Decision Analysis Services: A Framework for Distributed Decision Making over the Web

César Alfaro
Department of Statistics and Operations Research, Rey Juan Carlos University.
Fuenlabrada 28943, Madrid, Spain cesar.alfaro@urjc.es

There has been an increase in the desire to participate in governmental decisions by citizens in recent years. This has motivated the emergence of hundreds of participatory instruments to facilitate the transmission of citizen concerns and desires. Similarly, there has been an increased use of new technologies allowing citizens to contact and communicate their preferences to the political class. We develop here an architecture to support distributed decision making, allowing to construct and implement most participatory processes. It is based on a standard that enables information exchange among various phases of the process, using XML and Web services.

Key words: group decision making, decision support, decision analysis, e-democracy, e-participation, configurable software, XML, SOA, Web Services

1. Introduction

Since the 60's, an increasing apathy and feeling of alienation among citizens has led to the so-called democratic deficit (Steffek et al. 2007), which has entailed an increasing interest in promoting participatory processes, allowing citizens to take part in public policy decision making. Participatory processes are on a clear rise for several reasons including: increase of legitimacy, acceptance and transparency in the decisions made; approaching decisions to citizens; taking advantage of the local knowledge that citizens might have; educate politicians, remembering them that they are elected to represent citizens, possibly mitigating clientelism; educate citizens to make them understand that decisions entail benefits and costs that need to be balanced; enhance diversity, including additional perspectives on a problem; and, reduce the apathy which causes the above mentioned democratic deficit. In this context, participatory instruments are flourishing all over the world, see Rowe and Frewer (2005), including, to name but a few, classical ones like referenda or town meetings, to more recent ones like stakeholder workshops, participatory budgets or citizen juries, to more sophisticated ones like decision conferences, see Gregory et al (2005). Some of them are just means to allow citizens to inform politicians so as to guide decision making; others allow for co-participation among citizens and politicians in joint decision making; finally, some of the instruments do actually allow citizens to make the decisions.

As consequence of the rise in the use of these instruments for participation in decision making and the increasing use of the Internet, we find that, in fact, some of these instruments are being implemented through the Web. Thus, it seems timely to take advantage of the growing role of Web tools to support decision making and citizen participation, as well as the features of XML and its application in various fields, to create a standard to be used to promote and facilitate web based group decision making, AKA e-participation. Indeed, some of the few e-democracy and e-government initiatives suggest the benefits of using XML in this context. One of them is the ITALO project, see OficinaVirtual.mityc.es (2011), started by the Spanish government to facilitate the exchange of administrative information between public administrations. Another example is the GovTalk project, see CabinetOffice.gov.uk (2011), started by British government. While both proposals are groundbreaking in its conception, they simply use XML for administrative operations such as payment of taxes or the application of certain services. However, they do not provide support for decision making by citizens.

The development of technologies like XML, SOA or Web Services allow us to use a Software as a Service (SaaS) model, that is, software running on a computer accessible from anywhere and anytime through an Internet connection. Indeed, the increasing emphasis in cloud computing is putting SaaS at the core of ICT. The main advantage of this model is that it minimizes the investment cost in powerful machines with high capacity to run complex applications, because the application is executed in the service provider's machine. The best known example of SaaS is *Salesforce.com*, although, there are many more like *Google Apps.* In the field of Operations Research, two relevant projects are OSiL, see Fourer et al. (2010) to represent optimization problems and provide optimization services, and Decision Deck, see Bisdorff et al. (2009), to support multiple criteria decision making.

Our work here presents decision analysis services and it is based on the SaaS model. Since participatory instruments began to be used, participation has increased. However, despite they bring many benefits, each instrument has its advantages and disadvantages, including the large amount of resources, whether of space, time or finance, that may entail their use. We have observed that there are actually common tasks to all participatory instruments. Thus, we could develop a generic algorithm that permits to adapt our system to each specific participatory process by deleting or repeating stages. These tasks follow the order established by the problem designer. They could appear in virtually any order and repeated as often as necessary, as long as they satisfy a number of constraints. For example, some tasks may not appear after others, some tasks cannot be the first or last ones of the participatory process, and so on.

This paper develops a standard to facilitate information exchange between different devices with the aim of supporting group decision making over the Web. E-participation processes can take different paths to solve a problem. Therefore, our services must provide any information of the problem solution and the phases to follow. In addition, some of the e-participation tasks require high computation resources and, frequently, running the application on a client machine, quickly, becomes prohibitive.

In Section 2, we present different participatory processes and identify the common tasks that we have found in them. We also outline the proposed algorithm and architecture to support group decision making over the Web. In Section 3, we provide some key features of XML, its development, use and advantages. We also show different scheme languages and motivate our choice for a particular XML scheme. In Section 4, we describe the choosen solution to develop a decision analysis service oriented architecture and show the XML schemas created to support e-participation. Section 5 concludes with a brief discussion.

2. An architecture for distributed decision making

In this section, we present different processes that allow citizens to participate in decision making and its transformation into an electronic participation framework, based on the boom of new technologies like social networks and the Web 2.0. Then, we present an architecture to make participation processes accessible from anywhere and anytime with an Internet connection.

2.1. From participation to e-participation

The creation of new participatory processes can be understood as an attempt to gain the trust of citizens, to mitigate the feeling of disappointment that they currently have with politicians and to try to reduce abstention rates. Different concepts have been proposed to describe this development: manage proximity, new public administration, modernization of local management or participatory democracy. There are many physical participation methods used in public participation around the world. In fact, it is estimated that there are more than one hundred participatory instruments, see Rowe and Frewer (2005). Some important examples include:

- *Referenda*, see Budge (1996). It is the instrument par excellence of semi-direct democracy. It is aimed at the whole population by asking them a question which may be consultative or binding.
- *Consensus Conferences*, see Tekno.dk (2006). They are based on meetings of a group of citizens and experts on matters relevant to the proposed issue. The discussion is facilitated by an independent mediator.
- *Citizen Juries*, see Jefferson-center.org (2005), are groups of citizens, chosen at random from the population, who meet during several days. Citizen' Juries are only consultative.
- *Citizen' Panels*, see Brown (2006), constitute an instrument for consulting the community, in which a group citizens are asked to evaluate a problem through public discussion or online forums.
- *District Forums*, see Sousa Santos (2004), are meetings open to the public. The forums are consultative with intention to create a permanent dialogue between citizens and local

authorities.

- *Participatory Budgets* (PB), see Sintomer et al. (2008), constitute an attempt to allow citizens to have a word on the decision of how part of a public budget is spent. There are many variants of PBs, see Alfaro et al. (2010).

Through the analysis of most of the participatory instruments, we have identified the tasks which might be included in various participatory processes, and are scheduled within them in various ways. These are:

1. *Participant sampling*: In many of the mechanisms, participation of all citizens is impossible for logistic or physical reasons. Thus, a sample of citizens is sometimes chosen to represent the wider population.
2. *Election of representatives*: Additionally, sometimes the representatives can be elected in meetings or votings by other participants.
3. *Use of questionnaires*: They aid in focusing on the main issues of interest, revealing what is of most interest to citizens.
4. *Document preparation*: Before the beginning of any participatory process, it is necessary to generate documents to inform participants about the process.
5. *Distribution of information*: One of the most important elements in decision-making is having the best possible information available about an issue, whether it is theoretical or practical. This information must be available among participants to facilitate decision-making.
6. *Sharing of information*: Similarly, participants should be able to share information they might be able to gather.
7. *Problem Structuring*: In this phase, technicians determine criteria to evaluate the proposals, technical features and constraints among them. Also, the whole process is structured indicating the start and end dates of the remaining phases.
8. *Alternative generation*: Participants usually spend some time proposing alternative solutions to the problem at hand, possibly aided by brainstorming techniques.

