

Semantic of eGovernment Processes: a Formal Approach to Service Definition

Annalisa Barone, Paolo Di Pietro
Diviana e-consulting
Via Pisandro, 91
00124 Rome - Italy
abarone@acm.org, dipietro@acm.org

Abstract

This paper presents Arianna, the approach used in Diviana, a small Italian e-consulting organization, for defining a standard in describing the semantic of e-government services. Such a standard is born sharing (and not imposing!) information with Italian Local Public Administration (LPA) Entities, especially Comuni.

In order to describe semantically the LPA services, our approach models ontologies using the Unified Modeling Language (UML).

The UML model is automatically converted in a SVG site semantically browsable, as further explained later, and a set of XML Schema Definition (XSD) files, describing data structure used in the services.

Moreover, such XSD files represent the communication standard intra and inter LPA entities; in fact the XSD describes the base elements for implementing the application interoperability.

An ontology driven front-end generator allows the generation of an XForms application, fully compliant with the model. It is an universal front-end, fully customizable by the LPA, and standardize the relationship between citizens and LPA. In a global effort to reorganize the LPA, this approach completely solve the front-end site of the e-government, allowing focalize resources on the BPR site.

All the information contained in the model repository are made available through one e-government catalogue over the internet.

The results obtained during the tasks of standardization conclude the paper.

1. Introduction

Internet diffusion supported a high standardization level: everyone can browse millions of sites which use heterogeneous technologies without having any problem.

In the second half of 90s, the interest for the Application Interoperability grew up rapidly. The growth

did not correspond with an efficient technology but with the promises it holds.

In fact, nowadays, Application Interoperability is pervasive in many Internet activities and it is transparent for the users. Application Interoperability is the set containing everything needed by two or more applications to interact each others for reaching a specific business goal.

For example, let us consider a b2b environment: e-commerce sites are tightly integrated with just-in-time producers, transport/logistics and payment sites.

Usually, the Application Interoperability is implemented using *ad-hoc* interfaces: each site has its specific technology and language.

Such sites are islands in the cyberspace: the users can browse them but each site has laws, rules and languages for its own.

This situation could be acceptable in the early Internet age but today it is obsolete. In fact the islands want to gather together in archipelagos using common laws, rules and languages.

The Italian Public Administration (PA) is an archipelago of Administrative Entities including 8100 Towns (Comuni), 103 Province and 20 Regions, each one with its Administrative Autonomy and the need of interacting and integrating each other. The Central Government and several other government related Agencies complete the picture.

In order to accomplish the integration, the critical step is defining a communication standard between Administrative Entities.

A standard could be defined using an imposing way (by law) or using a sharing process. The first way has no chance of success because it reduces the decisional autonomy of the Administrative Entities, guaranteed by the Italian Constitutional Law.

The latter way aims to share the domain information, following the method successfully used by the Internet Engineering Task Force (IETF) [1], and the W3C[2] for introducing new Internet technologies.

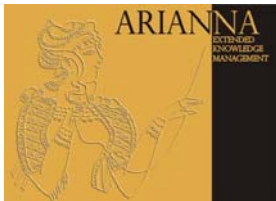
Defining and tuning a shared standard is a longer and more complex task than imposing a prefab one. In fact, the process has to overcome difficulties, rivalries and prejudices and the process managers must be *super-partes* and must have a strong authoritativeness.

Moreover, the complexity of defining a PA standard is enormous and it is easy losing the governance of the project.

In order to manage such a complexity we need a formal approach using a formal representation. We decided to use UML modelling for representing the standard, as we are going to explain in the following sections.

In order to build up a standard it is necessary to face various problems which concern with juridical, managing and monitoring aspects, beside the technology ones. Even the impacts of the change on persons have to be considered. Our goal is to describe this situation in an easy to understand and not ambiguous manner in order to discuss and share the standard. We propose the use of a formal model to accomplish such a goal.

The whole process and the tools described in this paper have been implemented by an articulate solution,



named *Arianna*. As in the myth, Arianna gave Theseus a wool thread to find his way out the labyrinth after he killed the Minotaur, our solution give the user the ability to discover his/her path into the complexity of

the Public Administration in general and, more specifically, into the Italian one. Arianna can be found at <http://Arianna.diviana.net>.

In the remaining of this paper, Section 2 introduces the concept of ontology, Section 3 summarizes the solution paradigm, Section 4 describes the architectural model, Section 5 describes the Interoperability Pattern, Section 6 makes some consideration about the reuse based approach, Section 7 shows the UML repository structure and section 8 shows the different ways used to made available the knowledge base to the users over the Internet. Section 9 makes a short description of the catalogue, while Section 10 describes the Ontology Definition and Usage Process. Section 11 presents some numerical results. Some final remarks and forwards in Section 12 and 13 conclude the paper.

2. The LPA ontology

The complexity described in the introduction must be managed, and the first enabling step is a shared knowledge. The key concept driving our approach on sharing knowledge is the LPA ontology definition.

The ontology definition is the basic to manage the interaction between a large amount of subjects, each one with its decisional autonomy, to identify the involved concepts, information, elements, subjects and roles and their mutual relationships, giving each one both a semantic definition and both intrinsic structure description. And everything must be shared, with time and patience.

