derived from the work in [1] and depict the potential interaction of the NOMAD stakeholders with the NOMAD platform to assess the behaviour of the social media users against the specified arguments attached to a particular policy model, which addresses the policy formulation needs of a certain domain.

The NOMAD system use cases are summarised in Table 1.

Table 1: The NOMAD Platform Use Cases

ID	Use Case Title	Use Case Overview	Accessed by Role
NOMAD Generic UI			
UC1	Manage User Roles	Define the list of roles accessing the platform functionalities and managing the role assigned to the registered users	Administrator
UC2	Monitor Platform Services	Have a complete view for the status of the platform services	Administrator
UC3	Check status analysis of domain and policy models	Retrieve the list of domain and policy models stored into the NOMAD DB and view whether their analysis is active or not. The Administrator has view access to all available models, while the Author can view only the public and their own models	Administrator, Author
UC4	Start/Stop analysis for the set of	Start or Stop the analysis for a defined list of domain or policy models. While the Administrator can manage all	Administrator, Author

ID	Use Case Title	Use Case Overview	Accessed by Role
	domain/policy models	models, the Author is able to start only their models	
Model Authoring Module			
UC5	Import Domain Model	Upload an external domain model (i.e. an existing ontology file - OWL) to the NOMAD platform	Author
UC6	Load Domain Models	Retrieve all the public domain models and those that are owned by the specific user. Navigate through the domain entities of these models.	Author
UC7	Edit Domain Model	Create, modify or delete a domain model owned by the specific user. Add metadata information to the domain model.Support for authoring the domain model in three languages (English, Greek and German). Based on the given access rights to a public domain model, this use case also involves the editing of a public model, which is saved as a new domain model.	Author
UC8	Edit Domain Entity	Create, modify or delete domain entities and change their relations in a certain domain model. Support for authoring the entities in three languages (English, Greek and German)	Author
UC9	Export Domain Model	Generate an ontology file for the domain model	Author
UC10	Share Domain Model	Set access rights to a domain model owned by the specific user. Define the list of users who can be granted privileged access rights	Author
UC11	Import Policy Model	Upload an external policy model (i.e. an existing ontology file - OWL) to the NOMAD platform	Author
UC12	Load Policy Models	Retrieve all the public policy models and those that are owned by the specific user. Navigate through the policy components and arguments of these models.	Author
UC13	Edit Policy Model	Create, modify or delete a policy model owned by the specific user. Add metadata information to the policy model, including the domain model to which is attached. Support for authoring the policy model in three languages (English, Greek and German). Based on the given access rights to a public policy model, this use case also involves the editing of a public model, which is saved as a new policy model.	Author
UC14	Edit Policy Component	Create, modify or delete policy components and change their relations in a certain policy model. Support for authoring the entities in three languages (English, Greek and German). Associate domain entities to the policy component	Author
UC15	Edit Argumentation	Create, modify or delete an argument to a policy component. Associate domain entities to the argument	Author
UC16	Export Policy Model	Generate an ontology file for the policy model	Author
UC17	Share Policy Model	Set access rights to a policy model owned by the specific user. Define the list of users who can be granted privileged access rights	Author
Visualisation Module			

ID	Use Case Title	Use Case Overview	Accessed by Role
UC18	Trend Discovery	View important topics in online discussions	End User
UC19	Time-comparative Trend Discovery	Observe differentiations on trending topics over time	End User
UC20	Audience-comparative Trend Discovery	Observe differentiations on trending topics across audiences	End User
UC21	In-field Trend Discovery	View important topics in field-specific discussions	End User
UC22	Temporal Comparative in-field Trend Discovery	Observe differentiations on field-specific discussions over time	End User
UC23	Audience-comparative in-field Trend Discovery	Observe differentiations on field-specific discussions across audiences	End User
UC24	In-Field Trends Projection	Acquire an estimation of trend evolution in the future	End User
UC25	Discovery of Public Sentiment towards In-field Entities	View sentiment analysis of attitude against entities related to a domain	End User
UC26	Cross-time Sentiment Changes Monitoring	Observe differentiations on sentiment towards domain-related entities over time	End User
UC27	Cross-audience Sentiment Change Monitoring	Observe differentiation on sentiment towards domain-related entities across audiences	End User
UC28	Public Sentiment Projection	Acquire an estimation of public sentiment towards domain-related entities in the future	End User
UC29	Inform Policy Model Authoring	Get polarized information related to norms and entities involved in a Policy Model	End User
UC30	Discover Public Stance against Policy Arguments	Get polarized information related to the argumentation provided by a Policy Model	End User
UC31	Monitor Public Stance against Policy Arguments	Observe differentiations in public stance towards policy arguments over time	End User
UC32	Observe Cross-Audience Stance against Policy Arguments	Observe differentiation in public stance towards policy arguments across audiences	End User
UC33	Project Argument Matching	Acquire an estimation of the public stance towards policy arguments in the future	End User

3. DESCRIPTION OF THE FIRST PROTOTYPE

This section describes the development of the services in the different layers, which constitute the first integrated prototype of the NOMAD platform. As already mentioned, NOMAD follows the logic of REST Web Services, so this section specifies the Web service method, the end point and the I/O objects for each service. It, also, gives an example of use.

3.1 Development of the Processing Layer

3.1.1 The NOMAD Service Orchestrator

The NOMAD Service Orchestrator implements the logic of breaking down the available work into batches or *threads.* It ascertains that each thread will be independently executed from other threads and that the order of execution of analysis in a specific thread will be as required by the analysis pipeline. This component allows on-demand routing of newly arrived data through the analysis pipeline of the processing layer.

The NOMAD Service Orchestrator supports status reporting and thread creating via a set of API calls as follows in the next paragraphs.

3.1.1.1 Thread status listing

- Method: GET
- Path: service_orchestrator/threads/list
- Receives: Nothing
- Returns: A JSON object

The method call returns a list of thread IDs that are currently running, their starting time of execution (number of milliseconds from specified point in time), and their status. The status is the maximum analysis level the thread items have reached. Thus, a sample returned object is as follows:

```
[{
    "thread_id": 12,
    "start_time": 1368532092683,
    "analysis_level": 5.0
},{
    "thread_id": 13,
    "start_time": 1368532000380,
    "analysis_level": 2.0
}]
```

3.1.1.2 Thread creation

- Method: POST
- Path: service_orchestrator/threads/create
- Receives: Nothing
- Returns: A JSON object

This method creates a set of threads to serve recently added data, breaking the required work into smaller processing batches. The method is used to support on-demand prioritization of work. The method call returns a list of new thread IDs that are commencing their execution, and the exact time the threads were launched. A sample return value is as follows:

```
[{
    "thread_id": 12,
    "start_time": 1368532000380
```

```
},{
    "thread_id": 13,
    "start_time": 1368532000380
}…]
```


3.1.2 Web Service implementation in the Processing Layer

This section presents the interface that the orchestrator uses to communicate with components of the processing layer. The interface consists of a set of RESTful web services which allow the orchestrator to call the various components and retrieve their processing results. The storage API is defined in terms of a prefix and a number of endpoints. It attempts to follow the principles of REST, and emits JSON documents to be parsed by the caller. Each of the following endpoints for the storage API is expected to be found on the web at `prefix + path`. For example, if the prefix was "http://server.nomad-project.eu/processing/components", then the "`argument_extraction`" endpoint would be found at "`http://server.nomad-project.eu/processing/components/argument_extraction`". Each processing component may reside under any `prefix`, which must be known to the orchestrator.

General rules are those common to most REST APIs. If a resource cannot be found, the service returns 404 NOT FOUND. If an action is not permitted for the current parameters, the service returns 401 NOT AUTHORIZED, otherwise 200 OK is returned. JSON data is sent with the header "Content-Type: application/json".

3.1.2.1 Linguistic Demographic Extractor

- Method: POST
- Path: demographic_extraction
- Receives: A JSON object
- Returns: A JSON object

This API call processes a document through the Linguistic Demographic Extractor Components, which is responsible for acquiring demographic information for the NOMAD content, in case it is not explicitly provided (e.g. from Twitter, Facebook etc.).

The input JSON object must contain the following information:

- url: The url of the document to be processed (string).
- clean_text: The clean text of the document to be processed (string).
- timestamp: The crawling timestamp of the document to be processed (timestamp).
- language: The language of the document to be processed (string).
- thread_id: The id of the processing thread, managed by the orchestrator (integer).
- processing_level: The processing level, managed by the orchestrator, which denotes how many components have already be run, prior to this call (integer).
- source_table: The source table id (integer).
- query_id: The id of the query associated with this processing call (integer). This id can be used by the component to query the term of the query, and the associated to the query arguments, domains, entities, norms, policies, and themes.

A sample input JSON object can be:


```
{
    "url": "http://some.url.eu",
    "clean_text": "the clean text of the document",
    "timestamp": "13/05/2013 22:15:00",
    "language": "el",
    "thread_id": 2,
    "processing_level": 10,
```

```
    "source_table": 12,
    "query_id": 1234
}
```

The output is a JSON object, which contains demographic information:

```
{
    "age": "18-24",
    "gender": "male",
    "education": "College",
    "region": "Athens",
    "geonames_id": "<some id>"
}
```

3.1.2.2 Segment Extractor

- Method: POST
- Path: segment_extraction
- Receives: A JSON object
- Returns: A JSON object

This API call processes a document through the Segment Extractor Component, which analyses the acquired content and associates specific segments with the relevant constituents of a model, via the association of the query with the relevant models.

The input JSON object must contain the following information:

- url: The url of the document to be processed (string).
- clean_text: The clean text of the document to be processed (string).
- timestamp: The crawling timestamp of the document to be processed (timestamp).
- language: The language of the document to be processed (string).
- thread_id: The id of the processing thread, managed by the orchestrator (integer).
- processing_level: The processing level, managed by the orchestrator, which denotes how many components have already be run, prior to this call (integer).
- source_table: The source table id (integer).
- query_id: The id of the query associated with this processing call (integer). This id can be used by the component to query the term of the query, and the associated to the query arguments, domains, entities, norms, policies, and themes.

A sample input JSON object can be:

```
{
    "url": "http://some.url.eu",
    "clean_text": "the clean text of the document",
    "timestamp": "13/05/2013 22:15:00",
    "language": "el",
    "thread_id": 2,
    "processing_level": 10,
    "source_table": 12,
    "query_id": 1234
}
```

The output is a JSON object, which contains a set of segments (encoded as JSON objects):

```
[{
```

```
    "text": "the argument text",
    "start_index": 10,
    "end_index": 28,
},{
    "text": "the argument2 text",
    "start_index": 50,
    "end_index": 78,
}]
```

3.1.2.3 Sentiment Analyser (document level)

- Method: POST
- Path: sentiment_analysis/overal
- Receives: A JSON object
- Returns: A JSON object

This API call processes a document through the Sentiment Analyser, which discovers the polarity of a document towards an argument.

The input JSON object must contain the following information:

- url: The url of the document to be processed (string).
- clean_text: The clean text of the document to be processed (string).
- timestamp: The crawling timestamp of the document to be processed (timestamp).
- language: The language of the document to be processed (string).
- thread_id: The id of the processing thread, managed by the orchestrator (integer).
- processing_level: The processing level, managed by the orchestrator, which denotes how many components have already be run, prior to this call (integer).
- source_table: The source table id (integer).
- query_id: The id of the query associated with this processing call (integer). This id can be used by the component to query the term of the query, and the associated to the query arguments, domains, entities, norms, policies, and themes.

A sample input JSON object can be:

```
{
    "url": "http://some.url.eu",
    "clean_text": “the clean text of the document”,
    "timestamp": "13/05/2013 22:15:00",
    "language": "el",
    "thread_id": 2,
    "processin_level": 10,
    "source_table": 12,
    "query_id": 1234
}
```

The output is a JSON object, which contains the overall sentiment of the document as a whole:

```
{
    "sentiment": 0
}
```

3.1.2.4 Sentiment Analyser (segment level)

- Method: POST
- Path: sentiment_analysis/segment
- Receives: A JSON object
- Returns: A JSON object

This API call processes a segment through the Sentiment Analyser, which discovers the polarity of the segment towards an argument..

The input JSON object must contain the following information:

- id: The id of the segment (integer).
- text: The text of the segment to be processed (string).
- start_index: The character offset of the segment start (integer).
- end_index: The character offset of the segment start (integer).
- language: The language of the document to be processed (string).
- thread_id: The id of the processing thread, managed by the orchestrator (integer).
- processing_level: The processing level, managed by the orchestrator, which denotes how many components have already be run, prior to this call (integer).
- source_table: The source table id (integer).
- query_id: The id of the query associated with this processing call (integer). This id can be used by the component to query the term of the query, and the associated to the query arguments, domains, entities, norms, policies, and themes.

A sample input JSON object can be:

```
{
    "id": 1234,
    "text": "the text of the segment",
    "start_index": 0,
    "end_index": 0,
    "language": "el",
    "thread_id": 2,
    "processing_level": 10,
    "source_table": 12,
    "query_id": 1234
}
```

The output is a JSON object, which contains the overall sentiment of the document as a whole:

```
{
    "sentiment": 0
}
```

3.1.2.5 Tag Cloud Generator

- Method: POST
- Path: term_extraction
- Receives: A JSON object
- Returns: A JSON object

This API call processes a document through the Tag Cloud Generator Component, which discovers the dominant terms in the NOMAD Content entries and computes their frequency of appearance and their information weight in the specific content.

The input JSON object must contain the following information:

- url: The url of the document to be processed (string).
- clean_text: The clean text of the document to be processed (string).
- timestamp: The crawling timestamp of the document to be processed (timestamp).
- language: The language of the document to be processed (string).
- sentiment: The sentiment of the whole document (integer).
- thread_id: The id of the processing thread, managed by the orchestrator (integer).
- processing_level: The processing level, managed by the orchestrator, which denotes how many components have already be run, prior to this call (integer).
- source_table: The source table id (integer).
- query_id: The id of the query associated with this processing call (integer). This id can be used by the component to query the term of the query, and the associated to the query arguments, domains, entities, norms, policies, and themes.

A sample input JSON object can be:

```
{
    "url": "http://some.url.eu",
    "clean_text": "the clean text of the document",
    "timestamp": "13/05/2013 22:15:00",
    "language": "el",
    "sentiment": 0,
    "thread_id": 2,
    "processing_level": 10,
    "source_table": 12,
    "query_id": 1234
}
```

The output is a JSON object, which contains a set of terms (encoded as JSON objects):

```
[{
    "term_string": "the term text",
    "term_frequency": 10,
    "tf_idf": 0.1234,
    "content_id": 12
},{
    "term_string": "the term2 text",
    "term_frequency": 1,
    "tf_idf": 0.01234,
    "content_id": 10
}]
```

3.1.2.6 Argument Extractor

- Method: POST
- Path: argument_extraction
- Receives: A JSON object
- Returns: A JSON object

This API call processes a document through the Argument Extractor Component, which discovers segments in the provided content that pertain to the arguments provided by a policy model.