9. *Preference Modeling*: Participants are sometimes required to express their preferences over consequences or alternatives, possibly through pairwise comparisons, goal setting or value functions.
10. *Individual problem exploration*: Participants are allowed some time to explore the problem individually to find their most preferred alternative.
11. *Debate*: Whether regulated or spontaneous, the interchange of ideas is vital for citizen participation because. If it does not occur, decisions tend to be unimaginative and poor.
12. *Negotiation*: When individuals disagree on their preferred alternative, they may try to deal with the conflict through negotiations in which participants exchange offers, ideas and arguments to reach a consensus. Furthermore, several negotiation methods could be used, see Benyoucef and Verrons (2008) for further information.
13. *Arbitration*: Some mechanisms include the figure of an arbitrator who makes the final decision, once the opinions and reasoning of the different parties have been presented.
14. *Voting*: Very frequently, voting is used as a last resort, particularly if achieving consensus is not possible. The decision to be applied comes from voting.
15. *Explanation (to citizens)*: The resulting documents should be available to the citizenship printed or digital.

Just as we have identified tasks common to all participatory instruments, we found that in all of them there are three different user profiles:

- The *problem owner*, is the entity which aims at solving a participatory problem, structures and publishes it.
- The *citizen*, provides input to the participatory decision process, that is, opinions, suggestions and preferences about the issue to be addressed.
- The *technical staff*, takes technical care of the process development from a technical point of view: supporting the problem owner and providing assistance to citizens in the rest of the phases.

2.2. An architecture for distributed decision making

We describe a configurable architecture whose goal is to provide a technological platform with innovative tools and techniques facilitating decision analysis services and encouraging the use of ICT, possibly increasing citizen satisfaction as well as transparency in public decision making. The architecture is designed to combine modules to support the above tasks, facilitating the build up of any of the participatory instruments described above, as well as creating new mechanisms depending on the need of the process.

The architecture follows a SOA (Service Oriented Architecture) architecture, see Davis (2009), to support our business requirements. The system has been divided into modules as it provides greater scalability, and can include new modules, thereby increasing the functionality without requiring any major modification. The architecture is designed to solve discrete multicriteria group decision problems making under certainty. In case of modeling individual preferences, this will be done through value functions. The general outline of the architecture is as in Figure 1:

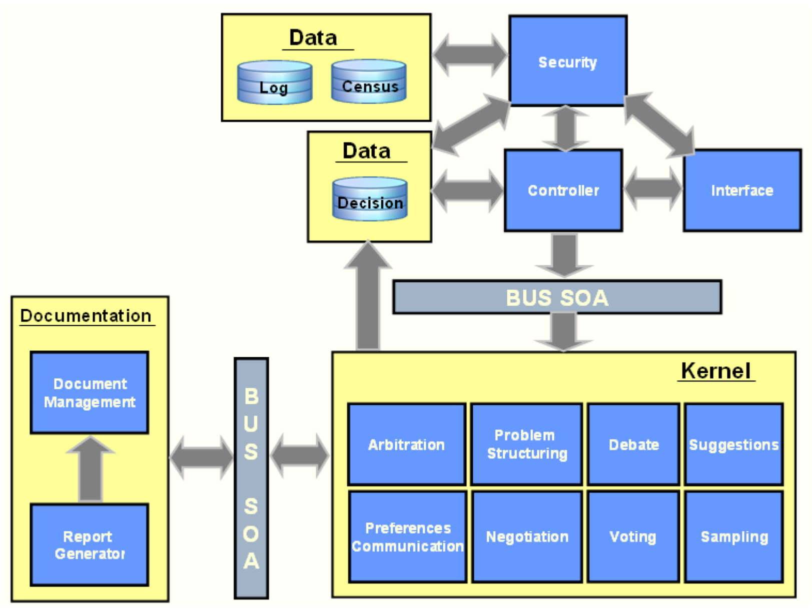


Figure 1 *General architecture with different modules.*

The architecture includes three main databases: one to store the population census, another one to store the information and data generated during a process, and another one to register the actions of participants. This also facilitates supervising whether the system had an unauthorized access.

We include five structures which, acting together, can properly support most group decision making processes:

1. *Controller.* It is a core module of the application as it coordinates the other subsystems. It also registers all services of the application. Therefore, when someone wants to register a new service, he must submit all relevant information, such as input and output parameters or location of the service, to the controller.
2. *Interface.* Responsible for facilitating interaction with users. In order to mitigate the digital divide, it should be graphical and easy to use.
3. *Security.* Responsible of preventing unauthorized accesses to the application and assign roles to participants.
4. *Documentation.* It is divided into two subsystems: *Report generator*, to create informative documents, and, *Document management*, responsible of storing the generated documents.
5. *Kernel.* It supports the business logic. It has a set of modules to solve each phase of the participatory process. These modules are executed in the order established by the controller. The main structures within this component are Problem Structuring, Preference communication, Negotiation, Arbitration, Voting, Suggestions, Sampling and Debate.

The controller allows us to define a participatory process by combining various participatory tasks as implemented in the modules. For example, a *Consensus conference* could be composed by following tasks: *Document preparation*, *Distribution of information*, *Debate* and *Explanation (to citizens)*. The definition of processes may be done with a business process modeling environment, as in Alfaro (2012).

As each of the decision analysis services could be implemented by different servers in different locations, we need an efficient means to facilitate information exchange among the corresponding modules. This is achieved through XML.

3. A primer on XML

XML (eXtensive Markup Language) evolves from the GML language (General Markup Language), with the aim of organizing technical documents with structural tags, as invented by IBM in the 70's. It appeared due to the need to manage lots of different information. In 1986, it became the Standard Generalized Markup Language and was adopted by ISO. Note, though, that SGML is not in itself a markup language, but rather a specification for defining markup languages. An application of SGML that became well known is HTML (Hypertext Markup Language). Despite being a good markup language, interpreters found serious problems when analyzing documents written in SGML, as it combines elements of different languages. To solve these problems, XML arose to: mix elements of different languages, so as to become extensible; create simple analyzers without special logic; and, reject documents with syntax errors.

XML has become an international standard developed by the World Wide Web Consortium (W3C) which allows the creation of a set of marks for information processing. One of its main features is its versatility, being able to process heterogeneous types of information. As a metalanguage, XML defines a syntax, which facilitates that the information is processed in a structured way, and managed more efficiently.

In summary, XML provides many advantages, with many initiatives that promote its use in a wide variety of fields. By facilitating data exchanges using a non-proprietary data format, XML is especially valuable in promoting interoperability in heterogeneous environments, such as the Internet. The growth of XML applications as a language for representing and exchanging data is reflected in several domain problems. Some examples include MathML (Mathematical Markup Language), for describing mathematical notation, maintaining its structure and content; or CML (Chemical Markup Language), which allows to describe molecules and their physico-chemical reactions.

In the context of optimization models, different languages have been proposed such as OSiL (Optimization Service Instance Language), to represent optimization problems, including linear, integer, quadratic and general, nonlinear programming problems, see Fourer et al. (2010), or LPFML, to represent linear programming models facilitating interoperability between modeling languages and solvers, reducing the number of required drivers, see Fourer et al. (2005). A very important project which also uses XML to exchange information is Decision Deck. Its goal is to develop software tools to support multiple criteria decision making. To do this, an XML standard called XMCDA has been introduced, see Bisdorff et al. (2009), which is defined by an XML schema.

There are several approaches for creating syntactic rules that define the tags of an XML document in e-participation. The more popular methods are DTD (Document Type Definition), see w3schools.com/dtd (2011), XML schema, see w3.org/XML/Schema (2011), RELAX NG, see Relaxng.org (2011) and Schematron, see www.schematron.com (2011). DTD and XML schema are the most popular, we decided to use the last one because it is a complex and powerful schema language that uses XML syntax. Therefore, there is includes different data types and it permits the creation of new types defined by the user, commonly called “archetypes”. Its most important feature is its ability to extend “archetypes”, called inheritance in object-oriented programming.

