

IST PROJECT 2001-35399



A Governmental Knowledge-based Platform for Public Sector Online Services

Project Number:	IST-2001-35399
Project Title:	A Governmental Knowledge-based Platform for Public Sector Online Services
Deliverable Type:	Public

Deliverable Number:	D62
Contractual Date of Delivery:	31-5-2003
Actual Date of Delivery:	31-7-2003
Title of Deliverable:	Implementation of SmartGov Services and Applications
WP contributing to the Deliverable:	WP6
Nature of the Deliverable:	Report
Editor(s):	Stelios Gorilas
Author(s):	Stelios Gorilas, Pablo Fernandez Pardo, Tomas Pariente Lobo, Costas Vassilakis, Akrivi Katifori, Anna Charissi, George Lepouras, Nick Adams, John Fraser, Ann Makynthos, Vassilis Stoumpos

Abstract: This deliverable constitutes the implementation report of WP6. It presents the implementation details of the following components: Integrator, SmartGovLang Translator, SmartGov Agent and SmartGov Information Interchange Gateway.

Project funded by the European Community under the "Information Society Technologies" Programme (1998-2002)

© Copyright by the SmartGov Consortium.

The SmartGov Consortium consists of:

Partner's Name	Acronym	Role	Country
University of Athens	UoA	Project Coordinator	Greece
T-Systems Nova	TNB	Partner	Germany
Indra Sistemas S.A.	Indra	Partner	Spain
Archetypon S.A.	ARC	Partner	Greece
Napier University	NU	Partner	UK
General Secretariat for Information Systems	GSIS	Partner	Greece
City of Edinburgh Council	CEC	Partner	UK

Table of Contents

Executive Summary.....	8
1 Introduction.....	9
2 Platform Overview	10
3 The Integrator Component.....	12
3.1 Introduction	12
3.2 The STRUTS Framework	12
3.2.1 CONTROLER COMPONENTS	12
3.2.2 MODEL COMPONENTS.....	15
3.2.3 VIEW COMPONENTS	16
3.2.4 STRUTS class and sequence diagram	16
3.3 STRUTS in the context of the Integrator	17
3.4 Integrator Input	19
3.4.1 Building modules.....	19
3.4.2 Taxonomies.....	20
3.4.3 Service elements.....	21
3.4.4 Functionality	28
3.4.5 Establishing links between the form visual elements and SmartGov semantic elements.....	29
3.5 Integrator Processing layers	30
3.5.1 Model	31
3.5.2 Processor	31
3.5.3 Builder.....	32
3.6 Integrator Implementation.....	32
3.6.1 Packages.....	32
3.6.2 Class diagrams	34
3.6.3 Sequence diagrams	38
3.7 Integrator Main Sub-components/Tasks.....	39
3.7.1 Help JSP	39
3.7.2 Integrator Ant Task	40
3.7.3 Data Storage	40
3.7.4 Statistics Storage.....	41
3.7.5 Design Time Service Model	43
4 The SmartGovLang Language.....	46
4.1 Specification of the SmartGovLang language	47

4.1.1	Specification of full rules	47
4.1.2	Specification of compact rules.....	50
4.2	Translation of the SmartGovLang language	53
4.2.1	Storage of the SmartGovLang language	56
4.2.2	API for the SmartGovLang language translator.....	62
5	SmartGov Agent – Information Interchange Gateway.....	64
5.1	Implementation of the SmartGov Agent – Information Interchange Gateway	
	66	
6	Conclusions	70
7	References	71
	Appendix A – SmartGovLang grammar	72
	Appendix B - List of functions available in SmartGovLang	74
	Appendix C – Integrator JavaDocs	76
	Appendix D - SmartGovLang Translator JavaDocs	108
	Appendix E - Agents JavaDocs	113

Table of Figures

Figure 1 Overview of the SmartGov platform	10
Figure 2 Struts, an MVC Web Application Framework	12
Figure 3 STRUTS class diagram.....	16
Figure 4 STRUTS sequence diagram	17
Figure 5 SmartGov service description layers.....	20
Figure 6 Taxonomies and Service elements	21
Figure 7 Example of a set of service description files.....	28
Figure 8 - Link establishment between visual and semantic entities	30
Figure 9 Integrator Layers.....	31
Figure 10 Integrator packages and interdependencies.....	33
Figure 11 Process launching sequence diagram	38
Figure 12 TS element processing sequence diagram	39
Figure 13 – Sample interface for defining full rules	50
Figure 14 – Sample interface for defining compact rules.....	52
Figure 15 - Translating a SmartGov rule to Java and Javascript.....	56
Figure 16 The SmartGovLang translator interface.....	63
Figure 17 – DTD for SmartGov Agent to Information Interchange Gateway Messages	65
Figure 18 – DTD for Information Interchange Gateway to SmartGov Agent Messages	65
Figure 19 - Class Diagram for SmartGov Agent	66
Figure 20 - Class diagram for the IIGServer	67
Figure 21 - Class diagram for the SSLIIGServer	68
Figure 22 - Class diagram for logging facilities	69

