Improving Risk Management

RIESGOS-CM

Análisis, Gestión y Aplicaciones

P2009/ESP-1685

Technical Report 2011.5

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

Cesar Alfaro

http://www.analisisderiesgos.org

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

In recent years there has been a boom in the desire to participate in governmental decisions by citizens. Thus has motivated the emergence of hundreds of participatory mechanisms to facilitate the transmission of citizen concerns, and desired. Similarly, there has been an increased use of new technologies allowing citizens to contact and communicate their preferences to the political class. We have developed an architecture to support distributed decision making that allows to construct and implement new participatory processes. It is based on a standard that enables the exchange of information between the phases of the process, using XML and Web services.

Key words: group decision making, e-democracy, e-participation, configurable software, decision support, 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 reasons including: increase of legitimacy, acceptance and transparency in 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 and mitigate clientelism; educate citizens to make them understand that decisions entail both 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 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. It is important to stress that many of them have been initiated and tested at local level politics, a context in which it is specially appreciated that citizens may provide very useful local knowledge that would otherwise remain hidden.

As consequence of the rise in the use of these instruments for participation in decision making and the increasing use of Web, we find that, in fact, a few of those instruments are being developed over the Web. Thus, it it seems timely to take advantage of the growing role of Web tools for 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 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 (electronic procedures) between public administrations, sending information about electronic files. 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 as Software as a Service model, that is, software running on a computer accessible from anywhere through an Internet connection. The best known example of SaaS is *Salesforce.com*, although, there are many more like *Google Apps*. The main advantage of this model is that it minimizes the investment cost in powerful machines with high calculation capacity to run complex applications, because the application is executed in the service provider's machine. Another advantage is that the software is accessible anytime and anywhere, with the disadvantage of requiring an Internet connection.

The main goal of this paper is to develop a standard to facilitate the information exchange between different devices with the aim of supporting decision making by citizens. 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 shall talk about some key features of XML, its development, its use, its advantages, and, also, some of its applications that use in. We also show different scheme languages and we explain why did we choose an XML scheme for this project. In Section 3 we present different participatory processes and identify the common tasks that we have found in them; We also outline the proposed architecture to support group decision making over the Web. In Section 4, we present the choosen solution to develop a service oriented architecture and we show the XML schemas created to e-participation. Section 5 concludes with a brief discussion.

2. A primer on XML

XML language (eXtensive Markup Language) evolves from the GML language (General Markup Language), with the aim of marking up 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 Generalised 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.
- 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 the 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 Internet. The growth of XML applications as a language for representing and exchanging data is reflected in several domains 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. The goal of this project is to develop software tools to support decision making with multiple criteria. To do this, an XML standard called XMCDA has been introduced, see Bisdorff et al. (2009), which are 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), XML schemas, RELAX NG and Schematron. Although DTD and XML schemas 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 a lot of different data types and it permits the creation of new types defined by the user, commonly called "archetypes". Its most important feature is the ability to extend "archetypes", called inheritance in object-oriented programming. XML schemas provide working power, scalability, its use and a lot of available documentation.

3. An architecture for distributed decision making

In this section we shall talk, about different processes that allow citizen participate in decision making and its transformation into an electronic participation framework, due to the boom of new technologies like social networks and the Web 2.0. Then, we present an architecture to make participation processes described above accessible from anywhere with an Internet connection.

3.1. From participation to e-participation

The creation of new participatory processes can be understood as an attempt to gain the trust of citizen, to mitigate the feeling of disappointment that they currently have with politician and to try to lower 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 are:

- *Referenda*, see Budge (1996), is the instrument par excellence of semi-direct democracy. It is aimed at the whole population by asking them a question. The answer can be consultative or binding.
- *Consensus Conferences*, see Tekno.dk (2006), are based on meetings between a group of citizens, and experts on matters relevant to the proposed issue. The discussion is facilitated by an independent mediator.
- *Citizen´s Juries*, see Jefferson-center.org (2005), are groups of citizens, chosen at random from the population, who meet during several days. Citizens' Juries are only consultative.
- *Citizens' 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 (Alfaro et al. (2010)).

Through the analysis of the several experiences and the tasks included, around the world, we have identified the tasks which might be included in various participatory processes, and are scheduled 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 chosen to represent the wider population.
2. **Use of questionnaires:** They aid in focusing on the main issues of interest, revealing what is of most interest to citizens.
3. **Document preparation:** Before the beginning of any participatory process it is necessary to generate documents to inform participants about the process.
4. **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, practical. This information must be available among participants to facilitate decision-making.
5. **Sharing of information:** Similarly, participants should be able to share information they might be able to gather.
6. **Problem Structuring:** In this phase, technicians determine criteria for choosing between proposals, technical features and constraints among them.
7. **Alternative generation:** Participants usually spend some time proposing alternative solutions to the problem at hand, possibly aided by brainstorming techniques.
8. **Preference Modeling:** Participants are sometimes required to express their preferences over consequences or alternatives, possibly through pairwise comparisons, goal setting or value functions.