4. GroupDecXML

In this section, we briefly describe GroupDecXML, the chosen solution to adapt the proposed architecture to a decision analysis service oriented architecture, which enables greater scalability together with the XML schemas for each of the modules of the proposed architecture.

4.1. Introduction

Intuitively, each module within the architecture should have its own XML schema indicating the structure in which the interrelating information is stored. In a first version we developed a centralized architecture, see Figure 2. However, one of the problems identified was that all requests to each module were going through a central server, which in our case, could be the controller. For example, if a user logs into the voting module, he should send a request to the controller and this would redirect this request to the server in which the voting module is hosted.

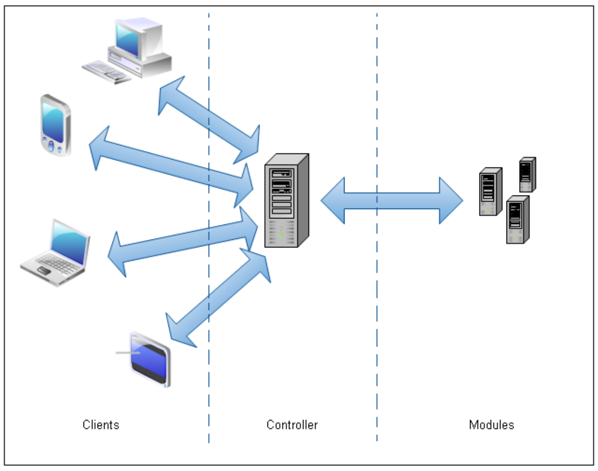


Figure 2 *Centralized architecture through the Controller.*

This solution can be improved, among other things, because if the controller is inaccessible, for reasons such as a system crash or saturation, the entire system would stop working. Thus, the system is not scalable and prevents that its fluid growth. The proposed solution is to create a common interface that is responsible of recording the services provided by each module, see Figure 3: when a customer wants to use a service, he sends the request to the common interface, redirecting the client to the server where the service is. Thus, the system load falls on the service provider server, not on the controller, and the common interface has a lighter load.

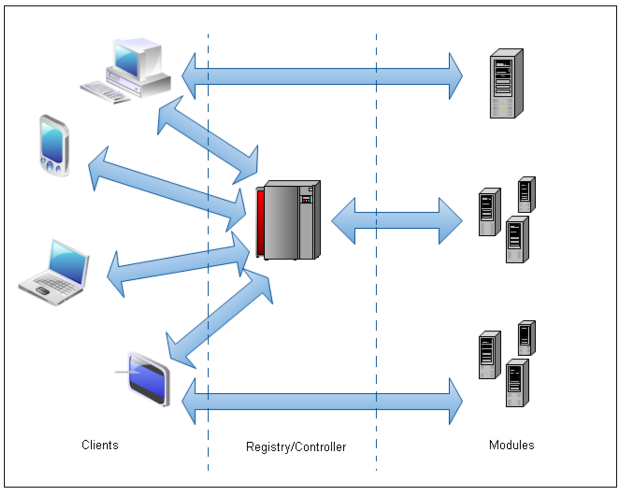


Figure 3 *SOA architecture.*

4.2. General description

The designed architecture, as mentioned above, is based the SOA paradigm. Thus, each service is accessible through the Internet from anywhere in the world. Conceptually, our proposed solution is supported by the architecture described in section 2.2, and each service could be located in a different location. For example, the voting service could be located in Barcelona, whereas the preference communication service server could be in Paris, the location being irrelevant to the final user.

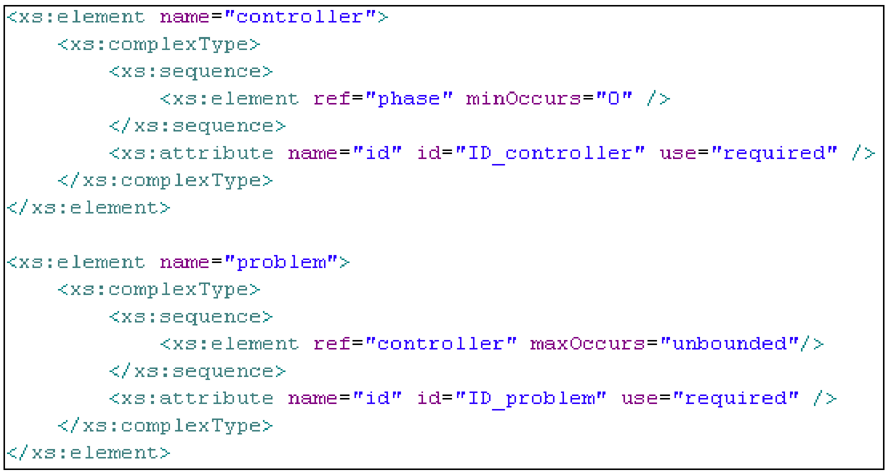
Should it be necessary to create a new module for solving some of the common tasks identified within a participatory process, we would register its services through the common interface, Register server, designed for this purpose. To do this, we must specify the required input and output parameters. Thus, when the *Controller* module, which is responsible of guiding the participatory process, detects that it is necessary to move to the next stage it knows which parameters must be sent. The *Registry server*, in addition to register the services provided, acts as middleware between the users and the system modules, facilitating a more fluid application.

4.3. Key Services

This section shows the basic structure of some of the XML schemes designed for various decision services analysis.

4.3.1. Controller It manages the participatory process. Therefore, it has the information required to determine which module should be invoked at a certain date, as well as its end date.

Its XML scheme, described in Figure 4, has a <problem> element, that allows to identify the decision analysis problem at each phase. This element has only as attribute an identifier and a reference to the <controller> element that stores the unique identifier that allows to distinguish it from the rest of controllers. This item also contains a reference to <phase> in which we find the phase of the participatory process, the start date (<start>) and the end date (<finish>) of each phase.

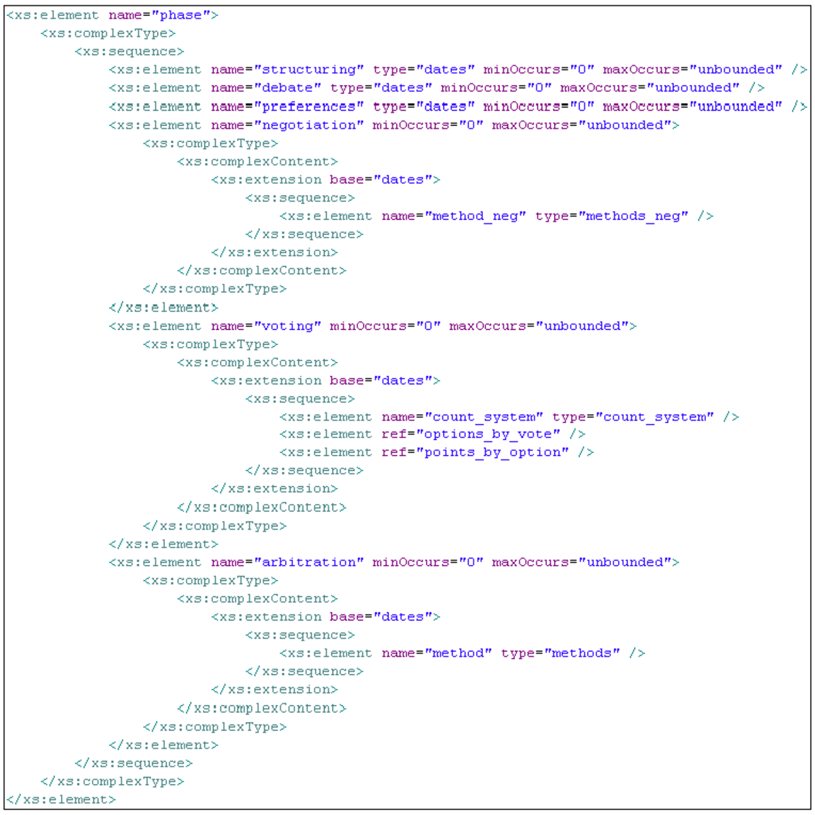


```
<xs:element name="controller">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="phase" minOccurs="0" />
        </xs:sequence>
        <xs:attribute name="id" id="ID_controller" use="required" />
    </xs:complexType>
</xs:element>

<xs:element name="problem">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="controller" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="id" id="ID_problem" use="required" />
    </xs:complexType>
</xs:element>
```

