An Object-Oriented Approach for Designing Administrative e-Forms and Transactional e-Services

Dimitris Gouscos[1], Stathis Rouvas[1], Costas Vassilakis[1], Panagiotis Georgiadis[1]

[1] e-Government Laboratory –Dept. of Informatics and Telecommunications
University of Athens, 15771, Ilissia, Athens, Greece
{d.gouscos,rouvas,costas,p.georgiadis}@e-gov.gr
http://www.e-gov.gr/uoalab

Abstract. Although electronic transaction services are considered to be a necessity for e-government, it has not been possible insofar to unleash their full potential. E-forms are central to the development of e-government, being a basic means for implementing the majority of the public services considered as required for local and central public administration authorities. In this paper, we present an object-oriented model for e-form-based administrative services, which spans the e-service lifecycle, including development, deployment and use by enterprises and citizens, data collection and communication with legacy information systems. The proposed approach encompasses semantic, structural and active aspects of e-forms, providing thus an inclusive framework for modelling electronic services.

1 Introduction

Administrative services being the most visible and contradictory aspect of Government for the majority of citizens and businesses, it certainly comes as no surprise that e-Government action plans on the national as well as EU level ([9], [3]) recognize the importance of bringing administrative services on-line as a cornerstone of e-Government projects. Administrative forms, on the other hand, are an indispensable part of public administrative services, since delivery of a public administrative service entails, at some point, an application or declaration form that has to be filled, submitted and processed. This situation is well acknowledged by the fact that eEurope benchmarking indicators, as well as the eEurope 4-stage framework for monitoring maturity of on-line public services, both make explicit reference to levels of on-line availability and submission of forms as e-Government indicators ([6], [4], [5]).

Therefore, an important part of public administrative services and information are delivered and represented, respectively, through forms, which means that an approach to better design administrative forms and exploit their informational content can provide substantial benefits. An essential part of designing an administrative form has to do with (i) its structure, i.e. which its component, sections and fields are and how they are nested, and (ii) its semantics, i.e. the intended meaning of each individual field. Field semantics, in particular, determine (a) how input data should be validated

when the form is filled and submitted and (b) how input data can be processed and related in back-office operations.

Controlling the structure of an administrative form allows to have new forms resembling existing ones as appropriate. In this way forms can have a standardised appearance, which lowers design costs and allows to capitalize on a sharp learning curve due to user familiarization. Most importantly, however, controlling the structure of a form facilitates the effort to keep the semantics of all forms consistent.

Having consistent semantics for corresponding fields in different forms provides two very important capabilities:

1. Common fields of different forms can be identified and eliminated, so that many forms can be re-engineered into a single one; this may the case of an individual public agency reducing its different forms or of different public agencies trying to establish a single shared form in the context of one-stop e-Government G2C or G2B services.
2. Fields of different forms can be related; this may be the case of seeking a common field to be used as correlation key, or of trying to make two fields comparable for cross-checking or statistical processing purposes. Both of these functions may be needed either in a back-office setting within a single public agency or in the context of G2G information flows between different agencies.

An important note to make here is that the last point about consistent semantics essentially treats administrative forms as generic information sources (for instance, databases); indeed, the terms could well be exchanged with the rest of the point still holding true. This is due to the fact that, on a conceptual level, an administrative form is nothing more than the schema of an information collection. Therefore, the same issues about semantics consistency arise in both cases. Exploring the implications of this analogy, identification and elimination of common fields during the merging of administrative forms corresponds to schema integration of two information collections; relating fields of different forms for correlation or comparability corresponds to the same operations on, e.g., database tables; validation rules for the data input in administrative forms correspond to constraints and triggers in databases; administrative forms themselves correspond to data entry screens for DB applications.

Therefore, the above discussion about administrative form semantics holds for the semantics of arbitrary information sources as well. Still, our approach is focused on administrative forms because of the additional issues that are raised in this area. Controlling the structure of an administrative form is equally important as controlling its semantics, since it allows to re-use components and standardize on user navigation and user support issues.

2 Basic Terms

In this section we present the basic concepts of the electronic service model, regarding the desired functionality, as well as the processes and items involved in the lifecycle of electronic services.

2.1 E-Forms

We are concerned with the problem of applying some methodology for designing administrative forms in a systematic way, that allows to control (a) the structure, (b) the appearance and (c) the semantics of a form. In all these aspects, it should be possible to re-use previous designs for standardisation and productivity reasons. What is more, we are concerned with electronically represented administrative forms (e-forms), i.e. with forms implemented as web pages in the context of e-Government service offerings.

