A System Generating CV through Intelligent Agents and Apache Cocoon

Evelio J. González, Alberto Hamilton, Lorenzo Moreno, Juan A. Méndez, José Sigut and Marta Sigut
Dpto. Ingeniería de Sist. y Automática y Arq. y Tecnología de Computadores
Universidad de La Laguna, CP 38207, La Laguna, SPAIN
E-mail: ejgonzal@ull.es

Keywords: MAS, Apache Cocoon, Development

Received: May 5, 2006

The aim of this paper is to present a dynamic system for the automatic and dynamic generation of CV documents in an academic and research environment. In particular, this system has been developed in the FES Department of the University of La Laguna, Spain. For that purpose, the authors have integrated Multiagent Systems (MAS) with XML and Apache Cocoon, designing a web portal where the users – in this case, professors and research students- can manage their CV data. Regarding to the use of Apache Cocoon and apart from showing its great potential, one of the main contributions of the work presented in this paper consists of the dynamic generation of the web environment, since the forms presented to the user change as soon as the structure of the XML files is modified. In other aspect, the agents will ensure the privacy and safety of the data.

Povzetek: Predstavljen je sistem za avtomatsko generiranje CV.

1 Introduction

The Spanish Universitary model requires its members to manage a big amount of personal data, such as publications in journals and conferences attended. Different official institutions – Ministry of Education and Science, regional governments, universities – often require researchers and students for these data in order to different purposes such as awarding a contract or research fellowship, annual reports, etc. Unfortunately, it is usual that each institution has its own template to fill, so researchers are often condemned to waste their time typing the same data in different documents. It has been calculated that the generation of a CV takes an average 3-hour period and that an automated system could save at least $25,000 per 100 generated CV's.

This scenario immediately brings XML back. This standard language provides a well-supported and powerful technology for the described scenario. Users would only need to type their data once in a XML file and apply a different XSL transformation in order to generate each type of document. Unfortunately, there are a good number of users in this scenario that find XML extremely difficult. For example, it is hard to imagine a standard Philosophy Ph.D. student typing a XML file and applying a XSL transformation to it. Thus, it would be desirable a user-friendly web environment in which researchers and students could manage their personal data – enter/delete/look for/update a publication, course, etc. – and generate their updated CV with only a click.

For that purpose, the authors have decided to use Apache Cocoon as base of the designed web environment. An interesting tool provided by Apache Cocoon is its forms (Cocoon forms or CForms), a XML way to build forms that can be filled by the users. In this sense, and this is one of the main contributions of this paper, it would be desirable that the forms were generated in a dynamic way. In other words, that the structure of the form should change as soon as the data architecture is modified, as several fields can be added in the CV data request by the official institutions. In addition to this, the data introduced through these forms should be validated against the restrictions codified in XML, e.g., maxInclusive in numeric data.

This system for the dynamic generation of CV documents has been integrated in a Multiagent System, originally developed for the automatic management of agendas in a Universitary Scenario [1][2][3]. This system, called MASplan, has been designed using FIPA specifications [4] and its aim is to help the members of the Universitary Scenario to organize internal meetings and to get resources such as portable computers and projectors. For the integration of the web environment into that system, the authors have implemented several agents whose behaviour can be studied in an independent way. In addition to this, the agents in MASplan, and therefore the new agents, take the advantage of the use of ontologies, expressed in a highly expressive language, OWL.

