An XML model for electronic services

C. Vassilakis[1], G. Lepouras[1], C. Halatsis[1], T. Pariente Lobo[2]
[1]e-Gov Lab, Department of Informatics, University of Athens, 15784, Greece
[2]Indra Sistemas S.A., Velásquez 132, 28006 Madrid, Spain
costas@e-gov.gr, gl@e-gov.gr, halatsis@di.uoa.gr, tpariente@indra.es

Abstract: With the need for electronic services to be developed and deployed more and more rapidly, it is imperative that concrete models of electronic services are developed, to facilitate systematic work of electronic service stakeholders, concrete semantics and coherent representations across services developed within an organisation. Using the XML language to develop such a model, offers a number of additional advantages, such as rich semantics, facilitation of data interchange, extensibility, high abstraction levels and possibility for mechanical processing. In this paper we present the design aspects of an XML model for electronic services, which has been used for building a repository of interlinked elements representing e-services. A web-based interface for the management of this repository and a tool for automatically compiling e-service descriptions into executable images have been developed alongside. The model has been evaluated by a mixture of electronic stakeholders, and the results of this evaluation are also presented.

Keywords: e-government, electronic service, XML model

1 Introduction

Electronic Government, driven by an ever increasing and pervasive use of information and communication technologies, is more and more affecting the public sector. (European Commission, 1999). At both national and European level, strong will has been declared for promoting electronic governance, mainly expressed through specific projects and initiatives for developing and promoting electronic services (European Commission, 2004; Italian Ministry of Innovations and Technology, 2004; US Government, 2002), or supporting frameworks (UK online, 2004a, 2004b) since the benefits from this area have become apparent to both service providers (administrations) and service users (businesses and citizens) (Top of the web, 2003). Insofar, however, the electronic service lifecycle does not employ any concrete, formal representation model; rather, *ad-hoc* models are used, either specifically drawn for the modelled electronic service or drafted by developers or integrators as templates.

Using a formal model for describing an electronic service is considered necessary, since this approach enables stakeholders (i.e. roles participating in the development of electronic services, such as managers, domain experts, technical staff etc) to work systematically on the task of recording all important aspects of the electronic service, without the risk of omitting key elements or essential element attributes. Semantics for each element and element attribute are also standardised, promoting the clear separation of the different concepts involved in e-service modelling and facilitating common understanding of e-service descriptions among e-service development stakeholders. Automated checks for description completeness and integrity can also be conducted against descriptions following a formal model.

Using an XML-based formal model for electronic services offers a number of additional advantages:

- XML is an expressively rich language, allowing for any structural or semantic concept to be appropriately modelled. Thus every aspect of the electronic service can be represented in the e-service XML model. Note that "aspects of an electronic service" may include forms and fields that comprise a service, validation checks expressed in a high-level specification, documentation that has emerged from the service analysis, related legislation, help that should be available to end-users, specification of deadlines, definition of statistics that need to be gathered etc. Interrelations between service elements or between services can also be modelled.
- XML is considered nowadays the standard for data interchange, thus an XML model for electronic services facilitates the communication and sharing of service portions across either divisions of the same organisation or between different organisations. This may be employed either for individual service elements (e.g. a law or directive exported by a legislative body may be directly imported in the description of a relevant service; a validation rule regarding the VAT number format may be shared by all organisations that involve VAT numbers in their services), or for service portions (e.g. the part of the service that models the citizen's personal data may be shared between divisions or organisations delivering electronic services).
- Schemata are easily expandable to accommodate new features by simply adding the new elements. Language features may be exploited to cater for backwards compatibility – e.g. by including a `minoccurs="0"` tag to a newly introduced element designates it as *optional*, ensuring schema-level compatibility for all existing instances of the particular schema.
- Since all aspects of the service that are considered important have been recorded and stored in a concrete and unambiguous format, these descriptions may be mechanically parsed and executable service images may be automatically generated. Such an executable image will target a specific deployment platform, e.g. a JSP container, a web server with a ColdFusion or a PHP engine installed etc. Different generators may be employed to produce code for different deployment platforms, which may potentially address different client device technologies (e.g. PC-based web browsers, WAP clients, iMode clients etc). Besides generating the executable service image, the generator may also automatically produce a storage schema for documents submitted through the specific service, since all data elements of the service are known, along with any relevant properties (e.g. mandatory vs. optional, single-occurrence vs. multiple occurrences, type of data etc).