To achieve this goal, an analysis of the services offered by the LPA has been made. The result has been the identification of the services, each one with their respective clients and providers.

For each service, the following items, needed to enable the service supplying, have been identified:

- Information, each one with the description of its:
 - ownership definition;
 - full information structure;
 - lifecycle.
- Normative;
- Administrative practices;
- Available best practices, if any.

Services are also been classified using different taxonomies, giving to specific classes of users/providers an easy way to retrieve and access them.

The service provider describes the specific LPA responsible for the service. In the case of complex services (i.e. services involving more than one cooperating LPAs) it describes the single LPA responsible for the entire service, usually representing the one facing with the user.

The client describes the specific subject that will use the service. It can be a citizen, a company or a third part acting as an intermediate.

The full ontology definition is based on a specific solution paradigm and is described using an UML knowledge base, as described in the remaining of then paper.

3. The solution paradigm

The solution paradigm is based on the following three main aspects:

1. The definition of an architectural reference model;
2. The definition of an interoperability pattern;
3. The reuse based approach.

4. Architecture: a Reference Model

As the main goal of our work is to define a semantic approach capable of running in different physical environment, we didn't prescribe a specific architecture, but only describe a generic logical layering, permitting hosting of specific implementations.

The result is an n-tier architecture, briefly described in the rest of this chapter and depicted by the following figure:

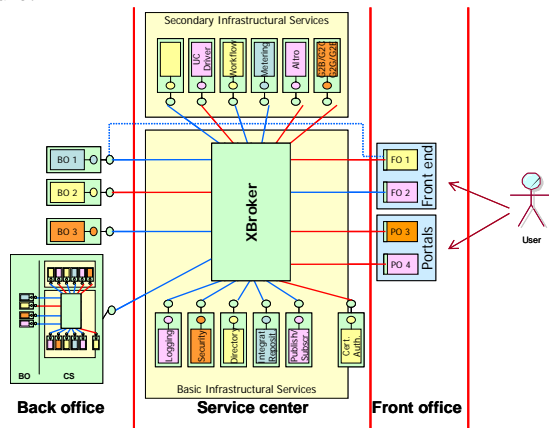


Figure 1. Architecture overview

On the above architecture, the user ask for a service using a portal or a front end application whose internals are not relevant: just the interfaces are relevant and these must be XML interfaces. The channel is still not important: it can be http or MOM (Message oriented Middleware). The receiver is always a broker, that is a central component of a Service Center. The broker, using a set of specific infrastructural services, is able to identify the final receiver and routing the request. The Back Office exposes a series of services: the service interface is in XML. As the services are often extracted from legacy systems, these systems have been first wrapped using their native technology and then wrapped again using XML. Of course, a recent Back Office implementation could directly expose an XML interface.

In the case of a federated architecture, the BackOffice could be another Service center.

The Service Center itself can be seen as a Virtual Service Layer (VSL) in a specific architecture implementation, meaning that it must not be a physical layer, but only a decoupling system between the Front-End Service Layer (FSL) and the Back-End Service Layer (BSL).

There can be several infrastructural services used by the VSL to accomplish their tasks, and all of them could be provided by third parts. These services can be grouped in two main classes: Basic and Secondary infrastructural Services. The Basic Infrastructural Services are Security services, Directory services, Publishing/Subscription services and Logging services, Certification Authority services. Samples of Secondary Infrastructural Services are Workflow and Use Case Driven workflow services, Metering services and Billing services.

5. Interoperability Pattern

After a deep analysis of the e-government services, and several intermediate steps, we were able to identify one pattern capable to hosts every e-government service.

This pattern, delineated in Figure 2, describes the different roles and the main activities they must perform to achieve the service implementation. Here is a brief description of the involved roles:

1. Requestor: the role who submits a request to the Public Administration: it is primarily a human user, but can also be another architectural layer. In both cases, it is already authenticated when the transaction begins.
2. The Front-end Service Layer (FSL), which manage the user interaction: it can typically be a portal or a generic front-end application.
3. The Virtual Service Layer (VSL), which provides common infrastructural services, (payments, Certification Authority, directory services, Orchestration services and so on). This layer provides a logical detachment between the requestor (FSL) and the provider (BSL) of the service.
4. The Back-end Service Layer (BSL), which implements the real Backoffice Service. The BSL exposes the logical interfaces for the provided services. In a real BSL implementation there should be an information system or, in the worst case, just a stub routing the requests to a human operator.
5. Administration Back Offices: here is where humans accomplish their administrative task implementing the business/government processes.
6. The Protocol: this is where a unique identifier is assigned to each received Service Request. This number has normative meaning and value: it is the real receipt from the administration, and can be used by the citizen as a proof of presentation. Depending on the particular service, the Protocol Date is the starting date to measure Service Level compliancy.

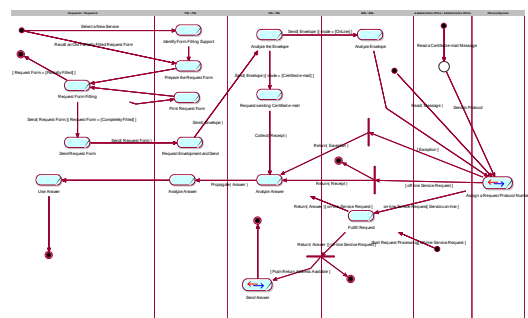


Figure 2. Interoperability Pattern

The Interoperability Pattern shown in Figure 2 is one of many interaction diagrams describing the workflow with the main activities to be performed by the different

roles. Each activity is further described by a lower level workflow.