Why do the authors use agents in the generation of CV documents when it seems that a simple database for each user could be sufficient? Some researchers may think that the decision of developing a MAS in this case is questionable. They could think that employing MAS is a bit like using a sledge-hammer on a thumb-tack. The answer lies in the human behaviour. Firstly, it has been observed that sometimes the corresponding author of an article forgets to communicate the acceptance of that paper to the rest of the authors. Even if the corresponding author sends, e.g., an e-mail to a co-author communicating the good news, this co-author is usually so busy that he/she prefers to update his/her CV database later, taking the risk of 'losing' the paper in his/her CV. This behaviour implies that each user sends an e-mail to every related colleague, looking for that 'lost' publications, whenever he/she needs to present a CV document, wasting a lot of time in this way. Thus, a general database – an initial attempt was initially implemented in Apache Cocoon-, covering all the users' merits, could be a good solution. However, once more the authors have bumped into the human behaviour. In spite of the security offered by the database manager, a significant number of members of the Universitary Scenario were reluctant to insert their data, claiming that they did not want leave their data in a centralized system where they could be accessed by malicious people. In this context, the features provided by the multiagent systems – distribution, reliability, proactivity, autonomous and reactive behaviour, etc.- seem to be especially useful.

The remainder of this paper is structured as follows. Firstly, state of the art in the three involved aspects in this paper – multiagent systems, Apache Cocoon and generation of CV documents- are described. After that description, the authors will detail the changes introduced in the original MASplan system in order to manage the desired environment. The next step will be the description of the web environment, paying special attention to those aspects related to the dynamic generation of CForms. After a brief inform about the experience using the designed system, some conclusions will be reported.

2 State of the art in MAS, Apache Cocoon and Generation of CV documents

As stated above, this section is dedicated to describing the state in the art of the main elements of the development of the system.

2.1 Multiagent systems and ontologies

Agents and Multi-Agent Systems (MAS) are part of a new programming paradigm. They have been successfully used in a wide range of applications such as robotics, e-commerce, agent-assisted user training, military transport or health-care. However, why using agents? This is not a trivial question, as there is not a uniform and widely accepted definition of agent. In fact, agents are often characterized by describing their features (long-lived, autonomous, reactive, proactive, collaborative, ability to perform in a dynamic and unpredictable environment…). Users can delegate to agents tasks that are designed to be carried without human beings intervention, for instance, as resource managers or personal assistants that learn from its user.

In most of applications, a standalone agent is not sufficient for carrying out the task: agents are forced to interact with other agents, forming a multi-agent system. Due to their capacity of flexible autonomous action, MAS can treat with open – or at least highly dynamic or uncertain- environments. On the other hand, MAS can effectively manage situations where distributed systems are needed: the problem being solved is itself distributed, data are geographically distributed, managed by different control systems and/or difficult to share, systems with many components and huge content…

In this context, having a shared ontology that all the agents utilize in their inter-agent communication is critical to successful communication because a shared ontology provides the common format in which express data and knowledge. An ontology is a set of classes, relations, functions, etc. that represents knowledge of a particular domain.

Ontologies have made a number of applications which are more capable of handling complex and disparate information. Agents do not offer any advantage if they are not intelligent and ontologies represent an intelligent way to manage knowledge. The main advantages of the use of both MAS and ontologies are extensibility and communication with other agents sharing the same language. These advantages are shown in the case of open systems, that is, when different MAS from different developers interact. Moreover, agents can use the information stored in the ontology (not only vocabulary, but instances and constraints/axioms) for achieving their goals.

There are several ontology languages such as KIF or OKBC (although this is actually the name of an API). Nevertheless, the authors prefer those languages known as "markup languages". The last generation of these languages (RDF, DAML+OIL, OWL) offers several advantages: source files are compact and portable, easy to learn and use, flexibility...

In this sense, OWL is on the top of the ontologies languages, although there are other markup languages, such as DAML+OIL that are sufficiently expressive for carrying out the project described in this paper. As W3C indicates:

"Where earlier languages have been used to develop tools and ontologies for specific user communities (particularly in the sciences and in company-specific e-commerce applications), they were not defined to be compatible with the architecture of the World Wide Web in general, and the Semantic Web in particular. OWL uses both URIs for naming and the description framework for the Web provided by RDF to add the following capabilities to ontologies: ability to be distributed across many systems, scalability to Web needs, compatibility with Web standards for accessibility and internationalization and openness and extensibility. OWL builds on RDF and RDF Schema and adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes."