The rest of the paper is organised as follows: section 2 surveys related work regarding models of electronic services. Section 3 summarises the key elements of electronic services and their interrelations; section 4 presents the important design aspects of the XML model and provides hints on how model elements can be mechanically processed and transformed into executable service schemata, while section 5 presents an evaluation of the XML model. Finally, section 6 concludes and outlines future work.

2 Related work

Although electronic services have received increased attention in the past few years, mainly in the contexts of e-government, e-commerce and B2B services, related frameworks and standards have not yet been developed accordingly. In (Piccinelli et al. 2003) an architecture for electronic service management systems is presented, which mainly analyses the business processes associated with electronic services in the context of organisations. The paper also proposes a meta-model covering concepts related to electronic services, such as business assets, workflow processes and business roles; formal semantics and a UML mapping for these concepts are also given. In (Vassilakis, 2003) an approach for enabling a holistic management of the electronic service lifecycle is described. This approach employs modelling and representation in high levels of abstraction and identifies business roles that are involved in each stage of the development.

Regarding the use of XML in electronic services development and delivery, insofar this has been limited to information interchange and modeling and filing of documents submitted through electronic services. A noteworthy activity is carried out by UK GovTalk for the development of standard XML schemata to be used in electronic services (UKGovTalk, 2003)], covering the issue of what data should be collected by specific services and how these should be structured. In (Juna Project, 2001) the use of XML for standardisation of interfaces is encouraged. ebXML is also a major development for enabling XML to be utilized in a consistent manner for the exchange of all electronic business data (ebXML committee, 2003).

BRML is another XML-related technology that can be used in the context of electronic services. BRML provides a rule-based framework for developing rule-based applications with major emphasis on maximum separation of business logic and data, conflict handling, and interoperability of rules (BRML committee, 2003). BRML however is a generic framework for the development of any application, not just electronic services, thus additional effort is required by any organisation to tailor the framework to its specific, e-service oriented needs. The event-condition-action language for XML proposed in (Bailey et al., 2002) can be utilised to model workflow aspects related to electronic services or validation rules, but is again too generic to be used directly.

Finally, XML has been used for the development of personalised e-shopping solutions, presenting adaptive menus and tailored pages (Weske, Schneider, 2002) and for moving relatively small catalogues online (Sims, Tikekar, 2001).

3 Key elements of electronic services

An electronic service is, in general, a computerised counterpart of a form submission business process in the paper-based world. In this context, the electronic service user is presented with a *set of forms* to fill in (lengthy documents are subdivided into multiple forms or form pages). Forms may be structured into *areas*, with each area containing some conceptually interrelated *fields*; for example a form area may be dedicated to collecting the citizen's personal details. Form fields are the individual elements that citizens need to fill; this is mainly performed by either writing some text within the field area (e.g. writing 10,000.00 in the area corresponding to the *Income* field) or by checking one of the available field options (e.g. *yes* or *no* for the *Are you married?* field). Some fields may have *repeating occurrences*; for instance when an enterprise declares the vehicles owned, the inputs corresponding to *Vehicle type, C.C.* and *Date of Purchase* have to occur multiple times -one for each owned vehicle- as

illustrated in Figure 1. In an electronic version, some fields may be automatically calculated (e.g. the sum of values in a column) or pre-populated (for example, the personal details of the service user that has been authenticated via a login procedure); in both cases, the values of these fields cannot be directly modified. Usually the forms contain also *instructions* for the citizens, to guide them through the process of filling in the form. Instructions are particularly useful in cases of complex forms, containing fields whose semantics are not obvious.

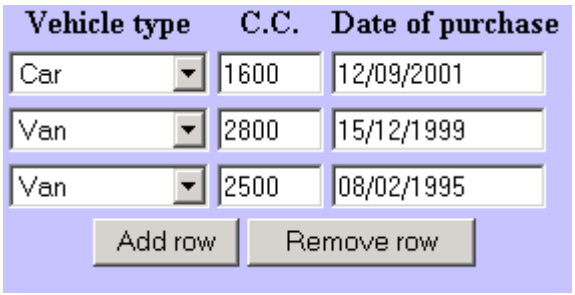

Figure 1 – Input fields with multiple occurrences

Besides the above listed elements, which are addressed to the citizen that will fill in the form, a form submission business process is associated with a number of elements that are addressed to front-desk and back-office workers. Firstly, a form is defined on the basis of some *legislation*, which describes the form purpose, contents, submission periods etc. The legislation also usually defines some *validation criteria*, which pertain to the values that are filled in by the citizens and must be met by every submitted form. Examples of such validation criteria are "The SSN is mandatory", "The value in the income field should be a positive number", "If the citizen declares to be not married, the *Spouse surname* field should be left blank" and *"Declared pre-paid taxes may not exceed the 25% of the declared income"*. In a paper-based environment, most of the validation criteria are checked by the front-desk staff that collects the form, whereas some other validation criteria (mainly those which either require cross-checking with other documents and those that are particularly time-consuming) are checked by the back-office workers.