List of Acronyms

Acronym	Explanation
APAQ	Adelante (Outgoing) Pending Actions Queue
API	Application Programming Interface
EPAQ	Entra (Incoming) Pending Actions Queue
IIG	Information Interchange Gateway
IIG-MYP	Information Interchange Gateway – Minimal Yoking Processor
IIG-NI	Information Interchange Gateway Notification Initiator
IIG-SEP	Information Interchange Gateway – Separate External Process
IT	Information Technology
JSP	Java Server Page
KU	Knowledge unit
MVC	Model-View-Controller
PAQ	Pending actions queue (in the context of the SmartGov Agent and the Information Interchange Gateway)
PAQUED	Pending Actions Queue Dispatcher (in the context of the SmartGov Agent and the Information Interchange Gateway)
SGA	SmartGov agent
SGA-NI	SmartGov Agent Notification Interceptor
SGovApp	SmartGov application
TS	Transaction service
TSE	Transaction service element
XML	Extensible Markup Language
JSP	Java Server Page
XHTML	eXtensible Hypertext Markup Language
RUP	Rational Unified Process
UML	Unified Modeling Language
WAP	Wireless Application Protocol
XSLT	Extensible Style sheet Language Template
WML	Wireless Markup Language
JDBC	Java Database Connectivity
API	Application Programming Interface
RDBMS	Relational Database Management System
LDAP	Lightweight Directory Access Protocol

DSN	Data source name
SOAP	Simple Object Access Protocol
DDL	Data Definition Language

Executive Summary

The SmartGov project, fully entitled as “A Governmental Knowledge-based Platform for Public Sector Online Services”, commenced on the 1st of February 2002. The aim of the SmartGov platform is to introduce a *holistic approach* for the use of electronic forms within the public sector. This approach is focused on two main axes:

- ❑ Integrating emerging standards with state-of-the-art technology and with advances in areas such as knowledge management, Web technologies, interoperability and accessibility.
- ❑ Introducing this technology in a systematic manner by adopting new process models and process re-engineering and process improving methods.

In order to support these axes, a complete knowledge-based development environment will be provided to the public authorities, supporting all phases of electronic, form-based services development, deployment and maintenance. This environment will be complemented with a generic framework, including process models, reference models and social acceptance models for the introduction of electronic services.

In the workpackages that have been completed insofar, the state of the art has been reviewed (WP3 [D31]), the current status of the participating public authorities has been captured (WP3), the user requirements have been analysed (WP4 [D41]) and the high-level specifications of the system have been derived (WP4). After that the software architecture document of the platform was produced, which reported on work carried out in WP5 and WP6 and provided detailed system specifications for the architectural modules identified in WP4 as parts of the SmartGov platform. The present document is an elaboration of the low level design that was delivered within the aforementioned joined deliverable (D51-D61 [D51-61]) and constitutes the outcome of a number of iterations that were performed during the development phase; as such it contains more implementation details than the previous deliverable. It corresponds to the implementation work carried out within workpackage 6 and does not include components that were developed outside of this workpackage. This is due to the reviewer’s suggestion to keep separate the two implementation deliverables (D52 and D62). Hence the implementation work reported here presents the implementation details of the following components: Integrator, SmartGovLang Translator, SmartGov Agent and SmartGov Interchange Gateway.

1 Introduction

The SmartGov platform software was specified, designed and developed by following the Rational Unified Process [RUP]. As this process suggests, the specifications, design and development do not happen strictly sequential but in an iterative manner. The present deliverable constitutes an implementation report that presents the outcome of the last iteration of the development phase. Hence it contains more implementation details than D51-61 [D51-61] and also slight changes in the design and implementation.

The document firstly provides a brief description of the overall SmartGov platform architecture, in order to familiarise the reader with the modules of the platform, their functionality and their interactions. The next section presents the Integrator component, which is the “heart” of the SmartGov platform. It is responsible for accessing the XML repository (presented in D51-61) and extracting the documents containing the service definitions. These are stored there by the Front-end tool (presented in D51-61 and D52). It then processes these descriptions and generates the run-time e-forms application, which is a STRUTS [STRUTS] application and deploys it in the specified web container/server, with the use of the ANT [ANT] utility. Since STRUTS plays a crucial role in the integrator design, it is shortly presented and analysed. Subsequently, the required input of the integrator is described and its core design and processing layers are given. Finally the main subcomponents tasks that the integrator includes are shortly described.

After the Integrator section, the SmartGovLang Language is described. SmartGovLang is a simple, yet powerful language, designed to enable domain experts with minimal computer skills to write validation checks. This language is processed and translated at generation time by the SmartGovLang Translator, which is invoked by the integrator, and the appropriate JavaScript and Java code snippets are created and planted in the generated service.

Finally the SmartGov agents are presented, which are responsible for the communication with third party systems and storage of the submitted instance data.

2 Platform Overview

In the following paragraphs, the high-level architecture of the SmartGov platform is summarised. This section aims to provide the reader with a global view of the SmartGov platform and outline the modules involved in the development and delivery of transaction services. These modules are detailed in the main part of this deliverable.