Figure 4 *Structure of the elements “problem” and “controller”.*

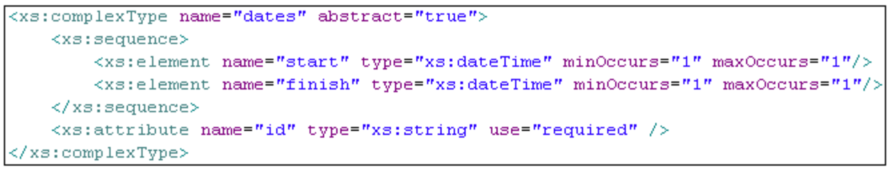
The <phase> element can take different values depending on the corresponding phase. These values are shown in the XML schema in Figure 5, including “structuring”, “debate”, “preferences”, “voting” and “arbitration” in consonance with the modules shown in Figure 1.



```
<xs:element name="phase">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="structuring" type="dates" minOccurs="0" maxOccurs="unbounded" />
            <xs:element name="debate" type="dates" minOccurs="0" maxOccurs="unbounded" />
            <xs:element name="preferences" type="dates" minOccurs="0" maxOccurs="unbounded" />
            <xs:element name="negotiation" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:complexContent>
                        <xs:extension base="dates">
                            <xs:sequence>
                                <xs:element name="method_neg" type="methods_neg" />
                            </xs:sequence>
                        </xs:extension>
                    </xs:complexContent>
                </xs:complexType>
            </xs:element>
            <xs:element name="voting" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:complexContent>
                        <xs:extension base="dates">
                            <xs:sequence>
                                <xs:element name="count_system" type="count_system" />
                                <xs:element ref="options_by_vote" />
                                <xs:element ref="points_by_option" />
                            </xs:sequence>
                        </xs:extension>
                    </xs:complexContent>
                </xs:complexType>
            </xs:element>
            <xs:element name="arbitration" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:complexContent>
                        <xs:extension base="dates">
                            <xs:sequence>
                                <xs:element name="method" type="methods" />
                            </xs:sequence>
                        </xs:extension>
                    </xs:complexContent>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figure 5 *Structure of the element “phase”.*

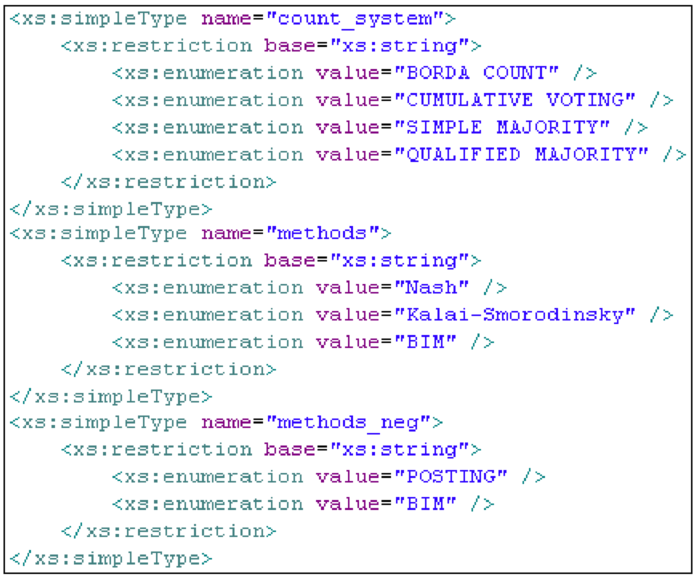
Each component in the <phase> element has child elements to store the corresponding start and end dates, as well as a unique identifier for each phase. These features are in the type "dates", see Figure 6.



```
<xs:complexType name="dates" abstract="true">
    <xs:sequence>
        <xs:element name="start" type="xs:dateTime" minOccurs="1" maxOccurs="1"/>
        <xs:element name="finish" type="xs:dateTime" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="required" />
</xs:complexType>
```

Figure 6 *Structure of the complex type "dates".*

In addition, the voting, negotiation and arbitration phases have an element to indicate the count system for voting (we have included so far Borda count, cumulative voting, simple majority and qualified majority, see Nurmi (2010)) and the resolution methods for negotiation (so far, POSTING and BIM, see Alfaro et al. (2010)), and arbitration (so far, Nash, Kalai-Smorodinsky or BIM, see Aliprantis and Chakrabarti (2010)), see Figure 7.



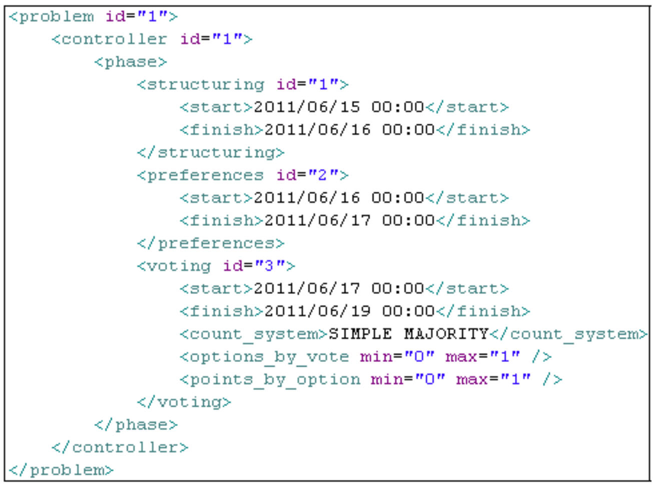
```
<xs:simpleType name="count_system">
    <xs:restriction base="xs:string">
        <xs:enumeration value="BORDA COUNT" />
        <xs:enumeration value="CUMULATIVE VOTING" />
        <xs:enumeration value="SIMPLE MAJORITY" />
        <xs:enumeration value="QUALIFIED MAJORITY" />
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="methods">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Nash" />
        <xs:enumeration value="Kalai-Smorodinsky" />
        <xs:enumeration value="BIM" />
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="methods_neg">
    <xs:restriction base="xs:string">
        <xs:enumeration value="POSTING" />
        <xs:enumeration value="BIM" />
    </xs:restriction>
</xs:simpleType>
```

Figure 7 *Available voting, arbitration and negotiation methods.*

Depending on the designed process, the corresponding XML document will have a different structure. An example could be:

- In the first stage, the participatory process includes a problem structuring phase lasting for one day.
- Next, a preference communication stage will be held, lasting also for one day.
- Finally, a voting stage based on simple majority, with one vote per option and only one option being selected, which would take two days.

as we can, see in Figure8.