E-forms are developed and placed on the web site by the service provider. To this end, some experts on the service provider side are assumed who are able to create new e-forms and publish them on the web. In creating a new form, an expert should be able to re-use components of existing forms, whether these have to do with the form's structure, appearance, semantics, user assistance information, validation logic or process logic. What is more, since the creation of e-forms is assumed to be an iterative and collaborative process, there is a need to treat e-forms as artefacts which also have some "life-cycle" information: version, history, author, approver, etc.

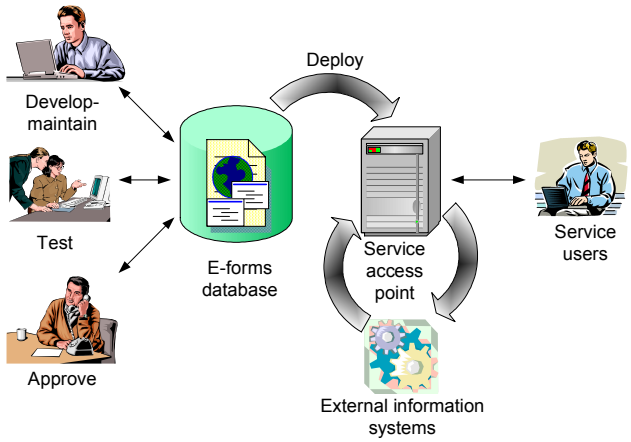

Fig. 1. E-Form Lifecycle

The basic user interaction scenario that we assume is that some administrative e-forms are available on the web; an end-user chooses the form of interest, navigates around its structure (sections, sub-sections etc), enters data as appropriate and submits it; a submitted form is processed by the service provider (the corresponding public agency) and some results are returned to the end-user. Input data validation occurs in two phases: some validation checks are applied on individual fields upon data entry whereas some others are performed after submission, since they apply to combinations of fields or may cross-check values with content from other sources (e.g. an administrative registry). It should be noted here that, in order to support these tasks, an e-form must accommodate, apart from its structure, appearance and semantics, also (d) user assistance information as well as (e) the appropriate

validation logic. Apart from that, the form should indicate (f) the associated process logic, i.e. it should determine (most possibly by pointing to it) the procedure with which the form should be processed after it is submitted and validated. Finally, processing of submitted forms may produce results or errors, which should be communicated to the users that filled the forms. The overall scenario for the e-form lifecycle is illustrated in Fig. 1.

2.2 Transaction Service Elements

A completed e-form, together with its structure, appearance, semantics, user assistance information, validation logic and process logic may be handled as a single entity that can be stored and re-used for generating new e-forms; and of course, this e-form can be published on the web as part of an e-Government service offering for transactional e-services.

As already mentioned, an e-form has some structure, i.e. it consists of sections, subsections, and individual fields just like paper-based administrative forms. Since most of the features of the e-form (appearance, semantics, user assistance info, validation logic) are best defined at the level of the e-form's components, these components (sections, fields) of an e-form are themselves considered as design artefacts which are autonomously created, stored and re-used; this policy evidently increases re-usability potential. Therefore, the term Transaction Service Element (TSE) refers to individual e-form sections and fields, while e-forms are referred to as Transaction Services.

2.3 Semantics

Semantic information as referred to above, has to do with the intended meaning of some values that are expected in the fields of an e-form at data entry time. Only if these semantics are correctly perceived and respected by end users, this a priori intended meaning will also be the actual meaning of the data a posteriori, i.e. when the e-form is inspected or processed after submission and validation. What is more, only if such a condition is met for the corresponding data of all administrative forms that are to be correlated, can such a correlation be successful. In order to facilitate the correct perception by end users of the intended meaning of a data value, this intended meaning can best be associated with the corresponding input field as a description; this description may be complemented by additional aids to the users, including online help references, examples etc. Therefore, field-level TSEs should be able to accommodate such information and references. Additionally, intended meaning descriptions can be optionally defined for section-level TSEs as well as for entire e-forms. The user assistance information of a TSE at any level can also be employed to clarify intended meaning semantics.