The input JSON object must contain the following information:

- url: The url of the document to be processed (string).
- clean_text: The clean text of the document to be processed (string).
- timestamp: The crawling timestamp of the document to be processed (timestamp).
- language: The language of the document to be processed (string).
- thread_id: The id of the processing thread, managed by the orchestrator (integer).
- processing_level: The processing level, managed by the orchestrator, which denotes how many components have already be run, prior to this call (integer).
- source_table: The source table id (integer).
- query_id: The id of the query associated with this processing call (integer). This id can be used by the component to query the term of the query, and the associated to the query arguments, domains, entities, norms, policies, and themes.

A sample input JSON object can be:

```
{
    "url": "http://some.url.eu",
    "clean_text": "the clean text of the document",
    "timestamp": "13/05/2013 22:15:00",
    "language": "el",
    "thread_id": 2,
    "processing_level": 10,
    "source_table": 12,
    "query_id": 1234
}
```

The output is a JSON object, which contains a set of segments (encoded as JSON objects), which represent the arguments:

```
[{
    "text": "the argument text",
    "start_index": 10,
    "end_index": 28,
    "argument_id": 12
},{
    "text": "the argument2 text",
    "start_index": 50,
    "end_index": 78,
    "argument_id": 1254
}]
```

3.1.2.7 Argument Summariser

- Method: POST
- Path: argument_summarisation
- Receives: A JSON object
- Returns: A JSON object

This API call processes a set of arguments through the Argument Summariser Component, which groups and summarises arguments related to a policy model query. The summary of arguments, related to the query, consist of a set of representative text snippets. Each representative snippet is expected to reflect the essence of a set of (partially or fully overlapping) expressed arguments. The return value of the call also integrates information about the importance of a

snippet. For example, a snippet representing 10 expressed arguments can be more important than a snippet representing 1 expressed argument. The importance is a floating point number, which can be a function of different importance indicators.

The input JSON object should contain is expected to contain the following information:

- query_id: The id of the query associated with this processing call (integer). This id can be used by the component to access the associated to the query arguments, domains, entities, norms, policies, and themes.

A sample input JSON object can be:

```
{
    "query_id": 1234
}
```

The output is a JSON object, which contains a representative list of segments (encoded as JSON objects), efficiently the query related arguments. A sample output is as follows:

```
[{
    "text": "the summarised argument text",
    "start_index": 10,
    "end_index": 28,
    "argument_id": 12,
    "importance": 5.0
},{
    "text": "the summarised text of the second argument set",
    "start_index": 50,
    "end_index": 78,
    "argument_id": 18,
    "importance": 2.0
}]
```

The “argument_id” field is optional and, if present, indicates the source of the summary text.

3.2 Development of the Presentation Layer

3.2.1 The NOMAD Generic UI Component

The main User Interface development environment has been deployed using Liferay [3]. Liferay is a free and open source enterprise portal written in Java and distributed under the GNU Lesser General Public License. It allows users to set up features common to websites and easily create corporate intranets and extranets. Liferay portal is also a content management system that allows users store, retrieve and edit documents and web content for their portals. Although Liferay offers a sophisticated programming interface for developers, no programming skills are required for basic website installation and administration.

The main parts of a portal are functional units that are called portlets. With Liferay portal, it also possible for developers to implement themes that define the look n’ feel of the web application’s pages, hooks that extend the portal’s abilities and layout templates that define the way portlets are spread inside a portal page. The development of these units is possible through a series of plugins provided freely by Liferay for the Eclipse Integrated Development Environment (IDE) [4].

Liferay Portal is Java based and runs on any computing platform capable of running the Java Runtime Environment and an application server. A number of application servers, databases technologies and open standards are supported, fact that gives the opportunity for a combination that best suits to the needs of each project (see [5] for a full list of technologies used and supported by current version of Liferay portal). Liferay portal comes out in two editions, a community edition freely available for use and an enterprise edition with a certain price.

In NOMAD, the Liferay Portal Community Edition, bundled with the Apache Tomcat server [6], has been chosen, in order to keep the under development web application light weighted yet powerful in terms of performance. This Liferay edition exhibits key features, necessary for the Argument Modelling Environment, which are summarised in the next paragraphs (a complete list of Liferay features and more detailed information can be found in [7]).

Simplified UI Development

Liferay Portal simplifies the development of internal, external, and channel websites and Web Application, notably those that allow users to login for personalized services or views and those that require a workflow approval process to update content and integrate or aggregate multiple existing services. Liferay Portal provides a single presentation layer for integrating all enterprise systems into a single easy to use interface for end users.

User-Driven Workflow and Approval

Not only is there embedded workflow for content, but also Liferay Portal allows users create their own workflow and define the number of approval paths based on their own unique business requirements and operational needs. For example, administrators can now implement an approval process for new document uploads before they appear in the Document Library

User Groups, Organizations and Sites

Liferay users can be intuitively grouped into a hierarchy of "organizations" or cross-organizational "user groups," providing flexibility and ease of administration.

For example, members of different geographic location, such as America and EMEA can be grouped into organizations, whereas project based or departmental teams such as a "Website redesign" that cross disciplines can be created as user groups. Liferay provides support for "sites" where both organizations and user groups can be added to a separate web property with its own set of pages, content management system, shared calendar, and authorizations. A user can belong to multiple sites and easily navigate between them.

Multi-language Support

International or multi-lingual organizations get out of the box support for 30+ languages. Users can toggle between different language settings with just one click. This is of an additional asset for NOMAD, which targets to the provision of multilingual functionalities for authoring policy and domain models.

User Defined Content Categories

Administrators can create their own custom metadata sets and document types into language familiar to users. For example, financial reports, surveillance videos, and so on can be defined in a way that makes sense for the business functions of the content (e.g., author, reporting period, etc.).

Web Content Structures and Templates

Liferay Portal allows users to easily create reusable templates for web pages and page sections. This enables users to quickly build pages and allows websites to maintain a common look and feel across the entire site by allowing new pages to be created with an approved set of templates. Users can quickly create templates within Liferay Web UI or by using external web development tools.

Community Features

Liferay offers wiki functionality to enable building up and documenting important or interesting information for community use. Each community can get their own wiki with its own set of authorizations. Anyone with editing rights can quickly contribute information to these online topical encyclopaedias.

Liferay Blogs features provide the best option of modern blogging tools coupled with the benefits of the community-centric nature of Liferay Portal. They allow users to convey information and facilitate conversations around blogs directly in the context of a website, intranet, extranet, or social network. Features include a user-friendly rich text editor, social bookmarking links, email notifications of blog replies, and an entry rating system. All blogs can be subscribed to via RSS and users can schedule entries to be published at specific times and dates.

Finally, Liferay community toolset offers multiple choice polls, which can be created with this tool to maintain a voting system. Many separate polls can be managed and a separate portlet can be configured to display a specific poll's results.

The communication of the Persistency Layer with the Liferay Development Environment has been built as resources exposed via the http protocol. These resources consist of the RESTful Web Services that on-demand expose the contents of the NOMAD Database to the presentation layer. For this purpose, the use of JAX-RS [8], Jersey [9] and JAXB [10] has been taken into consideration.

3.2.2 The Model Authoring Module

The Model Authoring Module facilitates domain experts and policy makers in defining the models to describe policies. The Module offers a visual representation of the policy argumentation models and assists policy makers in expressing their understanding on the policy formulation aspects on a conceptual level and delivering a machine-readable representation of the respective models.

The activities for the development of this module are described in the related documentation of WP3.

3.2.3 The Visualisation Module

The Visualisation Module is used by all NOMAD users to navigate through the results of the processing analysis towards associating the policy argumentation models with the opinions expressed in social media. The Visualisation Module considers different views on presenting the results to the end users, taking into advantage state-of-the-art technologies in visual analytics and exploiting various data streams coming from the analysis of the NOMAD processing layer components.

The activities for the development of this module are described in the related documentation of WP5.

3.2.4 Web Service Implementation in the Presentation Layer

This section describes the Web services, which facilitate the functionalities for the Presentation Layer Components. For each service, the following information is provided:

- A short description of the Web Service
- The HTTP method used
- The endpoint
- The I/O schema

3.2.4.1 Get domain/policy/norm/entity

Description	It returns a tree structure of a model (i.e. domain, policy,norm,entity)
Method	POST
Endpoint	http://nomad.atc.gr/nomadWS/services/visualization/modelStructure
Input	*application/json, application/xml*

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":false,
   "properties":{
      "request": {
         "type":"object",
         "id": "http://nomad.jsonschema.org/request",
         "required":false,
         "properties":{
            "category": {
               "type":"object",
               "id": "http://nomad.jsonschema.org/inputParameter/category",
```

```
            "required":true,
            "properties":{
               "modelID": {
                  "type":"number",
                  "id": "http://nomad.jsonschema.org/inputParameter/category/modelID",
                  "required":true
               },
               "modelType": {
                  "type":"string",
                  "id": "http://nomad.jsonschema.org/inputParameter/category/modelType",
                  "required":true
               }
            }
         },
         "demography": {
            "type":"object",
            "id": "http://nomad.jsonschema.org/inputParameter/demography",
            "required":false,
            "properties":{
               "age": {
                  "type":"object",
                  "id": "http://nomad.jsonschema.org/inputParameter/demography/age",
                  "required":false,
                  "properties":{
                     "max": {
                        "type":"number",
                        "id":
"http://nomad.jsonschema.org/inputParameter/demography/age/max",
                        "required":true
                     },
                     "min": {
                        "type":"number",
                        "id":
"http://nomad.jsonschema.org/inputParameter/demography/age/min",
                        "required":true
                     }
                  }
               },
               "education": {
                  "type":"array",
                  "id":
"http://nomad.jsonschema.org/inputParameter/demography/education",
                  "required":false,
                  "items":
                     {
                        "type":"string",
                        "id":
"http://nomad.jsonschema.org/inputParameter/demography/education/0",
                        "required":false
```

```
                    }


                },
                "gender": {
                   "type":"array",
                   "id": "http://nomad.jsonschema.org/inputParameter/demography/gender",
                   "required":false,
                   "items":
                      {
                         "type":"string",
                         "id":
"http://nomad.jsonschema.org/inputParameter/demography/gender/0",
                         "required":false
                      }


                },
                "geonames_id": {
                   "type":"array",
                   "id":
"http://nomad.jsonschema.org/inputParameter/demography/geonames_id",
                   "required":false,
                   "items":
                      {
                         "type":"number",
                         "id":
"http://nomad.jsonschema.org/inputParameter/demography/geonames_id/0",
                         "required":false
                      }


                },
                "region_name": {
                   "type":"array",
                   "id":
"http://nomad.jsonschema.org/inputParameter/demography/region_name",
                   "required":false,
                   "items":
                      {
                         "type":"string",
                         "id":
"http://nomad.jsonschema.org/inputParameter/demography/region_name/0",
                         "required":false
                      }


                }
             }
```

```
            },
            "languageCode": {
               "type":"string",
               "id": "http://nomad.jsonschema.org/inputParameter/languageCode",
               "required":true
            },
            "number of terms": {
               "type":"number",
               "id": "http://nomad.jsonschema.org/inputParameter/number of terms",
               "required":true
            },
            "timespan": {
               "type":"object",
               "id": "http://nomad.jsonschema.org/inputParameter/timespan",
               "required":false,
               "properties":{
                  "begin": {
                     "type":"string",
                     "id": "http://nomad.jsonschema.org/inputParameter/timespan/begin",
                     "required":false
                  },
                  "end": {
                     "type":"string",
                     "id": "http://nomad.jsonschema.org/inputParameter/timespan/end",
                     "required":false
                  }
               }
            }
         }
      }
   }
}
```


Output	*application/json, application/xml*


```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":false,
   "properties":{
      "modelStructure": {
         "type":"object",
         "id": "http://nomad.jsonschema.org/modelStructure",
         "required":true,
         "properties":{
            "children": {
               "type":"array",
               "id": "http://nomad.jsonschema.org/modelStructure/children",
               "required":true,
```

```
"minitems": "0",
              "maxitems": "-1",
              "items":
                 {
                 "$ref":"modelStructure"
           },
           "description": {
              "type":"string",
              "id": "http://nomad.jsonschema.org/modelStructure/description",
              "required":true
           },
           "modelID": {
              "type":"number",
              "id": "http://nomad.jsonschema.org/modelStructure/modelID",
              "required":true
           },
           "name": {
              "type":"string",
              "id": "http://nomad.jsonschema.org/modelStructure/name",
              "required":true
           },
           "numberOfSegments": {
              "type":"number",
              "id": "http://nomad.jsonschema.org/modelStructure/numberOfSegments",
              "required":true
           },
           "sentiment": {
              "type":"number",
              "id": "http://nomad.jsonschema.org/modelStructure/sentiment",
              "required":true
           },
           "type": {
              "type":"string",
              "id": "http://nomad.jsonschema.org/modelStructure/type",
              "required":true
           }
        }
     }
   }
}
```

3.2.4.2 Get term frequencies

Description	It returns the term frequencies related to a model (domain, policy)
Method	POST
Endpoint	http://nomad.atc.gr/nomadWS/services/visualization/termFrequencies
Input	*application/json, application/xml*
{	

```
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":false,
   "properties":{
      "inputParameter": {
         "type":"object",
         "id": "http://nomad.jsonschema.org/inputParameter",
         "required":false,
         "properties":{
            "category": {
               "type":"object",
               "id": "http://nomad.jsonschema.org/inputParameter/category",
               "required":true,
               "properties":{
                  "modelID": {
                     "type":"number",
                     "id": "http://nomad.jsonschema.org/inputParameter/category/modelID",
                     "required":true
                  },
                  "modelType": {
                     "type":"string",
                     "id": "http://nomad.jsonschema.org/inputParameter/category/modelType",
                     "required":true
                  }
               }
            },
            "demography": {
               "type":"object",
               "id": "http://nomad.jsonschema.org/inputParameter/demography",
               "required":false,
               "properties":{
                  "age": {
                     "type":"object",
                     "id": "http://nomad.jsonschema.org/inputParameter/demography/age",
                     "required":false,
                     "properties":{
                        "max": {
                           "type":"number",
                           "id":
"http://nomad.jsonschema.org/inputParameter/demography/age/max",
                           "required":true
                        },
                        "min": {
                           "type":"number",
                           "id":
"http://nomad.jsonschema.org/inputParameter/demography/age/min",
                           "required":true
                        }
```