OWL is usually recommended for ontology developments due to the following reasons:

- It is based on well-known existing technologies: XML, RDF, DAML+OIL...
- There are many tools with regard to OWL such as editors, reasoners, consistence checkers and translators from other ontology languages (DAML+OIL).
- It allows the representation of instances, not only classes and/or relations.
- It is the most likely candidate to lead Semantic web. In this sense, it is a recent W3C recommendation.

Nevertheless, some disadvantages have been found:

- Tools are at very early stage, especially for the most expressive subset (OWL Full).
- There are some limitations on what can be expressed, especially in the subsets called OWL Lite and OWL DL.
- Unfortunately, OWL does not allow developers to express all the constraints they would like, for instance, those related to the limitation in the range of a value or constraints between values. In this way, A MAS system could manage these situations, for instance, taking advantage of the fact that OWL datatype properties may make use of simple types defined in accordance with XML Schema datatypes.

It is noted that the main aim of this paper is not to justify the use of MAS and ontologies in a strict way (there is a lot of literature about that fact), but to present the software experience when using them in the development of an example application [5,6].

2.2 Apache Cocoon

The second component used in this work is Apache Cocoon. This is a mature and popular web development framework, developed in Java and based on the Servlet API model, that has been built around Separation of Concerns (SoC) and component-based development (COP), providing pipelined SAX processing. From a more formal point of view, Cocoon is said to adopt the "pyramid model of web contracts"(Figure 1). This model is composed of four separated contexts (management, logic, content and style) and five contract contents (the lines depicted in the figure) between the different contexts. As can be seen, the contract between style and logic is removed, as it is considered as a bugging web site development since the beginning of the Web.

It introduces design patterns, evolutionary guidelines and software that make easy the creation of web services, using a cache system in order to get a better performance. This technology, interfacing with many different back-end and data formats, is focused on human resource management rather then technological details, left in the middleware level. Apache Cocoon is a well-supported Open source project, as there can be found a good number of resources in the Internet, such as newsgroups, Wiki, etc. On the other hand, its modular structure and Model-view-controller (MVC) architecture, based on XML files, makes the development extremely customizable with minimal coding. Furthermore, there is a really plethora of tools and XML derivates that can interact with Apache Cocoon. In this sense, one of these XML languages is XML-FO (Formatting Objects), a XML namespace that allows to describe the way to build a document, that is, how to draw and place the text on screen on paper (size of the document, fonts, paragraphs, tables, etc.) and to generate different outputs (PDF, RTF, HTML, etc.).

From a practical point of view, Apache Cocoon uses a centralized configuration mechanism called the sitemap, a declarative XML document describing a set of pipelines that will be invoked upon a URI pattern match. A pipeline consists of three main components: a generator (which produces SAX events), one or more transformers (which operate on the SAX event stream, transforming it into some other grammar) and a serializer (which transforms the SAX event stream into an output stream for the client browser or file) [9].

In spite of Apache Cocoon started as a simple servlet for static XSL styling, it has become a powerful tool as many features have been included. One of these features is their CForms. It consists of an advanced forms framework that provides a solid basis for building interactive web applications. Describing the structure of the forms involves the definition of the widgets it consist of. A widget is an object that knows how to read its state from a Request object, how to validate itself, and can generate an XML representation of itself. Using widgets, developers can indicate, for example, that a specific field should contain an integer or a date [10]. In order to create a CForm, a developer needs to define two XML files: a form model – that describes the structure of the form- and a form template – that informs the place a widget is desired to appear in the form. An optional third file is the form binding, which avoids having to write actual code for the edition of things like the properties of a bean or data from an XML document [7,8,9].

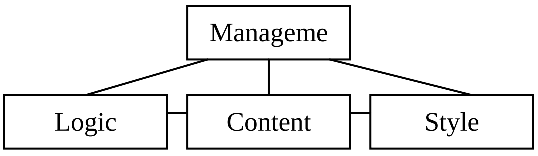

Figure 1: Pyramid model of web contracts

2.3 Generation of CV's

The problem of the automatic generation of CV's is not new. A good number of institutions and companies have realized that users often waste a lot of time when writing, updating and - not less important- formatting their CV's. For example, collecting the CV's of a relatively high number of members of an institution for an annual information and giving the impression of style homogeneity could become a heroic effort, even when strict guidelines were given. Thus, several developers have tried to solve this problem through different software. The authors will cite in this work three significant approaches.

