Designing a General Architecture to Support eGovernment

Carlos Grima-Izquierdo
Department of Statistics and Operation Research, Universidad Rey Juan Carlos (Madrid, Spain).
Email: carlos.grima@urjc.es

David Ríos Insua
Universidad Rey Juan Carlos and Royal Academy of Sciences (Madrid, Spain).
Email: david.rios@urjc.es

We propose a computer system that would support eGovernment activities, with emphasis in decision making, for a city, region, country or a supranational entity. We provide its high-level description, using UML as modeling language. First, we establish basic terminology in relation with eGovernment. Then, we carry out the general software architecture at design level. This architecture consists of the system decomposition in subsystems and databases, specifying all dependencies among them.

Keywords: eGovernment, eDemocracy, eProcedure, eAgency, eService, decision support.

1. Introduction

Electronic Government (eGovernment) is the application of concepts, techniques and tools of the "Information Age" to all activities related with the government, the Public Administration, and political decision making: execution of administrative procedures, tax payment, votings, public debates, public budget elaboration, law elaboration, etc.

EGovernment is becoming possible thanks to technologies such as digital signatures, computer security[1], bureaucratic procedure automation, etc. In addition, if we are in an eDemocracy, we would need algorithms for negotiations[2][3], electronic voting techniques and security[4][5], computer supported participatory budget elaboration[6], debate structuring through Internet[7], etc. Here we shall propose a generic, scalable and reusable architectural description of a complete computer system that integrates these ideas for a realistic and feasible development and deployment in a modern society. There have been some partial efforts made until now[8][9][10][11][3].

Although the system can be used for any eGovernment type, we have centered on eDemocracies, because these are the most desirable forms of government according to modern criteria[12]. In addition, the system is conceived to provide support to all public administrations of a whole society, at every level (local, regional, national, supranational, etc). This entails that it might be used daily by lots of citizens. All this implies that this system will be big and complex. Therefore, a careful description, as the one we give here, is required.

We begin describing the terminology we use. Then, we enunciate the involved actors and functional requirements, using UML use cases[13]. Next, we explain the nonfunctional requirements, their relevance and justification. Finally, we split the system in parts, subsystems and databases, enunciating their characteristics and inter-relations, with emphasis on decision making issues. We represent this system decomposition using UML package diagrams[13].

2. Terminology

As a working definition of eGovernment, we propose the set of computer procedures, resulting of the union of eAdministration and ePolitics.

- eAdministration is the set of computer procedures deriving from the computerization of the bureaucratic procedures necessary for the operation of a Public Administration. Some examples include: paying a tax, enrolling on a competitive public exam, enrolling in a public university, pressing charges against somebody, add/modify/delete bureaucratic procedures, etc.
- ePolitics is the set of computer procedures deriving from the computerization of all procedures to make executive, legislative and judicial decisions, in order to manage a Public Administration. Some examples could be the elaboration of laws, votings, elaboration of public budgets, etc.

All existing eGovernment procedures belong to either eAdministration or ePolitics. Some of them may belong, simultaneously, to both subsets, as described in Figure 1.

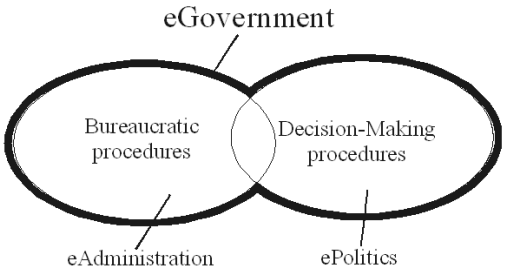

Figure 1: eGovernment, eAdministration and ePolitics definitions

We define eProcedure as an eGovernment procedure, independently of whether it belongs to eAdministration or ePolitics. Each eProcedure is formed by atomic administrative services, which we designate eServices, and by an execution order among them, not necessarily sequential. Each eService will be executed by a unique public agency or by an actor. The computer system that implements a public agency, at any level or field, is called eAgency. The complete system will actually include a set of eAgencies.

3. Requirements

We want to design a computer system that fully supports eAdministration and ePolitics in any eGovernment, specially eDemocracy. The citizens and politicians could connect to the system (server-side) from any place (client-side) through Internet, in order to execute an eProcedure. For that, we shall describe here the corresponding general requirements. Related detailed description of requirements in electronic voting is available[14]. Most of them could be applied to any eProcedure, not only votings. In addition, there is an informal description of requirements in any participation support system[11].

3.1. Actors

The relevant actors for the system will be:

- Citizens.
- Politicians: senators, councillor, mayors, ministers, etc.
- Civil servants.
- Private Organizations, such as companies, media, NGOs, religious institutions, etc.
- Political parties, which are a special type of Private Organization.