A short description of the workflow is the following:

1. The Requestor requires a specific service;
2. The FSL can use a set of services to download information from the BackEnd; then prepares the request form and preload some fields with the information just downloaded; examples of this kind of services are 'Request for family composition' or 'Request for Owned Real Estates'; they enable the FSL to preload some request form fields, giving the user the ability to select the information relevant to the service instance avoiding an error prone manual filling.
3. The Requestor fills the form fields. If, for any reason, he/she decides to suspend the form filling, the operation is suspended and the request goes in a *partially filled state*; this state can be later retrieved by the user. Note that this behavior is only admissible for complex forms, when a human requestor could be asked for data unknown at the filling time. The requestor can also decide to print the request.
4. When the Requestor decides to send the form, the FSL prepares it filling some system info then send it to the VSL. This is the decoupling point between the FSL and the rest of the world: the FSL send a request and wait for a synchronous answer.
5. The VSL, after reading the request type and the destination, identifies the BSL address and the physical way to be used to send it: the BSL, in fact, could not only be online or offline, but could also be lacking of an Information System behind. In the latter case, the service request will be send by the VSL using a Certified Electronic Mail (CEM), with the CEM system giving a receipt routed by the VSL to the requestor. Note that the CEM system is usually a third part service.
6. In the online case, the Service Request reaches the BSL, which dispatch it to the Protocol System (another candidate for a Third Part system).
7. The Service Request is now ready to be fulfilled; we can have two options:
 - a. a Synchronous request, i.e. a request with an immediate answer by the BSL (for instance a request to access data on a Back Office Information System),

In this case, the BSL fulfills the Service request and produces the required answer sending it to the requestor through the VSL.
 - b. an Asynchronous request, i.e. a request that requires a human participation, usually fulfilled in terms of hours or days.

In this case, the BSL sends a receipt to the requestor through the VSL and queue the service request to the appropriate employee/workflow.

Every communication with the requestor will be treated with a push approach to notify an event toward one or more delivery addresses, and then the requestor will connect accessing the real communication.

8. In the CEM case, the overall process is the same. The only difference is in a human intervention to open the mailbox, read the message and then forward it to the Protocol. The successive behavior corresponds to the one described in point 7 above.

In summary, the pattern described can be applied to every e-government service. In fact, it is used all over the entire model, so that every service reference it. Each service has its own Workflow model, that is the concrete instantiation of the above pattern and, precisely, redefines the first two roles describing the specific information to download to support the user during the form filling.

Moreover, the pattern describes a Web Services Based Approach in the communication between the FSL and the BSL (the VSL acting only as a *virtual* mediator), where each e-government service has one and only one signature, given by the couple Request/Receipt or Answer. The exceptions definition completes the specification.

6. The Reuse Based Approach

From the inception phase of the entire project, our pole star have always been the *reuse*. The reuse of everything, from ideas to artifacts, from the organization structures to processes. The only self limitation from the beginning were as follow:

- Respect the actual laws and regulations, but as a critical observer, extrapolate suggestions for enhancement;
- Respect the decisional autonomy of the Administrative Entities, guaranteed by the Italian Constitutional Law.
- Act as a standardization group, looking without party-spirit both at the local administration when discussing their organizational needs and solutions and both at the software companies when discussion their technical solutions.

With these points engraved on our memory, we start reasoning about the reuse.

The first question was *Reusing what?* Do we have to reuse organizational processes or software solutions, process definition or class definition? There was no easy answer, because the number of potential users can be

huge, the number of different solution is large, there are many different ways to aggregate/disaggregate the participants, the information, the users. So, we decide to select an approach enabling us to **reuse the knowledge**.

The work experience of the authors, deeply involved not only with IT themes like standardization, modeling and architectures, but also with business consultancy in the government field, led them to discover an obvious fact: the work of the business consulting firms usually produces a huge result in term of paper, but the language used is not comprehensible to the software implementation companies; as a result the latter usually throw away the work of the former and start again doing the job, usually with an implementation driven approach and a less strategic vision.

This approach must be overcome. There is the need to use a common language for both these roles, a language enabling the representation of both business and technical objects, strictly related together. But the language itself is still not enough: there is the need of a methodological approach, a complete path from the process inception thru its final implementation, joining together all the different aspects (the knowledge) in a melting repository where each actor can manifest its own knowledge and at the same time easily discover the others' one.

As a result, all the knowledge base content has been classified in a way suitable to be reused, describing *knowledge components* to be aggregated in different ways, depending on the specific user needs. The knowledge base thus contains both simple and complex objects, where the latter are aggregations of the formers. This approach leads toward a very advanced reuse model, very effective in practice.

To avoid model pollution, it requires a special role to be designated: the Model Manager, also known as the Ontology Manager, who guarantees the coherence of the information in the knowledge base.

7. UML Repository: the Knowledge Base

Our approach can also be described in terms of building a repository containing the description of the e-government processes by every different point of view. When we needed to select a modeling language, we decide to use UML[3][4] because it is a standard, it enables us to use a formal approach to describe the semantic and it supports extensibility mechanisms.