Vitae is a free Tcl/Tk-based curriculum vitae management tool for use with the Illinois Computer Affiliates Program, whose purpose is to standarize the format of CV's provided to visitors to the annual Illinois Computer Affiliates Program Conference. Another cited system is a more general application developed by i-Linksoft solutions for the Faculty of Medicine of the University of Malaya. Among other features, the system allows the faculty staff to avoid writing their CV's every time it is needed for submission. Finally, University of Windsor is said to save thousands of dollars using a commercial application, IntelliPRINTPLUS, for reporting from Lotus Notes applications. This tool is used for the automatic generation of CV's, eliminating manual CV processes.

All these attempts seem to be limited to an only style of CV and they do not take any of the advantages of using the XML technology. Moreover, two of them are based on commercial products, so a user is laid when a structural change in the application is desired. In this aspect the design of a system, based on XML and open-source technologies is justified.

3 MAS Architecture and Development

The application of MAS to this problem is justified by the following reasons.

- The environment is dynamic. For instance, the number of users and their preferences can change in an unpredictable way. The agents should adapt themselves to these situations.
- The agents form a distributed system and it is not necessary a permanent connection. The agents are who interact, not the users.
- Extensibility. Using both MAS and ontologies, new types of agents (or new instances of the same agents, even implemented by different developers) can be added easily to the system, making its functionality grow in a dynamic way. In general, this easiness cannot be reached by centralized systems, for example, a central server that every user interacts with via their Web browser.

For the development, the authors adopted FIPA specifications because they have become a stronger standard in MAS development and involves not only agent language specifications but agent management, conversations …

FIPA (Foundation for Intelligent Physical Agents) is an organisation whose purpose is to promote the development of specifications of generic agent specifications (Faratin et. al, 1998). Its agent management reference model provides the normative framework within which FIPA agents exist and operate. The Directory Facilitator (DF) provides yellow pages services to agents that query it to find out services offered by other agents. The Agent Management System (AMS) offers white pages services and maintain a directory, which contain transport addresses for agents registered in the Agent Platform (AP). The Message Transport Service (MTS) is the default communication between agents on different APs (FIPA Agent Management Spec.).

These specifications allow users not to be worried about technical aspects such as a detailed communication implementation. As indicated above, agent-based computing provides the decomposition, organization and abstraction of multifaceted applications in heterogeneous networks.

The authors have implemented a MAS for planning and scheduling in a University Research Group. This MAS, called *MASplan*, should help group members to find the best possible time frames to perform a meeting and to designate the use of the common resources. Originally the system MASplan was composed of 6 different types of agents. The agents for CV generation have been integrated in this system, thus the authors consider its brief description as illustrative.

User Agent (UA): This agent is an end-user interface, which shows the schedule to its related user and allows it to ask for a meeting or a resource. When it occurs, this agent tries to locate its negotiator agent and communicates what user needs. Once the negotiator has finished its work, the user agent receives the result and shows it to the user.

Negotiator Agent (NA): The implementation of the meeting and resource negotiation algorithm is applied via this agent. When it is asked by its related user agent for a meeting negotiation, it looks in the DF for the negotiator agents of the rest of the intended attendees. Then, the negotiation process begins. Alternatively, in the case of a resource negotiation, it looks for the resource agent.

Ontology Agent (OA): It provides ontology services to an agent community, so that the identification of a shared ontology for communication between two agents is facilitated. The definition of an external ontology, managed by an OA, provides numerous general advantages: it permits consultation with regard to concepts, the updating and use of ontologies and it eliminates the need to program the entire ontology in every agent, hence reducing required resources.