To enhance transparency and security in public performance, the "Public Organization" actor does not exist. Instead of it, civil servants or politicians will act in representation of public organizations, with the result that they are legally and, possibly, individually responsible for their actions.

In a direct eDemocracy, most citizens would be part-time politicians. In order to reduce the opportunity costs derived from the time which they use unselfishly in government activities, citizens could use "software agents" [15], that learn and use their owner's preferences. Each software agent could act in place of its owner whenever he does not have enough time or desire to participate directly. Software agents can be used also by the other types of actors in order to attain more efficiency in system use, due to automation of their interactions. This could be specially interesting for civil servants.

3.2. Functional requirements

In spite of its potential size and complexity, the system has only two large use cases:

- To interact directly with other actors.
- To execute an eProcedure.

The first one consists simply of the possibility that actors exchange among them messages and other computer resources, such as archives, links, etc. Using this internal messaging, we obtain several advantages: the possibility of preserving anonymity in communications; exchanging of political opinions in a structured format, something very useful for software agent learning[15]; sending messages to a group of actors depending on its geographic location; etc.

Both use cases can be initiated by any of the five actors of the system or by their software agents. In this way, we obtain homogeneity in system use, simplifying and rationalizing it. Then, the system will analyze whether that actor has permissions to execute the eProcedure, according to the actor type and other factors.

3.3. Nonfunctional requirements

There are four types of nonfunctional requirements for the system:

1. Usability. The system could be used, in principle, by millions of citizens. Therefore, usability should be enhanced. If not, we would be contributing to "the digital divide"[12].

2. Scalability. This is related with the fact that this system could be used at different administrative levels, simultaneously or not. Therefore, the system has to be designed so that it can easily adapt to different numbers of users, eProcedures and eAgencies. Also, eProcedure and eAgency modules should be added or removed easily.

3. Minimum coupling. It is demanded by the second requirement. Coupling among subsystems has to be minimal as the system may have a huge size. To achieve it, the subsystems have to communicate and exchange information in a homogenous and stable form.

4. Security. The system must be technologically secure so that citizens may use it with confidence and can trust it. There are numerous sensitive aspects related with security in eGovernment[5][8][17], including:

a. Reliability and robustness against any fault or attack. To achieve it, we require the system and its databases to be distributed and redundant[16].
b. Traceability and transparency. Everything made by the system has to be recorded in order to be consulted by political parties and authorized citizens with the purpose of validating correct operation. Also, physical surveillance of the system is desirable.
c. Confidentiality in communications. This is made possible by well-known techniques such as SSL encryption[1].
d. Safe identification among users and the system, and among subsystems. It is made possible through digital signatures and public-key cryptography[1].
e. Confidentiality in data recording within the system, in order to avoid that governments know individual votes or political preferences, for example. This problem would be solved if the system does not record citizen's real identity. Instead of it, the system will record an identifier for each citizen that only he/she knows that it belongs to his/her.
f. Avoiding fraud, mainly in votings.

Security requirement "e" is solved if a participant can consult his participation and that of others, protected by the identifier described before. A fraud can be made by the user or by the government behind the system, so, when one of them makes a fraud, the other part has to prove that it happened. To do it, on every activity, the participant and the system would exchange an encrypted document, digitally signed by both. The document specifies the activity content. All this process is applied to any eProcedure. For example, in votings, the voter can view all votes, and the exchanged document has the vote content.

In order to fulfill security requirements "e" and "f" when software agents participate, it is necessary that they run in the client side, because they may have sensitive data such as citizen's political preferences. If the software agent lives within the system (server-side) instead of the client-side, a malignant government might collect this sensitive data.

4. General Architecture

We divide the system in three subsystems:

- The "eAgency Container Subsystem", which is the set of eAgencies and, therefore, executes the eServices that form eProcedures.
- The "eProcedure Control Subsystem", which implements a language to describe and execute administrative procedures.
- The "Interface Subsystem", which lets actors to interact with the system.