A different sort of semantics has to do with the nomenclature in which a value of a given intended meaning is expressed. To consider two simple examples: the same input fields called "your sex" may be (correctly) filled by the same person as "female" on one form and "woman" on another; the same input fields called "salary" may be

(correctly again) filled by the same person as "340750" on the one form (expressed in Greek drachmas) and "1000" in another (expressed in euro). Although intended meanings are the same and have been perceived correctly in both cases, data values differ. In order to avoid such situations, it is necessary to included intended nomenclature semantics at field-level TSEs. Possible nomenclatures in which data values are expressed include (a) closed sets of acceptable values, (b) statistical classifications, (c) sets of values from third registries as well as (d) measurement units. Drawing from statistics, it should be noted that the intended nomenclatures of two fields whose values must be compared do not necessarily have to be identical; as long as they are known and at least an one-way mapping exists between them, the field values are comparable.

3 Modelling Transaction Service Elements: An Object-Oriented Approach

According to the above analysis, TSEs at any level, i.e. form-, section- and field-level TSEs must be stored in a way that they can be re-used for designing new e-forms. We adopt an object-oriented approach towards modelling Transaction Service Elements and their attributes. This approach allows, on the one hand, to exploit a significant amount of inheritance and, on the other hand, it facilitates TSE re-use. The resulting object-oriented model, called Transaction Service Object Model (TSOM) incorporates all TSE attributes mentioned above, i.e. structure, appearance, semantics, user assistance, validation logic, process logic as well as life-cycle attributes.

3.1 Modelling of Submitted Forms

Submitted forms, i.e. e-forms that have been filled-in with values by users and submitted, are hosted in TSOM in a special class, which is a specialisation of the respective e-forms class. The additional attributes of a submitted form include, of course, its values, as well submission data and a post-submission trace which is intended as an administrative log for post-submission validation and processing operations (such a trace is necessary in order to produce, e.g., application status reports in an e-Government service context). Providing a special subclass for submitted forms permits for redefinition or cancellation of certain methods defined for the generic class modelling forms. For example, the method catering for the form presentation should now consider the data already typed in by the user; moreover, while a *submit* method is required for e-forms, a submitted form should not have such a method (since it is already submitted!). Through the inheritance mechanism, the specialised class overrides the inherited method to produce an appropriate error.

3.2 Modelling of Agents

As has been discussed, e-forms and TSEs in general are designed by some domain experts on the service provider side. Therefore, the life-cycle information of each TSE also includes some pointers to authors of this TSE and other roles, such as contributors and approvers. On the other hand, form instances are filled and submitted by end-users. For reasons of completeness and uniformity, TSOM includes a sub-hierarchy for modelling all of these roles (TSE authors, etc. as well as end users that fill and submit forms) under the general category of Transaction Service Agents. It is worth noting that this uniform modelling of both e-form authors and users connotes to the possibility of providing, in a real-world setting, a uniform web-based environment both for the authoring of e-forms by experts (possibly external to the public agency) and for the filling and submission of stable and released e-forms by end users in the context of operational e-Government services.

3.3 Modelling of Active Behaviour

Although e-forms may be considered as passive objects being filled-in and submitted by users, they in fact encompass substantial active behaviour in all stages of their usage:

1. When users select an e-form to fill in, the values of various fields may need to be pre-computed before data entry is allowed; this may be the case, for instance, where registry information about the end user is pre-loaded in certain fields of the form.
2. Upon field value modification, certain validations may need to be performed, such as type checks (e.g. only digits are entered in numeric fields), value range assertion, etc. Additionally, some fields may be *read-only* (e.g. fields containing registry data), and consequently appropriate behavioural rules should be defined to prohibit the alteration of their pre-loaded values. Although such checks may be performed when the form is submitted, they are usually performed upon field modification so as to detect errors early and assist thus the user throughout the procedure of form filling.
3. Field value modification may also trigger the updating of the value of other fields. Inter-field dependencies may be necessitated for user convenience, e.g. within a complex multi-page form with many sections that appear through navigation links, it may be arranged that some values are automatically carried on between sections, in order to be readily available for users to look up, without any need for navigating between web pages. Moreover, some fields are automatically calculated via formulas, such as table column sums, VAT amounts corresponding to sales or purchases etc. In these cases, changing the value of any field appearing in the right hand side of the formula should trigger the updating of the field appearing at the left hand side of the formula.
4. Form submission initiates the execution of additional actions, such as field-value correlations, cross-checking of user input with other submitted forms or registries, or even repetition of checks conducted earlier at the organisation's back end

environment, since in a web environment front-ends (browsers) are not considered trustworthy and validation checks depending on them may be circumvented.