9. **Individual problem exploration:** Participants are sometimes allowed some time to explore the problem individually to find their most preferred alternative.

10. **Debate:** Whether regulated or spontaneous, the interchange of ideas is vital for citizen participation because, if it does not occur, decisions are tend to be unimaginative and poor.

11. **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 so as to try to reach a consensus. Furthermore, several negotiation methods could be used, see Benyoucef and Verrons (2008) for further infotmation.

12. **Arbitration:** Some mechanisms include the figure of a referee who makes the final decision, once the opinions and reasoning of the different parties have been presented.

13. **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.

14. **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 there are three different user profiles:

- The **problem owner**, which is the entity which aims at solving a participatory problem, structures and publishes it.
- The **citizen**, who provides input to the participatory decision process, that is, opinions, suggestions and preferences about the issue to be addressed.
- The **technical staff**, who 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.

3.2. 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 by increasing citizen satisfaction as well as transparency in public decision making. The architecture is designed to combine each of the modules, so as to build any of the participatory instruments described above, as well as to create new mechanisms depending on the need of the process. The schema follows a SOA (Service Oriented Architecture) architecture (Wikipedia.org (2011)), to support the business requirements. The system has been divided into modules because it provides greater scalability, and can add new modules, thereby increasing the functionality without requiring any significant modification. The architecture is designed to solve multicriteria decision problems under certainty, that in case of modeling individual preferences, it will be through value functions. The general outline of the architecture looks like the Figure 1:

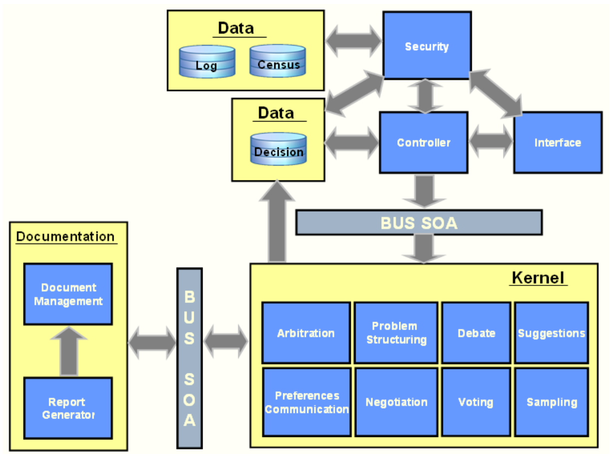


Figure 1 *General architecture with different modules.*

The architecture has 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 movements of participants. At the same time, it sees if the system had an unauthorized access.

We include five structures which acting together properly can give solutions to many decision making process, see Alfaro et al. (2010):

- **Controller**. It is a core module of the application and it coordinates the other subsystems.
- **Interface**. Responsible for interacting with users. In order to mitigate the digital divide, it should be graphical and easy to use.

- **Security**. Responsible for preventing unauthorized accesses to the application and assign roles to participants.
- **Documentation**. It is divided in two subsystems: *Report generator*, to create informative documents, and, *Document management* responsible of storing the generated documents.
- **Kernel**. It is one of the most important components of the architecture and hosts 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.

4. GroupDecXML

In this section, we describe briefly the solution chosen to adapt the proposed architecture to a service oriented architecture, which enables greater scalability. Then, we shall make several general description. Finally, we show the XML schemas for each of the modules of the proposed architecture.

4.1. Introduction

Each one of the modules in which the architecture has been divided, has its own XML schema indicating the structure in which the interrelating information is stored.

In a first version we developed a centralized architecture, but 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. See Figure 2.