```
<problem id="1">
    <controller id="1">
        <phase>
            <structuring id="1">
                <start>2011/06/15 00:00</start>
                <finish>2011/06/16 00:00</finish>
            </structuring>
            <preferences id="2">
                <start>2011/06/16 00:00</start>
                <finish>2011/06/17 00:00</finish>
            </preferences>
            <voting id="3">
                <start>2011/06/17 00:00</start>
                <finish>2011/06/19 00:00</finish>
                <count_system>SIMPLE MAJORITY</count_system>
                <options_by_vote min="0" max="1" />
                <points_by_option min="0" max="1" />
            </voting>
        </phase>
    </controller>
</problem>
```

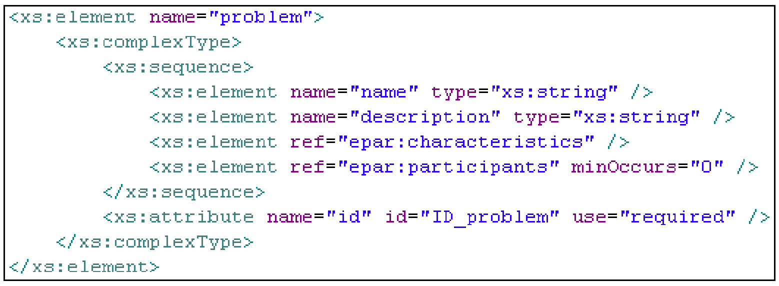
Figure 8 *Example XML schema for the "Controller".*

4.3.2. Problem structuring The problem structuring module facilitates the structuring of the problem, identifying proposals, constraints and criteria for a problem. Its XML scheme may be divided in three main sections:

- Basic problem data (name, description,...)
- Problem features (alternatives, criteria, constraints,...)
- Participants' data.

These three structures store the information of the group decision making problem. The XML schema definition of the problem structuring module is represented in Figure 9. The <problem>

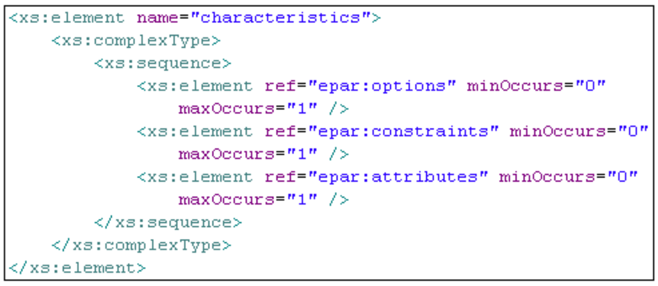
element is composed of four elements and a unique attribute. The <id> attribute represents a unique identifier for each participation problem. The <name> and <description> elements hold the basic data of the problem. The <characteristics> element stores the relevant information of the problem, such as the attributes, options and constraints. Finally, the <participants> element has the identifiers or "nicks" of the people allowed to take part in the process.



```
<xs:element name="problem">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string" />
            <xs:element name="description" type="xs:string" />
            <xs:element ref="epar:characteristics" />
            <xs:element ref="epar:participants" minOccurs="0" />
        </xs:sequence>
        <xs:attribute name="id" id="ID_problem" use="required" />
    </xs:complexType>
</xs:element>
```

Figure 9 *Structure of the element "problem".*

Some of these elements have descendants. For example, <characteristics> is formed by <options>, <attributes> and <constraints>, which store the options, attributes and constraints of the problem, respectively, providing its key features within a single element, see Figure 10.



```
<xs:element name="characteristics">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="epar:options" minOccurs="0"
                maxOccurs="1" />
            <xs:element ref="epar:constraints" minOccurs="0"
                maxOccurs="1" />
            <xs:element ref="epar:attributes" minOccurs="0"
                maxOccurs="1" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figure 10 *Structure of the element "characteristics".*

The <option> element has the descendant <attribute-option>, see Figure 11, that represents the relation between an attribute and an option. This element is repeated as many times as attributes has the problem and consists of an identifier, an attribute and a value.

```
<xs:element name="option">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string" />
            <xs:element name="description" type="xs:string" />
            <xs:element ref="attribute-option" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="id" id="ID_option" type="xs:string" use="required" />
    </xs:complexType>
</xs:element>
<xs:element name="attribute-option">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="attribute" type="string_attribute" />
            <xs:element name="value" type="xs:float" />
        </xs:sequence>
        <xs:attribute name="id" id="ID_attribute_option" type="xs:string" use="required" />
    </xs:complexType>
</xs:element>
```

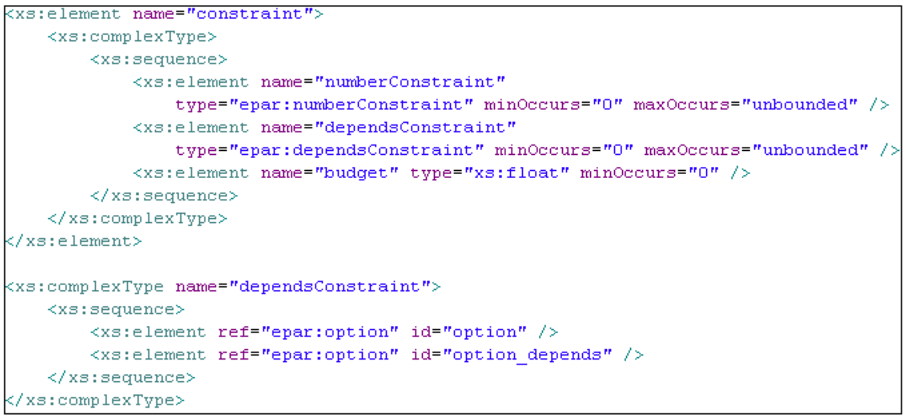
Figure 11 *Structure of the elements "option" and "attribute-option".*

Another element that should be noted are the constraints. In the current version, we have modeled tow types of constraints:number or dependence. Number constraints are those in which you can only take a number of options from all the available. For example, when a company needs to renew their computers they often buy all to just one provider. Therefore, if there are four providers, we have a number constraint as we must choose one manufacturer out of four. In our XML schema, this constraint is represented by the <numberConstraint> element, see Figure 12, which is composed of two elements: <numMax>, which represents the maximum number of options that can be implemented, and as many elements <option> as options can be chosen.

```
<xs:complexType name="numberConstraint">
  <xs:sequence>
    <xs:element name="numMax" type="xs:nonNegativeInteger" />
    <xs:element ref="epar:option" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>
```

Figure 12 *Structure of the element "numberConstraint".*

Dependence constraints represent the dependence of an option on others. For example, the construction of a parking in a hospital depends on the previous hospital building. Such restrictions are represented through the <dependsConstraint> element, see Figure 13, which has two elements, <option>, where the element with the identifier "option" depends of the element with the identifier “option_depends”. Thus, in the above example, the element with the identifier “option” would be the hospital parking, whose construction depends on the element with the identifier “option_depends” which, in this example, would be the hospital itself.



```
<xs:element name="constraint">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="numberConstraint"
                type="epar:numberConstraint" minOccurs="0" maxOccurs="unbounded" />
            <xs:element name="dependsConstraint"
                type="epar:dependsConstraint" minOccurs="0" maxOccurs="unbounded" />
            <xs:element name="budget" type="xs:float" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:complexType name="dependsConstraint">
    <xs:sequence>
        <xs:element ref="epar:option" id="option" />
        <xs:element ref="epar:option" id="option_depends" />
    </xs:sequence>
</xs:complexType>
```

Figure 13 *Structure of the element“constraint” and the type “dependsConstraint”.*

4.3.3. Preference communication The preference communication module is one of the most important in the architecture. It supports an additive value function model, see e.g. Winterfeldt and Edwards (1986), without much loss of generality on its applicability. The component value functions are piecewise linear with the information stored at three intermediate points between the worst and best attribute values, as exemplified in Figure 14.