Resource Agent (RA): This agent is invoked when a resource negotiation occurs. Firstly, it asks the Ontology Agent for the instances of the selected resource type.

Mail Agent (MA): When an agenda change is confirmed, the Mail Agent is requested by the respective negotiator agent to send an email to the user via the mail software. For this purpose, it asks to OA for the email address of the user, as these data are stored in the ontology.

Rule Agent (RuA): This agent provides the system with the ability of learning from the users. The RuA is consulted whenever there is an agenda change in order to organize a meeting. The NA will consult with this agent in order to determine whether or not the user is supposed to agree to a possible agenda change.

In this scenario, several mobile agents have been integrated. Each user owes its own mobile agent, called CVSearchAgent (CVSA) that is periodically migrating in the network looking for new merits in which its user is involved. It is clear from the nature of the scenario that the CVSA's do not need to be always active as users do not need to be continuously submitting their CV. Thus, their activity is reduced to a few hours each 7-15 days. This fact and the characteristic of mobility make that the network is not overloaded, as the interaction with other agents and the search of new merits can be done off-line.

Whenever a new merit is found, the CVSA asks the MASplan MA for sending an e-mail to the corresponding user, informing of the result of the search. This way, the user can obtain a copy of the found data for other purposes different to the generation of CVs. Apart from this interaction, the CVSA send these data to the UA for its inclusion in the local user CV database.

The implementation of these local databases does not affect the dynamic generation of the forms as the XSD's are accessed from a remote server.

As stated in the introduction of the paper, a centralized database could have been a good solution. Nevertheless, and in spite of the security offered by the database manager, a significant number of members of the Universitary Scenario were reluctant to insert their data, claiming that they did not want leave their data in a centralized system where they could be accessed by malicious people. Thus, each user owes its own database, stored in the local system and accessed through the Cocoon web portal environment with an authorization and verification process. That database should be managed by an agent in the system. In order to reuse code and resources, the authors have implemented that management functions in the UA code. Thus, the CVSA interacts with the corresponding UA when it is necessary. One of the actions to be carried out is to avoid redundancy in the merits.

CVSA's have been implemented using JADE tool. This is a well-supported agent framework and the authors consider that its use is more adequate for this implementation, as it allows a better management of the agent life cycle (active/inactive), although keeping in mind the restrictions mentioned above. The fact of having implemented agents in different frameworks (the original agents were implemented in FIPA-OS tool) is not a problem as both frameworks are FIPA-compliant.

The original MASplan one has complemented with the inclusion of new concepts related to CV activities. As example of these new definitions, the following OWL code points that research activities related to the Philosophy area are disjoint with those related to Biomedicine.

```
    <owl:Class
rdf:about="#Philosophy_Research_Activity">
      <rdfs:subClassOf>
        <owl:Class rdf:about="#Research_Activity"/>
      </rdfs:subClassOf>
      <owl:disjointWith>
        <owl:Class                    rdf:about="#Biomedicine_
Research_Activity "/>
      </owl:disjointWith>
    </owl:Class>
```

This type of definitions makes easier the interaction among the CVSA's and the UA's. When both agents start a conversation, they compare the research areas of their corresponding users. In case of disjointness, there is no need of search in the XML database, avoiding its computational cost, and maybe more important, avoiding future unproductive conversations. A more refined version, currently in progress, will consist in using the DF functionality as a yellow pages service of research activities.