When forms are submitted by citizens they need to be filed for reference and further processing. In the electronic paradigm, form filing corresponds to storing the electronic document in a database. Finally, from a form submission process certain *statistics* may be computed, which may be related to the service as a whole (e.g. total number of form submissions; average time to complete the form) or individual service elements (for instance, average value of inputs to a specific field; number of times that a validation check has failed; number of times a particular help text was retrieved).

4 The e-service XML model

A model for electronic services, besides being able to represent all key elements presented in the previous section, is strongly desirable to have a number of properties to enhance its functionality and usefulness:

- *Express concepts in the highest possible level of abstraction*. Using high levels of abstraction enables the immediate possessors of the knowledge to input this knowledge to the system without the intermediation of analysts. This feature supports the task of turning tacit knowledge into explicit, which is a valuable asset for the organisation. Lower-level information might be

provided in specific cases, however, to facilitate the task of mechanical processing.

- *Use a minimal set of "base concepts".* The model should employ a small number of orthogonal (non-overlapping) base concepts to formulate complete descriptions of e-services. This approach reduces the time needed by model users to learn and use the model and is in line with the minimality principle of conceptual modelling (Bergamaschi, Sartori, 2002).
- *Support collaborative work.* An electronic service is a complex artefact, which is jointly developed by stakeholders of different skills and backgrounds (e.g. domain experts, managers, IT staff etc), with each stakeholder contributing a set of elements to the overall description. The model should, to the maximum extent possible, facilitate the separation of the activities that need to be performed by each stakeholder and avoid introducing unnecessary restrictions in the order that activities should be performed by various stakeholders.
- *Facilitate linking between elements.* Various elements comprising an electronic service are interrelated - for example, an input element may be related to the piece of legislation that defines the electronic service contents, to the examples that are presented to the user on how the element is filled in, to the form it lies on, to the validation checks that verify that the input value provided is conferment to the instructions etc. The model should be able to represent such linkages, to allow developers to navigate along related objects and maintainers to easily locate elements that are affected due to some change (e.g. a change in the legislation may affect all linked elements).
- *Enable the execution of completeness checks.* Since an electronic service comprises of numerous interrelated elements that should be defined and elaborated by various actors, it is necessary for the model to make possible checks that identify if any required (or desired) elements are missing. Once the missing elements have been pinpointed, the respective stakeholders may be notified about their outstanding tasks.
- *Allow mechanical processing to produce executable service images.* To the extent that sufficient information has been included in the model, each individual electronic service description can be processed and an executable image for it can be produced for a particular execution environment (e.g. WAR files for JSP containers (Apache Group, 2003; Zuffoletto, 2002) ColdFusion scripts for the ColdFusion server engine (Hewit, 2001); PHP scripts for PHP-enabled servers (Lerdorf, Tatroe, 2002) etc) by employing generative programming techniques (Czarnecki, Eisenecker, 2000). This task is possible due to the fact that within electronic services the possible actions for a user are limited (complete a new form; edit an existing form; delete a previously submitted form), thus suitable code fragments may be generated for implementing these functionalities.

In order to support the dimensions presented above, a number of design-level decisions were made for the model[1]. Firstly, besides the object types representing electronic services, forms, element groups and individual elements (which are indispensable parts of electronic services) only one additional top-level concept was added to the model, namely the *Knowledge Unit*. The concept of a knowledge unit

[1] The full XML model is not included in this paper for brevity reasons. The full XML model can be found in (SmartGov Consortium, 2002)

encapsulates any information that may be associated with an electronic service or any portion of it, including related legislation, documentation, design rationale, help text or examples for the end users etc. A special tag indicates the purpose of the knowledge unit, providing thus means to formulate different subcategories, depending on the knowledge unit intended usage. Knowledge units may be interlinked with any top-level concept (electronic service, form, element group and individual element or another knowledge unit) in a many-to-many fashion, forming thus a semantically rich network of information, which can be traversed in any direction; starting thus from a specific point in this network, all related nodes may be easily reached. Knowledge units may be also linked to selected elements that do not reside on the top-level of the model, notably validation checks, which need to be documented, exemplified and associated with related legislation. In addition to links from/to knowledge units, the model supports links between service elements; for instance an electronic service is linked to the forms it comprises of and these forms are in turn linked to the individual elements or element groups that appear on each form. These links can be traversed as well, allowing navigation through the electronic service hierarchy.