```
                    }
                },
                "education": {
                    "type":"array",
                    "id":
"http://nomad.jsonschema.org/inputParameter/demography/education",
                    "required":false,
                    "items":
                        {
                            "type":"string",
                            "id":
"http://nomad.jsonschema.org/inputParameter/demography/education/0",
                            "required":false
                        }
                },
                "gender": {
                    "type":"array",
                    "id": "http://nomad.jsonschema.org/inputParameter/demography/gender",
                    "required":false,
                    "items":
                        {
                            "type":"string",
                            "id":
"http://nomad.jsonschema.org/inputParameter/demography/gender/0",
                            "required":false
                        }
                },
                "geonames_id": {
                    "type":"array",
                    "id":
"http://nomad.jsonschema.org/inputParameter/demography/geonames_id",
                    "required":false,
                    "items":
                        {
                            "type":"number",
                            "id":
"http://nomad.jsonschema.org/inputParameter/demography/geonames_id/0",
                            "required":false
                        }
                },
                "region_name": {
                    "type":"array",
                    "id":
"http://nomad.jsonschema.org/inputParameter/demography/region_name",
                    "required":false,
                    "items":
                        {
                            "type":"string",
                            "id":
"http://nomad.jsonschema.org/inputParameter/demography/region_name/0",
```

```
                        "required":false
                     }
                  }
               }
            },
            "languageCode": {
               "type":"string",
               "id": "http://nomad.jsonschema.org/inputParameter/languageCode",
               "required":true
            },
            "number of terms": {
               "type":"number",
               "id": "http://nomad.jsonschema.org/inputParameter/number of terms",
               "required":true
            },
            "timespan": {
               "type":"object",
               "id": "http://nomad.jsonschema.org/inputParameter/timespan",
               "required":false,
               "properties":{
                  "begin": {
                     "type":"string",
                     "id": "http://nomad.jsonschema.org/inputParameter/timespan/begin",
                     "required":false
                  },
                  "end": {
                     "type":"string",
                     "id": "http://nomad.jsonschema.org/inputParameter/timespan/end",
                     "required":false
                  }
               }
            }
         }
      }
   }
}
```


Output	*application/json, application/xml*


```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":false,
   "properties":{
      "frequencies": {
         "type":"array",
         "id": "http://nomad.jsonschema.org/frequencies",
         "required":true,
         "items":
```

```
            {
               "type":"object",
               "id": "http://nomad.jsonschema.org/frequencies/0",
               "required":false,
               "properties":{
                  "frequency": {
                     "type":"number",
                     "id": "http://nomad.jsonschema.org/frequencies/0/frequency",
                     "required":false
                  },
                  "sentiment": {
                     "type":"number",
                     "id": "http://nomad.jsonschema.org/frequencies/0/sentiment",
                     "required":false
                  },
                  "term": {
                     "type":"string",
                     "id": "http://nomad.jsonschema.org/frequencies/0/term",
                     "required":false
                  },
                  "tf_idf": {
                     "type":"number",
                     "id": "http://nomad.jsonschema.org/frequencies/0/tf_idf",
                     "required":false
                  }
               }
            }
      }
   }
}
```


3.2.4.3 Get database info

Description	It returns the different demographic values.
Method	GET
Endpoint	http://nomad.atc.gr/nomadWS/services/visualization/dbInfo
Input	*application/json, application/xml*
-	
Output	*application/json, application/xml*


```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":false,
   "properties":{
      "databaseStats": {
         "type":"object",
         "id": "http://nomad.jsonschema.org/databaseStats",
```

```
        "required":true,
        "properties":{
           "age": {
              "type":"object",
              "id": "http://nomad.jsonschema.org/databaseStats/age",
              "required":false,
              "properties":{
                 "max": {
                    "type":"number",
                    "id": "http://nomad.jsonschema.org/databaseStats/age/max",
                    "required":false
                 },
                 "min": {
                    "type":"number",
                    "id": "http://nomad.jsonschema.org/databaseStats/age/min",
                    "required":false
                 }
              }
           },
           "education": {
              "type":"array",
              "id": "http://nomad.jsonschema.org/databaseStats/education",
              "required":false,
              "items":
                 {
                    "type":"string",
                    "id": "http://nomad.jsonschema.org/databaseStats/education/0",
                    "required":false
                 }
           },
           "geonames_id": {
              "type":"array",
              "id": "http://nomad.jsonschema.org/databaseStats/geonames_id",
              "required":false,
              "items":
                 {
                    "type":"string",
                    "id": "http://nomad.jsonschema.org/databaseStats/geonames_id/0",
                    "required":false
                 }
           },
           "language_id": {
              "type":"array",
              "id": "http://nomad.jsonschema.org/databaseStats/language_id",
              "required":false,
              "items":
                 {
                    "type":"string",
```

```
                    "id": "http://nomad.jsonschema.org/databaseStats/language_id/0",
                    "required":false
                }
            },
            "region_name": {
               "type":"array",
               "id": "http://nomad.jsonschema.org/databaseStats/region_name",
               "required":false,
               "items":
                  {
                     "type":"string",
                     "id": "http://nomad.jsonschema.org/databaseStats/region_name/0",
                     "required":false
                  }
            }
         }
      }
   }
}
```

3.2.4.4 Get bucket size

Description	It returns the number of segments and the average sentiment of a domain/policy/entity/norm/argument
Method	POST
Endpoint	http://nomad.atc.gr/nomadWS/services/visualization /groupInfo
Input	*application/json, application/xml*

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":false,
   "properties":{
      "inputParameter": {
         "type":"object",
         "id": "http://nomad.jsonschema.org/inputParameter",
         "required":false,
         "properties":{
            "category": {
               "type":"object",
               "id": "http://nomad.jsonschema.org/inputParameter/category",
               "required":true,
               "properties":{
                  "modelID": {
                     "type":"number",
                     "id": "http://nomad.jsonschema.org/inputParameter/category/modelID",
                     "required":true
                  },
```

```
                "modelType": {
                   "type":"string",
                   "id": "http://nomad.jsonschema.org/inputParameter/category/modelType",
                   "required":true
                }
             }
          },
          "demography": {
             "type":"object",
             "id": "http://nomad.jsonschema.org/inputParameter/demography",
             "required":false,
             "properties":{
                "age": {
                   "type":"object",
                   "id": "http://nomad.jsonschema.org/inputParameter/demography/age",
                   "required":false,
                   "properties":{
                      "max": {
                         "type":"number",
                         "id":
"http://nomad.jsonschema.org/inputParameter/demography/age/max",
                         "required":true
                      },
                      "min": {
                         "type":"number",
                         "id":
"http://nomad.jsonschema.org/inputParameter/demography/age/min",
                         "required":true
                      }
                   }
                },
                "education": {
                   "type":"array",
                   "id":
"http://nomad.jsonschema.org/inputParameter/demography/education",
                   "required":false,
                   "items":
                      {
                         "type":"string",
                         "id":
"http://nomad.jsonschema.org/inputParameter/demography/education/0",
                         "required":false
                      }
                },
                "gender": {
                   "type":"array",
                   "id": "http://nomad.jsonschema.org/inputParameter/demography/gender",
                   "required":false,
                   "items":
```

```
                        {
                           "type":"string",
                           "id":
"http://nomad.jsonschema.org/inputParameter/demography/gender/0",
                           "required":false
                        }
                  },
                  "geonames_id": {
                     "type":"array",
                     "id":
"http://nomad.jsonschema.org/inputParameter/demography/geonames_id",
                     "required":false,
                     "items":
                        {
                           "type":"number",
                           "id":
"http://nomad.jsonschema.org/inputParameter/demography/geonames_id/0",
                           "required":false
                        }
                  },
                  "region_name": {
                     "type":"array",
                     "id":
"http://nomad.jsonschema.org/inputParameter/demography/region_name",
                     "required":false,
                     "items":
                        {
                           "type":"string",
                           "id":
"http://nomad.jsonschema.org/inputParameter/demography/region_name/0",
                           "required":false
                        }
                  }
               }
            },
            "languageCode": {
               "type":"string",
               "id": "http://nomad.jsonschema.org/inputParameter/languageCode",
               "required":true
            },
            "number of terms": {
               "type":"number",
               "id": "http://nomad.jsonschema.org/inputParameter/number of terms",
               "required":true
            },
            "timespan": {
               "type":"object",
               "id": "http://nomad.jsonschema.org/inputParameter/timespan",
               "required":false,
```

```
            "properties":{
               "begin": {
                  "type":"string",
                  "id": "http://nomad.jsonschema.org/inputParameter/timespan/begin",
                  "required":false
               },
               "end": {
                  "type":"string",
                  "id": "http://nomad.jsonschema.org/inputParameter/timespan/end",
                  "required":false
               }
            }
         }
      }
   }
}
```


Output	*application/json, application/xml*


```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":false,
   "properties":{
      "groupInfo": {
         "type":"object",
         "id": "http://nomad.jsonschema.org/groupInfo",
         "required":true,
         "properties":{
            "numberOfSegments": {
               "type":"number",
               "id": "http://nomad.jsonschema.org/groupInfo/numberOfSegments",
               "required":false
            },
            "sentiment": {
               "type":"number",
               "id": "http://nomad.jsonschema.org/groupInfo/sentiment",
               "required":false
            }
         }
      }
   }
}
```


3.2.4.5 Get model trees

Description	It accepts a language id, a model type (domain, policy, entity) and a list of model ids and it returns a list the model tree structures for these ids.
Method	POST

Endpoint	http://nomad.atc.gr/nomadWS/services/authoring /getModelTrees
Input	*application/json, application/xml*

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":false,
   "properties":{
      "languageID": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/languageID",
         "required":true
      },
      "modelIDs": {
         "type":"array",
         "id": "http://nomad.jsonschema.org/modelIDs",
         "required":true,
         "items":
            {
               "type":"number",
               "id": "http://nomad.jsonschema.org/modelIDs/0",
               "required":false
            }
      },
      "modelType": {
         "type":"string",
         "id": "http://nomad.jsonschema.org/modelType",
         "required":true
      }
   }
}
```

Output	*application/json, application/xml*

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":false,
   "properties":{
      "modelNode": {
         "type":"object",
         "id": "http://nomad.jsonschema.org/modelNode",
         "required":false,
         "properties":{
            "attr": {
               "type":"object",
               "id": "http://nomad.jsonschema.org/modelNode/attr",
               "required":true,
               "properties":{
```

```
                "id": {
                   "type":"number",
                   "id": "http://nomad.jsonschema.org/modelNode/attr/id",
                   "required":true
                },
                "modelID": {
                   "type":"number",
                   "id": "http://nomad.jsonschema.org/modelNode/attr/modelID",
                   "required":false
                },
                "modelType": {
                   "type":"null",
                   "id": "http://nomad.jsonschema.org/modelNode/attr/modelType",
                   "required":false
                },
                "type": {
                   "type":"string",
                   "id": "http://nomad.jsonschema.org/modelNode/attr/type",
                   "required":true
                }
             }
          },
          "children": {
             "type":"array",
             "id": "http://nomad.jsonschema.org/modelNode/children",
             "required":false,
   "items":{
                      "$ref":"modelNode"
                   }
          },
          "data": {
             "type":"object",
             "id": "http://nomad.jsonschema.org/modelNode/data",
             "required":false,
             "properties":{
                "access_level": {
                   "type":"null",
                   "id": "http://nomad.jsonschema.org/modelNode/data/access_level",
                   "required":false
                },
                "depiction": {
                   "type":"null",
                   "id": "http://nomad.jsonschema.org/modelNode/data/depiction",
                   "required":false
                },
                "id": {
                   "type":"number",
                   "id": "http://nomad.jsonschema.org/modelNode/data/id",
```

```
                    "required":false
                },
                "owner": {
                    "type":"null",
                    "id": "http://nomad.jsonschema.org/modelNode/data/owner",
                    "required":false
                },
                "title": {
                    "type":"string",
                    "id": "http://nomad.jsonschema.org/modelNode/data/title",
                    "required":false
                },
                "translations": {
                    "type":"object",
                    "id": "http://nomad.jsonschema.org/modelNode/data/translations",
                    "required":false,
                    "properties":{
                        "en": {
                            "type":"string",
                            "id": "http://nomad.jsonschema.org/modelNode/data/translationsen",
                            "required":false
                        },
                        "el": {
                            "type":"string",
                            "id":
"http://nomad.jsonschema.org/modelNode/data/translations/el",
                            "required":false
                        },
                        "de": {
                            "type":"string",
                            "id":
"http://nomad.jsonschema.org/modelNode/data/translations/de",
                            "required":false
                        }
                    }
                },
                "uri": {
                    "type":"string",
                    "id": "http://nomad.jsonschema.org/modelNode/data/uri",
                    "required":false
                }
            }
        }
    }
  }
  "type":"object",
  "$schema": "http://json-schema.org/draft-03/schema",
  "id": "http://nomad.jsonschema.org",
```

```
   "required":false,
   "properties":{
         "type":"array",
         "id": "http://nomad.jsonschema.org/modelTree",
         "required":true,
         "items":
            {
               "type":"object",
               "id": "http://nomad.jsonschema.org/modelTree/0",
               "required":false,
               "properties":{
                  "modelRoots": {
                     "type":"array",
                     "id": "http://nomad.jsonschema.org/response/0/modelRoots",
                     "required":true,
                     "items":{
                        "$ref":"modelNode"
                     }
                  }
               }
            }
         }
      }
   }
}
```

3.2.4.6 Get model tree

Description	It accepts a language id, a model type (domain, policy, entity) and a model id and returns the model tree structure for this model.
Method	POST
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring /getModelTree
Input	*application/json, application/xml*

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":false,
   "properties":{
      "languageID": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/languageID",
         "required":true
      },
      "modelID": {
         "type":"number",
```