We then start with building taxonomies. The repository main menu is shown in the following figure:

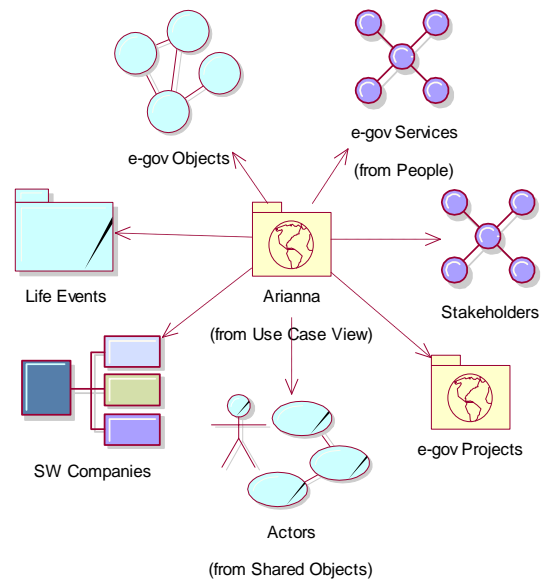


Figure 3. Repository Main Menu

Such a menu allows the users to access Services, Objects, Projects, Actors, SW companies and Class of Users. Surfing the e-gov Services taxonomy, we reach the first level of the service taxonomy:

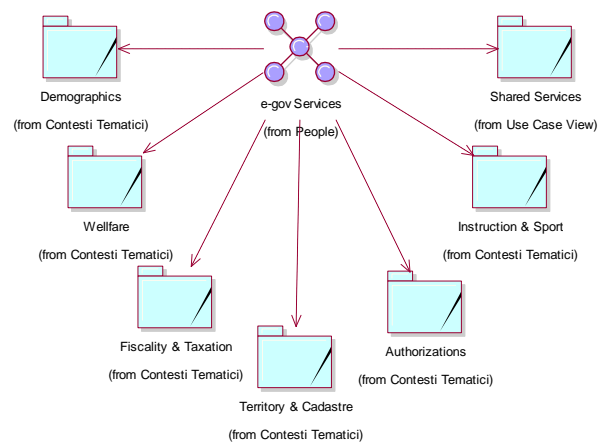


Figure 4. Main Service Taxonomy by the Administrative Organizational Point of View

Proceeding in the Demographic Area, we reach the Single Taxonomy Point of View, which groups all the artifacts related with that particular Area, and specifically

1. Services, containing all the government services belonging to the area;
2. Objects, containing all the objects declared and used only by the area;
3. Services by the FSL point of view, grouping all the service for which a user interface must be developed;

4. Services by the BSL point of view, grouping all the services for which a web service interface must be provided by the back office implementation; as shown in the following figure:

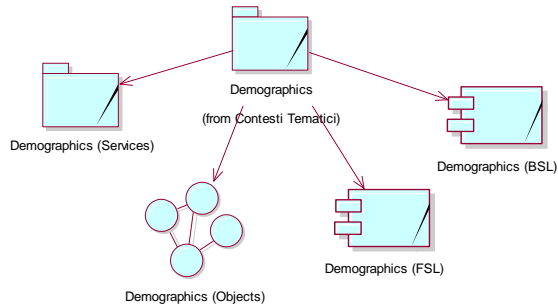


Figure 5. Main Menu for a Specific Area

Finally, selecting Services, and after an intermediate selection step, we reach the service list for the Registry Office sub area, where all the services are grouped.

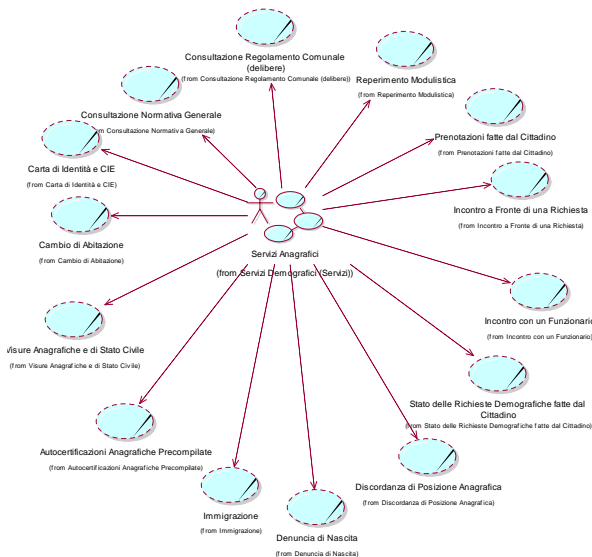


Figure 6. Services for the Registry Office sub area

Selecting a single service, we reach the main Service Diagram, showing:

1. The admissible requestor, in this case an Authenticated User;
2. The Service itself;
3. the Request Form for the Service;
4. the Receipt Form for the service;
5. other optional messages received from the Service provider.