Regarding the abstraction level of the model concepts, the elements comprising each concept have been chosen so as to be direct or close counterparts to notions that are used by electronic service users, in order to enable stakeholders to work directly with the model (through an appropriate front-end). For instance, an electronic service is described (Figure 2) through an identifier (ESId), a short and long human-readable description (serviceName and description), the form sets it includes (includedFormSets – a "form set" is a set of forms targeted for a specific service access environment e.g. web browser, WAP client, I-mode client etc) and the knowledge units to which it is linked (linkedKUNodes). Additionally, the validation rules that apply to documents that are submitted through the service are defined (serviceValidationRule) and the authentication requirements for this service are specified, by selecting among a library of methods (user name and password, smart cards, unauthenticated etc). Finally, the allowed operations for the service are designated (whether the users can submit, modify or delete documents), a date is set after which the service becomes inoperative (deadline), and the statistics that need to be collected for the service are listed (ESStatistics). The *lifeCycle* element is system-maintained and records the modification dates and the users that have performed the modifications.

```
<xs:element name="ES">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ESId" type="xs:string"/>
            <xs:element name="serviceName" type="multilingualText" maxOccurs="unbounded"/>
            <xs:element name="description" type="multilingualText" maxOccurs="unbounded"/>
            <xs:element name="includedFormSets" type="formSet" minOccurs="0"
                maxOccurs="unbounded"/>
            <xs:element name="linkedKUNode" type="xs:string" minOccurs="0"
                maxOccurs="unbounded"/>
            <xs:element name="serviceValidationRule" type="validationMethod"
                minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="authenticationRequirements" type="xs:string"/>
            <xs:element name="allowSave" type="xs:boolean"/>
            <xs:element name="allowEdit" type="xs:boolean"/>
            <xs:element name="allowDelete" type="xs:boolean"/>
            <xs:element name="deadline" type="xs:date"/>
            <xs:element name="lifeCycle" type="lifeCycleType"/>
            <xs:element name="serviceStatistics" type="ESStatistics"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figure 2 – XML schema for the "electronic service" model concept

Special care has been taken regarding the abstraction level of validation rules, which are "traditionally" considered a task for IT staff. By analysing, however, a number of electronic services[2] it was found that the 80% approximately of the required validation checks can be modelled after the following prototypes:

1. $L1 \leq A \leq L2$, where A is a document field and $L1$ and $L2$ constant values. This prototype models cases where the value of a field should fall within a given range, e.g. *the number of days worked in a year may range from 0 to 300.*
2. *A Requires B*. If a value is provided for field *A* then a value must be provided for field *B*. For instance, if the *Car Owner* field is filled in, the field *Car licence plate number* should be filled in as well.
3. *A Precludes B*. If a value is provided for field *A* then no value should be provided for field *B*. For example, if the user fills in the field *losses from trade business*, the field *profits from trade business* should be left blank, since it is impossible to have simultaneously profits and losses from the same activity.
4. $A\ cmp\ B\ *\ c$, where *A* and *B* are document fields, *cmp* is a relational operator ($=, \neq, >, \geq, <, \leq$) and *c* is a constant value. This prototype enables the specification of arithmetic constraints on the values of form fields, such as *profits from trade business cannot exceed total profits* (in this example c is equal to one) or *cargo insurance fees should be less than the 2% of the declared value of the transported goods*.

Validation checks modelled after the above prototypes are coupled with a *severity level* (either *error*, or *warning*) and with a *message*, which is displayed to the service user when the check fails.

Validation checks at this abstraction level can be directly expressed by domain experts, through an appropriate user interface. There is still, however, a 20% of validation checks which are too complex to be modelled using these prototypes. To this end, the ability to express a validation check in any general-purpose programming language has been provided in the electronic service schema. Naturally, the

[2] Seven electronic services were analysed to obtain the listed results. Two of them were simple, one-form services with few fields, three services were of medium complexity (one-two pages with 40-60 fields) and two services were highly complex including more than 300 fields spread along 3-4 pages.

programming language that will be selected should match the architecture of the service execution environment (e.g. Java for JSP containers, PHP for PHP-enabled servers etc). This code may be either typed-in directly or a pointer to the file may be provided. The XML schema used for modelling validation checks allows for specification of either validation checks following the prototypes listed above, or for validation checks directly expressed in a programming language, as shown in Figure 3. In Figure 3, the top-level entity is the *validationRule* complex type, which is subsequently refined into the various sub-cases by means of the *choice* XML schema construct.