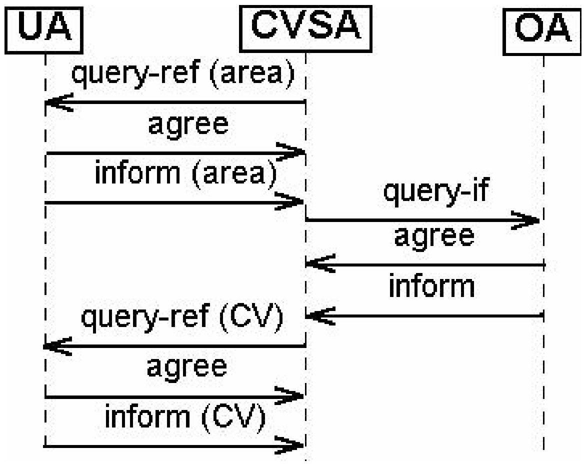

Figure 2: Message flow in the system

Another open line in this project consists of providing more intelligence to the system. The purpose would be that the CVSA's themselves were able to extract the rules of interaction among users regarding analogue research areas- for example, if two areas are compatible or not – from the list of keywords in the titles of publications or common authors in the merits. For that purpose, several well-known techniques can be used, e.g., Dempster-Shafer method [10]. After that deduction, the CVSA would interact with the RuA and OA for the inclusion of the deduced axiom in the ontology.
Figure 2 shows a standard simplified message flow between the CVSA and UA agents.

4 Description of Cocoon Web portal

In this section, the Cocoon web portal will be described. It is noted, as stated above, that the interaction of the web portal with the MAS is done via the UA's. The Cocoon distribution offers a good number of useful developing examples. From one of these examples, in particular from the address *samples/blocks/portal-fw/sunspotdemoportal*, the authors have developed their web environment.
After an initial verification process, the user is presented a frame configuration where several parts can be distinguished. The most important one is located on the left, including a set of controls. Clicking on the controls, a user can insert, modify, look for and update their data, as generating several formats of CV. An example of generated CV CForm is shown in Figure 3. It is important to remark that the number of the identity card is the parameter used to match the data with the corresponding user. This is necessary due to it is clearly not desirable that a user could access data related to other users. The use of this parameter in the session is not trivial and requires modifying some XML and XSL files, such as *sunrise-newuser.xml*, *sunrise-changeuser.xml*, *portal.xsl* and *sunriseconfHTML.xsl*.
The data involved in the structure of the system have been divided into four categories, declared in their respective schemas.

- Courses (lectures, seminaries): name of the course, recommended bibliography, etc.
- Personal data: name, address, spoken languages, etc.
- Merits: list of the merits to be included in a CV, such as publications in journals, conferences, books, chapters, research activities, etc.
- Groups: list of research groups, members, address, director, etc. – accessed only by the administrators of the system.

It is noted that for avoiding repetition in this structure, each document includes a reference field, a kind of main key in a database. This key is the number of identity card (DNI) of the user.
As stated above, the web environment is generated in a dynamic way. It is interesting that the information could change following the structure of the XML files, in other words, following the changes produced in the XML schemas. This purpose has been reached through the design of a set of XSL transformations that allow generating Cocoon forms. As expected, a change in the XML schema involves a change in the corresponding CForm.
For this purpose, three transformation files were designed: *form-template*, *form-definition* and *form-binding*. These transformations are applied to the XSD files in order to generate the forms presented to the user. A possible problem could be that the application of three transformations whenever a form is generated could be excessively slow. Fortunately, Cocoon offers an effective cache system. When a form is loaded for the first time, the rest of forms can be showed in a more quick way. This methodology has been simultaneously used by other developers, in particular, Arje Cahn and Max Pfingsthorn from Hippo.nl. However, that implementation is even much more limited than the presented in this work. As example, the date type is treated as a mere string.
The mentioned files collect the XSD structure and extract recursively the data and turn it into a CForm structure. All of them have a similar structure, based on recursive invocations with attribute inheritance and recollection in groups of information. Thus, the differences are not in their structure but in their content, due to their specific syntax using their respective namespaces.
It is noted that for each transformation three possible simple types have been considered:

- Those defined as *xs:simpleType*.
- Those defined as complex type, but they are *xs:extension*.
- Those types that have directly defined from the complex type, for example, a complex type with attribute *type="xs:string"*.