```
        "id": "http://nomad.jsonschema.org/modelID",
        "required":true,
      },
      "modelType": {
        "type":"string",
        "id": "http://nomad.jsonschema.org/modelType",
        "required":true
      }
   }
}
```

Output	*application/json, application/xml*

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":false,
   "properties":{
      "modelNode": {
         "type":"object",
         "id": "http://nomad.jsonschema.org/modelNode",
         "required":false,
         "properties":{
            "attr": {
               "type":"object",
               "id": "http://nomad.jsonschema.org/modelNode/attr",
               "required":true,
               "properties":{
                  "id": {
                     "type":"number",
                     "id": "http://nomad.jsonschema.org/modelNode/attr/id",
                     "required":true
                  },
                  "modelID": {
                     "type":"number",
                     "id": "http://nomad.jsonschema.org/modelNode/attr/modelID",
                     "required":false
                  },
                  "modelType": {
                     "type":"null",
                     "id": "http://nomad.jsonschema.org/modelNode/attr/modelType",
                     "required":false
                  },
                  "type": {
                     "type":"string",
                     "id": "http://nomad.jsonschema.org/modelNode/attr/type",
                     "required":true
                  }
               }
```

```
        },
        "children": {
           "type":"array",
           "id": "http://nomad.jsonschema.org/modelNode/children",
           "required":false
        },
        "data": {
           "type":"object",
           "id": "http://nomad.jsonschema.org/modelNode/data",
           "required":true,
           "properties":{
              "access_level": {
                 "type":"null",
                 "id": "http://nomad.jsonschema.org/modelNode/data/access_level",
                 "required":false
              },
              "depiction": {
                 "type":"null",
                 "id": "http://nomad.jsonschema.org/modelNode/data/depiction",
                 "required":false
              },
              "id": {
                 "type":"number",
                 "id": "http://nomad.jsonschema.org/modelNode/data/id",
                 "required":false
              },
              "owner": {
                 "type":"null",
                 "id": "http://nomad.jsonschema.org/modelNode/data/owner",
                 "required":false
              },
              "title": {
                 "type":"string",
                 "id": "http://nomad.jsonschema.org/modelNode/data/title",
                 "required":true
              },
              "translations": {
                 "type":"object",
                 "id": "http://nomad.jsonschema.org/modelNode/data/translations",
                 "required":true,
                 "properties":{
                    "en": {
                       "type":"string",
                       "id": "http://nomad.jsonschema.org/modelNode/data/translationsen",
                       "required":false
                    },
                    "el": {
                       "type":"string",
```

```
                        "id":
"http://nomad.jsonschema.org/modelNode/data/translations/el",
                        "required":false
                     },
                     "de": {
                        "type":"string",
                        "id":
"http://nomad.jsonschema.org/modelNode/data/translations/de",
                        "required":false
                     }
                  }
               },
               "uri": {
                  "type":"string",
                  "id": "http://nomad.jsonschema.org/modelNode/data/uri",
                  "required":false
               }
            }
         }
      }
   }
   "type":"object",
   "required":true,
   "properties":{

      "type":"object",
      "id": "http://nomad.jsonschema.org/modelTree/0",
      "required":true,
                  "properties":{
           "modelRoots": {
           "type":"array",
           "id": "http://nomad.jsonschema.org/response/0/modelRoots",
           "required":true,
              "items":{
                 "$ref":"modelNode"
                     }
              }
           }
        }
     }
   }
}
```

3.2.4.7 Get languages

Description	It returns the languages supported by the system (i.e. Greek, English, German)
Method	GET
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring /getLanguages

Input	*application/json, application/xml*
Output	*application/json, application/xml*

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":true,
   "properties":{
         "type":"array",
         "id": "http://nomad.jsonschema.org/response",
         "required":true,
"minitems":"0",
         "items":
            {
               "type":"object",
               "id": "http://nomad.jsonschema.org/response/0",
               "required":false,
               "properties":{
                  "depiction": {
                     "type":"string",
                     "id": "http://nomad.jsonschema.org/response/0/depiction",
                     "required":false
                  },
                  "id": {
                     "type":"number",
                     "id": "http://nomad.jsonschema.org/response/0/id",
                     "required":true
                  },
                  "languageCode": {
                     "type":"string",
                     "id": "http://nomad.jsonschema.org/response/0/languageCode",
                     "required":true
                  }
               }
            }
      }
}
```

3.2.4.8 Get Public Domains

Description	It returns the available public domains	
Method	GET	
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring /getPublicDomains	
Input		
Int languageID		The database id of the language the user selected
Long owner		The current user's id. Used to return domains **not** belonging to this user.

Output	*application/json, application/xml*


```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":false,
   "properties":{
         "type":"array",
         "minitems": "0",
         "id": "http://nomad.jsonschema.org/response",
         "required":true,
         "items":
            {
               "type":"object",
               "id": "http://nomad.jsonschema.org/response/0",
               "required":false,
               "properties":{
                  "access_level": {
                     "type":"number",
                     "id": "http://nomad.jsonschema.org/response/0/access_level",
                     "required":true
                  },
                  "depiction": {
                     "type":"string",
                     "id": "http://nomad.jsonschema.org/response/0/depiction",
                     "required":false
                  },
                  "id": {
                     "type":"number",
                     "id": "http://nomad.jsonschema.org/response/0/id",
                     "required":true
                  },
                  "owner": {
                     "type":"number",
                     "id": "http://nomad.jsonschema.org/response/0/owner",
                     "required":true
                  },
                  "title": {
                     "type":"string",
                     "id": "http://nomad.jsonschema.org/response/0/title",
                     "required":true
                  },
                  "translations": {
                     "type":"object",
                     "id": "http://nomad.jsonschema.org/response/0/translations",
                     "required":true,
                     "properties":{
                        "de": {
```

```
                        "type":"string",
                        "id": "http://nomad.jsonschema.org/response/0/translations/de",
                        "required":false
                     },
                     "el": {
                        "type":"string",
                        "id": "http://nomad.jsonschema.org/response/0/translations/el",
                        "required":false
                     },
                     "en": {
                        "type":"string",
                        "id": "http://nomad.jsonschema.org/response/0/translations/en",
                        "required":false
                     }
                  }
               },
               "uri": {
                  "type":"string",
                  "id": "http://nomad.jsonschema.org/response/0/uri",
                  "required":false
               }
            }
         }
      }
}
```


3.2.4.9 Get Public Policies

Description	It returns the available public policies	
Method	GET	
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring /getPublicPolicies	
Input		
Int languageID		The database id of the language the user selected
Long owner		The current user's id. Used to return policies**not** belonging to this user.
Output	*application/json, application/xml*	


```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":true,
   "properties":{
         "type":"array",
         "minitems": "0",
         "id": "http://nomad.jsonschema.org/response",
         "required":true,
         "items":
            {
               "type":"object",
```

```
"id": "http://nomad.jsonschema.org/response/0",
"required":true,
"properties":{
   "access_level": {
      "type":"number",
      "id": "http://nomad.jsonschema.org/response/0/access_level",
      "required":true
   },
   "depiction": {
      "type":"string",
      "id": "http://nomad.jsonschema.org/response/0/depiction",
      "required":false
   },
   "id": {
      "type":"number",
      "id": "http://nomad.jsonschema.org/response/0/id",
      "required":true
   },
   "owner": {
      "type":"number",
      "id": "http://nomad.jsonschema.org/response/0/owner",
      "required":true
   },
   "title": {
      "type":"string",
      "id": "http://nomad.jsonschema.org/response/0/title",
      "required":true
   },
   "translations": {
      "type":"object",
      "id": "http://nomad.jsonschema.org/response/0/translations",
      "required":true,
      "properties":{
         "de": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/response/0/translations/de",
            "required":false
         },
         "el": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/response/0/translations/el",
            "required":false
         },
         "en": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/response/0/translations/en",
            "required":false
         }
```

```
                    }
                },
                "uri": {
                    "type":"string",
                    "id": "http://nomad.jsonschema.org/response/0/uri",
                    "required":false
                }
            }
        }


    }
}
```

3.2.4.10 Get User Domains

Description	It returns the specified user's domains	
Method	GET	
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring /getUserDomains	
Input		
Int languageID		The database id of the language the user selected
Long owner		The user's id.
Output	*application/json, application/xml*	

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":true,
   "properties":{
        "type":"array",
        "minitems": "0",
        "id": "http://nomad.jsonschema.org/response",
        "required":true,
        "items":
           {
              "type":"object",
              "id": "http://nomad.jsonschema.org/response/0",
              "required":false,
              "properties":{
                 "access_level": {
                    "type":"number",
                    "id": "http://nomad.jsonschema.org/response/0/access_level",
                    "required":true
                 },
                 "depiction": {
                    "type":"string",
                    "id": "http://nomad.jsonschema.org/response/0/depiction",
                    "required":false
```

```
                },
                "id": {
                   "type":"number",
                   "id": "http://nomad.jsonschema.org/response/0/id",
                   "required":true
                },
                "owner": {
                   "type":"number",
                   "id": "http://nomad.jsonschema.org/response/0/owner",
                   "required":true
                },
                "title": {
                   "type":"string",
                   "id": "http://nomad.jsonschema.org/response/0/title",
                   "required":true
                },
                "translations": {
                   "type":"object",
                   "id": "http://nomad.jsonschema.org/response/0/translations",
                   "required":true,
                   "properties":{
                      "de": {
                         "type":"string",
                         "id": "http://nomad.jsonschema.org/response/0/translations/de",
                         "required":false
                      },
                      "el": {
                         "type":"string",
                         "id": "http://nomad.jsonschema.org/response/0/translations/el",
                         "required":false
                      },
                      "en": {
                         "type":"string",
                         "id": "http://nomad.jsonschema.org/response/0/translations/en",
                         "required":false
                      }
                   }
                },
                "uri": {
                   "type":"string",
                   "id": "http://nomad.jsonschema.org/response/0/uri",
                   "required":false
                }
             }
          }
       }
}
```

3.2.4.11 Get User Policies

Description	It returns the specified user's policies	
Method	GET	
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring /getUserPolicies	
Input		
Int languageID		The database id of the language the user selected
Long owner		The user's id.
Output	*application/json, application/xml*	

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required":true,
   "properties":{
         "type":"array",
         "minitems": "0",
         "id": "http://nomad.jsonschema.org/response",
         "required":true,
         "items":
            {
               "type":"object",
               "id": "http://nomad.jsonschema.org/response/0",
               "required":false,
               "properties":{
                  "access_level": {
                     "type":"number",
                     "id": "http://nomad.jsonschema.org/response/0/access_level",
                     "required":true
                  },
                  "depiction": {
                     "type":"string",
                     "id": "http://nomad.jsonschema.org/response/0/depiction",
                     "required":false
                  },
                  "id": {
                     "type":"number",
                     "id": "http://nomad.jsonschema.org/response/0/id",
                     "required":true
                  },
                  "owner": {
                     "type":"number",
                     "id": "http://nomad.jsonschema.org/response/0/owner",
                     "required":true
                  },
                  "title": {
                     "type":"string",
                     "id": "http://nomad.jsonschema.org/response/0/title",
```

```
                  "required":true
               },
               "translations": {
                  "type":"object",
                  "id": "http://nomad.jsonschema.org/response/0/translations",
                  "required":false,
                  "properties":{
                     "de": {
                        "type":"string",
                        "id": "http://nomad.jsonschema.org/response/0/translations/de",
                        "required":false
                     },
                     "el": {
                        "type":"string",
                        "id": "http://nomad.jsonschema.org/response/0/translations/el",
                        "required":false
                     },
                     "en": {
                        "type":"string",
                        "id": "http://nomad.jsonschema.org/response/0/translations/en",
                        "required":false
                     }
                  }
               },
               "uri": {
                  "type":"string",
                  "id": "http://nomad.jsonschema.org/response/0/uri",
                  "required":false
               }
            }
         }
      }
}
```


3.2.4.12 Get Argument Types

Description	It returns the argumenttypes by the system (i.e. Negative, Positive, Neutral)
Method	GET
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring /getArgumentTypes
Input	*application/json, application/xml*
–	
Output	*application/json, application/xml*


```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required": true,
   "properties":{
```

```
        "type":"array",
        "id": "http://nomad.jsonschema.org/response",
        "required":true,
"minitems":"0"
        "items":
           {
              "type":"object",
              "id": "http://nomad.jsonschema.org/response/0",
              "required":false,
              "properties":{
                 "depiction": {
                    "type":"string",
                    "id": "http://nomad.jsonschema.org/response/0/depiction",
                    "required":false
                 },
                 "id": {
                    "type":"number",
                    "id": "http://nomad.jsonschema.org/response/0/id",
                    "required":true
                 },
                 "languageCode": {
                    "type":"string",
                    "id": "http://nomad.jsonschema.org/response/0/languageCode",
                    "required":true
                 }
              }
           }
     }
}
```

3.2.4.13 Add Entity

Description	It adds a new entity of a domain into the database
Method	POST
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring /addEntity
Input	*application/json, application/xml*

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required": true,
   "properties":{
      "depiction": {
         "type":"array",
         "id": "http://nomad.jsonschema.org/depiction",
         "required":false,
         "items":
```

```
        {
           "type":"number",
           "id": "http://nomad.jsonschema.org/depiction/0",
           "required":false
        }
    },
    "id": {
       "type":"number",
       "id": "http://nomad.jsonschema.org/id",
       "required": false
    },
    "modelID": {
       "type":"number",
       "id": "http://nomad.jsonschema.org/modelID",
       "required":true
    },
    "modelType": {
       "type":"string",
       "id": "http://nomad.jsonschema.org/modelType",
       "required":true
    },
    "parentID": {
       "type":"number",
       "id": "http://nomad.jsonschema.org/parentID",
       "required":true
    },
    "translations": {
       "type":"object",
       "id": "http://nomad.jsonschema.org/translations",
       "required":true,
       "properties":{
          "de": {
             "type":"string",
             "id": "http://nomad.jsonschema.org/translations/de",
             "required":false
          },
          "el": {
             "type":"string",
             "id": "http://nomad.jsonschema.org/translations/el",
             "required":false
          },
          "en": {
             "type":"string",
             "id": "http://nomad.jsonschema.org/translations/en",
             "required":false
          }
       }
    },
```

```
      "uri": {
         "type":"string",
         "id": "http://nomad.jsonschema.org/uri",
         "required":false
      }
   }
}
```

Output	
A long number indicating the database id of the newly added entity.	