```
<xs:complexType name="validationRule">
    <xs:choice>
        <xs:element name="compactRule" type="compactRule"/>
        <xs:element name=" nativeCodeFragment" type=" nativeCodeFragment"/>
    </xs:choice>
    <xs:element name="LinkedKUs" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
</xs:complexType>

<xs:complexType name="nativeCodeFragment">
    <xs:sequence>
        <xs:element name="langId" type="xs:string"/>
        <xs:choice>
            <xs:element name="codeText" type="xs:string"/>
            <xs:element name="fileSpec" type="xs:anyURI"/>
        </xs:choice>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="compactSmartGovLang">
    <xs:sequence>
        <xs:choice>
            <xs:element name="betweenCheck" type="SGbetweenCheck"/>
            <xs:element name="requiresCheck" type="SGrequiresCheck"/>
            <xs:element name="precludesCheck" type="SGprecludesCheck"/>
            <xs:element name="relationCheck" type="SGrelationCheck"/>
        </xs:choice>
        <xs:element name="validationMessage" type="multilingualText" minOccurs="0"
            maxOccurs="unbounded"/>
        <xs:element name="severity">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="warning"/>
                    <xs:enumeration value="error"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

Figure 3 – The XML schema for validation checks

In the area of collaborative work, the model for an electronic service includes numerous items, which can be independently developed by different stakeholders. For example, the visual layout of a form, essential for an electronic service, can be independently developed from the knowledge units required for the service or the validation checks that will check the values input to the form fields. These items may then be linked together by including appropriate references in the XML schema instances. Furthermore, an XML model inherently subdivides each object type into its elements, which are analogous to *frame slots* (Krishnamoorthy, Rajeev, 1996). These slots may be independently filled in by different stakeholders; for instance when defining a new electronic service, the domain experts may develop the associated knowledge units, the managers may define the statistics needed for the evaluation of

the service and the domain experts, jointly with the IT staff, can define the validation checks. Concurrent updates to the same object should however be protected using appropriate concurrency control schemes such as (Hadzilacos, Hadzilacos, 1991; Malta, Martinez, 1993).

Having a structured representation of the electronic service model, performing completeness checks is quite straightforward, provided that elements have been appropriately characterised as *compulsory* or *optional*. Note that this can not always be derived from the XML schema by exploiting the *minOccurs* and *maxOccurs* tags, since these have been chosen so as to facilitate the development process. For example, the *includedFormSets* element in the electronic service is tagged with *minOccurs* set to zero, while a "complete" electronic service definition should have at least one form set. If, however, the *minOccurs* tag for the aforementioned element was set to one, no electronic service description could be created until one form set for it would be ready, forcing for an "unnatural" development path. Moreover, the elements for an electronic service that are considered as compulsory may differ from one organisation to another: for example, some organisation may require that every element be linked to the related legislation, designating thus KUs as mandatory, while this requirement may not hold for another organisation. The development of a tool that traverses the electronic service model and finds elements that are required but not yet filled in is however an easy task. Once the missing elements have been identified, notifications to the respective stakeholders may me sent, to inform them about the pending tasks.

When all required elements of an electronic service have been provided by the respective stakeholders, it is possible to exploit the information gathered in the model to automatically generate executable service images. Service generation can be performed through the following procedure:

1. for each form object that the service contains, a separate file is generated incorporating:
    a. the elements that belong to the form.
    b. any validation checks that need to be conducted for the elements belonging to the form.
    c. navigational controls, allowing the user to move to the previous/next form of the service. In the last form, the "next" navigation control is replaced with a "finish" control, which invokes a separate operation arranging for saving the values provided by the service user to a database.