It has been observed that there are some fields that are rather difficult to manage. A significant case is that of the "author" field in the file including the CV Merits. This field should include the number of the identify card of each author involved in the merit (for example, the list of authors of a paper accepted for publication in a journal). Nevertheless, these numbers are difficult to remember by the users. It would be desirable an easier way for the inclusion of the authors. This problem has been resolved including a new attribute, called "src", to the tags *xs:element* and *xs:attribute*, called "Combo-Professor" when it is necessary. This name is related to the match actions -in the sitemap file located in the directory forms- that indicate that users should be inserted through a combo widget instead of a textfield one.
As example of design of XSL transformation, the authors will describe the transformation in order to get the form template file.

4.1 Design of the form template

The algorithm for the creation of this form is based on the following steps:

```
ↂWhen a complex type is found
      ↂFor each element, a group of items is created:
          ↂ If (maxOccurs=1) and (minOccurs=1):
            ❧Recursive call
        ❧ Else:
            ❧Add a repeater
            ❧Recursive call.
            ❧Add bottoms for its management.
    ❧For each attribute, recursive.
❧When a simple or similar type is found
    ❧Add a widget with a field "id", that is a
      parameter inherited from the previous
      recursive calls. This field is the chain of the
      id's of the previous elements that make up a
      unique path to the node. That also allows to
      establish the style of the widget.
```

Mainly, the idea consists in reading the schema from the root element to its children elements, at the same time as each child element inherits the attributes of its parent element. This structure will be used whenever the system needs to extract information from a XSD document.

4.2 Restrictions in the transformations

The designed transformations do not cover all the possible XSD syntax. As it is said in Apache web site "The difficulty with schemas is that they can contain an implicit structure by means of references to elements" However, due to the designed structure, it is possible to make easily updates and code improvements. The main restrictions in the XSD structure are summarized as follows:

- There can be as many complex and simple types as desired, but it is not allowed that the root element – or their children – has those types as children elements.
- Each immediately children element of the root element must define an attribute "xs:ID", that identifies univocally that element. The name of that **ID element must be the same for all the elements of the same document.** This element is used due to two reasons:
  - Avoiding repeated id's.
  - Allowing the search of elements for modifications and deletions.

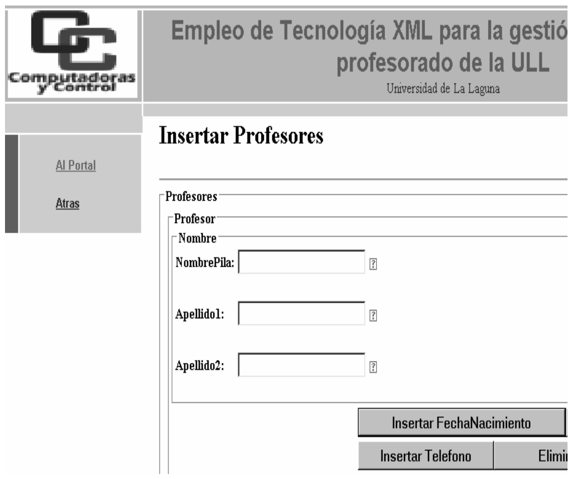

Figure 3: A detail of generated CV form

- In a complex type the following defined elements are included xs:sequence/xs:element|xs:choice/xs:element|xs:all/ xs:element|xs:simpleContent/xs:extension.
- In a complex type the following defined attributes are included: xs:attribute|xs:simpleContent/xs:extension/xs:attribute.
- Datatypes ,documentation, cardinality and validation tags can be applied to a simple type. In the case of the validation tags, they are restricted to xs:pattern, xs:minLength, xs:maxLength, xs:maxInclusive, xs:minInclusive.

The first phase of the project was only focused on this web environment. When a user desires to generate an updated CV, a XSL-FO transformation is applied in order to get that CV which is accessed in the browser. The system has been used successfully for the generation of the documentation regarding the PhD studies program of three departments of the University of La Laguna. That program covers nearly 100 professors, thus the recollection of information of the lectures and giving homogeneity to the presentation of that information could involve a great amount of time without that web environment (as stated above, the mean time for the generation of a CV has been calculated as 3 hours). Moreover, the users can get their CV with only a click when it is required or generate their web page with their research data.