The system includes also two distributed and redundant databases:

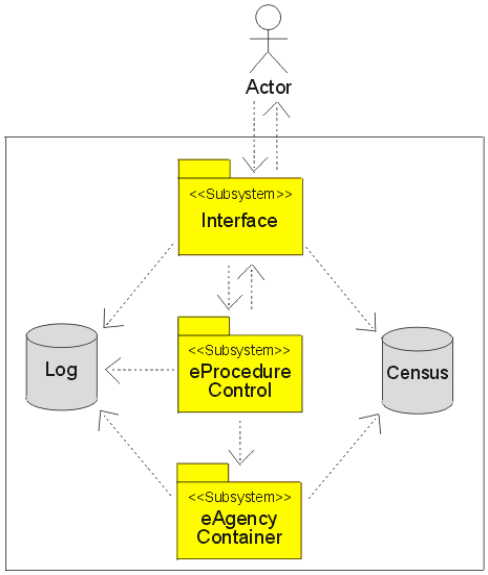

Figure 2: General architecture of the whole system

Figure 2 shows the subsystems, databases and dependencies. According to the UML notation, a dotted arrow means that any submodule in the origin uses (depends on) any submodule in the destination. An actor begins the execution of the use case "Interact directly with other actors" or "Execute an eProcedure" through the Interface Subsystem. In the first use case, the request is executed by the Interface Subsystem, allowing the actor to communicate with the receiver (human, organization or his software agent). In the second one, the Interface Subsystem requests to the eProcedure Control Subsystem the execution of an eProcedure. In both use cases, the actor has had to identify himself previously. This is possible thanks to the dependence between the Interface Subsystem and the Census Database.

The eProcedure Control Subsystem knows all eProcedures and their structure. Therefore, after receiving an eProcedure execution request from the Interface Subsystem, the eProcedure Control Subsystem analyzes it and sends each eService to the appropriate place, which processes and gives results back. This place may be the eAgency Container Subsystem or an actor, probably a civil servant (e.g., for signing or approving a document). The eProcedure Control Subsystem uses the Interface Subsystem to communicate with these other actors.

Internally, the eAgency Container Subsystem is simply the set of all eAgencies. Therefore, the eAgency Container Subsystem dispatches each received eService to the appropriate eAgency. In order to process an eService, it is possible that the eAgency, and, therefore, the eAgency Container Subsystem, needs the Census Database.

The eAgency Container Subsystem also manages the right operation of the system, through a special eAgency inside, called Computer Administration eAgency. It can add, modify and delete eProcedures, and prepares reports about the system activity, using the Log Database.

5. The eAgency Container Subsystem

The eAgency Container Subsystem contains all eAgencies, independently of their administrative level or scope. It implements the "business logic" of the system, as it processes most eServices that form eProcedures, and gives results back to the eProcedure Control Subsystem. Examples of eAgencies would be the computer system that support the Spanish Agency for Tax Administration, the Berlin City Council, Oxford University, the F.B.I., the Decision-Making eAgency (discussed later), the Computer

Administration eAgency (discussed later), and so on. The eAgency Container Subsystem enables the system to work for any eGovernment type, only updating the appropriate eAgencies in this subsystem.

5.1. Objectives and dependencies

Each eAgency is internally different from the rest, but all of them have, at least, the following goals:

- To pursue the objectives of the public agency that implements. In order to do this, it executes its eServices as required.
- To manage the security of its eServices, using the Census Database.
- To remember its execution trace in order to record it later on the Log Database.

All eServices, except those executed by actors, are executed in one of the eAgencies included in the eAgency Container Subsystem. Therefore, we ensure scalability because we can add new eAgencies to the eAgency Container Subsystem, by specifying their supported eServices.

5.2. Internal description

Internally, the eAgency Container Subsystem is divided into several parts (Figure 3):

- The eAgency Selector
- The different eAgencies, represented by the eAgency Abstract Template.
- A special eAgency, called the Computer Administration eAgency.

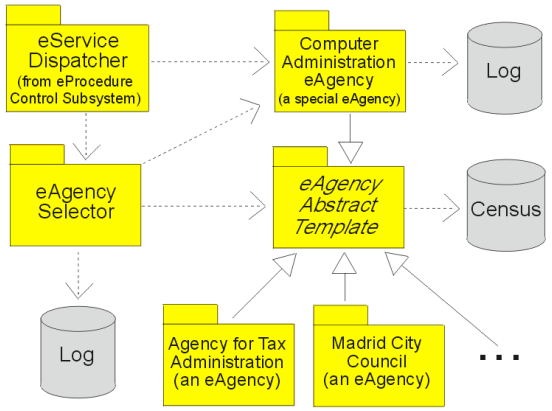

Figure 3: General architecture of the eAgency Container Subsystem

According to UML notation, a solid arrow means that the origin module has, at least, the characteristics of the destination module, and the italic letters means that the module is abstract, it is a template.

The operation is simple. The eAgency Selector receives an eService to execute from the eService Dispatcher, and sends it to the specific eAgency, descending from the eAgency Abstract Template, where it will be executed. Periodically, the eAgency Selector gets the latest execution traces from all eAgencies, in order to join them with its own and, then, to record the result on the Log Database.