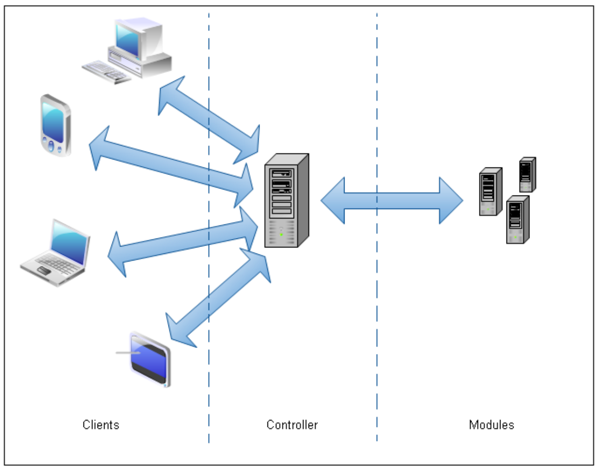


Figure 2 *Centralized architecture through the Controller.*

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

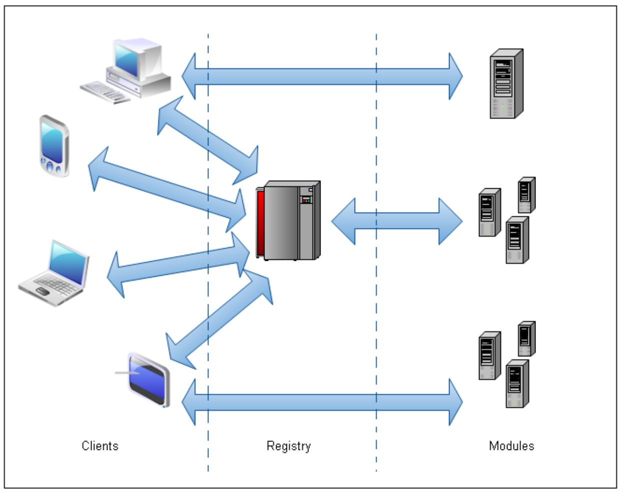


Figure 3 *SOA architecture.*

4.2. General description

The designed architecture, as mentioned above, follows the SOA paradigm. Thus, each of the services are accessible through the Internet from anywhere in the world. Conceptually, our proposed solution follows the architecture described in section 3.2, and each service could be located in a different location. For example, the voting service could be located in Barcelona, whereas the communication preferences server in Paris, the location being irrelevant to the final user.

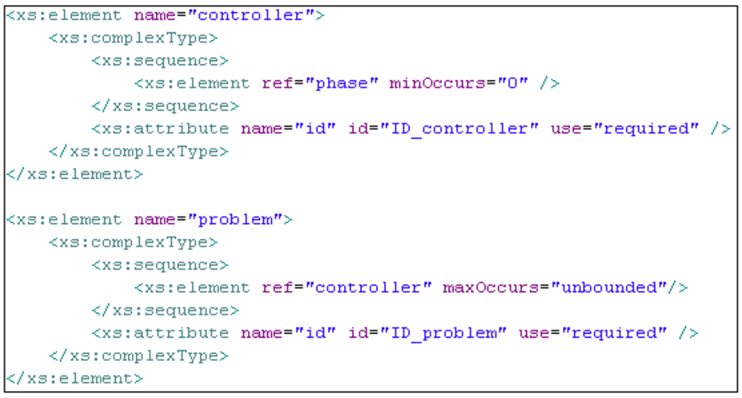
Should be necessary to create a new module for solving some of common tasks identified in a participatory process, we would register its services in the common interface, Register server, designed for this purpose. To do this, we must specify the required input and output parameters. Thus, when the *Controller* module, responsible of guiding the participatory process, detects that 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 users and 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 the services included in implementing any participatory process.

4.3.1. Controller It is the structure that manages the participatory process. Therefore, it has the information to determine which module should act in a certain date, as well as its end date.

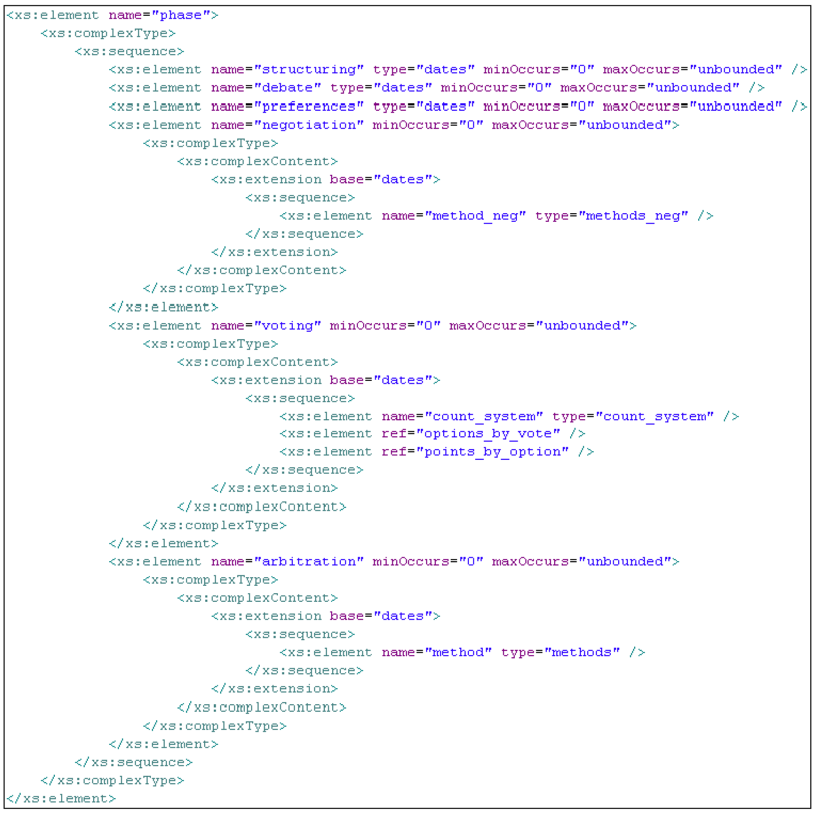
Its XML scheme has a <problem> element, in Figure 4, that allow to identify the problem at each phase. This element only stores as attribute an identifier and a reference to the <controller> element that stores the unique identifier that allows to distinguish it from the rest. This item also stores a reference to <phase> in which we find each of the phases of 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".*