    The generated file contains code suitable for the execution environment (JSP files, PHP scripts etc). Besides the automatically generated code that handles interception and validation of values provided by the user, this file contains the visual part of the form, extracted from the description of the respective "Form" object in the XML repository. Knowledge units that are linked with the form, or form elements, and are tagged as "help for end users" are made accessible through appropriate hyperlinks on the generated form.
2. for each input element within the form, a respective control is generated and incorporated. The generated control couples semantic information from the element description (e.g. maximum length, description, data type etc) with visual information for the same input element, extracted from the form layout (e.g. font family, size and colour).
3. for each validation check that has been defined, the appropriate code is either generated (if the validation check is modelled after the prototypes described above) or simply extracted, if the validation check has been specified in the

execution environment's language. Generating code for the validation check prototypes is straightforward, since the semantics expressed are simple; for example, a validation check $L1 \leq A \leq L2$ with a severity characterisation set to "error" and an error message set to "Error Message" is translated to the code

```
if ((A < L1) || (A > L2))
      errorMessage("Error message");
```

where *errorMessage* is a library procedure suitable for the execution environment that arranges for emitting the message to the user and inhibiting further user operations with the service, until the error is corrected. The code conducting the validation check, generated or extracted, is bundled in the file pertaining to the form that the input element appears in. If the validation check includes more than one element appearing on different forms, then the code is bundled in the file associated with the "finish" control of the last form, since at this stage all necessary values will be available.

4. storing of values entered by the user into a database is handled by code bundled in the file associated with the "finish" control of the last form. This code can be automatically generated when the executable image of the service is created, since all input elements of the service are known, together with any related details (e.g. data type and maximum length for each input element). For mapping to relational databases, the procedure starts off with an empty table schema and for each input element an extra field is added to the table. If fields with multiple occurrences are used (see Figure 1), then a new table is introduced for each such group of fields. Database restrictions regarding maximum fields in a row or maximum row size in bytes (e.g. Microsoft Corporation, 2001) have to be addressed in this mapping by dividing the single table schema into smaller table schemata meeting the restrictions. Storing user documents to object databases or XML databases is more straightforward, since multiple occurrences are allowed (*collections* in object databases and elements with *MaxOccurs* greater than one in XML databases).

Once the executable service image has been put together, it can be deployed to the execution environment. The deployment technique depends on the execution environment; e.g. in a PHP-enabled server simple file copying to the web server's document area usually suffices, whilefor deploying a service through the Tomcat JSP container, the Tomcat deployer (Apache Group, 2003) has to be used.

5 XML model evaluation

The XML model has been developed in the context of the SmartGov IST project (Georgiadis et al. 2002; SmartGov Consortium, 2004) and has been used for the development of a number of electronic services for the public sector. Electronic service stakeholders were able to create and manage the elements of the electronic services through a web-based front-end, while an engine for automatic e-service generation was also built, creating executable service images for the Tomcat JSP container. The XML documents describing the elements of the electronic services were stored in an XML repository.

Although electronic service stakeholders did not directly work with the XML documents, but accessed the content through the web-based development environment, the evaluation conducted through questionnaires completed by electronic service stakeholders after a training period contained items that would help assessing whether the XML model has met the design goals described in sections 1 and 4. Eighteen questionnaires were gathered and processed; the stakeholder sample

was a mixture of managers (2), domain experts (7), IT staff (5) and help-desk workers involved in electronic service delivery (4). The degree of computer literacy within the user group ranged from expert (6) to naïve (4), with the remaining 8 falling between these two extremes. An excerpt of the questionnaire is shown in Figure 4. Figure 5 summarises the results from questionnaire processing, including only the questions that are directly or indirectly relevant to the XML model. In all questions the rating *1* corresponds to "*Strongly disagree"* while the rating *9* corresponds to *"Strongly agree"*.

1. I could understand all the base concepts used by the system

   Strongly disagree 1 2 3 4 5 6 7 8 9 Strongly agree N/A

2. All the key concepts I needed to model an electronic service were present

   Strongly disagree 1 2 3 4 5 6 7 8 9 Strongly agree N/A

3. Each concept was described in the right level of detail

   Strongly disagree 1 2 3 4 5 6 7 8 9 Strongly agree N/A

4. Starting from a specific element, I could locate all related information

   Strongly disagree 1 2 3 4 5 6 7 8 9 Strongly agree N/A

Figure 4 – Excerpt from the evaluation questionnaire

Question	Mean	Std. dev
I could understand all the concepts used by the system	7,8	0,91
All the key concepts I needed to model an electronic service were present	7,5	0,67
When I needed to create a new item, I always knew which concept to use	8,3	0,47
Each concept was described in the right level of detail	7,2	1,02
Elements were missing from some concepts	2,8	0,98
I shouldn't be shown some elements that are not related to my work	6,7	0,94
Starting from a specific element, I could locate all related information	7,6	0,72
I could always link two objects that I considered to be related	7,9	0,60
I could easily locate elements that were missing from a service description	6,4	0,85
I could easily create validation checks	7,3	0,95
I would like the validation checks language to be more expressive	3,9	0,78
I could easily incorporate into the system information from other sources (word processor files, databases etc)	7,2	1,28