5.3. The Computer Administration eAgency

There is a special eAgency within the eAgency Container Subsystem, which we designate Computer Administration eAgency, that must always exist. It is dedicated to manage the right operation of the system, and some of the eServices that it supports are: getting data from the Log Database whenever an authorized actor (such as a political party) requests it; executing eServices that add new eAgencies to the eAgency Container Subsystem; executing eServices that add, modify or delete eProcedures (using an internal database called eProcedure Database); registering new eServices and associate them to eAgencies; answering with the list of eServices supported by an eAgency, etc.

To redirect an eService request to the appropriate eAgency, the eAgency Selector must know what eServices and eAgencies there are, and which eAgency an eService belongs to. This information is obtained directly from the Computer Administration eAgency. Because of it, there is a direct dotted arrow between the eAgency Selector and the Computer Administration eAgency.

Also, in order to get an appropriate operation of the system from the early beginning, the Computer Administration eAgency must have, al least, the set of eServices described in Table 1. Of course, for the same reason, the eProcedure Database must have, from its start, some important eProcedures, which are formed by the mentioned eServices. For example, the eProcedure called "Create new eProcedure" would be one of them, and it would be formed by the following eServices: "Modify eProcedure program" (selecting the eServices to put inside and its execution order), "assign eProcedure permissions", "modify eProcedure properties" and, finally, "register new eProcedure". Furthermore, some of the eServices in Table 1 are used directly by the eService Dispatcher (see section 6).

eService category	Compulsory eServices in the category
About eProcedures	Register new eProcedure Delete eProcedure Modify eProcedure program Modify eProcedure properties Assign eProcedure permissions View eProcedure permissions View eProcedure program List eProcedures
About eAgencies	Register new eAgency Modify eAgency properties Delete an eAgency List eAgencies
About eServices	Register new eService to an eAgency Delete an eService of an eAgency List eServices of an eAgency

Table 1: Compulsory eServices in the Computer Administration eAgency

5.4. The eAgency internal description

The eAgency Abstract Template is the abstract skeleton, the template, of any eAgency. Internally, the eAgency Abstract Template is divided into several parts (Figure 4):

- The eAgency main database. It contains the eAgency internal state, specific activities/requests state, and other important data related to the eAgency.

- The eAgency Kernel, that undertakes the eAgency's tasks, without considering security issues or eService related subjects.
- The Control of eAgency eServices (CEE), that coordinates the eAgency operation. To complete the execution of each eService, CEE calls the appropriate modules of this eAgency.
- The eAgency Security Manager. It checks security constraints for each eAgency operation.

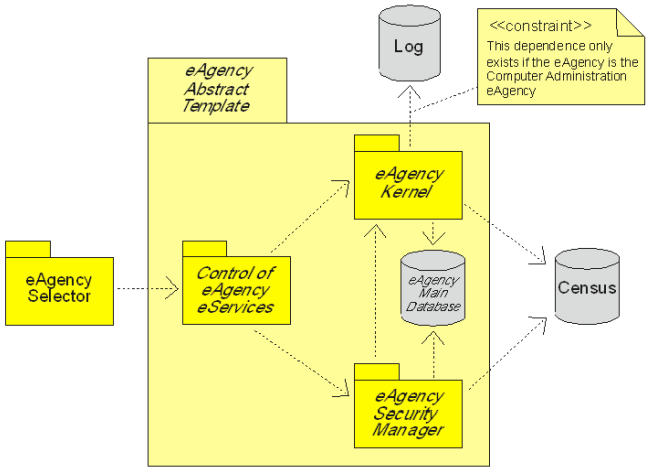

Figure 4: Architecture of the eAgency Abstract Template

The operation of the eAgency begins when the eAgency Selector requests to the CEE the execution of any eService supported by this eAgency. The CEE has, indeed, the responsibility of mantaining the list of eServices supported by this eAgency. Then, the CEE asks the eAgency Security Manager if the eService may be executed according to security constraints. In order to answer, the eAgency Security uses the Census Database, the eAgency Main Database and probably the eAgency Kernel. If the eAgency Security Manager responds affirmatively, the CEE calls the eAgency Kernel, which executes the eService, using the Census Database and the eAgency Main Database, updating them if necessary. Periodically, the CEE obtains the eAgency Kernel and the eAgency Security Manager execution traces, joins them with its own, and passes the result to the eAgency Selector when required. The eAgency Security Manager has also the responsibility of generating, for each eService, the corresponding digitally signed certificate, discussed above in security requirements, which proves that the eService has been executed.