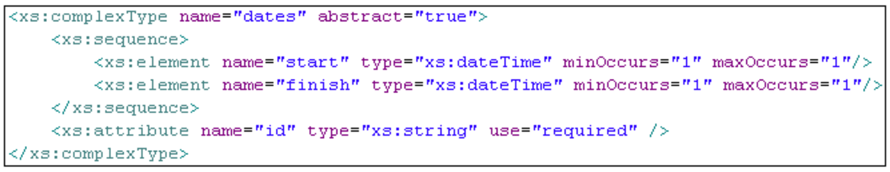
The <phase> element can take different values depending on the phase stored. These values are shown in the XML schema in Figure 5.



```
<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".*

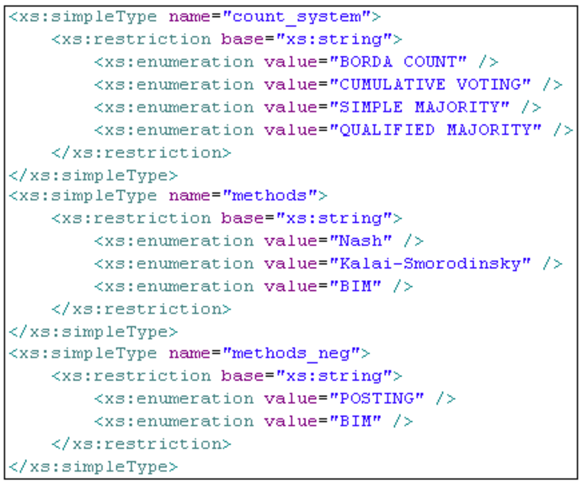
Each component in the <phase> element has child elements to store start and end dates as well as a unique identifier for each phase. These features are in the type "dates", see Figure 6.



```
<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 and the resolution methods for negotiation and arbitration, see Figure 7.



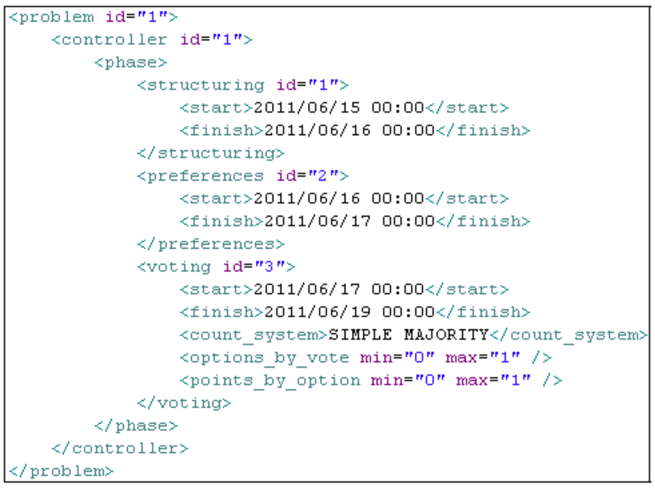
```
<xs:simpleType name="count_system">
    <xs:restriction base="xs:string">
        <xs:enumeration value="BORDA COUNT" />
        <xs:enumeration value="CUMULATIVE VOTING" />
        <xs:enumeration value="SIMPLE MAJORITY" />
        <xs:enumeration value="QUALIFIED MAJORITY" />
    </xs:restriction>
</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 XML document will have a different structure. An example could be, see Figure 8:

- In a first stage, the participatory process undertakes a problem structuring which last one day.
- Next, a preference communication stage, lasting also 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.



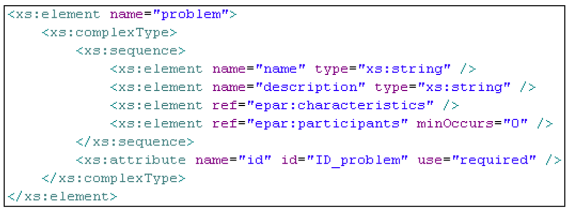
```
<problem id="1">
    <controller id="1">
        <phase>
            <structuring id="1">
                <start>2011/06/15 00:00</start>
                <finish>2011/06/16 00:00</finish>
            </structuring>
            <preferences id="2">
                <start>2011/06/16 00:00</start>
                <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 helps to structure the problem, identifies 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.