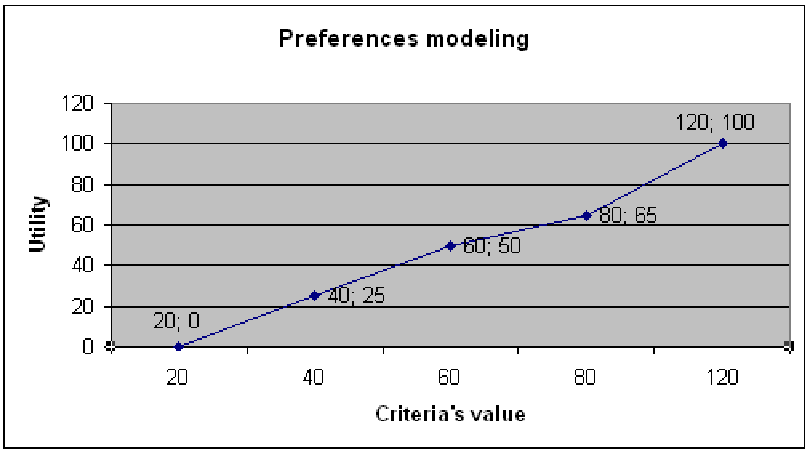


Figure 14 *Example of piecewise linear value function obtained once the user has indicated its preference.*

Once the system receives a user's preferences, it may compute the value of each alternative for such participant. Specifically, for each participant the architecture gets the following information:

P(w_i, $v_i(x_j,y_j)$) where: $i \in (1..n)$ refers to the number of criteria;
$j \in (1..5)$ refers to the number of points in each criteria;
w_i represents the weight of attribute i; $w_i \geq 0$; $\sum w_i = 1$
v_i represents the array with the five pairs of points for attribute i;
x_j represents the value of the attribute;
$y_j \in [0,1]$ represents the assigned value to x_j.

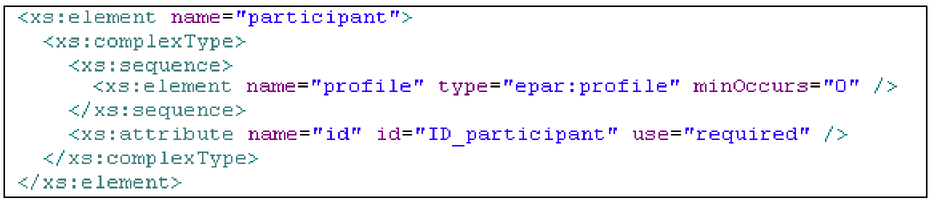
Then, for a given alternative o_k, with criteria values z_{ik}, the value of such alternative would be:

$$v(o_k) = \sum_{i=1}^{n} w_i \cdot d_i(v_i, z_{ik})$$

where to compute $d(v_i, z_{ik})$ we seek which is the position of the value z_{ik} in the array v_i and calculate its value through:

```
WHILE $z_{ik} \geq x_i$
    $i=i+1$;
$d_i(v_i, z_{ki}) = (x_i - z_{ik}) \cdot (y_{i+1} - y_i)$
```

For this, the schema of this module stores a profile with the preferences of each participant. The <participant> element has the structure, see in Figure 15.

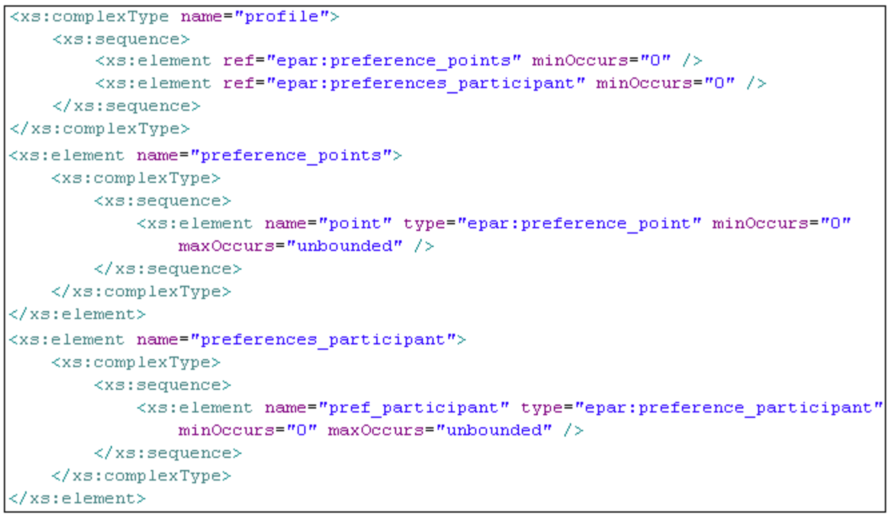


```
<xs:element name="participant">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="profile" type="epar:profile" minOccurs="0" />
    </xs:sequence>
    <xs:attribute name="id" id="ID_participant" use="required" />
  </xs:complexType>
</xs:element>
```

Figure 15 *Structure of the element "participant".*

The <profile> complex type, see Figure 16, has two elements that facilitate the storage of preferences of each participant for each attribute of the problem:

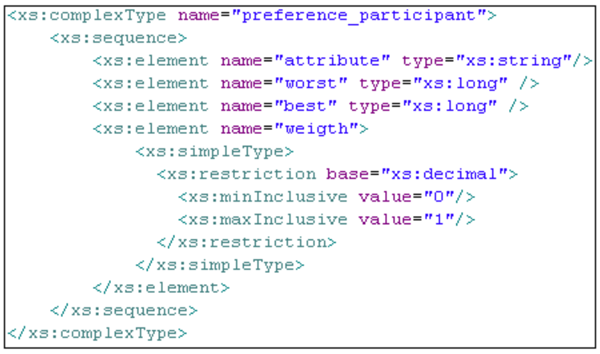
- The <preferences_participant> element stores the weight that the participant gives to each attribute, and the best and worst value of the attribute.
- The <preference_points> element stores the values assigned by the participant to five attribute points.



```
<xs:complexType name="profile">
    <xs:sequence>
        <xs:element ref="epar:preference_points" minOccurs="0" />
        <xs:element ref="epar:preferences_participant" minOccurs="0" />
    </xs:sequence>
</xs:complexType>
<xs:element name="preference_points">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="point" type="epar:preference_point" minOccurs="0"
                maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="preferences_participant">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="pref_participant" type="epar:preference_participant"
                minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figure 16 *Structure of the type "profile".*

The participant communicates his best and worst values based on the maximum and minimum ones for each attribute of the problem. Also, the participant indicates the weight assigned to each attribute, which is a value between 0 and 1, whose sum must be 1. This information is stored in the <preference_participant> type which has four elements: a reference to the <attribute> element; a <best> element to store the favorite value of the participant for this attribute; a <worst> element to store the worst value of the attribute; finally, a <weight> element to indicate the weight assigned by the participant to this attribute. The type of this last element is <range> and indicates the minimum and maximum values that can be assigned to an element of this type, 0 for the worst value and 100 for the best. These elements and types are provided in Figure 17:



```
<xs:complexType name="preference_participant">
    <xs:sequence>
        <xs:element name="attribute" type="xs:string"/>
        <xs:element name="worst" type="xs:long" />
        <xs:element name="best" type="xs:long" />
        <xs:element name="weigth">
            <xs:simpleType>
              <xs:restriction base="xs:decimal">
                <xs:minInclusive value="0"/>
                <xs:maxInclusive value="1"/>
              </xs:restriction>
            </xs:simpleType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

Figure 17 *Structure of the types "preference_participant".*

All this information is stored in the <preference_point> element, see Figure 18. It has an "utility" attribute with a value between 0 and 100 to indicate the utility of a criteria value, the "value" attribute which is an intermediate value between the worst and best attribute values and the "iteration" attribute to identify each point. The type of the "iteration_value" attribute is restricted to integer values one, two or three identifying the criteria points. Besides, the <attribute> element stores the attribute evaluated.

```
<xs:complexType name="preference_point">
    <xs:sequence>
        <xs:element name="attribute" type="epar:string_attribute" />
    </xs:sequence>
    <xs:attribute name="iteration" type="epar:iteration_values" />
    <xs:attribute name="utility" type="epar:range" />
    <xs:attribute name="value" type="xs:long" />
</xs:complexType>

<xs:simpleType name="iteration_values">
    <xs:restriction base="xs:integer">
        <xs:minInclusive value="1" />
        <xs:maxInclusive value="3" />
    </xs:restriction>
</xs:simpleType>
```

