ePolicy

Engineering the Policy-making Life Cycle

Seventh Framework Programme – Grant Agreement 288147

Deliverable D8.1: ePolicy Decision Support System Architecture

Document type:	Deliverable
Dissemination Level:	CO (Confidential for project members)
Editor:	Federico Chesani
Document Version:	1.0
Contributing Partners:	(Values: UNIBO, UCCNUIC, US, INESC PORTO, FRAUNHOFER, RER, PPA, ASTER, UNIFE)
Contributing WPs:	WP8
Estimated P/M (if applicable):	12
Date of Completion:	30 September 2012
Date of Delivery to EC	30 September 2012
Number of pages:	20

ABSTRACT

The ePolicy Decision Support System supports the policy maker within the whole process of creating policies and plans. This deliverable introduces the architecture of the DSS, providing an overview of how it is organized, and describing from a functional viewpoint the components being part of the DSS.

Authors of this document:

ALL

http://www.epolicy-project.eu/

Contents

This page has been intentionally left blank.

1 The ePolicy Decision Support System

The project ePolicy aims to "support policy makers in their decision process across the engineering of a policy making life-cycle"[1]. To this end, ePolicy integrates into the life-cycle both global and individual perspectives on the decision process, bringing to the policy maker's attention global concerns (e.g., impacts, budget constraints and objectives), and individual concerns (e.g., opinions, and reactions).
ePolicy tackles both concerns by means of a number of components, each component providing a perspective on one or more aspects. The policy maker exploits these components to get some aid within the policy development process.

This document introduces the architecture of the ePolicy Decision Support System (DSS): with the term "ePolicy DSS" we mean the whole set of components, and their integration into a comprehensive framework supporting the policy life-cycle. In particular, the architecture presented in this document has two major objectives:

1. primarily, **to allows and support the implementation of the DSS**, as defined by the project proposal. The DSS requirements have been already discussed in the Deliverable D2.1 [2];
2. secondarily, **to support the research activity**: indeed, as a part of their research activities, ePolicy members aim also to develop the single components as well as the DSS as a whole.

In particular, w.r.t. the second objective (support the research activity), some aspects have been taken into deep consideration:

a) **Component-based.** A key point of the ePolicy project is to tackle the policy life-cycle and the various aspects within a unified approach/framework. While each ePolicy partner contributes to the project with its own skills about specific aspects, all partners contribute to develop a unified framework. To this end, the architecture must support an approach based on components, and allow to define tight integration/collaboration models between such components.

b) **Openness to different technologies.** The architecture must be flexible enough to allow the interaction between the various components, without forcing any partner to commit to specific technological choices that might undermine or constrain the research activities. Each partner should have the freedom of making the technological choices that better fit the specific research needs.

c) **Openness to new components.** The ePolicy research activity will continue along the whole duration of the project: hence, research results are to be expected across the project's life. The architecture must be flexible enough to cope with components that will be added at different moments.
Notice that adding new components might require to re-design the interactions between the components, as well as the decision process itself. The architecture must support also various possible interaction models supporting different decision processes.

d) **Synchronous and asynchronous user interaction.** Some components might greatly differ form others in many aspects: some components might require synchronous interactions, while others might be long-running process and would require asynchronous interactions models. Some components might be completely stateless, while other could require a stateful approach. Some components might require persistence of huge quantity of data, hence hinting the need for storage and/or database support.

e) **Support to different integration models.** The integration-related aspects pose as well many issues: the integration can be achieved by considering the components as "black boxes" and by designing/implementing some interaction models, or rather the integration might be achieved by a direct integration/merge of two or more components. In any case, the architecture should be flexible enough to support the integration of many components, and to allow the definition of several different interaction processes.

On the basis of the considerations above, the architecture of the DSS will be organized as a set of software services, integrated within a SOA-based system. Hence, this deliverable describe the ePolicy DSS architecture in terms of three different viewpoints. In Section 2 we provide a general overview of the architecture, describing the components of the DSS, the roles of the involved human users, and the selected technologies. In Section 3 we give a more detailed overview of each component, while in Section 4 we discuss a first, possible interaction process.

2 DSS Architecture - General Overview

The ePolicy DSS will be composed of several different components, each one providing state-of-the-art solutions and algorithms for specific aspects of the decision process. Moreover, the DSS will support the policy maker by directly implementing one (or more) *interaction processes*. With the term interaction process, we mean using all the components in a specific way, with a specific order, and sharing specifics data between the components.

The DSS will be organized as a set of services (partly organized following Service Oriented Architectures-SOA principles): the policy maker will be allowed to access the single components, as well as to follow an interaction model supported within the DSS. We distinguish two different groups of components: a first group gathers the *task oriented* components, i.e. those components that address a particular issue or aspect related to the policy life-cycle. Examples of these components are the Global Optimizer, or the Social Simulator. A second group is made of those components that provide services and *support* to the first group, and allow easy access to the integrated framework. Examples of this second group are the Persistence Support Layer, that provides persistence storage capabilities, or the Scheduler, that takes care to invoke and manage long-running processes. The architecture, showing the components, is depicted in Figure 1.