5.5. The Decision-Making eAgency

This eAgency must exist if we are in an eDemocracy, but it could also exist in other eGoverment types. It implements all eServices related with decision-making (voting, negotiation, debate, etc). These eServices are included within the ePolitics eProcedures. Therefore, we should only update this eAgency if we want to change the decision-making structure or the political form of the society.

The Decision-Making eAgency objectives are:

- Those of the eAgency Abstract Template.
- To support the creation and structuring of any decision-making process.
- To manage the state transition of a decision making process.
- To manage information collection from the involved actors about a decision-making problem and its structure.
- To allow the involved actors to provide their preferences, in the required format, about a decision-making process.
- To support collaborative work when it is necessary to make decisions.

- To support opinion generation and comparison[7], including Habermassian discourses.
- To support negotiations or implement an arbitration scheme in order to try to promote a consensus with respect to the decision to be made.
- To manage votings, e.g. when a consensus has not been reached through negotiation.

The Decision-Making eAgency is divided into several parts (Figure 5), according to the eAgency Abstract Template description (Figure 4):

- The Decision-Making Database. It is the main database of this eAgency and contains all data related with each decision-making process.
- The eService Control. This module coordinates the remaining components from a computer point of view, according to the eServices to be executed at each moment.
- The Security Manager. It is responsible of fulfilling the security requirements typical of an eDemocracy, such as avoiding frauds, as well as having the security functionality of the eAgency Abstract Template.
- The Kernel. It is internally formed by several modules. All of them query and update the Decision-Making Database and are directly used by the Control or the Security Manager.

The Decision-Making Process Control allows to create and control the process of a decision-making problem. After creating the problem, there are several phases/states, not necessarily sequential[18]: "problem structuring", "preference modeling", "searching, generating or obtaining related information resources", "debating"[7], "arbitrating", "negotiating"[3][19] and "voting"[17][14].

The state transition is governed manually, with eProcedures initiated by actors, or automatically with some of the algorithms included in the Decision-Making Process Control. In general, this module has to be able to support any possible decision making process[20]. It should also support every decision domain type: budget elaboration[6], law elaboration, judicial decision making, etc. This module will call, at every moment, the corresponding module of the Kernel.

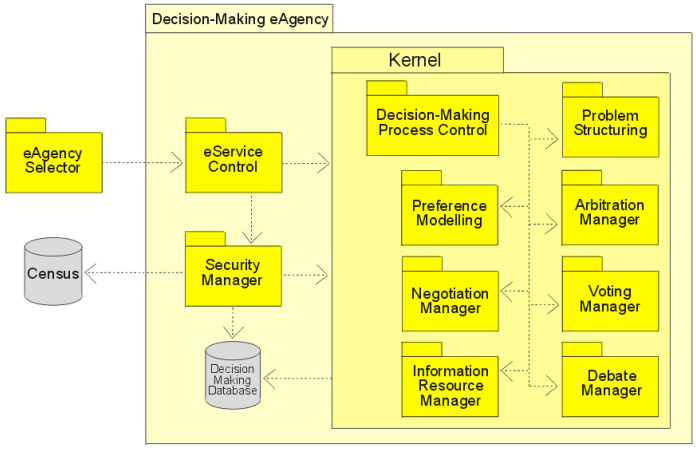

Figure 5: Architecture of the Decision-Making eAgency

When an eService execution request arrives from the eAgency Selector, and after verifying with the Security Manager that security specifications are fulfilled, the eService Control calls the Decision-Making Process Control if the action is to create or control the process flow of an existing decision making process. However, if the eService refers to structuring the problem, modeling preferences, putting/getting related information, debating, arbitrating, negotiating or voting, the eService Control calls directly the appropriate module in the Kernel.

The Information Resource Manager manages all information related with the decision problem: Internet links, book references, electronic documents, previous related debates, etc. It could include tools to obtain information cooperatively among several participants, such as wikis or videoconferences.

6. The eProcedure Control Subsystem

The eProcedure Control Subsystem controls the execution of eProcedures. It calls the eAgency Container Subsystem or an actor for each eService of an eProcedure in order to complete its execution.

The eProcedure Control Subsystem objectives are:

- To check the security at eProcedure level, such as to check if the actor has permissions to execute the eProcedure.
- To interpret an eProcedure, deducing the eServices that have to be executed at each moment.
- To send to the eAgency Container Subsystem or to an actor each eService to be executed at every moment, and get the corresponding results back.