3.2.4.14 Add Domain Data

Description	It adds a new domain
Method	POST
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring /addDomainData
Input	*application/json, application/xml*

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required": true,
   "properties":{
      "access_level": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/access_level",
         "required":true
      },
      "depiction": {
         "type":"array",
         "id": "http://nomad.jsonschema.org/depiction",
         "required":false,
         "items":
            {
               "type":"number",
               "id": "http://nomad.jsonschema.org/depiction/0",
               "required":false
            }
      },
      "id": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/id",
         "required": false
      },
      "owner": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/owner",
         "required":true
      },
```

```
      "translations": {
         "type":"object",
         "id": "http://nomad.jsonschema.org/translations",
         "required":true,
         "properties":{
            "el": {
               "type":"string",
               "id": "http://nomad.jsonschema.org/translations/1",
               "required":false
            },
            "en": {
               "type":"string",
               "id": "http://nomad.jsonschema.org/translations/2",
               "required":false
            },
            "de": {
               "type":"string",
               "id": "http://nomad.jsonschema.org/translations/3",
               "required":false
            }
         }
      },
      "uri": {
         "type":"string",
         "id": "http://nomad.jsonschema.org/uri",
         "required":false
      }
   }
}
```

Output	
A long number indicating the database id of the newly added domain.	

3.2.4.15 Add Norm

Description	It adds a new norm into the db
Method	POST
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring /addDomainData
Input	*application/json, application/xml*
long policyID	The database id of the policy under which the norm will be added

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required": true,
   "properties":{
      "access_level": {
         "type":"number",
```

```
      "id": "http://nomad.jsonschema.org/access_level",
      "required":true
   },
   "depiction": {
      "type":"array",
      "id": "http://nomad.jsonschema.org/depiction",
      "required":false,
      "items":
         {
            "type":"number",
            "id": "http://nomad.jsonschema.org/depiction/0",
            "required":false
         }
   },
   "id": {
      "type":"number",
      "id": "http://nomad.jsonschema.org/id",
      "required": false
   },
   "owner": {
      "type":"number",
      "id": "http://nomad.jsonschema.org/owner",
      "required":true
   },
   "translations": {
      "type":"object",
      "id": "http://nomad.jsonschema.org/translations",
      "required":true,
      "properties":{
         "el": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/1",
            "required":false
         },
         "en": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/2",
            "required":false
         },
         "de": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/3",
            "required":false
         }
      }
   },
   "uri": {
      "type":"string",
```

```
        "id": "http://nomad.jsonschema.org/uri",
        "required":false
      }
   }
}
```

Output	
A long number indicating the database id of the newly added norm.	

3.2.4.16 Add Policy Data

Description	It adds a new policyinto the database
Method	POST
Endpoint	http://nomad.atc.gr/nomadWS/services/ authoring /addPolicyData
Input	*application/json, application/xml*

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required": true,
   "properties":{
      "access_level": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/access_level",
         "required":true
      },
      "depiction": {
         "type":"array",
         "id": "http://nomad.jsonschema.org/depiction",
         "required":false,
         "items":
            {
               "type":"number",
               "id": "http://nomad.jsonschema.org/depiction/0",
               "required":false
            }
      },
      "id": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/id",
         "required": false
      },
      "owner": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/owner",
         "required":true
      },
      "translations": {
```

```
        "type":"object",
        "id": "http://nomad.jsonschema.org/translations",
        "required":true,
        "properties":{
           "el": {
              "type":"string",
              "id": "http://nomad.jsonschema.org/translations/1",
              "required":false
           },
           "en": {
              "type":"string",
              "id": "http://nomad.jsonschema.org/translations/2",
              "required":false
           },
           "de": {
              "type":"string",
              "id": "http://nomad.jsonschema.org/translations/3",
              "required":false
           }
        }
     },
     "uri": {
        "type":"string",
        "id": "http://nomad.jsonschema.org/uri",
        "required":false
     }
  }
}
```


Output	
A long number indicating the database id of the newly added policy.	

3.2.4.17 Add Argument

Description	It adds a new argument into the database, under a norm
Method	POST
Endpoint	http://nomad.atc.gr/nomadWS/services/ authoring /addArgument
Input	*application/json, application/xml*


```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required": true,
   "properties":{
      "description": {
         "type":"string",
         "id": "http://nomad.jsonschema.org/description",
         "required": false
```

```
      },
      "content": {
         "type":"array",
         "id": "http://nomad.jsonschema.org/content",
         "required":false,
         "items":
            {
               "type":"number",
               "id": "http://nomad.jsonschema.org/content/0",
               "required":false
            }
      },
      "id": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/id",
         "required": false
      },
"argumentTypeId": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/argumentTypeId",
         "required": true
      },
      "normId": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/normId ",
         "required": false
      },
"entityId": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/entityId",
         "required":true
      },
"contentFragment": {
               "type":"string",
               "id": "http://nomad.jsonschema.org/contentFragment",
               "required":false
      },
      "translations": {
         "type":"object",
         "id": "http://nomad.jsonschema.org/translations",
         "required":true,
         "properties":{
            "el": {
               "type":"string",
               "id": "http://nomad.jsonschema.org/translations/1",
               "required":false
            },
            "en": {
```

```
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/2",
            "required":false
         },
         "de": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/3",
            "required":false
         }
      }
   },
   "uri": {
      "type":"string",
      "id": "http://nomad.jsonschema.org/uri",
      "required":false
   }
 }
}
```


Output	
A long number indicating the database id of the newly added argument.	

3.2.4.18 Update Entity Data

Description	It updates an entity of a domain or policy
Method	PUT
Endpoint	http://nomad.atc.gr/nomadWS/services/ authoring /updateEntityData
Input	*application/json, application/xml*


```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required": true,
   "properties":{
      "depiction": {
         "type":"array",
         "id": "http://nomad.jsonschema.org/depiction",
         "required":false,
         "items":
            {
               "type":"number",
               "id": "http://nomad.jsonschema.org/depiction/0",
               "required":false
            }
      },
      "id": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/id",
```

```
        "required":true
      },
      "translations": {
        "type":"object",
        "id": "http://nomad.jsonschema.org/translations",
        "required":true,
        "properties":{
          "de": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/de",
            "required":false
          },
          "el": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/el",
            "required":false
          },
          "en": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/en",
            "required":false
          }
        }
      },
      "uri": {
        "type":"string",
        "id": "http://nomad.jsonschema.org/uri",
        "required":false
      }
   }
}
```

Output	
A Boolean indicating the success of the update.	

3.2.4.19 Update Entity relation

Description	It changes an entity relation
Method	PUT
Endpoint	http://nomad.atc.gr/nomadWS/services/ authoring /updateEntityRelation
Input	*application/json, application/xml*

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required": true,
   "properties":{
      "id": {
        "type":"number",
```

```
         "id": "http://nomad.jsonschema.org/id",
         "required":true
      },
      "oldParentID": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/modelID",
         "required":true
      },
      "newParentID": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/parentID",
         "required":true
      }
   }
}
```

Output	*application/json, application/xml*

A Boolean indicating the success of the update.

3.2.4.20 Update Domain Data

Description	It updates a domain's metadata
Method	PUT
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring/updateDomainData
Input	*application/json, application/xml*

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required": true,
   "properties":{
      "access_level": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/access_level",
         "required":true
      },
      "depiction": {
         "type":"array",
         "id": "http://nomad.jsonschema.org/depiction",
         "required":false,
         "items":
            {
               "type":"number",
               "id": "http://nomad.jsonschema.org/depiction/0",
               "required":false
            }
      },
      "id": {
         "type":"number",
```

```
        "id": "http://nomad.jsonschema.org/id",
        "required": true
      },
      "owner": {
        "type":"number",
        "id": "http://nomad.jsonschema.org/owner",
        "required":true
      },
      "translations": {
        "type":"object",
        "id": "http://nomad.jsonschema.org/translations",
        "required":true,
        "properties":{
          "el": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/1",
            "required":false
          },
          "en": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/2",
            "required":false
          },
          "de": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/3",
            "required":false
          }
        }
      },
      "uri": {
        "type":"string",
        "id": "http://nomad.jsonschema.org/uri",
        "required":false
      }
   }
}
```


Output	
A Boolean indicating the success of the update.	

3.2.4.21 Update Policy Data

Description	It updates a policy's metadata
Method	PUT
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring/updatePolicyData
Input	*application/json, application/xml*


```
{
   "type":"object",
```

```
"$schema": "http://json-schema.org/draft-03/schema",
"id": "http://nomad.jsonschema.org",
"required": true,
"properties":{
   "access_level": {
      "type":"number",
      "id": "http://nomad.jsonschema.org/access_level",
      "required":true
   },
   "depiction": {
      "type":"array",
      "id": "http://nomad.jsonschema.org/depiction",
      "required":false,
      "items":
         {
            "type":"number",
            "id": "http://nomad.jsonschema.org/depiction/0",
            "required":false
         }
   },
   "id": {
      "type":"number",
      "id": "http://nomad.jsonschema.org/id",
      "required": true
   },
   "owner": {
      "type":"number",
      "id": "http://nomad.jsonschema.org/owner",
      "required":true
   },
   "translations": {
      "type":"object",
      "id": "http://nomad.jsonschema.org/translations",
      "required":true,
      "properties":{
         "el": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/1",
            "required":false
         },
         "en": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/2",
            "required":false
         },
         "de": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/3",
```

```
            "required":false
         }
      }
   },
   "uri": {
      "type":"string",
      "id": "http://nomad.jsonschema.org/uri",
      "required":false
   }
}
}
```

Output	
A Boolean indicating the success of the update.	

3.2.4.22 Update Norm Data

Description	It updates a norm's metadata
Method	PUT
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring/updateNormData
Input	*application/json, application/xml*

```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required": true,
   "properties":{
      "access_level": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/access_level",
         "required":true
      },
      "depiction": {
         "type":"array",
         "id": "http://nomad.jsonschema.org/depiction",
         "required":false,
         "items":
            {
               "type":"number",
               "id": "http://nomad.jsonschema.org/depiction/0",
               "required":false
            }
      },
      "id": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/id",
         "required": true
      },
      "owner": {
```

```
        "type":"number",
        "id": "http://nomad.jsonschema.org/owner",
        "required":true
      },
      "translations": {
        "type":"object",
        "id": "http://nomad.jsonschema.org/translations",
        "required":true,
        "properties":{
          "el": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/1",
            "required":false
          },
          "en": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/2",
            "required":false
          },
          "de": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/3",
            "required":false
          }
        }
      },
      "uri": {
        "type":"string",
        "id": "http://nomad.jsonschema.org/uri",
        "required":false
      }
   }
}
```


Output	
A Boolean indicating the success of the update.	

3.2.4.23 Update Argument

Description	It adds a new argument into the database, under a norm
Method	POST
Endpoint	http://nomad.atc.gr/nomadWS/services/ authoring /updateArgument
Input	*application/json, application/xml*


```
{
   "type":"object",
   "$schema": "http://json-schema.org/draft-03/schema",
   "id": "http://nomad.jsonschema.org",
   "required": true,
```

```
   "properties":{
      "description": {
         "type":"string",
         "id": "http://nomad.jsonschema.org/description",
         "required": false
      },
      "content": {
         "type":"array",
         "id": "http://nomad.jsonschema.org/content",
         "required":false,
         "items":
            {
               "type":"number",
               "id": "http://nomad.jsonschema.org/content/0",
               "required":false
            }
      },
      "id": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/id",
         "required": true
      },
"argumentTypeId": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/argumentTypeId",
         "required": true
      },
      "normId": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/normId ",
         "required": false
      },
"entityId": {
         "type":"number",
         "id": "http://nomad.jsonschema.org/entityId",
         "required":true
      },
"contentFragment": {
               "type":"string",
               "id": "http://nomad.jsonschema.org/contentFragment",
               "required":false
      },
      "translations": {
         "type":"object",
         "id": "http://nomad.jsonschema.org/translations",
         "required":true,
         "properties":{
            "el": {
```