5 Conclusions

In this paper, the authors have presented a multi-agent system for the automatic generation of CV's for a Universitary scenario. The agents related to this purpose have been integrated with a previously designed system, called MASplan, for planning and scheduling in a University Research Group Scenario. The included agents, called CVSA, are mobile- one for each user in the system- and their task consists of the search of merits related to their corresponding user. This search is carried out interacting with other agents, called UA, in charge of the management of local merits databases. The main reason for this distribution, and thus for the use of a multiagent system, lies in the human behaviour. Firstly, it has been observed that sometimes the corresponding author of an article forgets to communicate the acceptance of that paper to the rest of the authors and that a co-author is usually so busy that he/she prefers to update his/her CV database later, taking the risk of 'losing' the paper in his/her CV. Secondly, a significant number of members of the Universitary Scenario were reluctant to insert their data, in a centralized system. In this context, the features provided by the multiagent systems – distribution, reliability, proactivity, autonomous and reactive behaviour, etc.- seem to be especially useful.

The original MASplan system included an ontology for the automatic meeting negotiation in an intelligent way amongst several researchers, among other several features. This ontology was implemented in OWL, a highly expressive markup language. In the context of this work, some new concepts have been introduced, for example axioms related to the disjointness of different research areas, facilitating the interaction among the agents involved. Currently, the authors are working on the addition of new interesting features in the system, as the ability of the agents of deducing axioms on-line.

Other interesting aspect of this work is the implementation of a dynamic web environment for the management of the CV generation. The development of this environment has been carried out using Apache Cocoon, a mature and popular web development framework, developed in Java and based on the Servlet API model. The web environment is generated in a dynamic way. The information changes follow the structure of the XML files, in other words, following the changes produced in the XML schemas. This purpose has been reached through the design of a set of XSL transformations that allow to generate Cocoon forms. As expected, a change in the XML schema involves a change in the corresponding CForm.

The system has been used successfully for the generation of the documentation regarding the PhD studies program of three departments of the University of La Laguna, Spain. That program covers nearly 100 professors, so the use of this web environment has saved a lot of time in the data treatment.

References

[1] E.J. González, A. Hamilton, L. Moreno, R. Marichal, S. Torres (2003) "Masplan: A Multi-Agent System for Automated Planning and Scheduling in a University Research Group Scenario". 1st Multidisciplinary International Conference on Scheduling: Theory and Applications. Nottingham.

[2] E.J. Gonzalez (2004) "Diseño e implementación de una arquitectura multipropósito basada en agentes inteligentes: Aplicación a la planificación automática de agendas y al control de procesos". Ph.D. Thesis, University of La Laguna.

[3] E.J. González, A. Hamilton, L. Moreno, R. Marichal, J. Toledo (2004) "Ontologies in a Multi-Agent System for Automated Scheduling". Computing and Informatics, Vol. 23, No 2, 157-178.

[4] FIPA Abstract Architecture Specification. Technical Report, SC00001L. FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. December, 2002.

[5] S. Falasconi, G. Lanzola, M. Stefanelli (1996) "Using ontologies in Multi-Agent Systems". Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop.

[6] Fonseca S.P., Griss M.L. and Letsinger R. (2002) "Agent Behavior Architectures. A MAS Framework Comparison". Proceedings of the first international joint conference on Autonomous agents and multiagent systems, Bologna, Italy.

[7] Stefano Mazzocchi (2000) “Adding XML Capabilities with Cocoon”, ApacheCON 2000 - Orlando, 2000

[8] Introducing Apache Cocoon”, Available: http://cocoon.apache.org/2.1/introduction.html

[9] Steven Noels. (2003) “Standards Applied: Using Apache Cocoon and Forrest”, XML Europe 2003, London, England.

[10] M. Bauer (1995), "A Dempster-Shafer Approach to Modelling Agent Preferences for Plan Recognition", User Model. User-Adapt. Interact. 5(4): 317-348.