2.1 Support-Oriented Components

Currently, we envisage different components that will play the role of supporting the use and the functioning of the DSS:

The Persistence Support Layer will provide support to (i) store partial results coming from the invocation of the services; (ii) storing data for further inspection, simulation, data mining and visualization; and (iii) storing the state of user interaction, since it is the case that each decision process is a "many days" process. In such perspective, it is necessary to store the state of the user interaction by mean of proper data structures. This component will be mainly based on Relational Database Management Systems.

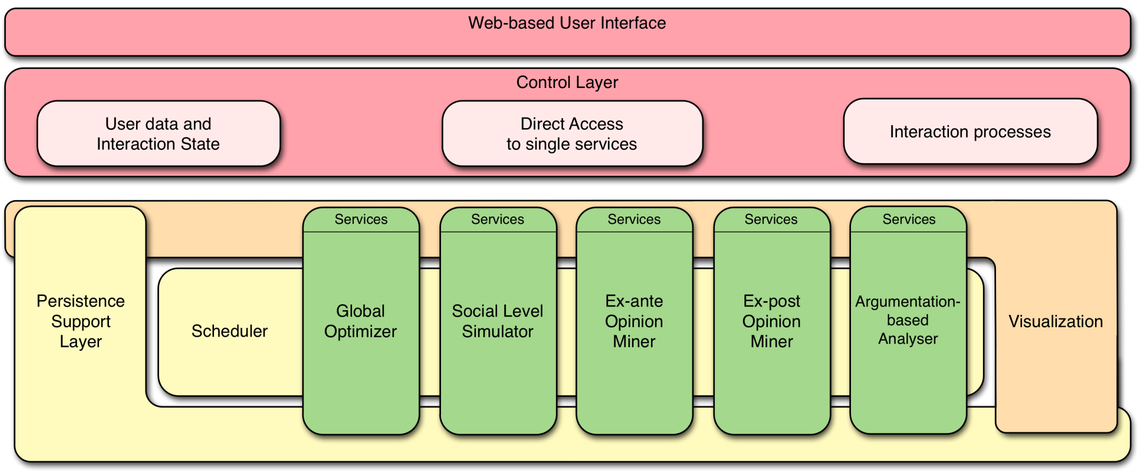


Figure 1: Components-based architecture of the ePolicy DSS.

The Scheduler Component will take care of invoking the components based on long-time processes. E.g., task-oriented components such as the Opinion Mining services need to be frequently updated to reflect opinions currently available on the web and on e-Participation platforms; while this update process can be triggered directly by the user requests, it is reasonable to schedule the updates off-line, thus making data available without waiting for the user intervention.

The Control Layer will provide support to the policy maker, storing personal and authentication data, allowing the invocation of the single services and the storing/retrieval of the results. Moreover, the control component will implement and support the *interaction processes* identified within the project, thus enabling the ePolicy life-cycle.

The User Interface will be based on a web access, and it will allow domain experts and policy makers to respectively configure and use the ePolicy DSS.

We do not mention here the Visualization Component, that will provide support and access to the others components, in terms of innovative graphic-based interfaces. More exactly, it will allow the users to "explore" and "navigate" huge quantity of data provided by other components. From a functional viewpoint, the Visualization might be perceived as a support-oriented component. However, since the focus of this component is explicitly on new visualization techniques, we decided to consider it as task-oriented component.

2.2 Task-Oriented Components

In the following we provide a brief overview of each task-oriented component: each of them is focused on specific aspects or issues that arise in the policy life-cycle. A detailed description of the components is given later in Section 3. Note that the following list of components is not complete or exhaustive: newly components might be added at a later moment within the project, as outcome of the research activities.

Global Optimizer. This component receives as input many different type of information, such as constraints on the energy production, citizen preferences extracted by the opinion mining, expected citizen behaviour on the base of social simulation etc. Its main objective is to compute plans that optimize objective functions defined by the policy maker. The objective functions can take into account economical objectives, environmental impacts, social and political desiderata.

Social Simulator. By means of simulations, it provides probabilistic estimations of how the citizen will behave w.r.t. certain aspects. E.g., a probabilistic distribution of how many citizen will adopt solar panels as a consequence of a certain economic incentives plan, or after an aggressive information campaign.

Ex-ante Opinion Mining. It aims to give the policy maker a feedback on the citizens preferences about specific topics. To this end, it continuously crawl public web sites and e-Participation sites to gather updated information.

Ex-post Opinion Mining. Very close to ex-ante, ex-post opinion mining focuses the research of citizen preferences but with respect to possible plans that the policy maker might have shared through e-Participation tools and sites. The aim is to get citizen's feedbacks about possible alternative plans.

MicroDebates Analyser. As a part of opinion mining techniques, this component exploits argumentation-based techniques to better understand how argumentations are brought on in public debates among citizens. The goal is to automatically identify which are the most important arguments in favour or against some policy-related choices, so as to help the policy maker to engage further actions, such as build a new plan or promote information campaigns to better inform citizens.