```
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/1",
            "required":false
         },
         "en": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/2",
            "required":false
         },
         "de": {
            "type":"string",
            "id": "http://nomad.jsonschema.org/translations/3",
            "required":false
         }
      }
   },
   "uri": {
      "type":"string",
      "id": "http://nomad.jsonschema.org/uri",
      "required":false
   }
 }
}
```


Output	
A Boolean indicating the success of the update.	

3.2.4.24 Copy Policy Node

Description	It copies a node (argument, entity, norm) of a policy to another policy.
Method	POST
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring /copyPolicyNode
Input	
Long nodeID	The database id of the node to be copied
String nodeType	The type of the node to be copied (argument, entity, norm)
Long parentID	The database id of node's new parent
String parentType	The type of node's new parent in the case we copy entity (norm, argument)
Output	
A Boolean indicating the success of the copy function	

3.2.4.25 Delete model

Description	It deletes a model from the database.
Method	DELETE
Endpoint	http://nomad.atc.gr/nomadWS/services/authoring /deleteModel
Input	
long modelID	The database id of model to be deleted

String modelType	The type of the model to be deleted (domain, policy, entity)
Output	
A Boolean indicating the success of deletion	

3.3 Development of the Storage Layer

3.3.1 The Persistency Component

The Persistency Layer of the NOMAD platform maintains all the data access objects that link the core Web application with the NOMAD database. The Persistency Layer is based on Object-relational mapping (ORM). This is a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages. This creates, in effect, a “virtual object database” that can be used from within the programming language [11]. For the specific ORM implementation the JPA framework [12] through Eclipse Link[13] has been used.

The actual storage is based on the MySQL Community Server database [14]. This is considered to be the most popular object relational database system. The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation. MySQL works on many different system platforms, including AIX, BSDi, FreeBSD, HP-UX, eComStation, i5/OS, IRIX, Linux, Mac OS X, Microsoft Windows, NetBSD, Novell NetWare, OpenBSD, OpenSolaris and others. It is a very mature system, very good documented and with a wealth of resources available. Many graphical interface applications for the management and administration of a MySQL database are available, both free and paid. Although MySQL was the database of choice for almost all open source projects, the acquisition of Sun by Oracle, which brought MySQL under Oracle's control, made many developers sceptical about using it in new projects. Oracle promised to continue supporting the database system and keep offering the community edition, but it is generally thought that a sudden policy change from Oracle's part is not to be excluded. Currently, the status of the terms of use for the MySQL Community Server fit to the needs of the NOMAD project.

3.3.2 Overview of NOMAD Databases

The NOMAD databasesarepopulated with sufficient content by means of the NOMAD Crawler, which fetches data periodically from a variety of sources. The domain-related search queries are used from the crawler to acquire content over Bing, Facebook, Twitter, and various other web sources.

The models used by NOMAD consist of:

- Domains
- Entities relevant to these domains
- Norms defining relations between the entities, and
- Arguments related to the defined norms.

Each one of the above carries descriptive characteristics that are used by the Thematic Classifier in order to define the multidimensional keyword spaces required by the aforementioned NOMAD Crawler.

The multidimensional keyword spaces produced by the analysis of the domain and policy models created by the users are therefore stored in the NOMAD databases in order to continuously be used by the NOMAD Crawler.

The content that is stored in the NOMAD databases, as well as its schema, is properly described in the Deliverable 4.1: Data Acquisition and Management Module.

3.4 Overview of Service-based Interfaces

The following Table 2 summarises the services that are available to interact with the NOMAD repositories. The first column contains the name of the service. The second column shows the http Method that must be used when calling the

respective service. The third column displays the type of the response expected after the call, while the last column provides a reference to the actual service description.

Table 2: Web Service Interfaces for the 1st NOMAD platform prototype

Service Name	Method	Response Type	Reference Section
Thread status listing	GET	JSON / XML	3.1.1.1
Thread creation	POST	JSON / XML	3.1.1.2
Linguistic Demographic Extractor	POST	JSON / XML	3.1.2.1
Segment Extractor	POST	JSON / XML	3.1.2.2
Sentiment Analyser (document level)	POST	JSON / XML	3.1.2.3
Sentiment Analyser (segment level)	POST	JSON / XML	3.1.2.4
Tag Cloud Generator	POST	JSON / XML	3.1.2.5
Argument Extractor	POST	JSON / XML	3.1.2.6
Argument Summariser	POST	JSON / XML	3.1.2.7
Get domain/policy/norm/entity	POST	JSON / XML	3.2.4.1
Get term frequencies	POST	JSON / XML	3.2.4.2
Get database info	POST	JSON / XML	3.2.4.3
Get bucket size	POST	JSON / XML	3.2.4.4
Get model trees	POST	JSON / XML	3.2.4.5
Get model tree	POST	JSON / XML	3.2.4.6
Get languages	GET	JSON / XML	3.2.4.7
Get Public Domains	GET	JSON / XML	3.2.4.8
Get Public Policies	GET	JSON / XML	3.2.4.9
Get User Domains	GET	JSON / XML	3.2.4.10
Get User Policies	GET	JSON / XML	3.2.4.11
Get Argument Types	GET	JSON / XML	3.2.4.12
Add Entity	POST	Long	3.2.4.13
Add Domain Data	POST	Long	3.2.4.14
Add Norm	POST	Long	3.2.4.15
Add Policy Data	POST	Long	3.2.4.16
Add Argument	POST	Long	3.2.4.17
Update Entity Data	PUT	Boolean	3.2.4.18
Update Entity relation	PUT	Boolean	3.2.4.19
Update Domain Data	PUT	Boolean	3.2.4.20
Update Policy Data	PUT	Boolean	3.2.4.21
Update Norm Data	PUT	Boolean	3.2.4.22
Update Argument	POST	Boolean	3.2.4.23
Copy Policy Node	POST	Boolean	3.2.4.24
Delete model	DELETE	Boolean	3.2.4.25

3.5 Status of the NOMAD PlatformDevelopment

The service-based interfaces analysed in the previous Section 3.4 facilitate the communication among the different NOMAD platform components. Based on the three different UI components, this section presents the completion rate of the NOMAD platform development, taking as reference the NOMAD platform use cases identified in Section 2.5.

For each NOMAD UI component, we describe which functionalities have been integrated (even as draft) and which ones have been left for the next prototype releases (a plan for the next releases is given in Section 5). The ID is the use case unique identification number and provides reference to the specific description of the use case in Table 1.

Table 3 summarises the status of integration for NOMAD platform use cases referring to the Generic UI. In particular, the processing layer components have been integrated, with the exception of the Argument Summariser, which is planned for the next period. The exposition such integration on the Generic UI has not been implemented yet. Still, the NOMAD platform does not consider for the different roles, so no role-based access to the NOMAD functionalities is supported.

Table 3: Development status for the use cases of the Generic UI

ID	Use Case Title	Completed	Next Prototype
UC1	Manage User Roles	*Not developed yet*	Add user management functionality, based on the respective features of Liferay, so that NOMAD offerings are attributed to registered users assigned to specific platform roles.
UC2	Monitor Platform Services	Currently, the management of processing services is performed through command line. The NOMAD Server maintains logs of all the service activities. All the processing layer components have been integrated with the exception of the Argument Summariser.	Exploit the logs of the NOMAD Server to provide Web-based access to the status of the analysis and processing components.
UC3	Check status analysis of domain and policy models	Currently the status of the models' analysis can be seen directly in the NOMAD DB.	Develop a Web-based access to the list of domain and policy models stored into the NOMAD DB to assess whether their analysis is active or not. The Administrator has view access to all available models, while the Author can view only the public and their own models
UC4	Start/Stop analysis for the set of domain/policy models	Currently, the analysis can be invoked through command line for all the model sin the NOMAD DB	Develop a Web-based access to the list of domain and policy models stored into the NOMAD DB to enable end users start or Stop the analysis for a defined list of models.

Table 3 summarises the status of integration for NOMAD platform use cases referring to the Model Authoring Module. In particular, the Authoring Module functionalities have been partially developed and integrated in the context of NOMAD to enable target end users perform the use cases as highlighted in this table.

Table 4: Development status for the use cases of the Model Authoring Module

ID	Use Case Title	Completed	Next Prototype
UC5	Import Domain Model	*Not developed yet*	The functionality for mapping an RDF structure to the NOMAD specific model schema will be implemented, so that

ID	Use Case Title	Completed	Next Prototype
			external domain models can be loaded to the NOMAD platform
UC6	Load Domain Models	A first draft of this use case is already available (see Figure 5). A NOMAD user can load existing NOMAD domain models, which are flagged as public, and navigate through their entities	The user management features will enable to load private own domain models. A better visualisation (mind-map based) for the loaded models will probably be provided
UC7	Edit Domain Model	A first draft of this use case is already available (seeFigure 5). A NOMAD user can create a new domain model and modify or delete an existing domain model. Authoring is supported in three languages (English, Greek and German). A domain model can be associated with a list of metadata fields. All functionalities are performed on the tree view, while the mind map view is synchronised.	This use case will be implemented in full mind map view and all the edit functionalities will be supported on the mind map.
UC8	Edit Domain Entity	A first draft of this use case is already available (see Figure 6). A NOMAD user can create a new domain entity and update or delete an existing domain entity. Authoring is supported in three languages (English, Greek and German). Entities from a public domain model can be inherited in the model under development. All functionalities are performed on the tree view, while the mind map view is synchronised.	This use case will be implemented in full mind map view and all the edit functionalities (including the change on the relation among entities) will be supported on the mind map.
UC9	Export Domain Model	*Not developed yet*	The functionality for mapping the NOMAD specific model schema to an RDF structure will be implemented, so that a NOMAD domain model can be edited by existing RDF tools
UC10	Share Domain Model	*Not developed yet*	Based on the user access rights, which will be implemented through the relevant Liferay features, the end users will be able to share domain models with a list of other users
UC11	Import Policy Model	*Not developed yet*	The functionality for mapping an RDF structure to the NOMAD specific model schema will be implemented, so that external policy models can be loaded to the NOMAD platform
UC12	Load Policy Models	A first draft of this use case is already available (see Figure 7). A NOMAD user can load existing NOMAD policy models, which are flagged as public, and navigate through their policy components, arguments and their associated domain entities.	The user management features will enable to load private own policy models. A better visualisation (mind-map based) for the loaded models will probably be provided.

ID	Use Case Title	Completed	Next Prototype
UC13	Edit Policy Model	A first draft of this use case is already available (see Figure 7). A NOMAD user can create a new policy model and modify or delete an existing policy model. Authoring is supported in three languages (English, Greek and German). On top of that, the policy model can be linked to public domain entities loaded in the platform. A policy model can be associated with a list of metadata fields, such as the linked domain model. All functionalities are performed on the tree view, while the mind map view is synchronised.	This use case will be implemented in full mind map view and all the edit functionalities will be supported on the mind map.
UC14	Edit Policy Component	A first draft of this use case is already available (see Figure 8). A NOMAD user can create a new policy component and update or delete an existing one. Authoring is supported in three languages (English, Greek and German). Policy components from a public domain model can be inherited in the model under development. On top of that, the policy components can be linked to public domain entities loaded in the platform. All functionalities are performed on the tree view, while the mind map view is synchronised.	This use case will be implemented in full mind map view and all the edit functionalities (including the change on the relation among policy components and entities) will be supported on the mind map.
UC15	Edit Argumentation	A first draft of this use case is already available (see Figure 8). A NOMAD user can create a new argument and update or delete an existing one, including the argument polarity. Authoring is supported in three languages (English, Greek and German). The arguments can be linked to public domain entities loaded in the platform. All functionalities are performed on the tree view, while the mind map view is synchronised.	This use case will be implemented in full mind map view and all the edit functionalities (including the change on the relation among arguments and entities) will be supported on the mind map.
UC16	Export Policy Model	*Not developed yet*	The functionality for mapping the NOMAD specific model schema to an RDF structure will be implemented, so that a NOMAD policy model can be edited by existing RDF tools