An example of this diagram is shown in the following figure:

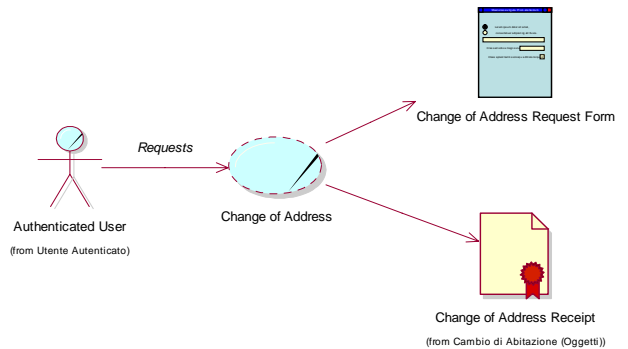


Figure 7. Main Service Diagram sample

At this point, clicking a Request or a Receipt bring directly to the UML class diagram containing the detailed description of the specific object.

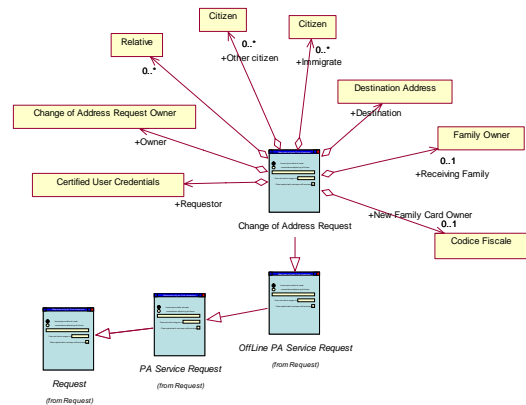


Figure 8. Class Diagram of a Specific Service Request

Of course, every item in the diagram brings to a detailed description of its components. The endpoint is an elementary item mapped on a simple type, each one, in turn, directly maps on a specific XSD simple type.

Every UML element can contain optional attributes, describing both quantitative and qualitative characteristics. Acting this way, we can tie up specific new or legacy documents, describe XML attributes (field length, pattern, number of digits, and so on), describe e-government attributes (the kind of the service: a petition, a payment, a certification, an information request), and other as needed.

Considering Figure 3, we can browse in the same way the other parts of the model. The approach is the same as described for the e-gov taxonomies. The overall idea is that the knowledge is reticular and not hierarchical, so the model can have multiple entry points, each one reflecting a specific user point of view of the subordinate information set. Each item is modeled only once in the

entire knowledge base, but can be reached through a combination of different user driven paths.

Here is a short overview of the main menu elements, representing the main model entry points:

- e-gov objects: describing all the infrastructural and the shared objects;
- Stakeholders: the model entry point for citizens (G2C), industries (G2B), other public administrations (G2G) and Civil Servants (G2E).

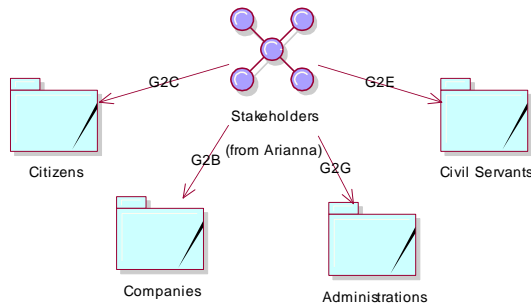


Figure 9. Model Entry Point for Stakeholder

- E-gov projects: the model entry point for each project instance;
- SW development companies: the model entry point coupling companies and service implementation;
- Life events: a taxonomy describing the service by the point of view of the citizen and company lifecycle.

8. Knowledge Base publishing

The Knowledge Base publishing is the activity needed to make available the full knowledge base content to the potential users. There are two families of users: new users and recurring ones. The goal is to give the new user the ability to easily discover the potential of the approach, becoming a recurring user, which in turn need to access the information in the easiest possible way, without following predefined paths.

The KB publishing must then make available the following items:

- The SVG Model, to navigate the knowledge using UML;
 - The XML Schema, to represent the information structure;
 - The XForms to prototype the user interfaces;
- To complete the process, the Catalogue is generated, binding together all the knowledge and giving the user homogeneous views with separate access points.

8.1. The SVG Model

The UML model must be made available over the Internet to give the users the ability to access, understand and browse it. As we use a commercial product for modeling, the first choice was trying to rely on the product capabilities for web publishing, but the result was not satisfying. In fact, we are using an UML modeler to model mainly at a Process level than at an object level and this is not the usual target for such tools. So, we decide to produce a web version of the knowledge base: our primary target is administrative and organizational people.

In order to achieve this goal, we build a tool to automatically deploy the knowledge base onto the web in a suitable way to be used by roles other than bare technicians. We decide to deploy it using SVG[5], so the results can be easily viewed in a device independent manner.

We develop the SVG generator in order to achieve:

- The ability to create logical links related with each model item;
- The ability to easily model links to every target, both internal and external to the model itself, so we can connect each model item to documents, graphs, other models, technical specifications, XSD definitions, prototypal user interface samples; in other words everything we want to logically relate with each item.