2.3 Users Roles

The principal aim of the ePolicy DSS is to aid policy makers to develop plans and implementation policies for specific aspects. This prototype in particular will be fitted for the energy-related field, and for alternative energy sector in specific. Beside the policy maker, we envisage at least other two possible type of users that are involved within the policy making process: the administrator, responsible for the correct functioning of the whole prototype, and a domain expert.

- The **system administrator** is responsible for managing the DSS infrastructure. Being based on services and web interfaces, the installation and maintenance of the DSS require technical skills and a proper role of system administrator. With respect to powers and rights, the system administrator has all the possible privileges within the system.
- The **policy maker** is the main intended user of the DSS. In particular, he/she might be able to invoke the services, following some specific interaction process or just querying one single service. When using the system, the policy maker can provide inputs depending on its own goals, such as for example the desiderata of minimizing costs or maximizing the amount of produced energy. Moreover, he/she should be able to store computed data and retrieve them at a later moment.
- The **domain expert**: is responsible for properly configuring the components, based on the peculiarities of the domain and of the scenario taken into account. E.g., the efficiency of a photovoltaic energy plant highly depends on the geographical position, as well as on the plant technology. The domain expert can access the system, and provide these information to properly configure the components. For example, the Global Optimzer needs these data to compute a more reliable plan, while the Social Simulator needs trustworthy information about the population it is going to simulate. Notice

that the policy maker should not be allowed to modify these data: indeed, these values should be based on scientific claims, and not subject to the desiderata of the policy makers. With respect to the privileges, the domain expert should be able to simulate the construction of a plan, by using each component as a single service, or the components as a whole; he/she must be allowed to change configuration parameters of the single components; finally, the domain expert should be able to store/retrieve current/previous computed results and plans form the persistence support layer.

The support to the roles, and the enforcement of rights and privileges will be part of the control layer depicted in Figure 1 .

2.4 Technologies

As specified in [1], the DSS will be developed with an open-source approach, when possible. To this end, the architecture will be based, on well-established and open-source solutions. In the following, we briefly describe the technological choices, with pros and possible cons.

Each single component will offer its own services using the Web Services standard [3]. This choice is motivated by the following reasons:

- Web Services frameworks are available for almost any hardware/software platform, and for almost any modern programming language; moreover, open-source frameworks as well as proprietary solutions are available. When developing components, any partner can freely choose hardware and software basing on its own specific needs.
- Each partner can develop its own components without worrying about the integration with other partners: the integration between the components, that is a key point of this project, is addressed at a higher level, and at a later moment w.r.t. the implementation of the components. Moreover, different (alternative) ways of integrating the components can be added later during the project.

This choice has an immediate drawback: communication between the components can be delayed because of the network. This aspect might be particularly relevant if huge quantity of data needs to be moved between different components. To avoid this we envisage to run the services as "close" as possible to each other (same sub-network). If needed, we plan to resort to services running on top of Virtual Machines, running in turn on top of the same physical server. In any case, a persistence support is envisaged to store computed data: hence, the persistence layer can play also the role of asynchronous access to the data.

As a general guideline, by no means mandatory, we provide also the following suggestions:

- Components should be running on top of physical/virtual servers with Ubuntu[1] installed as operating system; version 12.04 or any LTS (Long Term Support) version should be preferred.
- As a development language for the services, the open source Java Technology[2] is the first choice, mainly for its flexibility and the large support to internet based applications and services.

[1] `http://www.ubuntu.com/`

[2] `http://www.java.com/`

- As application container, components could exploit the Apache Tomcat[3] application server: it is a well known, largely used and established Servlet/JavaServerPages Container, it is open source, and it is available on the majority of hardware/software platforms.
- As a framework for supporting the Web Services implementation, our preference is about the Apache CXF Framework[4], based on the light-weight application container Spring[5].
- As a "light-weight" alternative to the technological stack "Tomcat+CXF+Spring", we consider the Jolie Framework[6], that allows a easier and faster development of single services.

In general, it is highly plausible that large quantity of data will be accessed by one or more components. For example, the Visualization Component will access different data sets provided by the components, helping the policy maker to compare different scenarios and solutions. Such capability might require to access data as fast as possible. To this end **the architecture envisages a persistence layer** implemented on MySQL DBMS[7]. MySQL is a well-known, fully established DBMS that is available on the majority of the hardware platforms, it is open-source, it is fully scalable and supports replication and redundancy; moreover, it has been recognized to be an industrial-quality product.

The integration between the java-based components and the persistence layer will be based on Object-Relational Mapping (ORM) technologies, and in particular by exploiting the Java Persistence Annotation (JPA[8]) standard. Hibernate[9] will be the preferred implementation of the JPA standard.