EProcedures must be described with some standard and structured form so that the eProcedure Control Subsystem may interpret them. Internally, we could describe the eProcedure input/output data with XML[21][22], but we would still need a language for the execution flow. This language should be a high-level one and easily understandable, if we want civil servants to create and modify eProcedures without requiring advanced computer expertise. We are currently developing a skeleton of it.

The eProcedure Control Subsystem is divided into the following components (Figure 6):

- The eService Dispatcher. It sends eServices to the eAgency Container Subsystem or to an actor (using the Interface Subsystem). It also coordinates the execution flow in order to complete the eProcedure.
- The eProcedure Interpreter. It obtains, from an eProcedure, the eService to execute at each moment.

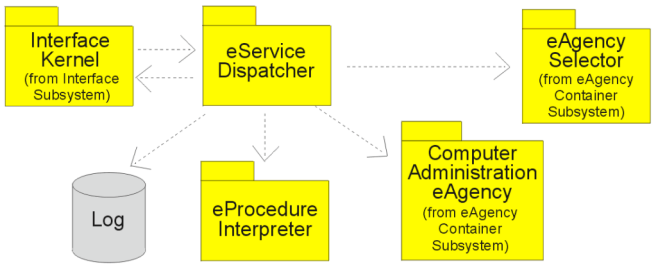

Figure 6: Architecture of the eProcedure Control Subsystem

When a request to execute an eProcedure arrives from an actor (through the Interface Kernel), the eService Dispatcher follows these steps:

1. It checks if the actor can execute the eProcedure. To do it, the eService Dispatcher asks to the Computer Administration eAgency, executing the eService called "View eProcedure permissions" (see Table 1).
2. It obtains the eProcedure program description from the Computer Administration eAgency again, using an eService called "View eProcedure program" (see Table \ref{compulsoryeservices}), and passes it to the eProcedure Interpreter.

3. It begins executing the eProcedure program. At every moment:
    a. The eService Dispatcher asks to the eProcedure Interpreter what is the next eService to execute.
    b. The eProcedure Interpreter analyzes the eProcedure program to give back the next eService to be executed.
    c. The eService Dispatcher sends this eService to the eAgency Selector or to an actor (using the Interface Kernel).
    d. The eService Dispatcher saves intermediate results, eventually.
4. When the execution flow finishes, the eService Dispatcher answers to the Interface Kernel with the eProcedure final results.

EProcedures that create, modify and delete eProcedures are processed in the same manner, sending its eServices to the eAgency Selector. Later, inside the eAgency Container Subsystem, the eAgency Selector will redirect these eServices to the Computer Administration eAgency (see Table 1).

The eProcedure Dispatcher should periodically obtain the eProcedure Interpreter execution trace and join it with its own, in order to record it on the Log Database.

7. The Interface Subsystem

The Interface Subsystem is the part in charge of all interface tasks with actors (persons or software agents). Its aim is the friendly and efficient communication between actors, and between actors and the system, through the execution of eProcedures. More specifically, the subsystem objectives are:

- To be the user graphical interface for human actors.
- To be the user interface for software agents.
- To support communication-interaction among actors (persons and/or software agents), to allow them to directly exchange resources and communicate with each other.
- To allow the eProcedure Control Subsystem (through eService Dispatcher) to warn actors about any eService in which they must participate.

The Interface Susbsystem is divided into the following parts (Figure 7):

- The Interface Kernel. It contains all possible system commands, and it is thought to be used automatically by software agents, but could be used also by human actors. It could be, for example, a command line interface.
- The Interface Security Manager.
- The Actor Inter-Communication Platform (AICP). It is the session manager and communication system between actors. It could be implemented through a multi-agent system[15].
- The Graphical User Interface, for human actors.

First, the actor must identify himself, executing the right command at the Interface Kernel. In order to complete this identification, the Interface Kernel verifies whether the actor currently exists and his password is correct calling the Interface Security Manager, which queries the Census Database. To avoid double identification, the Interface Security Manager verifies that the actor is not in the system yet, asking the AICP, which keeps all computer sessions.