Figure 5 – Results from questionnaire processing

The evaluation results showed that users could easily understand all the concepts presented to them by the front-end, which is attributed to the high level of abstraction used in concept modelling and the small number of base concepts used, which enabled the stakeholders to quickly obtain a holistic view of the platform scope and capabilities. Users also noted that no key elements were missing from the model, which indicates that the minimality goal has been attained without sacrificing completeness and expressiveness. Some users pointed out few useful attributes that were initially missing from the model (e.g. the preferred display size of an input element), which were subsequently added to it by including the appropriate element tags in the electronic document. When the model describing an element type was

extended, either the *minOccurs="0"* attribute was used for the new element to ensure conformance of the existing XML documents describing already created elements of this type, or an upgrade script was run which extracted documents from the repository, added the required element with a default value and stored back the updated version. The latter technique was used when the new elements were considered to be mandatory and thus the *minOccurs="0"* attribute could not be used.
Linking between elements was also considered adequate, since users stated that they could always easily locate information related to the items they examined. Although no separate tool for completeness check was implemented, users found adequate the feature of the e-service generation engine to generate error messages for missing elements, stating however that they would prefer a specific tool that would also pinpoint *desired* elements that were missing (the e-service generation engine reported only the missing *compulsory* elements) and would actively notify the stakeholders responsible for providing these elements. The separation of responsibilities was also well rated, with the comment however that it would be preferable for the front-end to be more "personalised", in the sense that information not directly related to the current user would preferably be hidden, rather than be displayed and having to be ignored.
The XML model has also enabled the development of a number of peripheral tools that enhanced the overall platform functionality. Firstly, an XML document import and export facility was developed, which facilitated document exchange with other information systems. This feature was mainly used for knowledge units, where documents were extracted from legal databases, appropriately formatted and then imported into the XML repository. Linking of such imported documents with other items (other knowledge units, forms or form fields) was performed through the front end, after the import phase. Knowledge units containing instructions and examples were also exported from the repository and imported into word processor files to formulate documents with instructions to end-users.
Another tool that was developed automatically generated HTML pages for the forms modelled within the repository. This was possible since all information regarding form fields and their descriptions and semantics were present in the XML repository. The layout of these pages was admittedly basic, they could serve however as a template to be elaborated on using professional HTML page editing tools, such as DreamWeaver™ or GoLive™. For DreamWeaver™ in particular, an extra tool was developed which enriched the built-in tag set with tags corresponding to the items modelled in the XML repository. These tags could then be used by web page designers to place e-service elements on the HTML page (field descriptions, help texts etc). This tool effectively extracted certain elements of the XML documents within the repository and reformatted them into new XML documents, as required by the DreamWeaver™ extension API (Macromedia Inc., 2002).

6 Conclusions

In this paper we have presented the key design aspects of an XML model for electronic services. The XML model has been used for the development of electronic services, in conjunction with a web-based front end and an engine for automatic generation of executable electronic service images. The XML model has also been evaluated, both in terms of (indirect) user satisfaction and in terms of ability to interface with other systems and develop value-added tools. Future work will focus on the incorporation of workflow aspects in the XML model, to handle the intra-organisation workflow of documents submitted through the electronic services, and

the reverse engineering of existing electronic services into the XML model, to cater for the consolidation of all knowledge related to electronic services within the organisation into a single, high-level, reusable repository.

7 References

Apache Group, 2003. The Tomcat 5 Servlet/JSP container: Deployer How To. Available at http://jakarta.apache.org/tomcat/tomcat-5.0-doc/depoyer-howto.html

Apache Group, 2004. Jakarta Project. Accessible at http://jakarta.apache.org/

Bailey, J., Poulovassilis, A., Wood, P.T., 2002. An Event-Condition-Action Language for XML. Proceedings of the WWW 2002 Conference, Hawaii, May 2002, pp 486-495

Bergamaschi, S., Sartori, C. On taxonomic reasoning in conceptual design. ACM Transactions on Database Systems, Vol. 17, Issue 3, 1992, pp. 385 – 422.