The integration models and processes that will be identified during the project will be implemented within the architecture. To this end, we have identified two key technologies:

- Persistence w.r.t. the decision process lead by the policy maker will be supported by means of the standard Enterprise Java Beans 3.0[10]. This standard, principally aimed to enterprise-class applications, directly provide support for persistence of complex processes, support to ORM-based persistence, replication, transaction and concurrency. The application container will be Spring, already used for the support to the CXF framework.
- Orchestration of different components will be achieved by exploiting the Jolie Framework, that offers a very simple language to compose simple services into complex ones, it is open-source, and comes equipped with an interpreter/compiler of the service orchestrations to directly execute the specifications. Moreover, Jolie is fully based on Java technology and is available for the majority of hardware/software platforms.

[3]`http://tomcat.apache.org/`
[4]`http://cxf.apache.org/`
[5]`http://www.springsource.org/`
[6] `http://www.jolie-lang.org/` and `http://www.italianasoftware.com`
[7]`http://www.mysql.com/`
[8]`http://www.jcp.org/en/jsr/detail?id=317`
[9]`http://www.hibernate.org/`
[10]`http://www.oracle.com/technetwork/java/javaee/ejb/index.html`

3 Task-Oriented Components

In this section we provide a detailed overview of each task-oriented component, with the aim of clarifying their role within the DSS, and to foster discussion about possible interactions processes and models. While the internal architecture of each component will be explained in details in specific deliverables within other work packages, here we describe the components in terms of "black boxes", and in terms of input and output parameters that are needed to invoke the components, and that are provided back from the components, respectively.

Although it might not be always possible, we distinguish three types of parameters. *Input parameters* are provided by the policy maker to the components: among them, we consider the policy maker's objectives, and other constraints that are related to specific plans, such as for example the amount of financial resources available for implementing that specific plan. *Output parameters* represent the outcome of each component computation: such parameters are explored by the policy maker, or can used as input parameters to other components.

Finally, we introduce the third group of parameters, namely the *configuration parameters*, to indicate those parameters that are strictly related to the specific domain, but not to a specific plan. For example, the Global Optimizer component evaluates how much energy production activities affect environmental receptors such as the quality of the air, or the quality of surface waters. The mathematical relation between a type of energy production system and the environment is given by environment experts. Clearly, the policy maker should not be allowed to change such relations: otherwise, it would be possible to generate any plan, but it would not be possible to properly evaluate it, since environmental impacts would be biased by the policy makers wills.

The distinction between input and configuration parameters is specific to each component. The DSS architecture indeed must support this distinction by providing different privileges to different user roles.

3.1 The Global Optimization Component

The global optimization component has the main objective of providing optimal plans given the user's requests. It will take into account environmental constraints, economic constraints and provide the optimal solution.

3.1.1 Input parameters

The component takes as input:

- **Objective function**: the function to be optimized. It can involve environmental receptors (e.g., the quality of the water, quality of the air, etc.), economic factors (like the total cost of the plan), other requirements (e.g., the total produced energy in a year, for an energy plan). The function could be a linear combination of the various terms. In the future, if it is considered necessary, we could consider non-linear functions as well.

3.1.2 Output parameters

The component provides as output:

- **one or more plans**: for each activity, a plan establishes how much of that activity should be developed. Each plan will be optimal with respect to different aspects, such as the cost, or the energy produced, etc.
- **cost**: the total cost of each plan.
- **the environmental footprint**: for each environmental pressure, an indicator (obtained from qualitative values) of the impact of each plan on the environment.
- **the modification of environmental receptors**: for each environmental receptor, how much it is improved or worsened by each plan (qualitative value).

3.1.3 Configuration parameters

Configuration parameters will be:

- **Possible Activities**: the list of the possible activities that can be developed in the plan, together with their unit cost.
- **Bounds on the activities**: for some of the activities there are bounds to be respected. For example, the total energy from solar power depends on the available surface that can be dedicated to such activity.
- **Primary/secondary activities**: Some of the activities are considered primary for a regional plan. For example, an energy plan considers as primary activities power plants and activities that can generate power (e.g., incinerators). Other activities may be necessary to support the primary activities; in an energy plan, the primary activities (mainly, power plants) require pylons and power lines, (large) hydroelectric plants require dams, etc.
- **Outcome of the (primary) activities**: the type of outcome depends on the type of plan. For an energy plan, the outcome can be the total energy (kTOE) produced by a plant in a year.
- **Constraints between activities and receptors**: the co-axial matrices relate the impacts of the activities with environmental pressures, and link pressures to receptors.

Configuration parameters may include also the type of algorithm to be used for obtaining the optimal plan (in case more than one algorithm will be developed).

3.1.4 Roles involved within the component

The roles involved in the component are:

- **policy maker**: selects the objective function to be optimized, imposes some of the bounds (e.g. maximum allowed budget);
- **domain expert**: depending on the domain, the expert provides specific parameters, to properly tune the component; e.g., the environmental expert provides the relationships (e.g., co-axial matrices) between activities and environmental receptors/pressures. He can also provide some of the bounds (e.g., the total availability of some energy type in the region).