A full example of the resulting work can be seen at <http://arianna.diviana.net>, following the link as shown in the following figure:

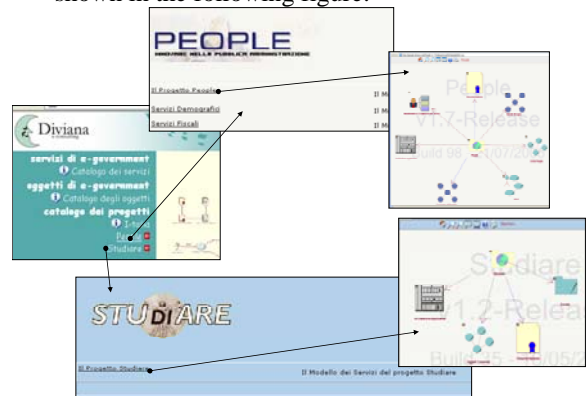


Figure 10. Internet path to real SVG model

8.2. Creating an XML Schema from the UML Model

The first logical step was to export the model in a way suitable to be used by developers. As we are not able to control the technology used for the different implementations, because it depends on a free choice made by the developing companies and/or its contractors,

we needed to define the *contracts* using a model both abstract and formal. So we decide to build an XML Schema (XSD)[6] generator.

The main question to face was how to partition the model: have we to create a very large XSD containing the description of the entire universe or, on the opposite, have we to create hundreds of small XSD containing the definition of a very small subset of objects? Both approaches have pros and cons. Building a very large XSD have a negative performance issue; moreover, it contains a lot of information not needed by a user approaching a specific service implementation. On the opposite, building a very large number of schema get the user confused, and create an unneeded complexity when using standard tools as XMLBeans to manage the generated schemata.

So we decide to use a different approach: we generate the XSD schema on a *per service* base, that is, each service has its own schema, containing only the elements needed by the specific service. The conceptual structure of the entire model can be described by the following figure:

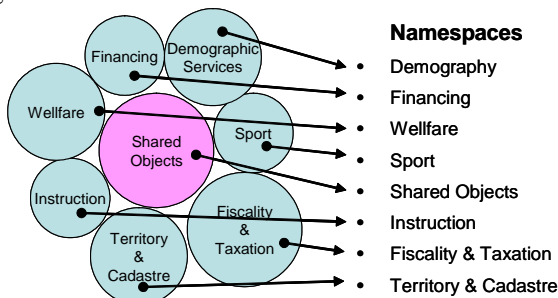


Figure 11. Model Conceptual Structure w/Namespaces

The central container contains all the shared objects, which are objects common to the different logical sub areas. The orbital containers contain all the objects private to a specific logical sub area. Each container has its own namespace.

As each service involves only a subset of objects from a subset of namespaces, the generated XSD is limited in scope and therefore in size.

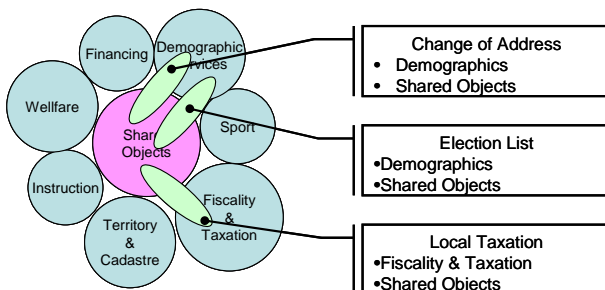


Figure 12. Service Oriented Model Partition

Of course, this approach can only be pursued having an automatic XSD generator, because the overall goal is to keep all the XSD consistent at every time with the UML model. Different XSD in different service definition can contain the same object specification. The logical union of all the Service XSD results in the entire modeled universe.

Using this approach produced a huge reduction of the XSD complexity, enhancing their usability. The XSD content is also described as types, avoid using instance elements: this approach lead to the build of an XML Type Library further reducing the XSD complexity and facilitating reuse.

8.3. XForms

Another critical point has been the lack of competences by the users to understand both UML and XSD models: as we were deeply concerned with reuse, both at a logical and physical level, the resulting model is composed by a large number of diagrams, each modeling elementary items. The user can navigate the model, but the information doesn't appear immediately in an easy to understand way. The XSD situation is worst: a lot of non tech users try to open XSD files using notepad-like tools, with dramatic results; we produce HTML documentation for the XSD, but it is still too fragmented (and huge) to be appreciate by these roles. They asked for a PowerPoint slide sequence describing the prototype of the user interface, but this was a costly approach, both by the development and by the maintenance point of view.

So we decide to try to generate an XForms[7] user interface, derived directly from the model. In the first version we decided to generate from the model '*as it is*', without any specific enhancement for this kind of task. The result has been appreciated by the users, enabled for a validation of the model content and its completeness in respect of a *business (non technical)* point of view.

But we decide to go further: we built a complete *Ontology Driven XForms generator*. The tool enable the LPA manager to discover the full complexity of the underlying ontology, thus using it both as a support for BPR activities and to deploy a fully functional sophisticated user interface.