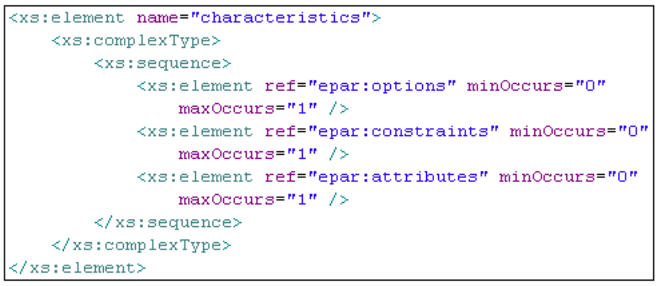
These three structures store the information of a citizen participation problem. The XML schema definition of the problem structuring module is represented in Figure 9. The <problem> element is composed of four elements and a unique attribute. The <id> attribute represents a unique identifier for each 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, <participants> element has the identifiers or "nicks" of the participants who can 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".*

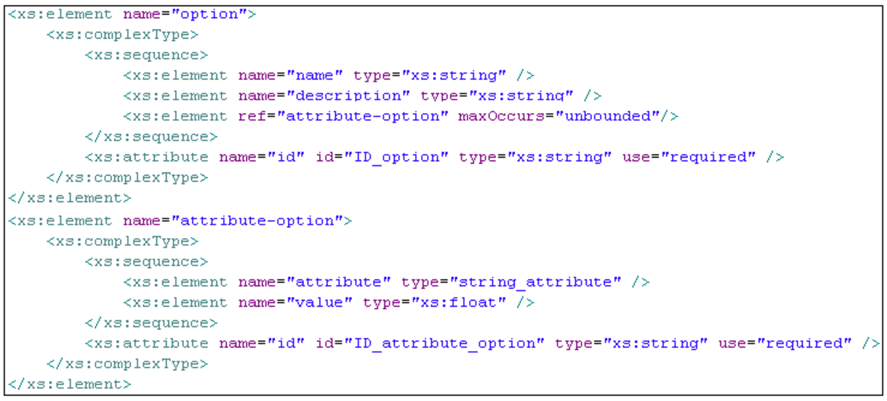
Some of the elements have descendants. For example, <characteristics> is formed by <options>, <attributes> and <constraints>, which store the options, attributes and constraints of the problem, respectively, providing, the key features of the problem within a single element, see Figure 10.



```
<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".*

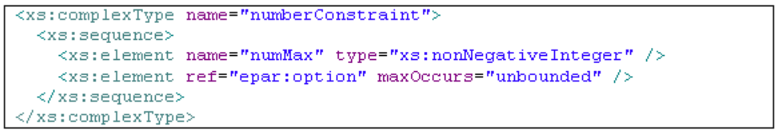
The <option> element has the descendant <attribute-option> element, see Figure 11, that represents the relation between an attribute and an option. This element is repeated as many times as attributes have the problem and consists of an identifier, an attribute and a value.



```
<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".*

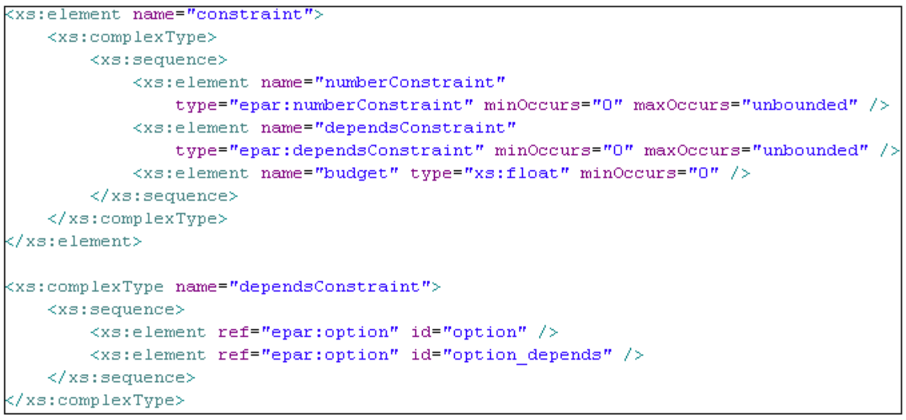
Another element that should be noted in the diagram are the constraints. So far we have modeled constraints on number or dependence. Number constraints are those in which you can only take a number of options from all the available. For example, when a company needs to renew their computers they often buy all to same manufacturer. Therefore, if there are four manufacturers, we have a number constraint because we must choose a manufacturer from four. In the scheme this constraint is represented by <numberConstraint> element, see Figure 12, which is composed of two elements: <numMax>, which represents the maximum number of options that can be implemented, and 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".*

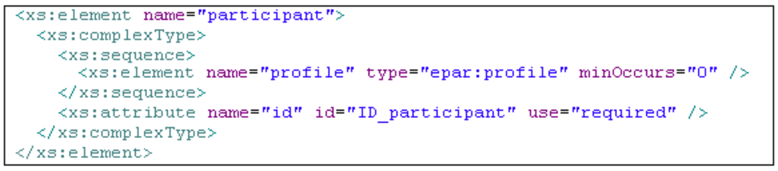
Dependence constraints represent the dependence of an option on others. For example, the construction of a parking in a hospital depends on the prior construction of the hospital. Such restrictions are represented thorugh the <dependsConstraint> element, see Figure 13 which has two elements: <option>, 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 of 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 is based on an additive value function, see Winterfeldt and Edwards (1986). This schema stores a profile with the preferences of each participant. The <participant> element has the structure, see Figure 14:

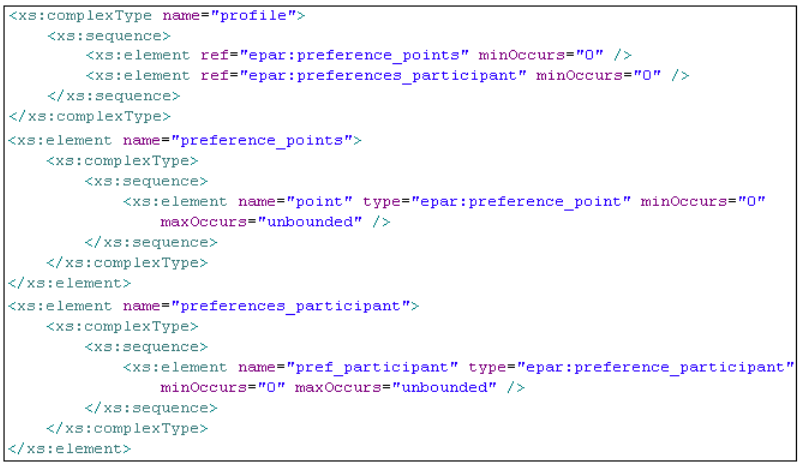


```
<xs:element name="participant">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="profile" type="epar:profile" minOccurs="0" />
    </xs:sequence>
    <xs:attribute name="id" id="ID_participant" use="required" />
  </xs:complexType>
</xs:element>
```

Figure 14 *Structure of the element "participant".*

The <profile> complex type, see Figure 15, has two elements that permit the store of the preference of the participant for each attribute of the problem. These elements are:

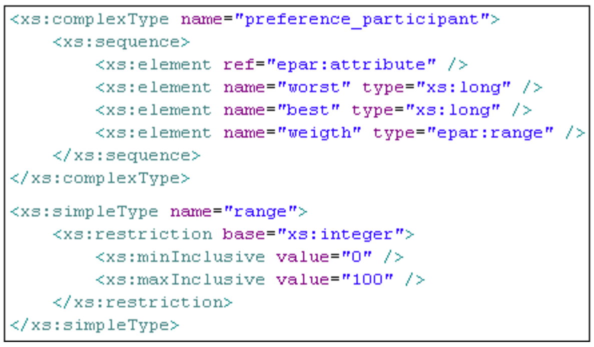
- The <preferences_participant> element stores the weight that the participant gives to each attribute, and the best and worst value of which of the attribute.
- The <preference_points> element stores the values assigned by the participant to a few attribute points.



```
<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 15 *Structure of the type "profile".*

Firstly, the participant communicates his best and worst value by the maximum and minimum value of each attribute of the problem. Also, the participant indicates the weight assigned to each attribute. This weight must be a value between zero and one hundred, and the sum of the weights given to all attributes must be one hundred. 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> that 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 16:



```
<xs:complexType name="preference_participant">
    <xs:sequence>
        <xs:element ref="epar:attribute" />
        <xs:element name="worst" type="xs:long" />
        <xs:element name="best" type="xs:long" />
        <xs:element name="weigth" type="epar:range" />
    </xs:sequence>
</xs:complexType>
<xs:simpleType name="range">
    <xs:restriction base="xs:integer">
        <xs:minInclusive value="0" />
        <xs:maxInclusive value="100" />
    </xs:restriction>
</xs:simpleType>
```

Figure 16 *Structure of the types "preference_participant" and "range".*

The uniattribute preference model named is based on a linear intermediate points between the worst and best attribute values as, exemplified in Figure 17

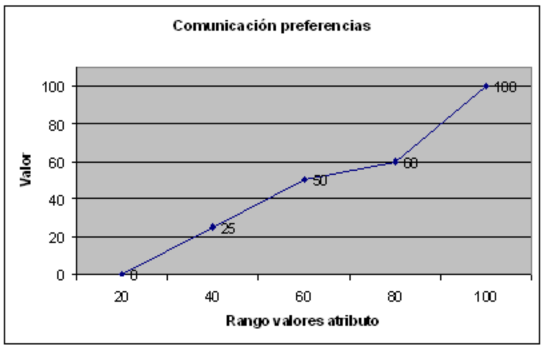
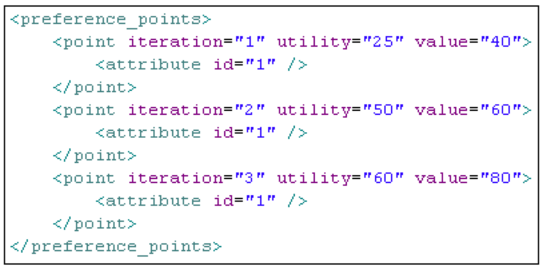


Figure 17 *Example of the straight line obtained once the user has indicated its preference.*

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 that point, the "value" attribute which is an intermediate value within 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 point. Also, the <attribute> element stores the attribute evaluated.

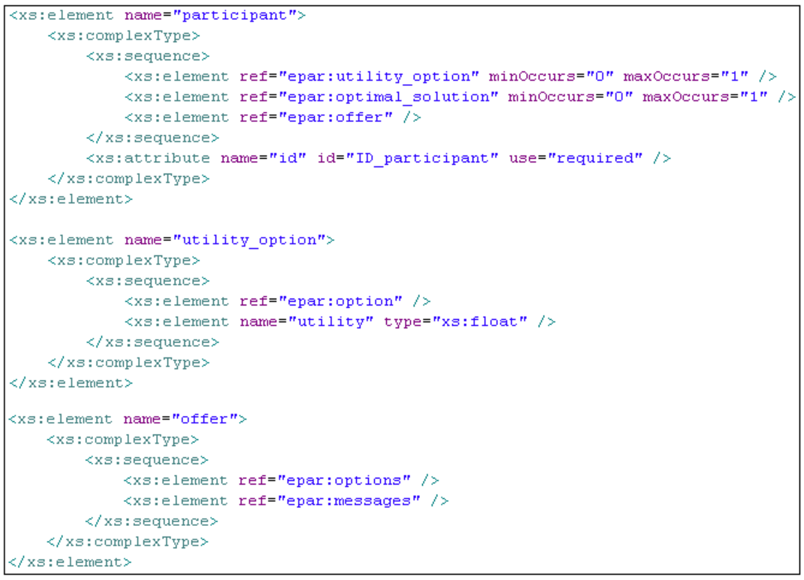