3.2 The Social Simulation Component

When performing a social simulation, similar to any other scientific experiment, one wants to observe the behaviour of the system (the simulation) with respect to the parameter-setup of the simulation. Rephrasing this, one wants to examine its output (emergent system properties) resulting from some input parameter specification. These parameters can for example be numeric values (e.g. the number of agents to be modelled), boolean values (e.g. whether a particular policy instrument shall be used or not) or proportions (e.g. percentage of people that already have a photovoltaic system).

For the social simulation the tool NetLogo will be used. NetLogo is an agent-based programming language and integrated modelling environment.

3.2.1 Input parameters

From the input perspective, NetLogo allows for two main ways to alter parameters of a simulation setup: (i) changing the parameters by hand before each simulation experiment, or (ii) using a tool to automate this process.

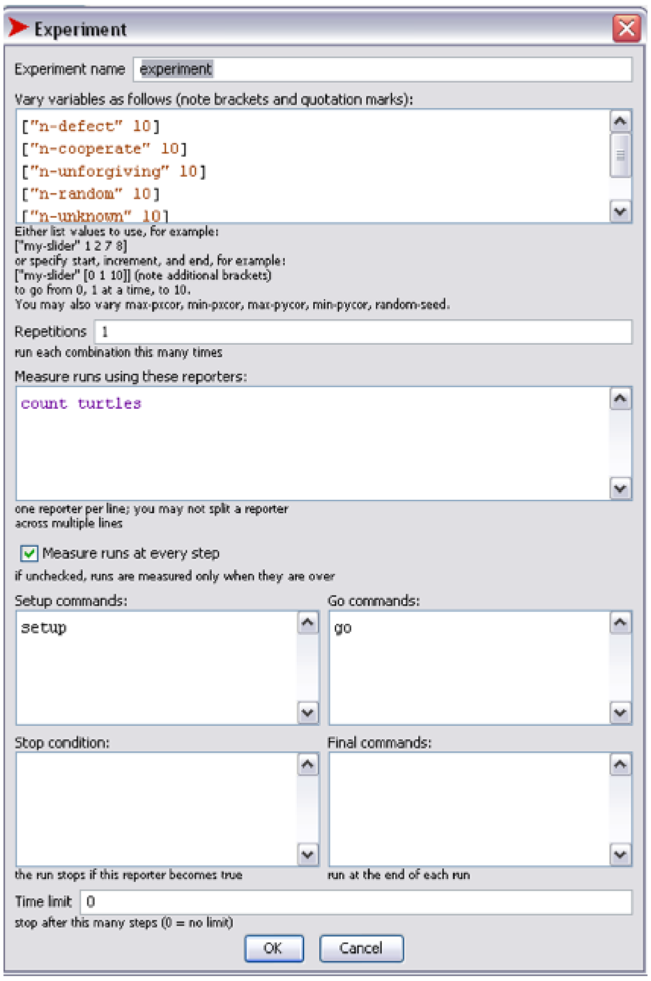
Due to the large number of experiments that we aim to perform, we shall use the second option. For this purpose a feature built into NetLogo called "BehaviourSpace" is typically used (see Figure 2).

Using the dialog shown in Figure 2, the value for each parameter can be specified using the simple form ["parameter name" [value1 value2 ...]] and once all parameters have been defined, experiments sweeping through all possible parameter combinations can be performed. It is possible to run BehaviorSpace experiments "headless", that is, from the command line, without any graphical user interface (GUI). This is useful for automating runs on a single machine or a cluster of machines. No Java programming is required for running the BehaviorSpace headless, instead experiment setups can be created in the GUI and then run later from the command line. Detailed information on the commands to run the BehaviorSpace headless can be found on the following website: `http://ccl.northwestern.edu/netlogo/docs/behaviorspace.html`.

It has to be noted that in Netlogo the specification of parameter values does not need to be done through the interface of the BehaviourSpace program, although this is the most common way. Other options include reading values from Excel or CSV-files (e.g. with the NetLogo's *import-world* command), or programming parameter sweeps into the NetLogo program itself.

3.2.2 Output parameters

For each of the experimental runs with one parameter combination, the results of the simulation experiment may be written to output files that can either be specified by the user of the simulation or programmed into the code. They are used to track output parameters that are of interest over time.These output files are typically in plain-text, "comma-separated values" (.csv) format. CSV files can be read by most popular spreadsheet and database programs as well as any text editor. The functions used for this export in Netlogo are the functions



Experiment

Experiment name: experiment

Vary variables as follows (note brackets and quotation marks):

```
["n-defect" 10]
["n-cooperate" 10]
["n-unforgiving" 10]
["n-random" 10]
["n-unknown" 10]
```

Either list values to use, for example:
["my-slider" 1 2 7 8]
or specify start, increment, and end, for example:
["my-slider" [0 1 10]] (note additional brackets)
to go from 0, 1 at a time, to 10.
You may also vary max-pxcor, min-pxcor, max-pycor, min-pycor, random-seed.