8.3.1. The process to build an XForms

The process to build an XForms starts from a hierarchical view of the ontology content, as shown in the following figure:

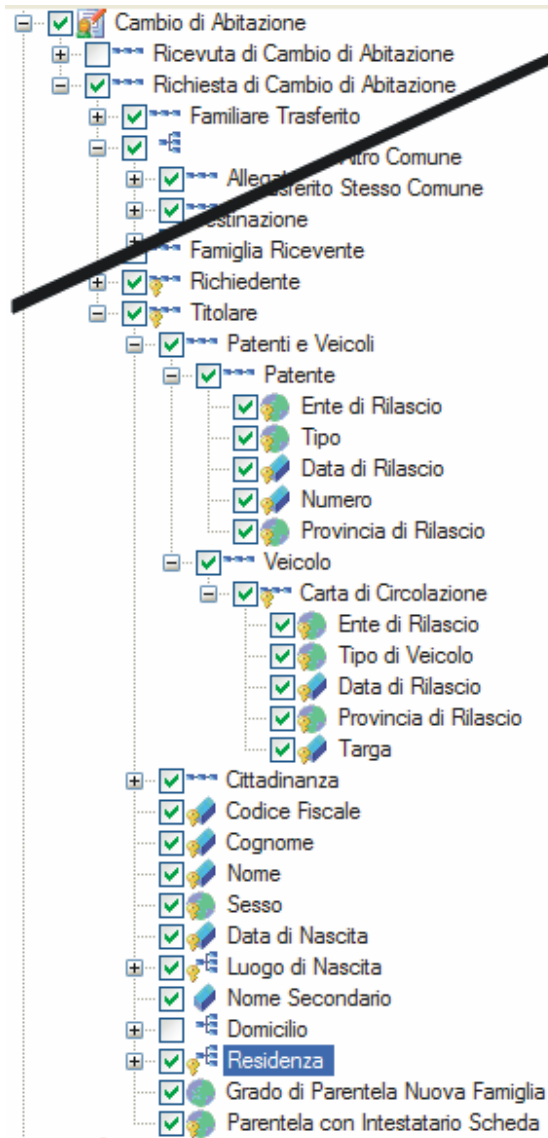



Figure 13. Basic hierarchical view of the ontology content

The different icons show different kind of knowledge items, and a small key icon  represents a mandatory knowledge element. At this point, the civil servant can select all the knowledge items he/she want to be asked to the citizen simply by checking them; the tool ensure the congruence with the model.

The final product of this activity is another hierarchical view, showing the knowledge by a logical user interface point of view, as described by the following figure:

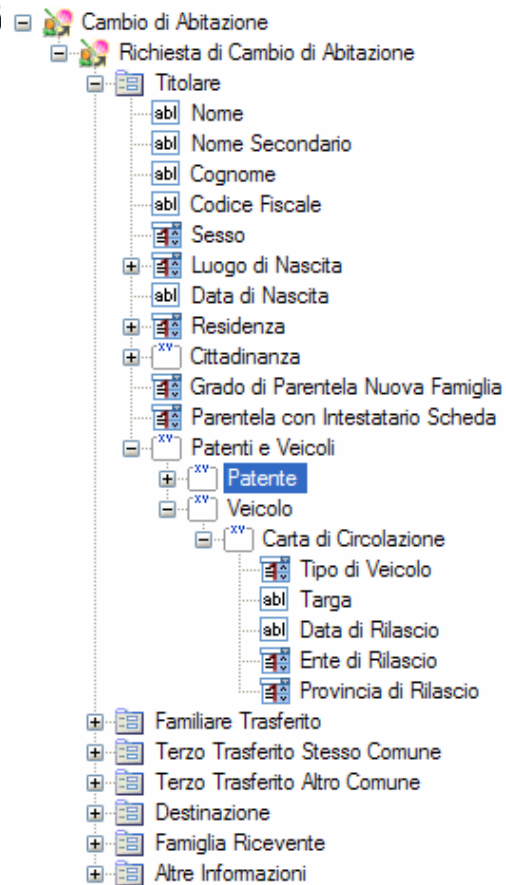
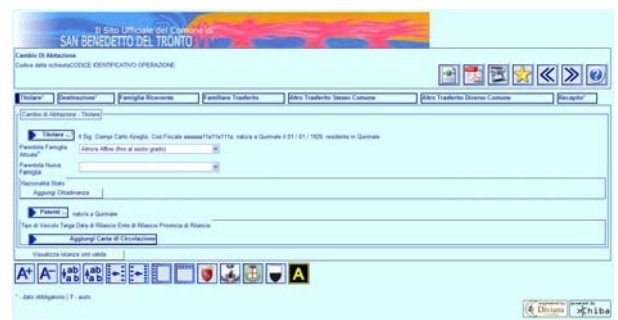


Figure 14. Hierarchical view of the user interface

The information can be logically grouped in pages and in groups and moved up and down. Different label can be specified for each UI element.

A successive step allows to translate the UI in different languages. The final result is represented by the following user interface, which can be styled using a CSS's.



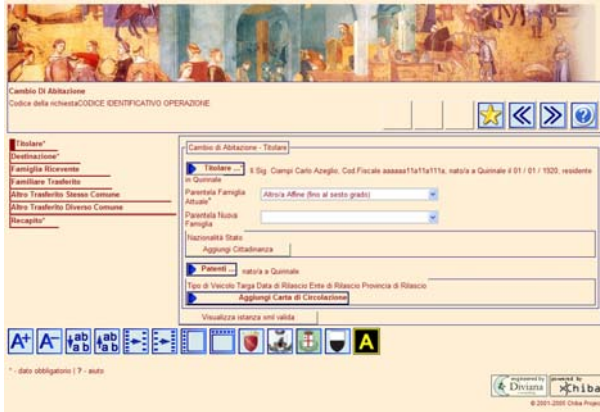


Figure 15. XForms UI for a real service: 2 samples

The list below reports the most interesting result derived by using this approach. The first three bullets represents more political issues, while the latter are more technical.