UC17	Share Policy Model	*Not developed yet*	Based on the user access rights, which will be implemented through the relevant Liferay features, the end users will be able to share policy models with a list of other users

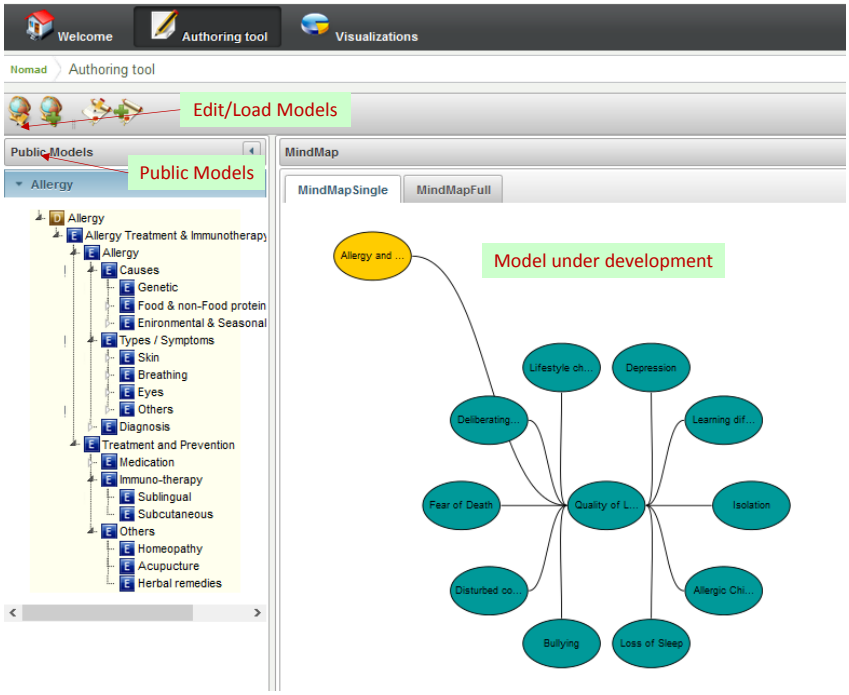

Figure 5: Loading public domain models and edit a domain model in the NOMAD platform

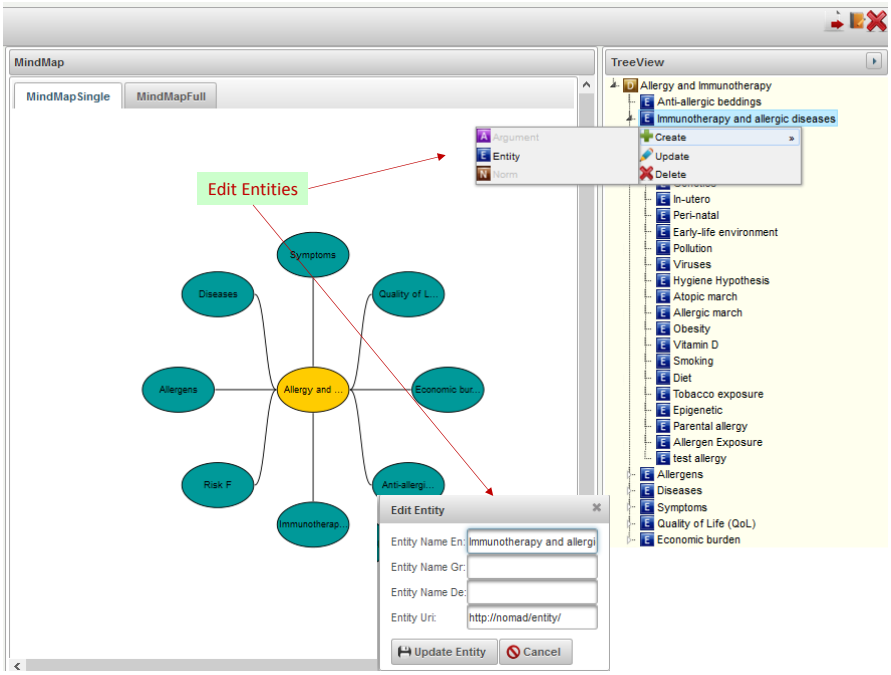

Figure 6: Editing domain entities in the NOMAD platform

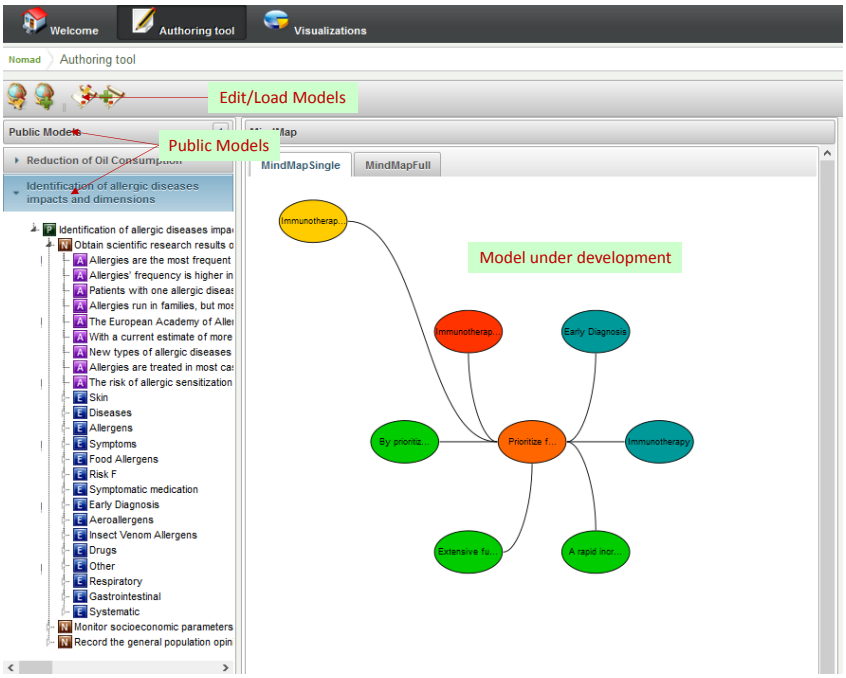

Figure 7: Loading public policy models and edit a policy model in the NOMAD platform

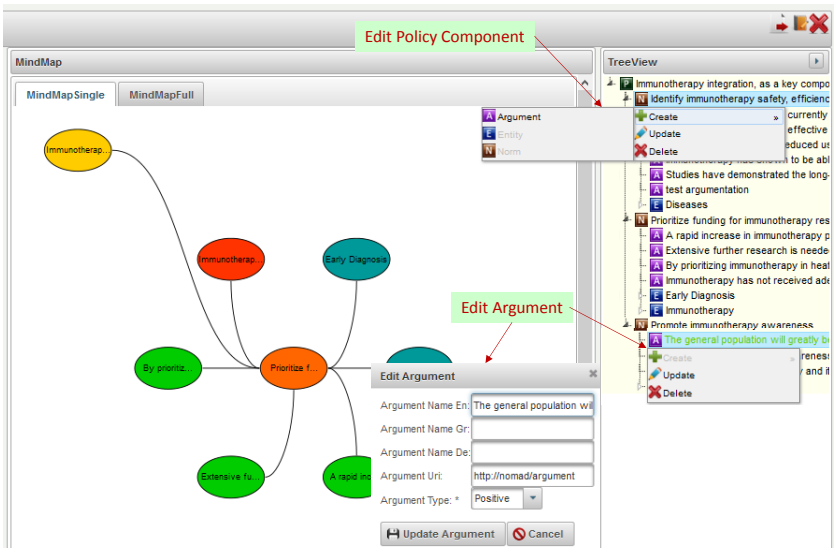

Figure 8: Editing policy components and arguments in the NOMAD platform

When coming to the Visualisation part, the Visualisation Module has been integrated as a Liferay portlet to the Generic UI, which acts as the User Interface container for all the Presentation Layer components. The integration is achieved through communicating with the remote Visualisation Server, which is responsible for providing the proper visualisation result to a given request.

Table 5summarises the status of integration for NOMAD platform use cases referring to the Visualisation Module. Currently, only a word cloud format for discovering trends in the sense of terms used in a domain or policy model has been implemented (see Figure 9), while an analysis of how the frequency of terms evolves in time is provided in a very draft format (see Figure 10).

Table 5: Development status for the use cases of the Visualisation Module

ID	Use Case Title	Completed	Next Prototype
UC18	Trend Discovery	Retrieve the list of terms relevant to a domain and their frequency in NOMAD sources in a word cloud format	View important topics in online discussions

ID	Use Case Title	Completed	Next Prototype
UC19	Time-comparative Trend Discovery	Retrieve a time span analysis of terms relevant to a domain	Observe differentiations on trending topics over time
UC20	Audience-comparative Trend Discovery	*Not developed yet*	Observe differentiations on trending topics across audiences
UC21	In-field Trend Discovery	Retrieve the list of terms relevant to a policy and their frequency in NOMAD sources in a word cloud format	View important topics in field-specific discussions
UC22	Temporal Comparative in-field Trend Discovery	Retrieve a time span analysis of terms relevant to a policy	Observe differentiations on field-specific discussions over time
UC23	Audience-comparative in-field Trend Discovery	*Not developed yet*	Observe differentiations on field-specific discussions across audiences
UC24	In-Field Trends Projection	*Not developed yet*	Acquire an estimation of trend evolution in the future
UC25	Discovery of Public Sentiment towards In-field Entities	*Not developed yet*	View sentiment analysis of attitude against entities related to a domain
UC26	Cross-time Sentiment Changes Monitoring	*Not developed yet*	Observe differentiations on sentiment towards domain-related entities over time
UC27	Cross-audience Sentiment Change Monitoring	*Not developed yet*	Observe differentiation on sentiment towards domain-related entities across audiences
UC28	Public Sentiment Projection	*Not developed yet*	Acquire an estimation of public sentiment towards domain-related entities in the future
UC29	Inform Policy Model Authoring	*Not developed yet*	Get polarized information related to norms and entities involved in a Policy Model
UC30	Discover Public Stance against Policy Arguments	*Not developed yet*	Get polarized information related to the argumentation provided by a Policy Model
UC31	Monitor Public Stance against Policy Arguments	*Not developed yet*	Observe differentiations in public stance towards policy arguments over time
UC32	Observe Cross-Audience Stance against Policy Arguments	*Not developed yet*	Observe differentiation in public stance towards policy arguments across audiences
UC33	Project Argument Matching	*Not developed yet*	Acquire an estimation of the public stance towards policy arguments in the future

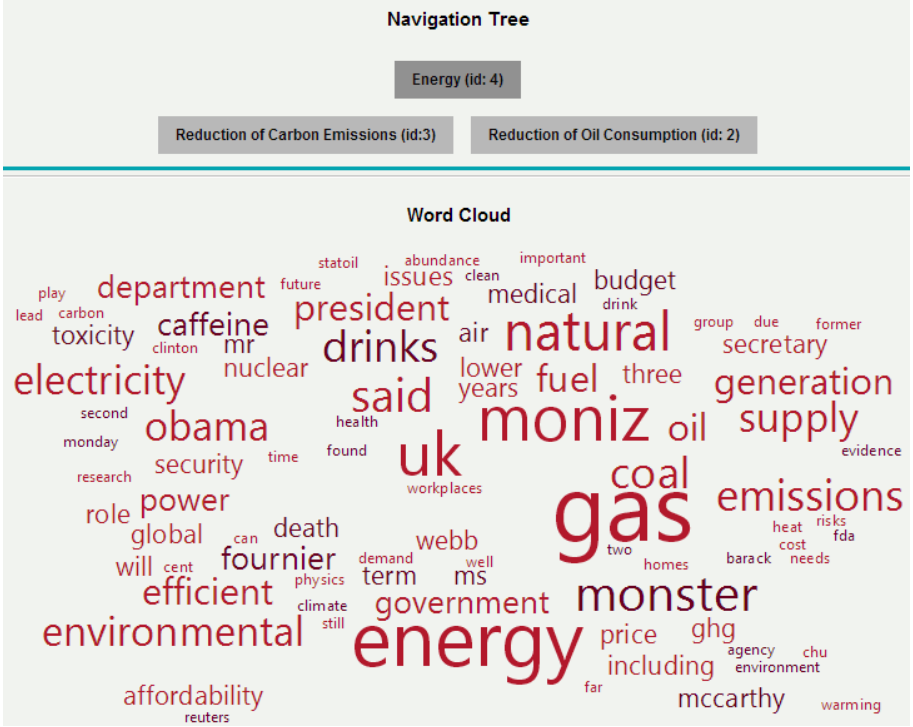

Figure 9: Example of word term visualisations

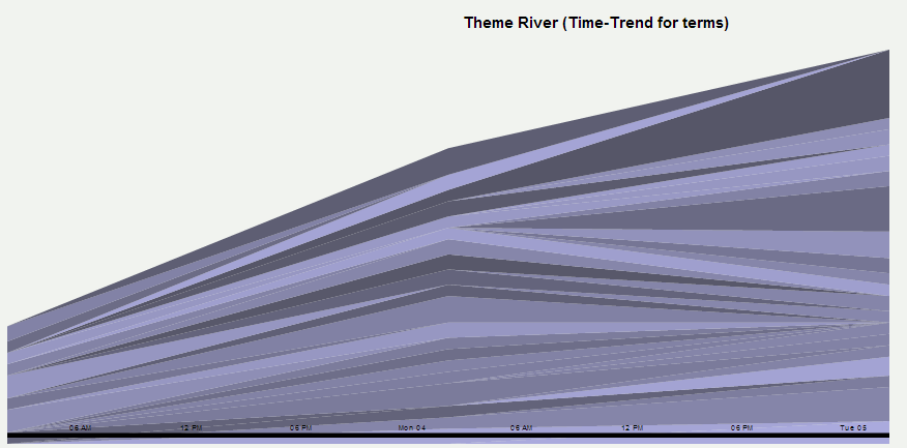

Figure 10: Example of time span visualisations

As per NOMAD Description of Work planning, the visualisation is expected to be provided by M20.

4. THE NOMAD TESTING FRAMEWORK

4.1 The Software Assurance Process

The NOMAD Integrated Prototype will be evaluated in the context of WP6 to assess the maturity of the technical implementation and the alignment to the user requirements from a technical perspective. The technical assessment of the NOMAD platform is supported by monitoring the technical parameters of the platform performance and aims to determine how far the integrated prototype meets the technical requirements and the functional specifications. It is an internal self-assessment made by technical experts engaged in the project development phase. The experts are allocated a dedicated area of work in the form of small projects and, during the testing and technical assessment, they assess the impact of the projects they are involved in on the basis of their perception of success.

NOMAD development is tested, according to established standards on software assurance process. This process aims to assess the efficiency of the platform functionalities and provide evidence that the integrated prototype is fully functional and available for release through a software assurance process. In principle, software assurance can be realised by evaluating both the software itself (the product) and how it has been developed (the process). Both aspects are important to a platform targeting to support scalable functionalities, like NOMAD. However, in research products such as the NOMAD platform, the software assurance of the process itself is almost impossible. It is mostly due to the constant change in both the design and the requirements of the software. In a research project, these changes occur during the project cycle itself, as a result of the ongoing research in the other work packages (reflected as changes in different components of the architecture) and since NOMAD has been mainly based on agile software development. Thus, the software assurance process means testing the outcome of the research prototype, considering this as an integrated platform, consisting of the individual components, which can be reached via Web service implementation.

The NOMAD technical assessment practises are based on the ISO/IEC 12207:2008standard [15]about software life cycle processes. From this standard, the software quality assurance process in clause 7.2.3 and the clause 7.2.5 about software validation process are adopted for the NOMAD research prototype. Software validation is the confirmation that the software specifications conform to user needs and intended uses through examination and provision of objective evidence, and that the particular requirements implemented through software can be consistently fulfilled. Since software is usually part of a larger hardware system, the software validation typically includes evidence that all software requirements have been implemented correctly and completely.

Software validation is realised through quality models. In the past, different quality models have been proposed, each of which addresses different quality attributes that allow evaluating the developed software. Some of the most well-known are:

- McCall's model of software quality [16](GE Model, 1977), which incorporates 11 criteria encompassing product operation, product revision, and product transition.
- Boehm's spiral model [17](1978) based on a wider range of characteristics, which incorporates 19 criteria. The criteria in both, this and the GE model, are not independent as they interact with each other and often cause conflicts.
- ISO 9126-1 [18]incorporates six quality goals, each goal having a large number of attributes. These six goals are then further split into sub-characteristics, which represent measurable attributes (custom defined for each software product).

Recently the BS ISO/IEC 25010:2011 standard [19] about system and software quality models has replaced ISO 9126-1. Applying any of the above models is not a straightforward process. There are no automated means for testing software against each of the characteristics defined by each model. For each model, the final attributes must be matched against measurable metrics and thresholds for evaluating the results must be set. It is then possible to measure the results of the tests performed (either quantitative or qualitative/observed).

For the NOMAD case, we have adopted the ISO/IEC 25010:2011 standard, which is the most widespread reference model and it includes the common software quality characteristics that are supported by the other models. This standard defines

two quality modelsproviding a consistent terminology for specifying, measuring and evaluating system and software product quality:

- Quality in use model, which is composed of five characteristics that relate to the outcome of interaction with the system and characterises the impact that the product can have on the stakeholders.
- Product quality model, which is composed of eight characteristics that relate to static properties of software and dynamic properties of the computer system.

For our case, the product quality model is adopted. The eight characteristics, are further divided into sub-characteristics, are shown in Figure 11.

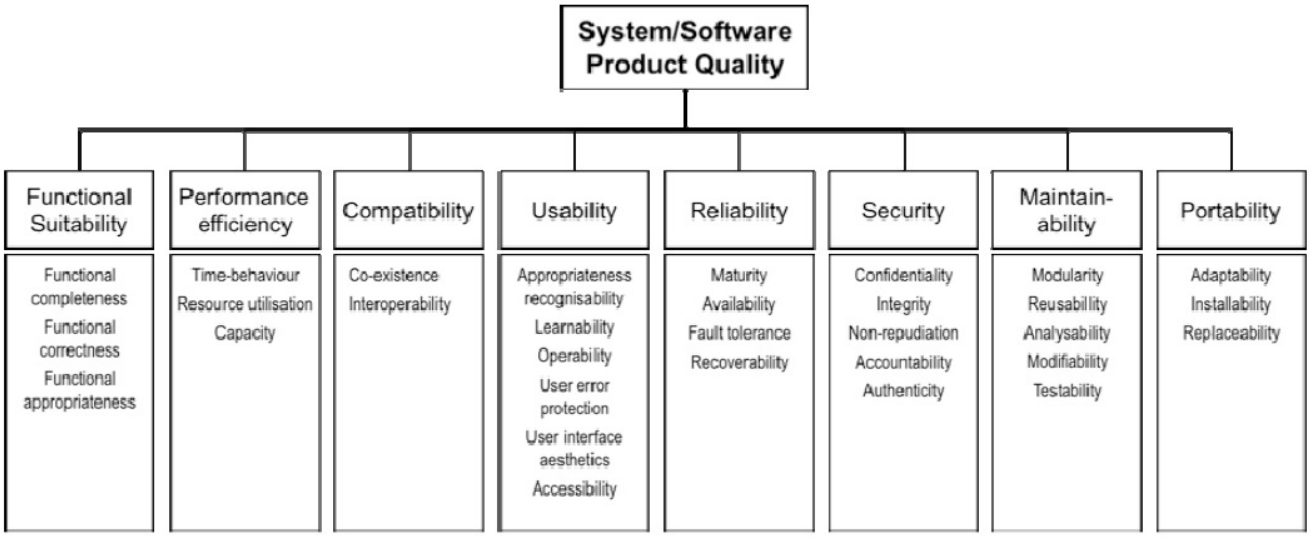

Figure 11: The ISO/IEC 25010:2011 system/software quality model characteristics [19]

For each of the sub-characteristics, a metric/measurable attribute is defined, along with thresholds. These metrics and thresholds are customised for each software product, which in our case is the NOMAD platform (consisting of individual components). By evaluating the complete set of metrics, we will be able to assess the overall quality of our platform and the percent to which we were able to meet the user requirements (reflected to system specifications and functionalities) defined during the design phase of the project.

4.2 The NOMAD Quality Model

The software quality model of the BS ISO/IEC 25010:2011 standard is adapted to the needs of the NOMAD platform, in order to define appropriate metrics and to be able to evaluate the platform capabilities. These metrics need to reflect the characteristics that they represent. They also need to allow appropriate measurements to be obtained, either through quantitative methods (e.g. by software tests/simulations, usability tests) or qualitative methods (e.g. through user observations). Three types of classes of metrics are defined in this standard:

- Internal metrics associated with static internal properties of a system such as number of function calls, number of rules.
- External metrics associated with dynamic external properties. These are metrics that are observable when the user interacts with the system (i.e. the user performs a task/function/operation and observes the response in the sense of time required, results obtained etc.).
- Quality-in-use metrics, which refer to metrics that evaluate the extent to which a system meets the needs of the user.

Since NOMAD is a research project and the aim of the relevant platform is that of a proof of concept and not a commercial product by the end of the project, there will be less focus on the internal metrics, which are essentially used in the development phase. As such, the different components and the platform itself will be mainly optimised for their functional suitability and usability. We will therefore concentrate on external metrics and quality in use metrics, which

are respectively used in the testing phase and the piloting phase. Various target users should actually make use of these metrics, as follows:

- Developers and IT experts will make use of external metrics prior to the release of the NOMAD platform prototype to the evaluation phase
- Platform end users (i.e. policy makers, domain experts, policy consultants, journalists, etc.) will make use of quality-in-use metrics, during evaluation to assess the suitability of the prototype to address the needs in the policy formulation domain.

It must be clarified that in the scope of WP6, only the external metrics will be reported, through the planned period for testing the NOMAD prototype prior to the release to other WPs. The quality-in-use metrics are attributed to the work in WP7 and will be closely related to the defined user scenarios for instantiating NOMAD in real policy formulation paradigms.

Due to the nature of the NOMAD platform, we concentrate on the following BS ISO/IEC 25010:2011 characteristics for the NOMAD platform, as shown in Table 6.

Table 6: Definition of evaluation metrics and success criteria

ISO/IEC 25010:2011 characteristics	Description	Method for measuring quality metric	Threshold[1]
Functional Suitability			
Functional Completeness	Assess the implemented functionalities with respect to requirements and project objectives	Observation Tests	# of implemented functionalities > 95%
Functional Correctness	Assess whether the NOMAD prototype provides the correct results with the needed degree of precision	Log Analysis, Observation Tests	Precision of results > 75%
Functional Appropriateness	Assess whether the implemented functionalities facilitate the accomplishment of specified tasks and objectives	Log Analysis, Observation Tests	# of correct actions > 95%
Performance Efficiency			
Time Behaviour	Assess the response and processing times and throughput rate per user	Log Analysis, Observation Tests, Simulation Tests	Mean Response Time < 1 min Mean Throughput < 2Mbps
Resource Utilisation	Assess the resource usage	Log Analysis, Simulation Tests	Memory and CPU usage < 50%
Capacity	Assess the number of concurrent users	Simulation Tests	# of con_users > 10
Compatibility			
Interoperability	Assess the interoperability capabilities of the NOMAD prototype	Observation Tests	Interoperability with at least two other separate systems
Usability: It is left to WP7 activities			

[1]For the given configuration of the NOMAD Server and at least for the NOMAD official evaluation periods

ISO/IEC 25010:2011 characteristics	Description	Method for measuring quality metric	Threshold[1]