Repetitions: 1
run each combination this many times

Measure runs using these reporters:

```
count turtles
```

one reporter per line; you may not split a reporter across multiple lines

[✓] Measure runs at every step
if unchecked, runs are measured only when they are over

Setup commands:

```
setup
```

Go commands:

```
go
```

Stop condition:

the run stops if this reporter becomes true

Final commands:

run at the end of each run

Time limit: 0
stop after this many steps (0 = no limit)

Figure 2: The Netlogo BehaviourSpace

x	y
01-2013	1298
02-2013	1659
03-2013	1819
04-2013	1892
05-2013	1935
06-2013	1964

Table 1: Number of Households with Photovoltaic Systems

export-plot, *export-all-plots* and *export-world*.

One output parameter that is of interest for the social simulation is the total number of households adopting photovoltaic systems over time. Table 1 shows what the logged output for this parameter might look like. The x-variable in Table 1 indicates the time (MM-YYYY), and the y-variable, the total number of households having a photovoltaic system. Using this information, one could, for example, develop theories of the dissemination of photovoltaic systems over time.

The outputs generated by the social simulation component is primarily going to be used by the Visualization component and by the Global Optimizer. We see the role of Visualization as being in the reprocessing of the results of the individual runs, the incorporation of the relation between simulation setup and simulation results, the comparison of simulation experiments with slightly different setups and the presentation of consolidated results in a form that is easily understandable for the stakeholders of the decision support system (e.g. policy makers). The social simulation therefore provides the visualization component with simulation results (which will vary according to the input parameters of the simulation). The Global Optimizer instead can exploit the social simulation outputs when computing a plan: for example, the estimated behaviour of the citizens can affect the total amount of installed solar panel, hence affecting the total amount of renewable energy produced.

3.2.3 Configuration parameters

Before running a simulation in Netlogo, the size of the java heap (especially the maximum heap) assigned to the simulation (and Netlogo in general) may need to be set. Adjusting the heap is advisable because large heap size is required at the start of the simulation when the GIS information is loaded. Information on how to change the (maximum) size of the java heap in Netlogo (depending on the operating system) can be found on the Netlogo FAQ website (`http://ccl.northwestern.edu/netlogo/docs/faq.html#howbig`).

3.2.4 Roles involved within the component

The main actors interacting with the social simulation component besides the other components of the DSS are policy maker which want to analyse how different simulation input settings effect the results of the simulation. If they interact directly with the simulation component (rather then the complete DSS), they can use the visual interface of the simulation to specify the values of input parameters. This interface is provided by Netlogo. When interacting with the complete DSS, we envision that the values of the simulation parameters are going to be provided by the visualization component which acts as main user interface for the DSS.

3.3 The Opinion Mining Components

The opinion mining components will be responsible by inferring the sentiment/opinion of the population concerning a series of alternative policies and their respective configurations. At the global optimization level the opinion mining components will produce *exante* policy evaluation by collecting the citizens' opinions on particular websites. As soon as the global level optimization produces a series of feasible alternatives the opinion mining components will infer the *expost* citizens opinions on the policy options available.

3.3.1 Input parameters

The typical interaction of the user with the opinion mining components will consist on asking for the opinion of the population concerning a specific topic. This means that the main input parameter will be the topic of interest to the user.

Other less relevant input options might be the possibility of letting the user to restrict the query concerning the opinion of user to a particular time window or even to some particular websites.

3.3.2 Output parameters

As output of the opinion mining components we propose a set of opinion scores consisting of integer numbers in the range $-2, 2$, for each of the selected topics. A score of -2 will correspond to the maximally negative opinion, whilst 2 is the maximally positive opinion. We will also produce information on the trends of these scores in the form of temporal graphs. Figure 3 shows an example of this type of output.

Finally, some form of justification of the opinion score will also be considered as output of these components.

3.3.3 Configuration parameters

The opinion mining components are based on data available at selected thematic websites that allow the users to express their opinions. In this context, the main parameter for these components are the sites that should be monitored for these opinions. The system will use by default a series of pre-defined websites, but the user should be allowed to extend this set by adding new websites that she/he deems relevant for the problem under consideration. However, there will be a series of restrictions on the type of sites that will be possible to add without needing further tuning of the opinion mining module.

The opinion mining components will collect user messages from the selected websites and analyse these messages to try to extract/infer the opinion of the poster concerning a series of relevant topics. There will be a pre-defined set of topics but we will allow the user to extend