BRML committee, 2003. Business Rules Markup Language (BRML). Accessible at http://xml.coverpages.org/brml.html

Czarnecki, K., Eisenecker, U., 2000. Generative Programming: Methods, Tools, and Applications. Addison-Wesley Professional, 2000, ISBN: 0201309777

ebXML committee, 2003. ebXML technical specifications. Accessible at http://www.ebxml.org/specs/

European Commission, 1999. Public Sector Information: A Key Resource for Europe, Green paper on Public Sector Information in the Information Society. http://europa.eu.int/ISPO/docs/policy/docs/COM(98)585/

European Commission, 2004. eEurope 2005 Action Plan. http://europa.eu.int/information_society/eeurope/index_en.htm

Georgiadis, P., Lepouras, G., Vassilakis, C., Boukis, G., Tambouris, T., Gorilas, S., Davenport, E., Macintosh, A., Fraser J., Lochhead, D., 2002. A Governmental Knowledge-based Platform for Public Sector Online Services. Proceedings of the 1st International Conference on Electronic Government-EGOV 2002, pp. 362-369.

Hadzilacos, T., Hadzilacos, V., 1991. Transaction Synchronization in Object Bases. Journal of Computer and System Sciences, 43(1):pp. 2-24, 1991.

Hewitt, E. 2001. Core ColdFusion 5. Prentice Hall, 2001, ISBN: 0130660612

Italian Ministry of Innovations and Technology, 2004. E-Government for development.
http://www.innovazione.gov.it/ita/egov_sviluppo/introduzione/egov1.shtml

Juna Project, 2001. Development Project for e-Government. Accessible at
http://www.intermin.fi/intermin/images.nsf/files/E54458C833DF46B4C2256BCF00259A99/$file/XML_juna.pdf

Krishnamoorthy, C.S., Rajeev, S., 1996. Artificial Intelligence and Expert Systems for Engineers. CRC Press, 1996, ISBN: 0849391253

Lerdorf, R., Tatroe, K. 2002. Programming PHP. O'Reilly & Associates, 2002, ISBN: 1565926102

Macromedia Inc., 2002. Extending Dreamweaver MX. June 2002

Malta, C., Martinez, J., 1993. Automating Fine Concurrency Control in Object-Oriented Databases. Proceedings of the International Conference on Data Engineering, pp. 253-260, 1993.

Microsoft Corporation, 2001. Frequently asked questions - SQL Server 2000. Available at http://support.microsoft.com:80/support/kb/articles/Q260/4/18.asp

Piccinelli, G., Emmerich, W., Williams, S., Stearns, M.. A Model-Driven Architecture for Electronic Service Management Systems. Proc. of the 1st Int. Conference on Service-Oriented Computing, Trento, Italy, LNCS 2910, pp. 241-255, 2003.

Sims, J., Tikekar, R. An XML model for small business e-commerce. Journal of Computing Sciences in Colleges, Volume 16, Issue 2 (January 2001), pp. 21-28
SmartGov Consortium, 2003. Deliverable D51-61: Low-level Specifications of SmartGov Services and Applications and the Knowledge-Based Core Platform. Available through http://www.smartgov-project.org/
SmartGov Consortium, 2004. SmartGov web site", http://www.smartgov-project.org
Top of The Web, 2003. Survey on quality and usage of public e-services. http://www.topoftheweb.net/docs/Final_report_2003_quality_and_usage.pdf
UK GovTalk, 2003. XML Schema Library. Accessible at http://www.govtalk.gov.uk/schemasstandards/schemalibrary.asp
UK online, 2004a. E-Government Interoperability Framework. http://www.govtalk.gov.uk/schemasstandards/egif.asp
UK online, 2004b. The e-Government Metadata Standard. http://www.govtalk.gov.uk/schemasstandards/metadata.asp
US Government, 2002. The E-Government Act of 2002. http://www.whitehouse.gov/omb/egov/pres_state2.htm
Vassilakis, C., Laskaridis, G., Lepouras, G., Rouvas, S., Georgiadis, P., 2003. A framework for managing the lifecycle of transactional e-government services. Telematics and Informatics Vol. 20, Issue: 4, pp. 315-329, Elsevier Publications, November, 2003
Weske, M., Schneider, B., 2002. An XML-Centred System Architecture For Flexible Electronic Services. International Journal of Information Technology & Decision Making, Vol. 1, No. 3, 2002 pp.525-540
Zuffoletto, J. 2002. BEA WebLogic Server Bible. Hungry Minds, Inc., New York, 2002, ISBN: 0−7645−4854−9