Reliability			
Availability	Assess whether the NOMAD platform functionalities are operational and accessible when required for use	Log Analysis, Observation Tests	Availability > 90%
Fault tolerance	Assess whether the NOMAD prototype operates as intended despite the presence of hardware or software faults	Log Analysis, Observation Tests	Fault tolerance > 90 %[2]
Security			
Confidentiality	Assess whether the NOMAD prototype ensures that data are accessible only to those authorised to have access	Observation Tests	Confidentiality > 99,9%
Maintainability			
Modularity	Assess whether the NOMAD prototype operation is affected by changes to one or more components	Observation Tests	Operation breaks < 0,1%

4.3 Definition of test cases

4.3.1 Functional Suitability

In order to test the functional suitability of the implemented functionalities, the NOMAD platform has been used by developers and IT experts to identify the prototype behaviour with respect the planned one and to check whether the user requirements, as they are presented in the NOMAD Deliverable D2.2[2], and the functional requirements, as they are presented in the NOMAD Deliverable D6.1[1], have been addressed.

The testing process follows a number of test use cases, which are iteratively repeated. These tests should be successfully accomplished from the NOMAD platform and in all iterations they should provide the same results. The test cases relate to the NOMAD requirements and for each of them the result of the testing process is reported. For all the test cases, the http client of the NOMAD platform prototype has been used as the method for observation testing, while direct access to the logs of the NOMAD Server ensure that the intended use of the functionalities is performed (with no programming level exceptions to be occurred).

4.3.1.1 Functional Completeness

The functional completeness of the NOMAD platform refers to satisfying the user requirements and functional requirements as introduced in the first year of the project. Currently, the development of the NOMAD platform partially meets the planned functionalities, as depicted in

Table 7. The degree of fulfilment is justifiable from the fact that the project targeted to the delivery of core functionalities at this phase, so that a first evaluation iteration from real stakeholders can be achieved.

[2]Thus up to 10% of errors

Table 7: The functional completeness of the NOMAD platform

Requirement	Current Status	Degree of fulfilment
View important topics	As per Table 5. The back-office functionality is completed	80%
View sentiment analysis of attitude for a domain	As per Table 5. The back-office functionality is completed	30%
View polarized information related to a policy domain	As per Table 5. The back-office functionality is in draft version	20%
Acquire an estimation of trend evolution in the future	As per Table 5. The back-office functionality is completed	0%
Acquire an estimation of public sentiment for a domain in the future	As per Table 5. The back-office functionality is completed	0%
Acquire an estimation of the public stance for a policy in the future	As per Table 5. The back-office functionality is in draft version	0%
Observe differentiations on trending topics	As per Table 5	20%
Observe differentiations on sentiments for a domain	As per Table 5. The back-office functionality is in draft version	20%
Observe differentiations in public stance for a policy model	As per Table 5. The back-office functionality is in draft version	20%
Create/Enhance policy models	As per Table 4	80%
Share policy models	As per Table 4	0%
Create/Enhance domain models	As per Table 4	80%
Share domain model	As per Table 4	0%
Create account – register to platform	As per Table 3	0%
Login	As per Table 3	0%
Manage Users	As per Table 3	0%
Manage Roles	As per Table 3	0%
Monitor System	As per Table 3	50%
Setup application	As per Table 3	50%
The system should cover all the major sources of Web 2.0 content (News, Blogs, Social Media, etc.)	NOMAD Crawling can retrieve content from defined news and blogs sites, social media (i.e. Facebook, twitter, YouTube), Bing Search Engine and Google+	100%

Requirement	Current Status	Degree of fulfilment
The system should extract and aggregate demographic information from the retrieved Web 2.0 resources	A draft version of the relevant service is ready	50%
NOMAD should allow the creation, updating & sharing of a domain model	Both front-end and back-end functionalities are in draft version	75%
NOMAD should allow the creation, updating & sharing of a policy model	Both front-end and back-end functionalities are in draft version	75%
The system should allow the user to easily introduce arguments for or against a policy, using natural language	Both front-end and back-end functionalities are in draft version	75%
The retrieved content should be classified in one or more thematic categories	A draft version of the relevant service is ready	50%
The system should present relevant segments, extracted from larger textual content (articles, blog posts etc.)	A draft version of the relevant service is ready	50%
NOMAD should produce tag clouds with respect to a given domain or policy entity (key terms, topics and polarised phrases etc.)	As per Table 5. The back-office functionality is completed	80%
The user should be able to observe the change of results by posing different temporal or demographic constraints. He/ she should also be able to compare the results for different time periods and obtain an estimate of the results at the future.	As per Table 5. The back-office functionality is completed	20%
The system should provide user/role based access	As per Table 3	0%

4.3.1.2 Functional Correctness

Since this is the first working prototype of the NOMAD platform, the functional correctness is left for the next version of the NOMAD prototype and the results on this quality metric will be reported in D6.3, which is due M30.

4.3.1.3 Functional Appropriateness

The functional appropriateness has been tested through iterative test cases performed in 20 cycles and for a period of time of one week.

Table 8 shows the results of this test process for selected functionalities offered so far from the NOMAD platform. For each test case, the relevant NOMAD platform use case is indicated, so that a connection of the test cases with the defined tasks and objectives (as per Table 1) and the current status of the platform (as per Table 3, Table 4 and Table 5) is realised.

Table 8: The functional appropriateness of the NOMAD platform

Test Case	Relevant Use Case	Result
Load public domain models and navigate through their entities	UC6	Success
Create a new domain model in three languages	UC7	Success
Open an existing domain model in a specific language and browse through it	UC7	Success
Create a new domain entity and update the term in three languages	UC8	Success
Drag and drop an entity from a public domain model to the model under development	UC8	Success
Delete the structure of a domain entity and rebuild from scratch	UC8	Success
Load public domain and policy models and navigate through their contents	UC12	Success
Create a new policy model in three languages	UC13	Success
Open an existing policy model in a specific language and browse through it	UC13	Success
Create a new policy component and update the definition in three languages	UC14	Success
Drag and drop an entity from a public domain model to the model under development	UC14	Success
Delete the structure of a policy component and rebuild from scratch	UC14	Success
Drag and drop a policy component from a public policy model to the model under development	UC14	Success
Create a new argument, update the definition in three languages and specify polarity	UC15	Success
Drag and drop an argument from a policy component to another policy component of the same policy model	UC15	Success
Invoke word cloud for a domain model	UC18	Success

Test Case	Relevant Use Case	Result
Invoke time analysis for a domain model	UC19	Success
Invoke word cloud for a policy model	UC21	Success
Invoke time analysis for a policy model	UC22	Success

4.3.2 Performance Efficiency

Since this is the first working prototype of the NOMAD platform, the performance efficiency is left for the next version of the NOMAD prototype and the results on this quality metric will be reported in D6.3, which is due M30. Specific log analysis screenshots will be also collected during the piloting and evaluation phase.

4.3.3 Compatibility - Interoperability

The NOMAD platform is designed and developed under the principle of a loosely coupled architecture, taking into consideration international standards on the data transformation and information exchange, as well as the potential interoperability with third party systems. The individual components communicate with each other through well-defined APIs over REST Web Services, while JSON data exchange schema has been adopted.

Although currently the need for interfacing with external systems is not necessary, the NOMAD platform can automatically crawl data from various different sources, including social media and news and blogs sites.

On the presentation layer, the Visualisation Module is delivered as a service and is currently integrated as a Liferay portlet, which makes it being independent from the specific technology used for the presentation of the results from the NOMAD analysis. Furthermore, the Model Authoring Module uses JSF technologies and is currently developed both through two different Mind mapping tools.

4.3.4 Reliability

Since this is the first working prototype of the NOMAD platform, the reliability is left for the next version of the NOMAD prototype and the results on this quality metric will be reported in D6.3, which is due M30. Specific log analysis screenshots will be also collected during the piloting and evaluation phase.

4.3.5 Security

4.3.5.1 Confidentiality

The NOMAD platform prototype defines role-based access to specific tasks and operations for the Authoring and Visualisation parts. The different users of the NOMAD platform, as we are introduced in Section 2.4, should have different access privileges. The authentication part of the NOMAD platform is planned for the next version of the NOMAD prototype and the results on this quality metric will be reported in D6.3, which is due M30.

4.3.6 Maintainability

4.3.6.1 Modularity

The NOMAD platform prototype has been built, based on the SOA-oriented modular architecture, thus components follow a loosely-coupled approach to connect to each other. Specific results on the modularity tests will be provided in the final prototype, due M30.

5. CONCLUSIONS AND NEXT STEPS

This document is the report following the prototype deliverable D6.2 about the first integration of the NOMAD platform. This version of the platform comprises the testbed for the NOMAD end users to build the envisaged pilots in the three specified countries and experience with the NOMAD provisions in the policy making field. The current prototype integrates draft versions of the main individual components and modules, including most of the processing layer services and the model authoring functionalities.

The implementation and integration is not limited to the current level. Most research modules, as well as the platform itself are designed to be flexible and easily customisable. The overall architecture of the system facilitates modifications and makes the platform improvement to be easy, at any point. This will be the aim of the following period: to continue the enhancement of this platform capabilities, make it more stable, more sophisticated and more efficient for the target end users, by plugging additional functionalities and calibrating existing ones.

This version of the prototype will be enriched with more use cases from the visualisation module by M20 (August 2013), so that the planned first evaluation iteration will be performed in the next period (M21 – M24). The results of this evaluation cycle, along with the already scheduled work in this and the other technical WPs, will lead to another prototype release of the NOMAD integrated platform by M26 (February 2014). That version should be closer to the final one, which is expected by M30 (June 2014), in order to facilitate the conduction of the second evaluation iteration.

To sum up, this version of the NOMAD platform prototype is going to be updated, according to the results of the first evaluation period, as well as the ongoing research work in the technical WPs. An updated official release is expected by M30 of the project (June 2014), but intermediate releases are planned to facilitate the project time plan and the needs of the other WPs.

6. REFERENCES

[1] NOMAD Consortium, “Deliverable D6.1: NOMAD Architecture Design”, October, 2012

[2] NOMAD Consortium, “Deliverable D2.2: Report on User Requirements”, October, 2012

[3] Liferay Enterprise open source portal and collaboration software: http://www.liferay.com/

[4] The Eclipse Foundation open source community website: www.eclipse.org/

[5] Liferay Technical Specifications: http://www.liferay.com/products/liferay-portal/tech-specs

[6] The Apache Software Foundation - Tomcat Server: http://tomcat.apache.org/

[7] Liferay Portal Features: http://www.liferay.com/products/liferay-portal/features/portal

[8] http://jcp.org/en/jsr/detail?id=311

[9] http://wikis.sun.com/display/Jersey/Main

[10] https://jaxb.dev.java.net/

[11] Object relation mapping concepts: http://en.wikipedia.org/wiki/Object-relational_mapping

[12] Java Persistency API: http://en.wikipedia.org/wiki/Java_Persistence_API

[13] Eclipse Community – Eclipse Link: http://www.eclipse.org/eclipselink/

[14] MySQL Community Server: http://www.mysql.com/downloads/

[15] ISO/IEC 12207, “Systems and software engineering - Software life cycle processes”, IEEE Std 12207-2008, Second edition, 2008-02-01

[16] J.A. McCall, P.K. Richards, and G.F. Walters, “Factors in Software Quality,” vols. 1, 2, and 3, AD/A-049-014/015/055, Nat'l Tech. Information Service, Springfield, Va., 1977.

[17] Barry W. Boehm, “ A spiral model of software development and enhancement”, TRW, Defence Systems Group

[18] ISO/IEC 9126-1:2001, Software engineering -- Product quality -- Part 1: Quality model

[19] BS ISO/IEC 25010:2011, Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models