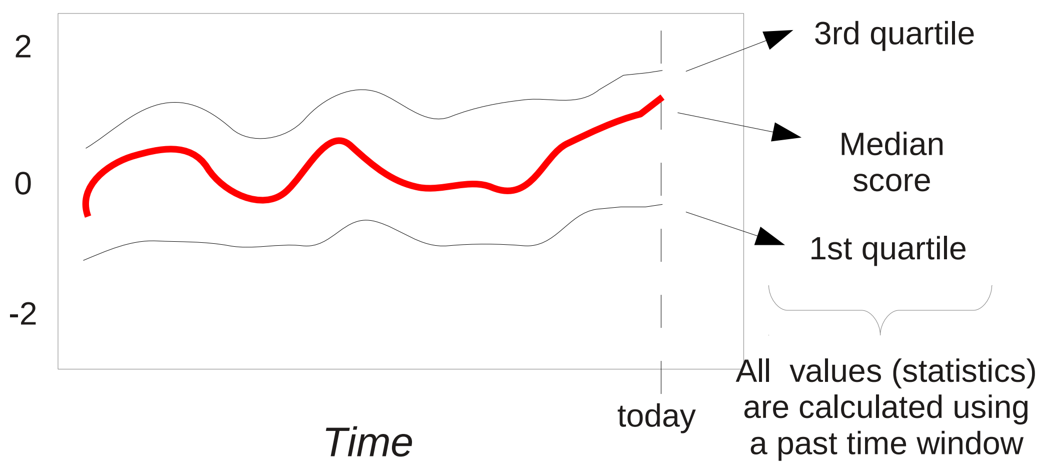


Figure 3: Example output of opinion mining trends.

this set by adding new topics of interest. We should remark that adding new topics of interest will be a computationally demanding operation as it will require the components to collect historical data on these topics and develop models for inferring sentiment/opinion on these new topics.

3.3.4 Roles involved within the component

The opinion mining components will have two main roles: an administrator role that will be able to take care of things like model tuning, updating and construction; and a standard user role that will essentially query the module for sentiment scores for a selected set of topics.

3.4 The Argumentation-based Analyser

MicroDebates aim at resolving argumentation framework retrieved from Twitter by means of computational argumentation models and basic network analysis techniques. Such argumentation frameworks consist of stream of tweets, called *micro-debates*, where users use two kind of tags, # and $ (along with its opposite !$). The first one is standard Twitter syntax, used to refer a stream of contents. The second one is used to refer the argument pro ($) or against (!$) which the user is tweeting. The module is able to retrieve a users-generated micro-debate, map tweets according to the arguments each tweet supports or attacks, and finally calculate semantics, i.e. models of reasoning, in order to reveal the winning positions in the argumentation framework.

3.4.1 Input parameters

The module requires three inputs:

- *Debate name:* the micro-debate name, a string which represents the name of the microdebate that the user wants to analyze;
- *Strategy:* the strategy used to parse the stream of tweets, one of "Naive" or "Smart". In the former case, each argument supported is considered a valid argument, while in the latter semantic techniques are used to filter and retain only well-formed argument;
- *Semantic:* the semantic used to reveal winning positions in the argumentation framework; the main semantics available in computational argumentation literature have already been implemented in the module: "Admissible", "Complete", "Grounded", "Ideal", "Preferred", "Stable", "Semi Stable", and "Stage".

3.4.2 Output parameters

The output result is a graph that represents the argumentative framework. Each node represents an argument and contains all the tweets retrieved supporting that argument. Links represent attacks between arguments. The user can inspect comments and change semantics in order to visually evaluate possible results.

3.4.3 Configuration parameters

The current prototype of MicroDebates does not envisage any configuration parameter.

3.4.4 Roles involved within the component

The MicroDebates component is principally aimed to policy makers, that will use it to capture the debates, and identify major arguments used in such debates.

3.5 Visualization

The visualization component will consist of a web service implementing several visualization techniques. Its purpose is the visual-interactive access to the data and the functionality provided by the other technical components. Here, the focus lies on:

- the Global Optimization component
- the Social Simulation component
- the Opinion Mining component.

For each of these components visualization techniques will be designed and implemented. The user will use these visualization techniques to

1. get an overview of the output data generated by the other technical components. Here, the user will define a visual query by choosing predefined parameter settings and get pre-computed output data from a database.
2. control the generation of new output data by choosing new parameter settings, and start new analysis runs. The output data will be generated by the other technical components and then, visualized by the visualization module.

In the first case the pre-computed data is transferred from the database to the visualization component. In the second case the data is calculated on-line and transferred directly from the other technical components to the visualization component.

3.5.1 Input parameters

As described above, the input data for the visualization consist of the output data generated by the other technical components. This data is visualized in the visualization component.

3.5.2 Output parameters

The output data of the visualization consists of graphical queries that the user can define supported by the visualization. With these queries she will get access to the pre-computed data available in the database. Moreover the computation of new data by the other technical components can be executed by these queries.

3.5.3 Roles involved within the component

The visualization component will possibly consist of two different modes. An expert mode and a politician mode. Since most politicians are no IT-experts the functionality of the visualization module will be restricted to the most important. That way she can get access to the relevant information provided by the other technical components while not being overwhelmed by technical complexity. The expert mode will provide full access to the functionality.

4 Interactions between the components

A key point of the ePolicy proposal is the exploitation of all the different components/services in a comprehensive manner to support the policy life cycle. Indeed, this means to identify one or many possible interaction processes that support the policy maker activity. Many possible interactions between the components can be achieved: in turn each possible interaction may depict a different way the policy maker will use the DSS. In the following, we describe one interaction process, being aware that more different processes might be expected as project outcomes. The process is depicted in Figure 4.