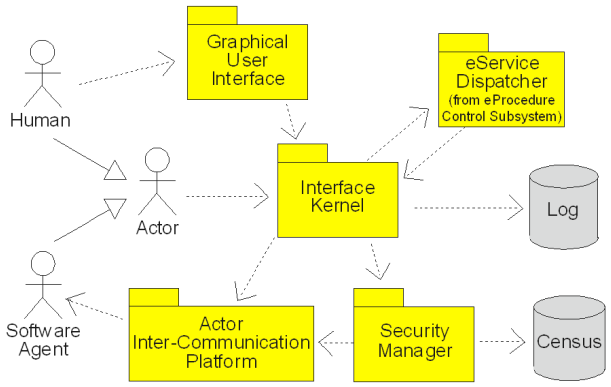

Figure 7: Architecture of the Interface Subsystem

Once the actor has been successfully identified, he may execute other commands through the Interface Kernel. Before executing each command, the Interface Kernel calls the Interface Security Manager to ask whether the actor has sufficient permissions to do it. To achieve it, the Interface Security Manager uses the Census Database to verify the actor permissions, and the AICP to verify if the actor session is still live in the system. If the Interface Security Manager responds affirmatively, the Interface Kernel executes the command. If the command is communicating with another actor, the Interface Kernel calls the AICP, which gets in touch with the other actor (synchronous communication), or leaves him a message (asynchronous communication). If the order concerns eProcedure execution, the Interface Kernel calls the eService Dispatcher instead of the AICP. The Interface Kernel gets periodically AICP and Security Manager execution traces, joins them with its own, and records the result on the Log Database.

Whenever an actor tries to execute a command in the Interface Kernel, this module communicates with the AICP in order to manage the session. Also, the AICP knows in which place of Internet, that is, which IP, each software agent is running to send it messages in real time. This information can be periodically consulted by the Interface Security Manager with the purpose of verifying whether it is coherent with itself in order to verify that the system is not under a subtle and/or concealed attack.

The Graphical User Interface module is necessary for the usability nonfunctional requirement[23]. It provides a web-based graphical user interface (GUI) for the Interface Kernel orders. We propose the following metaphor that we think is intuitive and simple: a map of the country or region, where each public building represents an eAgency. When the user clicks on a public building, a list of related eProcedures appears[24]. An eProcedure is related to an eAgency when most of its eServices belong to that eAgency.

8. Discussion

We have proposed here the terminology, requirements and general architecture to develop a computer system that fully implements eGovernment. The architecture is generic, scalable and reusable. It means that it can be implemented for any country at any level (local, national, etc.) and with any political and administrative layout. So the research question we trying to answer is not "Is possible to develop an eGovernment system?", which is obviously yes[10], but "Is possible to design a generic, extensible, reusable and highly secure architecture to develop an eGovernment system?".
First, we have defined some terms and concepts, such as eProcedure, eService or eAgency, and, later, we have enunciated the functional and nonfunctional global requirements. In future works, all these requirements would have to be clearly enumerated and described using some standard Software Engineering methodology, for example the ANSI/IEEE 830[25]. In addition, it is necessary to describe

deeply and develop some mechanisms enunciated in the security requirements, such as the encrypted document that proves that an eProcedure has been completed.

Next, to reduce system complexity, we have made a design architecture that consists of the decomposition in three parts (subsystems), with minimal coupling among them, enunciating the implied databases as well. For each of these subsystems, we have enunciated its main characteristics, objectives, operation, dependencies and its relation with the eProcedure, eService and eAgency concepts. Furthermore, we have divided them in smaller parts until a suitable complexity level has been achieved. We have studied more deeply one of these last parts: the Decision-Making eAgency, a very important computer system in a democratic eGovernment (eDemocracy). In future work, each of these parts will be independently described as a complete system, with its own objectives, requirements, actors, architecture, etc. Furthermore, an inter-operability strategy, e.g., SOA[26], is necessary to let all systems work together, keeping coupling at a minimum level. The databases should be detailed as well.

When we described the operation of the eProcedure Control Subsystem, we mentioned a high-level language to write eProcedures by humans. Defining this language and its interpreter system in future will be very important.

In order to achieve the usability requirement[23], essential to mitigate the "digital divide", we have proposed a metaphor for a web-based graphical user interface consisting of a map with public buildings. In future works, it will be clearly defined, using standards in graphical user interface design, e.g.: ISO 9241[27]. Moreover, the final obtained efficiency and usability has to be measured with statistical experiments using big enough sets of participants[28].

Acknowledgments

This research was supported by the "ESF-TED" Programme, the "EDEMOCRACIA-CM" programme, a "FPI" PhD scholarship from the Spanish Government and a "MEC" project.