5. Each step within the processing cycle of a submitted form may trigger a number of actions, such as appending entries to administrative logs for tracing purposes or sending electronic notification to the submitter regarding possible errors.

From the above analysis regarding active features within e-forms, we may derive the following requirements for the TSOM:

1. Since an active feature may involve a single field, a number of fields within the section or the whole e-form, the TSOM should allow the definition of active features in field-, section- and form-level TSEs.
2. TSOM should make provisions for specifying *when* each active feature should be fired. This is a two-fold issue, including the *event* that triggers the active feature (value change, form submission, back-end processing), and a *condition* which must hold (field is not empty, back-end processing resulted to an error etc.)
3. The functions that must be performed whenever appropriate may range from simple data type or value-range checks to much more complex validation checks that involve multiple fields or even correlation to information from external sources.

These requirements fit directly to the Event-Condition-Action (ECA) rules paradigm, which is encompassed in the Transaction Service Object Model through a dedicated class sub-hierarchy rooted at the Transaction Service Rules node. This generic category is further specialised to validation, pre-computation, protection, update and processing rules.

At the current state of work, a number of *primitive expressions, functions and constructs* are available for coding conditions and actions; these primitive elements may be combined using *operators*, to form arbitrarily complex constructs. In order to keep the scheme manageable, the number of primitive elements is kept small, sufficing however to model more than 90% of the checks usually encountered in electronic forms. For cases where the supplied expressive power is insufficient, the model supports the invocation of external methods, which may be coded in any general-purpose language with unconstrained expressive power. ECA rule execution clearly requires an appropriate engine; the choice of this engine depends on the environment within which the ECA rules will be executed. If the environment is a user's web browser, the Javascript language is a suitable option. Within an organisation's back-end, workflow engines, database triggers or general-purpose languages could be used. In all cases, the ECA rules should be mapped to the target environment.

3.4 Modelling Information Repository Access

While a transaction service is operational, it needs to access information repositories either to retrieve or to store and modify data. For instance, when a user selects to fill in an income tax declaration form, registry data must be retrieved from an information repository and filled in the corresponding form fields before the form is displayed. Subsequently, when the user submits the form, data filled in the various fields should be stored into an information repository, for future processing and/or reference. Data

access in the proposed environment is encapsulated in the *Information Repository* object class, which supports methods for invoking *predefined services*. Each such predefined service may accept input parameters and return, besides the execution status, appropriate information. For example, a taxation repository may offer a predefined service that accepts a citizen's tax registration number as an input parameter and returns a structure containing the citizen's data contained in the registry, or a failure indication. An environment offering transaction services may involve multiple instances of *Information Repository* objects, one for each actual information repository that needs to be accessed.

3.5 The Transaction Service Object Model

The class hierarchy of the Transaction Service Object Model is depicted in Fig. 2. The property protocol of TSOM classes is listed in Appendix A.

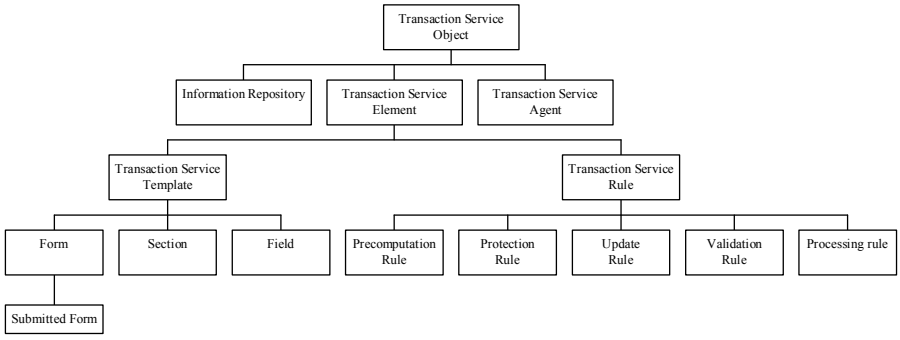

Fig. 2. Transaction Service Object Model class hierarchy

4 Sharing and Deployment of TSOM objects and Services

Form-level TSEs, together with their section- and field-level components, must be publishable on the web as forms that can be filled and submitted. What is more, TSEs at any level may need to be passed on to third parties that exploit a similar, but not necessarily identical approach towards designing administrative e-forms or e-services and could benefit from TSE understanding and re-use. Finally, e-forms may need to be exchanged with third parties in order to investigate capabilities for correlation, and submitted forms may also need to be exchanged in order to actually correlate field values. In the two latter cases, TSEs need to be exchanged together with their associated logic for validation and control, as well as their associated semantics for intended meanings and nomenclatures.