Figure 18 *Structure of the types "preference_point" and "iteration_values".*

As an example, the XML document generated for the example in Figure 14 would be:

```
<preference_points>
    <point iteration="1" utility="25" value="40">
        <attribute id="1" />
    </point>
    <point iteration="2" utility="50" value="60">
        <attribute id="1" />
    </point>
    <point iteration="3" utility="65" value="80">
        <attribute id="1" />
    </point>
</preference_points>
```

Figure 19 *XML document from Figure 14.*

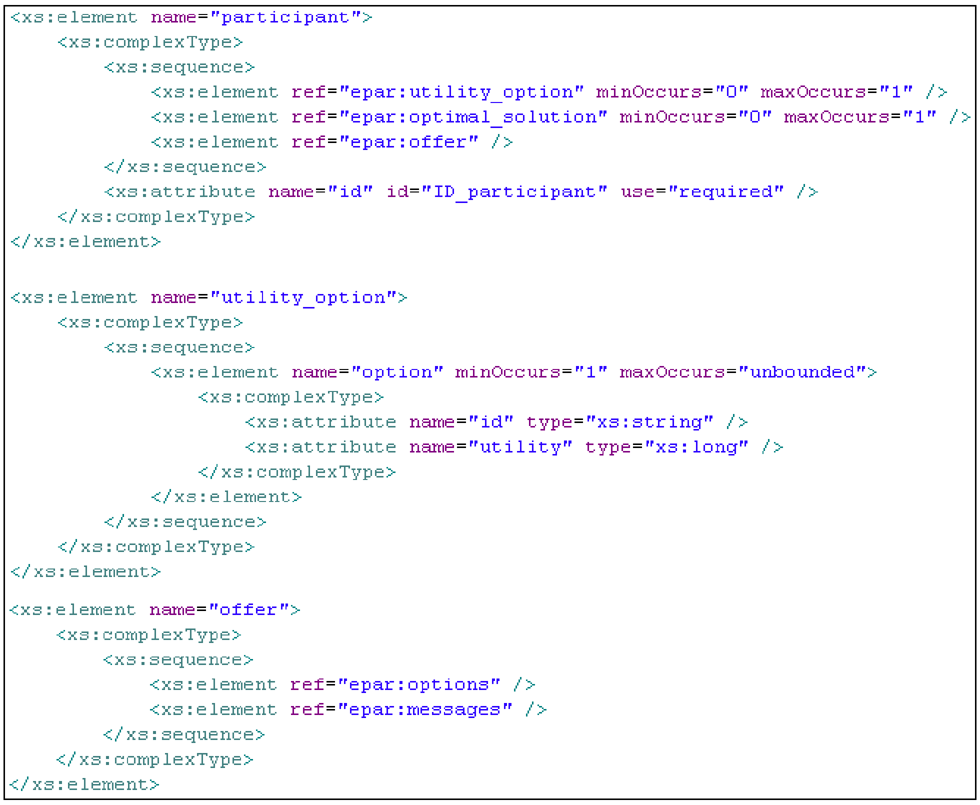
Finally, the value function of the participant for each of the options is computed. The method to obtain this utility function is independent of the method used to communicate the preferences. The result is stored in the element <utility_function> that includes an attribute "id" to identify this function and an element <option> which is the alternative that receives the value for that participant with an attribute "id" to identify the alternative and an attribute "utility" with the utility of the option for this participant.

```
<xs:element name="utility_function">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="option" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="id" type="xs:string" />
                    <xs:attribute name="utility" type="xs:long" />
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="id" id="utility_functionID" use="required" />
    </xs:complexType>
</xs:element>
```

Figure 20 *Structure of element "utility_function".*

4.3.4. Negotiation The negotiation module currently implements two different negotiation approaches, POSTING and BIM, see Ríos and Ríos Insua (2008) and Rios et al. (2010), for descriptions. Each one has its own protocol. In this paper, we shall only describe that of POSTING. In POSTING negotiations, the participants know their value functions, if they have communicated their preferences to the system. This is relevant because participants can see their most preferred options at any time, and more generally, quickly evaluate any option. In addition, the participants can post suggestions to obtain votes by other participants.

An XML schema of this module includes three main elements: "participant", "utility_option" and "offer", see Figure 21. The <utility_option> element stores the values that the participant sets for each option posted during the negotiation process. The <optimal_solution> element stores the optimal solution for the participant. It includes an <options> element with all the options of the optimal solution. Finally, the <offer> element has two elements: the <options> element, which contains the options of the offer, and the <messages> element, which may contain a text explaining the offer.

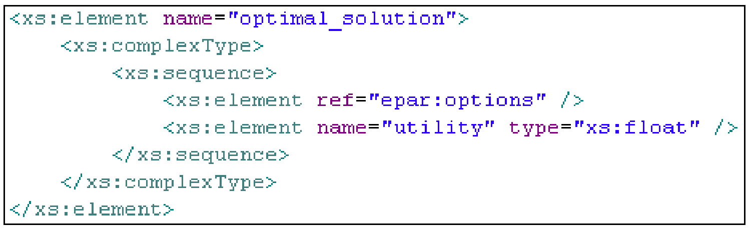


```
<xs:element name="participant">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="epar:utility_option" minOccurs="0" maxOccurs="1" />
            <xs:element ref="epar:optimal_solution" minOccurs="0" maxOccurs="1" />
            <xs:element ref="epar:offer" />
        </xs:sequence>
        <xs:attribute name="id" id="ID_participant" use="required" />
    </xs:complexType>
</xs:element>

<xs:element name="utility_option">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="option" minOccurs="1" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="id" type="xs:string" />
                    <xs:attribute name="utility" type="xs:long" />
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="offer">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="epar:options" />
            <xs:element ref="epar:messages" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figure 21 *Structure of the elements "participant", "utility_option" and "offer".*

Another relevant element of this module is <optimal_solution>, see Figure 22, which contains an <options> element with the options of the optimal solution and the <utility> element, with the value of all solutions.



```
<xs:element name="optimal_solution">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="epar:options" />
            <xs:element name="utility" type="xs:float" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figure 22 *Structure of the element "optimal_solution".*

Finally, the <offer_vote> element, see Figure 23, stores the votes received by the posted offers. It has the <offer> element with the set of options of the offer, the <participant> element describing the participant, who sent their votes and the <accept> element whose type is boolean, "true", if the vote is favorable, "false", otherwise.

```
<xs:element name="offer_vote">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="epar:offer" />
      <xs:element ref="epar:participant" />
      <xs:element name="accept" type="xs:boolean" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

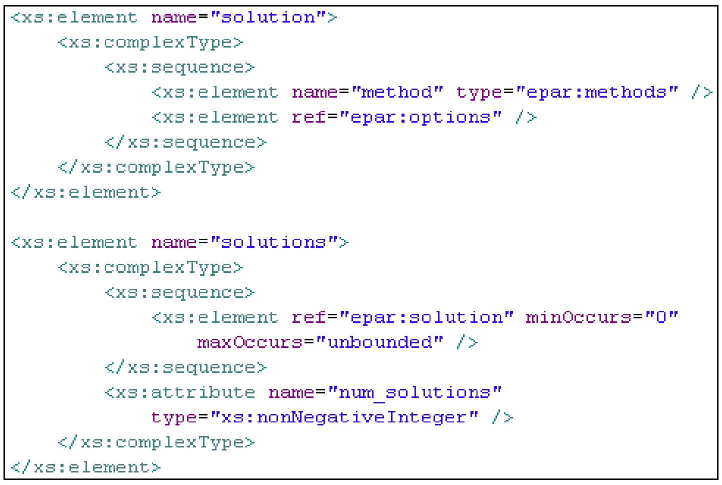
Figure 23 *Structure of the element "offer_vote".*

4.3.5. Arbitration If the arbitration module is included in the process, it must be executed only once the preference communication stage has finished. This is because the system needs to know the participant preferences to obtain a solution through arbitration. The XML schema of the arbitration module has a structure similar to that of communication of preferences, with a new element, <solutions>, as in Figure 24.