We describe an interaction process in terms of a use case scenario where the main actor is the policy maker. The policy maker starts by logging into the system. Once authenticated, he/she can opt for two different paths: if data or plans have been already computed, the policy maker can use the visualisation tools to explore such data or plans; otherwise, if data or plans are not available, the user can start to build a plan. Let us consider the latter situation.

A first step focuses on data from the Social Simulator (Section 3.2) and from the Ex-Ante Opinion Miner (Section 3.3). If data is not available yet, the policy maker can invoke the services to compute the outputs. However, depending on the input parameters specified by the policy maker, such process might require a long computation time: roughly speaking, we can estimate such computation time as spanning from few seconds up to several hours. Of course, better hardware architectures can lower the needed time, but anyway it depends by the input parameters specified by the policy maker. Note that to increase the user experience, the DSS architecture already envisages the use of a scheduler that takes care of invoking these components in advance, so as to pre-compute the data before the user really needs them.

A second step consist to build the plan: the policy maker invokes the Global Optimizer com-

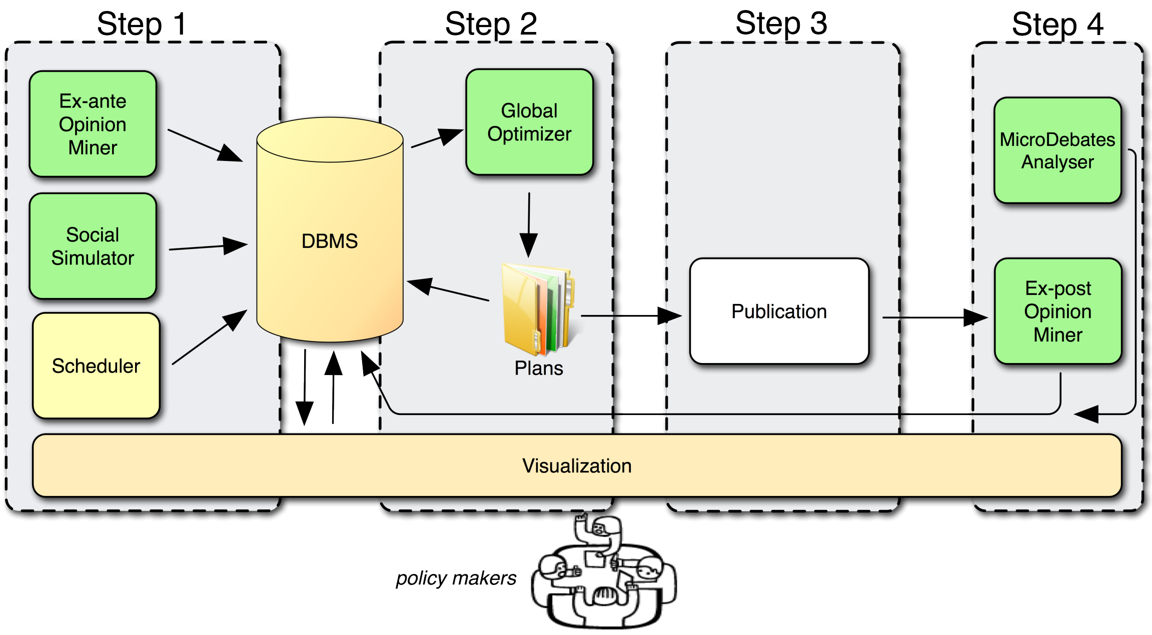


Figure 4: A first interaction model that will be supported by the ePolicy DSS.

ponent (GO), by specifying the proper parameters, as described in Section 3.1. Moreover, the GO will take into account also the results computed by the Social Simulator and by the Ex-ant Opinion Mining components. Although our preliminary tests on a GO prototype clearly show that the computation time is small enough to allow direct interaction, the policy maker can exploit the scheduler to run the component and store one (or more) plans.

Once one or more plans are available, the policy maker exploits the visualization tools to understand the peculiarities of each plan, and choose some of them for the third step, that is the publication (through e-Participation systems) of the plans (to be intended as possible alternatives).

As a fourth step, the policy maker exploits Ex-Post Opinion mining and the Argumentation Analyser to get a feedback about a plan (and possibly about several alternatives plans). At this stage, the policy maker can decide to select a plan and make it the first choice (thus exiting the process), or rather he/she can go back to the first or the second step, and re-run the services with different input parameters.

Of course, at any time within the process, the policy maker, if not satisfied by the current results, can go back to previous steps and recompute new results. The visualizations tools play a key role in this process, since they allow the policy maker to explore the computed results, and to understand the outputs of each single component.

References

[1] ePolicy Consortium. Annex i - description of work.
[2] ePolicy Consortium. Deliverable 2.1: Decision support system requirements.
[3] W3C. Web services activity. http://www.w3.org/2002/.