Most importantly, however, the need to exchange e-forms and submitted forms together with their associated logic and semantics calls for representing them by means of a semantics-neutral, syntax-level formalism where TSE attributes can be defined in a straightforward way. In this respect, XML is a natural fit, and has been generally accepted for communication between agencies [10].

The mapping of TSOM instances into XML documents can be approached quite simplistically. An instance o with values v_1, v_2 of an object class C with properties p_1, p_2 can be mapped to some XML code like

```
<instance>
          <of _class> C </of _class>
          <id> o </id>
          <p1> v1 </p1>
          <p2> v2 </p2>
</instance>
```

Fig. 1. XML representation of an instance

While this code excerpt arranges for the transfer of the actual values, communicating entities might also need to exchange schemas of the data transmitted. To this end, XML schemas [13] or XML DTDs [11] may be employed, facilitating the exchange of the data schema descriptions. XML schemas, in particular, may incorporate within data type definitions value constraints that apply to instances.

```
<xs:group name=" TransactionServiceRule">
   <xs:sequence>
      <xs:group ref="TransactionServiceObject"/>
      <xs:element name="trigger" type="Event">
      <xs:element name="condition" type="Expression">
      <xs:element name="procedure" type="ScriptAction">
   </xs:sequence>
</xs:group>
```

Fig. 2. Exchanging data schemas through XML

One interesting parameter of TSOM object sharing is that in some cases, certain aspects of the objects need not (or must not) be exchanged for the sake of simplicity, security or information volume reduction purposes. For example, when a tax declaration is forwarded for processing, the presentation details or active features contained within the involved TSOM instances are irrelevant, and may therefore be omitted, without any loss of functionality. Similarly, if the Ministry of Finance has included in its TSEs validation tests to control tax evasion, disclosing of these checks to cooperating taxation agencies (e.g. accountant offices) would void their efficiency. Therefore, the framework should provide the mechanisms for controlling which portions of the Transaction Service Object Model class hierarchy should be exchanged. To this end, a *content negotiation mechanism* is provided, through which the server (i.e. the offering machine) *advertises* the content that is available from it; subsequently, the client requests this data, or a subset of it, possibly providing some authentication credentials. Finally the server, after checking the presented credentials and the access constraints, sends the data or returns an appropriate error either

forbidding access or instructing the client to modify its request and ask for a smaller subset. The requesting client may optionally return a reply, either to simply indicate transfer status (success/failure) or to provide any relevant information. Replies, if provided, should be also coded in XML.

The content negotiation mechanism may additionally be used for avoiding to exchange redundant TSOM objects. This applies, for instance, to the case where a new e-form is sent which makes use of section- and field-level TSEs already exchanged. More importantly, this applies to exchanging multiple submitted forms without sending more than once the same e-form. Within the content negotiation phase, the requesting party provides the server with an identification of the objects it already has, so as to enable the server to limit its reply to the objects that will be actually new for the client. In all cases, however, the main issue is to provide full functionality, with optimisation issues being a highly desirable, but not absolutely necessary feature.

The communication mechanism described above is generic enough to accommodate all circumstances in which electronically submitted forms and/or their data schema need to be exchanged with other information systems. These information systems may be either external to the organisation deploying the electronic service (e.g. governmental agencies, business partners etc.), or internal, such as batch jobs that will process the data (e.g. tax computation procedures).

Service deployment, exploiting the Web as a primary channel, calls for mapping of the object-oriented constructs (i.e. instances of the Transaction Service Object Model) into some mark-up language that can be handled by browsers. The prime candidate for such a mapping is currently HTML since XML and other XML-oriented developments (such as X-Forms [12]) are not fully handled by the majority of browsers. Mapping of TSE information to the appropriate HTML code can be quite straightforward, by employing HTML forms and form elements to facilitate user input, hyperlinks to support navigation between form parts and using visual elements, such as format designators (<*b*>, <*i*>, etc.) or layout specifiers (e.g. <*table*>) to produce the effects designated by the related TSE attributes. A number of active features may also be supported on browser level by means of the Javascript language, which provides modelling constructs for *(event, action)* pairs. Additional active features, such as validation checks or information repository accesses, can be automatically generated for the organisation's back-end, based on the information contained in the TSEs. These features are actually realised through server-side scripting techniques, such as PHP and JSP. These mappings, however, decompose a semantically rich model (the object-oriented one) to low-level formatting and coding constructs, which makes controlling harder and minimises the capabilities for reverse engineering. It is expected that with the advent of the XML and X-Forms standards and their incorporation into browsers, a more straightforward and "non-lossy" mapping may be employed for deploying the electronic services through the Web.