```
<preference_points>
    <point iteration="1" utility="25" value="40">
        <attribute id="1" />
    </point>
    <point iteration="2" utility="50" value="60">
        <attribute id="1" />
    </point>
    <point iteration="3" utility="60" value="80">
        <attribute id="1" />
    </point>
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

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

4.3.4. Negotiation The negotiation module implements currently two different negotiation protocols, POSTING and BIS, see Ríos and Ríos Insua (2008) and Rios et al. (2010), for descriptions. Each one has its own protocol. In this paper we talk only about POSTING. In POSTING negotiation the participants know their value functions, if they have communicated their preferences to the system. This is important because participants can see their most preferred options. In addition, the participants can post suggestions to get votes by other participants.

XML schema of this module has three main elements: “participant”, “utility_option” and “offer”, see Figure 19:

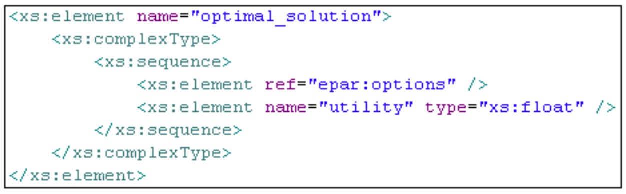


```
<xs:element name="participant">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="epar:utility_option" minOccurs="0" maxOccurs="1" />
            <xs:element ref="epar:optimal_solution" minOccurs="0" maxOccurs="1" />
            <xs:element ref="epar:offer" />
        </xs:sequence>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

<xs:element name="utility_option">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="epar:option" />
            <xs:element name="utility" type="xs:float" />
        </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 19 *Structure of the elements “participant”, “utility_option” and “offer”.*

The <utility_option> element stores the values that the participant receives for each option posted in the negotiation process. The <optimal_solution> element stores the optimal solution for the participant. It has an <options> element with all options of the optimal solution. Finally, the <offer> element has two elements: the <options> element contains the options of the offer, and the <messages> element, contains the text that explains the offer.

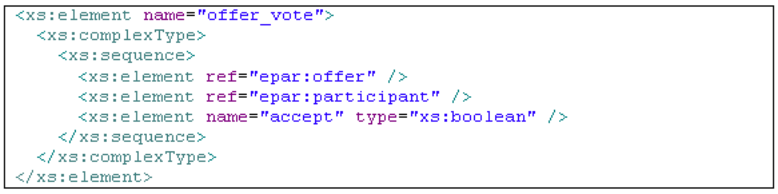
Another important element of this module is <optimal_solution>, see Figure 20, which contains an <options> element with the options of the optimal solution and the <utility> element with the value of all solutions.



```
<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 20 *Structure of the element "optimal_solution".*

Finally, the <offer_vote> element, see Figure 21, stores the votes received by posted offers. It has the <offer> element with the set of options of the offer, the <participant> element to know the participant who send the vote and the <accept> element whose type is boolean, with the "true", if the vote is favorable, or "false", otherwise.