```
<xs:element name="problem">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" type="xs:string" />
            <xs:element name="description" type="xs:string" />
            <xs:element name="start" type="xs:date" />
            <xs:element name="finish" type="xs:date" />
            <xs:element ref="epar:characteristics" />
            <xs:element ref="epar:participants" minOccurs="0"
                maxOccurs="1" />
            <xs:element ref="epar:solutions" minOccurs="0" maxOccurs="1" />
        </xs:sequence>
        <xs:attribute name="id" id="ID_problem" use="required" />
    </xs:complexType>
</xs:element>
```

Figure 24 *Structure of the "problem" of arbitration module.*

This element contains the optimal solution, or solutions in case there are several optimal solutions. It has an attribute called "num_solutions" to indicate the number of optimal solutions and the <solution> element with the solution proposed by the arbitration algorithm. In turn, the <solution> element, see Figure 25, has two elements, <options>, which shows the options of the solution, and <method>, which specifies the method used by the system to obtain the solution.

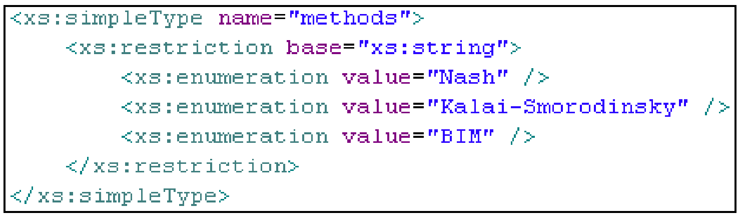


```
<xs:element name="solution">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="method" type="epar:methods" />
            <xs:element ref="epar:options" />
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="solutions">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="epar:solution" minOccurs="0"
                maxOccurs="unbounded" />
        </xs:sequence>
        <xs:attribute name="num_solutions"
            type="xs:nonNegativeInteger" />
    </xs:complexType>
</xs:element>
```

Figure 25 *Structure of the elements "solutions" and "solution".*

The values of the <method> element, see Figure 26, are specified in a simple type called "methods" which are strings indicating the arbitration method.



```
<xs:simpleType name="methods">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Nash" />
        <xs:enumeration value="Kalai-Smorodinsky" />
        <xs:enumeration value="BIM" />
    </xs:restriction>
</xs:simpleType>
```

Figure 26 *Structure of the "method".*

The algorithms currently implemented in the arbitration module are as shown in Figure 26, "Nash", "Kalai-Smorodinsky"and "BIS" (Balanced Increment Solution), see Raiffa et al. (2002).

5. Conclusions

There has been an interest in participatory processes to allow citizens to take part in public decisions that affect them. As a consequence, participatory instruments are flourishing all over the world, with purposes such as reporting to politicians for guiding them in decision making or allowing citizens to make themselves the decisions. In reviewing most of these participatory instruments, we have realized that there are common tasks in all such instruments. By combining innovatively such tasks, we could obtain new participatory instruments that could be, in some cases, more efficient that current ones.

In this paper, we have proposed an architecture and a set of XML schemes for designing, developing and deploying group decision making processes distributed over the Web. The architecture described is a web-services system to provide support for citizens in decision making processes, promoting virtual meetings in which participants express their views and preferences about the alternatives in the problem. The architecture illustrates how we might support groups in making decisions using ICT and decision technologies. We have also developed the necessary XML schemes (GroupDecXML) and Web services to exchange information between various phases of the decision making process. Each module of the architecture has its own XML schema to store the relevant information. Our goal is to ensure that users of these services will achieve better agreements faster and with less effort, hence moving a step closer towards achieving e-democracy.

It would be interesting to create a repository where we could store standard XML schemes for each of the participatory processes executed. Thus, when someone needs to launch a process he could search if a similar process has been previously executed. In that case, the user would only need to load it into the system. Thus, over time we would get a warehouse of participatory processes ready to be loaded.

Acknowledgments

Research supported by grants from the Spanish Ministry Science and Innovation, the Ministry of Industry, Tourism and Trade, the Government of Madrid, RIESGOS-CM, Agora.net, ALGODEC, Corporate Community, Democracy4All, EDUCALAB and HAUS.

References

Alfaro, C., Gómez, J., Ríos, J. 2010. From participatory budgets to e-participatory budgets, in Ríos Insua, French (eds), *e-Democracy: A Group Decision and Negotiation Approach.* Springer. 283-300.

Alfaro, C. 2012. A Business Process Modelling environment to support participatory processes definition, *Tech. Report.*

Aliprantis, Charalambos D., Chakrabarti, Subir K. 2010. Games and Decision Making, Oxford University Press Inc. Second Edition.

Benyoucef, M., Verrons, M-H. 2008. Configurable e-negotiation systems for large scale and transparent decision making *Group Decision and Negotiation*, Vol. 17, No. 3, Springer, pp. 211-224.

Bisdorff, R., Meyer, P., Veneziano, Th. 2009. XMCDA: a standard XML encoding of MCDA data. *Invited presentation at the 23rd European Conference on Operational Research Bonn (DE)*, July 5-8.

Brown, M. 2006. Citizen Panels and the concept of Representation, *Journal of Political Philosophy* 14, 203-225.

Budge, I. 1996. *The New Direct Democracy*, Policy Press.

GovTalk. 2011. http://cabinetoffice.gov.uk/

Davis, J. 2009. *Open Source SOA*, Manning Publications Co., Greenwich, CT.

DTD. 2011. http://www.w3schools.com/dtd/default.asp

Fourer, R., Lopes, L., Martin, K. 2005. LPFML: A W3C Schema for Linear and Integer Programming *INFORMS Journal on Computing* Vol. 17, No. 2, Springer, pp. 139-158.

Fourer, R., Ma, J., Martin, K. 2010. Optimization Services: A Framework for Distributed Optimization *Operation Research* Vol. 58, No. 6, November-December, pp. 1624-1636

Gregory, R., Fischhoff, B., McDaniels, T. 2005. Acceptable input: using decision analysis to guide public policy deliberations *Decision Analysis*, 2, 4-16.

Nurmi, H. 2010. Voting Theory, in Ríos Insua, French (eds), *e-Democracy: A Group Decision and Negotiation Approach.* Springer. 101-123.

ITALO. 2011. https://oficinavirtual.mityc.es/

Raiffa, H., Richardson, J., Metcalfe, D. 2002. *Negotiation Analysis: The Science and Art of Collaborative Decision Making.* Harvard University Press, Cambridge, MA.

Relax NG. 2011. http://relaxng.org

Ríos, J., and Ríos Insua, D. 2008. A methodology for participatory budget formation, *Jour. Oper. Res. Soc.*, 59, 203-212.

Ríos, J., and Ríos Insua, D. 2010. Balanced increment and concession methods for negotiation support, *RACSAM*, 104, 41-56.

Rowe, G., and Frewer, L. 2005. A Typology of Public Engagement Mechanisms, *Science, Technology and Human Values* 30/2: 251-290.

XML Schema. 2011. http://www.w3.org/XML/Schema

Schematron. 2011. http://www.schematron.com/

Sintomer, Y., Herzberg, C., Röcke, A. 2008. Participatory budgeting in Europe: potential and challenges, *Int. Jour. Ur. Reg. Research*, 32, 164-178.

Sousa Santos, B. 2004. *Democracia e participaçao. O caso do orçamento participativo de Porto Alegre*, Ediçoes Afrontamnto Lda. Portugal.

Steffek, J., Kissling, C., Nanz, P., (eds.), 2007. *Civil Society Participation in European and Global Governance: A Cure for the Democratic Deficit?*, Basingstoke: Palgrave.

Von Winterfeldt, D. and Edwards, W. 1986. *Decision Analysis and Behavioral Research.* Cambridge: Cambridge University Press. 604.

W3C. 2006. Consensus Conferences. http://www.tekno.dk.

W3C. 2005. Citizens' juries. http://www.jefferson-center.org