References

1. W. Stallings, *Cryptography and Network Security* (Prentice Hall, 2005).
2. D. Ríos Insua, J. Holgado and R. Moreno, An e-negotiation system to support edemocracy, in *Multicriteria Decision Analysis* **12**(2003) 177-190.
3. M. Benyoucef and M-H Verrons, Configurable e-Negotiation Systems for Large Scale and Transparent Decision Making, in *Group Decision and Negotiation*. Published online on 5th April 2007. doi:10.1007/s10726-007-9073-y.
4. Scytl secure electronic voting, *The Council of Europe's Standards on e-Voting. Pnyx Compliance with the Council of Europe's Security \& Audit Standards on e-Voting* (2004). Retrieved online on January 25th 2007 from http://www.scytl.com/docs/pub/science/Pnyx_Compliance_with_CoE_Standards.pdf
5. Scytl Online World Security and Accenture, *Electronic democracy and citizen participation. Technological and functional report on the electronic Citizen Consultation MadridParticipa* (2004). Retrieved online on January 25th 2007 from http://www.scytl.com/docs/pub/science/Madrid_Participa_Technological_Report.pdf
6. J. Ríos, Supporting *group decisions through the web: e-democracy and e-participatory budgets*. PhD thesis in Rey Juan Carlos University (2006).
7. R.P. Lourenço, J.P. Costa, Incorporating citizens' views in local policy decision making processes. To appear in *Decision Support Systems* (2006)

8. J.A. Rubio, D. Rios Insua, J. Rios and E. Fernandez, QuixoTe: Supporting Group Decisions Through the Web, in *Lecture Notes in Computer Science* **3416** (2005) 225-234.
9. M. Benini, F. De Cindio and L. Sonnante, Virtuose, a VIRTual CommUnity Open Source Engine for Integrating Civic Networks and Digital Cities, in *Lecture Notes in Computer Science* **3081** (2005) 217-232.
10. The Danish Government, *Architecture for eGovernment in Denmark*. Retrieved online on December 15th 2006 from http://www.oio.dk/arkitektur/eng
11. R.P. Lourenço and J.P. Costa, Requirements for a public participation support system, in *Proceedings of 14th Mini-EURO Conference (HCP 2003 - Human Centered Processes)* (Kirchberg Congress Centre, Luxembourg, May 5-7th 2003).
12. G. Browning, *Electronic Democracy: Using the Internet to Influence American Politics* (Prentice Hall, 2005).
13. G. Booch, J. Rumbaugh and I. Jacobson, The *Unified Modeling Language User Guide, the 2nd edition* (Addison-Wesley, 2005).
14. M. Volkamer and M. McGaley, Requirements and Evaluation Procedures for eVoting, to appear in *The Second International Conference on Availability, Reliability and Security* (Vienna, April 10th 2007).
15. W. Brenner, H. Witting and R. Zarnekow, Intelligent *Software Agents: Foundations and applications* (Springer-Verlag, 1998).
16. M. Castro and B. Liskov, Practical Byzantine Fault Tolerance, in *Proceedings of the Third Symposium on Operating Systems Design and Implementation* (New Orleans, USA, February 1999), 173-186.
17. R. Krimmer, *E-Voting 2006* (GI, 2006).
18. D. Ríos Insua, G.E. Kersten, J. Ríos and C. Grima-Izquierdo, *Handbook of Decision Support Systems* (Holsapple Ed., Springer, 2008).
19. G.L. Kolfschoten, A toolbox for the facilitation of E-Democracy, in *Proceedings of the Group Decision and Negotiation Conference* (Montreal, May 14-17th 2007).
20. C. Bayley and S. French, Designing a Participatory Process for Stakeholder Involvement in a Societal Decision, in *Group Decision and Negotiation* (Published online on 17th Feb 2007), doi:10.1007/s10726-007-9076-8.
21. The Danish Government, *The OIOXML Project*. Retrieved online on December 15th 2006 from http://www.oio.dk/dataudveksling/danishXMLproject
22. The British Government, *GovTalk: Information on policies and standards for eGovernment*. Retrieved online on December 28th 2006 from http://www.govtalk.gov.uk
23. Wikipedia, *Usability*. Retrieved online on January 10th 2008 from http://en.wikipedia.org/wiki/Usability
24. C. Grima-Izquierdo, A Graphical metaphor for an eGoverment user interface. Forthcoming on 2008.
25. American National Standards Institute (ANSI) and IEEE Computer Society, *ANSI/IEEE 830-1998 Recommended Practice for Software Requirements Specifications* (1998). Retrieved online on January 10th 2008 from http://www.ansi.org
26. Wikipedia, *Service-Oriented Architecture*. Retrieved online on January 10th 2008 from http://en.wikipedia.org/wiki/Service-oriented_architecture
27. International Organization for Standarization (ISO), *Ergonomics of human-system interaction* (2006). Retrieved online on January 10th 2008 from http://www.iso.org
28. Wikipedia, *Usability testing*. Retrieved online on January 10th 2008 from http://en.wikipedia.org/wiki/Usability_testing