5 Conclusions – Future Work

Work reported in this paper approaches the critical problem of automating the creation, management and processing of electronic administrative forms, in a way that supports the handling of rich form structures together with their associated front- and back-end logic. Object-oriented modelling of e-forms and their active behaviour allows for (a) semantic richness, (b) modelling extensibility, (c) high-level encapsulation of e-forms' data, metadata and associated logic as well as (d) uniform modelling of both submitted e-forms and e-form templates. Mapping of e-forms to XML messages allows forwarding of submitted e-forms to remote sites for processing, which means that front-end submission and back-end processing of an e-form may well be distributed over the web. What is more, XML mapping of e-form templates facilitates the exchange of e-form artefacts for collaborative e-forms design as well as for re-usability purposes.

An important direction of work to carry on, is to elaborate the modelling of e-forms' active behaviour by means of ECA rules and consider additional formalisms of equivalent expressive power (e.g. Horn clauses [8]). Any such representation shall have to be mapped to appropriate XML structures. This mapping may use techniques from existing work (e.g. [1], [2], [7]).

Still another direction of research is that of studying the middleware mechanisms necessary to accept or send e-forms, taking care of issues mentioned in this paper such as management of process traces and content negotiation. The integration of such middleware mechanisms with back-end processing infrastructures is also a subject of investigation.

6 References

1. J. Bailey, A. Poulovassilis, P.T. Wood "An Event-Condition-Action Language for XML", to appear in *Proc. WWW2002*, technical report available at http://www.dcs.bbk.ac.uk/~ap/pubs/wwwTechRep.ps
2. A. Bonifati, S. Ceri, S. Paraboschi, "Active rules for XML: a new paradigm for e-services", VLDB Journal 10(1), pp. 39-47, 2001
3. Commission of the European Communities, "eEurope Action Plan 2002: An Information Society For All", June 2000.
4. Commission of the European Communities, "eEurope 2002 Impacts and Priorities", March 2001
5. European Commission, DG Information Society, "Web-Based Survey on Electronic Public Services", November 2001.
6. European Union Council, "List of eEurope Benchmarking Indicators", November 2000
7. H. Ishikawa, M. Ohta, "An active Web-based Distributed Database System for e-Commerce", Proceedings of the Web Dynamics Workshop, London, 2001
8. J. W. Lloyd, "Foundations of Logic Programming", Springer Series in Symbolic Computation, Springer-Verlag, New York, 1984.
9. UK Cabinet Office, "E-Government: A Strategic Framework for Public Services in the Information Age", April 2000.
10. UK Cabinet Office, E-Government Interoperability Framework, September 2000.
11. W3 Consortium, "XML 1.0 (Second Edition)", available at http://www.w3.org/TR/REC-xml

12. W3 Consortium, “XForms-The Next Generation of Web Forms”, available at http://www.w3c.org/MarkUp/Forms
13. W3 Consortium, “XML Schema”, available at http://www.w3.org/XML/Schema

Appendix A – TSOM property protocol

object class TransactionServiceObject	☐ Id
object class TransactionServiceElement	☐ Name ☐ Description ☐ Version ☐ History ☐ Authors ☐ Contributors ☐ Approvers
object class TransactionServiceTemplate	☐ AdminName ☐ AdminCode ☐ AdminDescription ☐ Instructions ☐ Examples ☐ FAQs ☐ ApplicableRegulations ☐ MoreInfoPointer ☐ VisualEffects ☐ TransactionServiceRules
object class Form	☐ SectionSequence ☐ Language ☐ Provider ☐ RelatedForms ☐ AdminInfo ☐ ProcessingPointer
object class Section	☐ SubsectionSequence ☐ FieldSequence
object class Field	☐ Nomenclature ☐ DefaultValue ☐ FormatMask
object class SubmittedForm	☐ Values ☐ SubmittedBy ☐ SubmissionTime ☐ PostSubmissionTrace

object class TransactionServiceRule

- Trigger
- Condition
- Procedure

object class TransactionServiceAgent

- Name
- ContactCoordinates
- Credentials
- Privileges

object class InformationRepository

- Name
- Services
- ConnectionDetails