1. The LPA can customize the user interface without the need of a costly intervention by an ICT company, in a cost saving approach;
2. A normative change requires one time change to the model content and is then redistributed and reapplied with a minimum cost;
3. The time-to-market of a change is near to zero;
4. The model output is a Web Service implementation of an e-government request, as described by the model: the resulting XML is not affected by the UI customization.
5. The entire front end application derives directly from the model, thus ensuring a 100% compliance;
6. There can be multiple presentation layers per customer (standard, customizable, Accessibility compliant[8], ...);
7. The XForms component behavior is customized directly by the final customer: 1 model → 1 form → many behaviors;
8. The XForms model contains only data declarations: there are no formatting instructions (i.e. no DIV, no TAB, no HTML instructions): there is an effective separation between the data layer and the presentation layer;
9. The Presentation layer is easily styled using CSS;
10. There is a 1:1 mapping with the UML model and with the XSD: every change to the model automatically reflects in a new XForm;
11. The modeled choices are rendered using a ComboBox;
12. Complex information are collapsed giving the user a compact view expandable as needed;
13. The customer can easily customize the form, completely avoiding any optional element;

14. There is no need of an implementation using a specific language: the models directly maps on the XML instance;

9. Catalogue

The next logical step has been to bind altogether the single components we developed, giving users the ability to navigate all the information, easily switching from one point of view to another one. So we decide to develop an Internet application giving users a uniform way to access all the repository information.

The application is composed by two main modules or sub-catalogue: the Service Catalog and the Object Catalog: they describe the definitions of services and objects and are therefore *metadata catalogue*.

The **Service Catalog** allows users to discover e-government services in a set of given taxonomies. When a service is reached, the following info set is currently available:

- the service Use Case diagram, which can be used as a starting point to surf the entire UML model (see Figure 7);
- the XForms User interface prototype;
- the full XSD package for the service;
- the XSD documentation in HTML format;
- any documentation related with the service;
- the full list of the secondary services used to automatic pre-fill the user interface; each service is recursively described

The **Object Catalog** allows the users to access all the information related with every object used by the e-government services. When an object name is selected, the user can discriminate between homonymous objects if any, then access the full info set related with the selected object, containing:

- the basic Object Oriented information: class name, parent class if any, stereotype;
- all the documentation describing the object, usually a natural language document;
- the class diagram describing the object; for complex objects, the diagram can be used as an entry point to the UML model (see SVG);
- an Info Base, which collect all technical information related with the Object Oriented model. It includes:
 - the object attributes and relationships; for enumerated types, it shows the list of all the enumerated values;
 - the children classes, with an indication of where they are used;
 - a used-by list, with all the classes using the actual object;
 - a realized-by list, containing all the classes using a Realize relationship;

As the size of the knowledge base grown, it is critical to have tools helping to verify the completeness and the consistency of the content.

The interest for the achieved results is also leading to several new projects reusing and extending the approach and the knowledge base; here is a sample list:

- Integrating with an UDDI 3.0 catalog
The two approaches are complementary: the UDDI standard is a repository of technical information about a service instantiation, while Arianna is a metadata repository, containing information about service templates.
- Sharing the approach with Assinform, the Italian Association of Software Developers, a subsidiary of Confindustria, the leading organization representing manufacturing and service companies in Italy
Only 25 software development companies produce and sell the solution for more than 80% of the 8100 Italian Comuni. Sharing the approach enable the model to grow incorporating the knowledge from different experiences then converging on a shared model, giving the software companies the opportunity to face up with a decreasing number of different requirements.

Moreover, we are evaluating the opportunity to develop an evolved international version of the knowledge base, giving users other than Italian mother tongue the ability to access and use it. The 1.0 release of Arianna International is planned for 1Q2006.

13. Final remarks

This project is part of a dream, the dream to actively participate in the improving process of the Public Administration in our country. Improving this process is an IT problem only in a minimum part. It is mainly an organizational problem, with huge impacts on the overall internal organization.

Every e-government project is driven by visibility rules, i.e. the political choices are often tactical, related with events, elections, meetings.

Until now, every step have been made available by the imagination of few people, and by the visionary and far-seeing LPA entities who believed in the approach and sustained it investing their limited funds. We forebode an augmented interest by the political power and an official endorsement through the constitution of a consortium to maintain and enhance the actual knowledge base and to extend it to new area still not covered (Health and Labour just to cite).

Moreover, this systematic approach can be easily used to disseminate the experience in other countries: it can be a way to export best practices toward third world countries giving them a model to use as a starting point to

customize their own needs reducing the deployment time of new e-government solutions.

14. Acknowledgement

A special acknowledge from the authors to their daughter, Chiara, for her patience bearing the time stolen her and the never ending discussions.

15. References

- [1] <http://www.ietf.org>
- [2] <http://www.w3.org>
- [3] <http://www.uml.org>
- [4] <http://www.omg.org>
- [5] <http://www.w3.org/Graphics/SVG>
- [6] <http://www.w3.org/XML/Schema>
- [7] <http://www.w3.org/MarkUp/Forms>
- [8] <http://www.w3.org/wai>

16. Full Size images