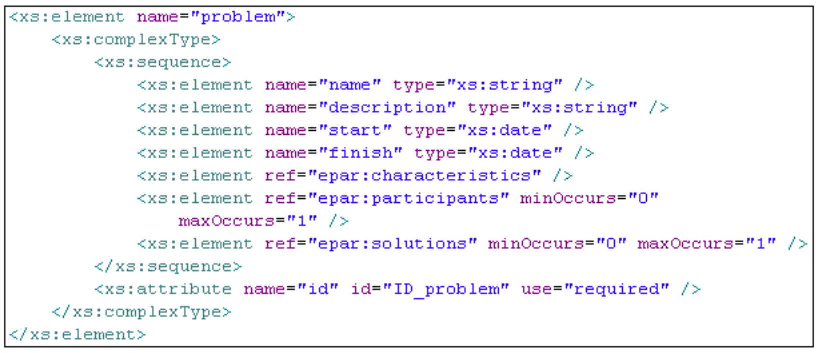
```
<xs:element name="offer_vote">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="epar:offer" />
      <xs:element ref="epar:participant" />
      <xs:element name="accept" type="xs:boolean" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 21 *Structure of the element "offer_vote".*

4.3.5. Arbitration If the arbitration module appears 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 the new elements. <solutions> as in Figure 22.

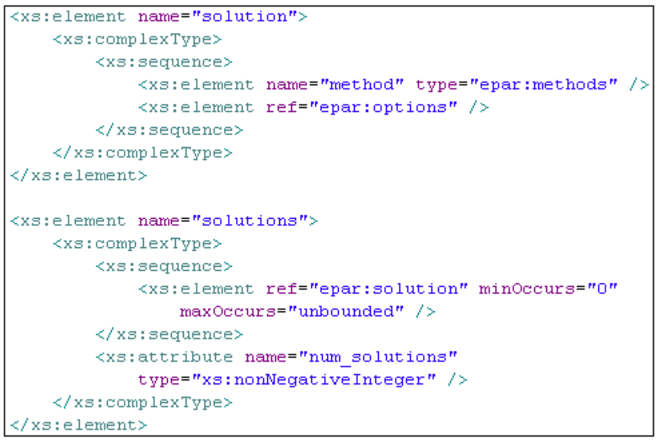
The <solutions> element contains the optimal solution, or solutions in case there are more of one optimal solution. It has an attribute called "num_solutions" to indicate the number of optimal solutions and the <solution> element with the solution proposed by the arbitrator.



```
<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 22 *Structure of the "problem" of arbitration module.*

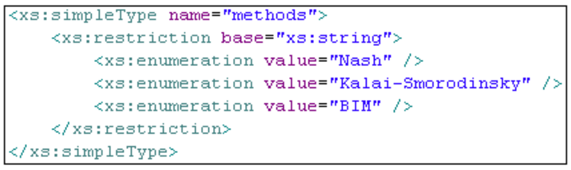
In turn, the <solution> element, see Figure 23, has two elements, the <options> element shows the options of the solution and the <method> element specifies the method used by the system to obtain the solution.



```
<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 23 *Structure of the elements "solutions" and "solution".*

The values of the <method> element, see Figure 24, are specified in the simple type called "methods" which are strings indicating the resolution 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 24 *Structure of the "method".*

The algorithms implemented in the arbitration module are, currently, as shown in Figure 24, “Nash”, “Kalai-Smorodinsky”and “BIS” (Increment Balanced Solution), see Raiffa et al. (2002).

5. Conclusions

In last years there has been an interest in participatory processes to allow citizens to take part in public decision making that affect them. As consequence, participatory instruments are flourishing all over the world to allow citizens, from reporting to politicians for guiding them in decision making or to allow citizens to take their own decisions. We have reviewed some of these participatory instruments and we realized that there are many common tasks in all instruments. If we combine each of these tasks we could obtain new participatory instruments that could be, in some cases, more efficient that currently participatory instruments.

In this paper, we proposed an architecture and a set of XML schemes for designing, developing and deploying e-participation processes. The architecture described is a web-services system to provide support for citizens in participatory 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 to make decisions using ICT and decision technologies. We have also developed the necessary XML schemes (GroupDecXML) and Web services to exchange information between the phases of the participatory process. Each module of the architecture has its own XML schemes 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 stage of the various participatory processes. Thus, we save time when it is necessary to use a common stage for different process. The repository would have initially with the process defined above and when a new process was defined, it would add to the repository. Thus, over time we get a store of participatory processes ready to be loaded on the system.

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 and ALGODEC. Discussions with David Ríos Insua are gratefully acknowledged.

References

Alfaro, C., Gómez, J., Lavín, J.M. 2009. A configurable architecture for e-participation support, *MeTTeG Conference*, Vigo.

Alfaro, C., Gómez, J., Ríos, J. 2010. From participatory budgets to e-participatory budgets, in Springer. 5: 283-300.

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/

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.

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.

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

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

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

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

SOA. 2011. http://wikipedia.org

Sousa Santos, B. 2004. *Democracia e participaçao. Ocaso do oçamento 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?*

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

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

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