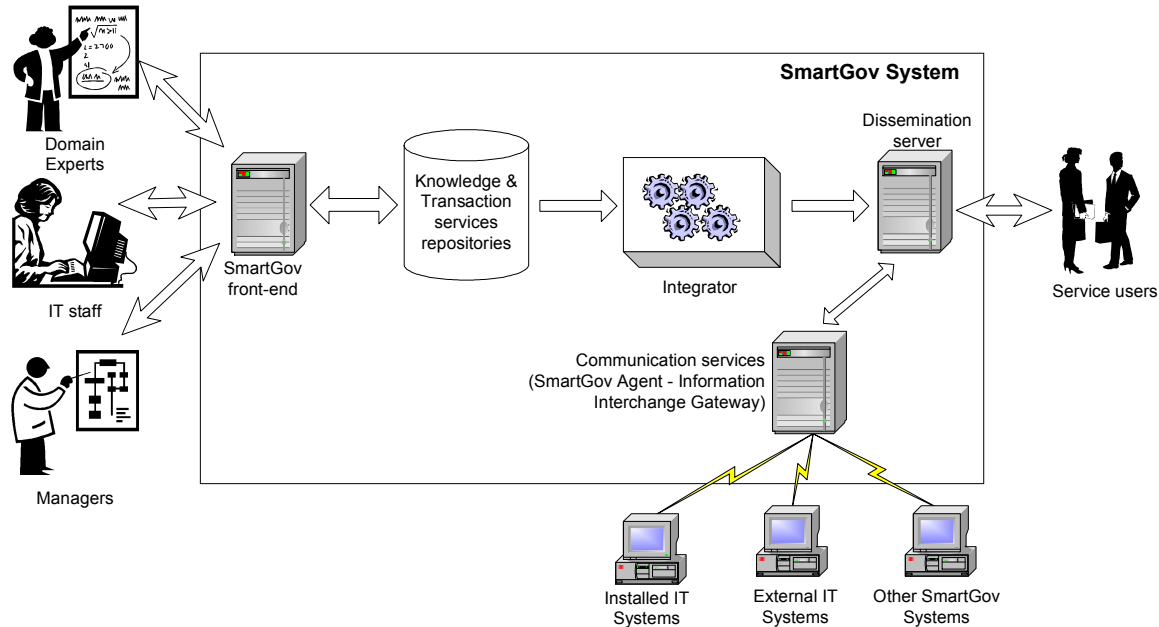


Figure 1 Overview of the SmartGov platform

Figure 1 illustrates the SmartGov platform architecture, in which the following modules may be identified:

- The SmartGov knowledge and transaction services repositories. These are general depots for storing organisational knowledge and information pertaining to the transaction services that are developed using the SmartGov platform. In order to provide a semantically rich environment and facilitate extensibility and interoperability, all data is stored in XML format.
- The SmartGov front-end, which constitutes of personalised application development environments which are available to the actors involved in the lifecycle of electronic transaction services, namely domain experts, IT staff and managers. The actors employ the SmartGov front end to populate, query and modify the knowledge and transaction services repositories.

- The integrator, a software module that reads the contents of the knowledge and transaction services repositories, and automatically generates all necessary elements (files, objects, components etc) for a fully operational transactional service. These elements are then deployed on a dissemination server, initiating service delivery to the users.
- The communication services, comprising of two units, namely the SmartGov agent and the information interchange gateway. This module provides generic communication mechanisms with installed IT systems for the purposes of data exchange, hiding idiosyncrasies and peculiarities of information system platforms and facilitating resilience against temporary failures.

3 The Integrator Component

3.1 Introduction

The integrator component is responsible for generating, compiling and deploying the e-forms service that will be used by the final users i.e. the citizens. The generated service is installed at the dissemination server, which is a web container, in our case the Apache Tomcat. The generated service is a STRUTS application (i.e. it is built according to the STRUTS framework) and thus requires the installation of STRUTS on top of Tomcat. Due to its crucial role in the implementation, STRUTS is explained in the first section of this chapter.

3.2 The *STRUTS* Framework

Figure 2 depicts the architecture of STRUTS.



Figure 2 Struts, an MVC Web Application Framework

3.2.1 CONTROLLER COMPONENTS

Struts Controller Servlet

ActionServlet is the command part of the MVC implementation and is the core of the Framework. It is a central servlet that mediates application flow. It is a command design pattern implemented as a servlet and is provided by the Struts framework.

- Parses the Struts configuration file
- Checks for defined ActionMappings
- Delegates requests to the appropriate application actions, through Action mappings
- Checks session for instance of bean of appropriate class
- If no session bean exists, creates one automatically
- Automatically populates an ActionForm bean with request parameters
- Determines which Action or JSP to dispatch to

Struts Configuration file (struts-config.xml).

This file contains all the configuration setting and is a vital part of a Struts application. It defines the action mappings for the application and allows for performing certain changes without recompilation of code.

The configuration file contains:

- Action Mappings: These are mappings between user actions and business methods by delegating user requests to the later and then decide which JSP should be rendered next.

Action Mappings

- define mappings between a logical Action name and the physical Action class

- specify an associated ActionForm name
- declare the input page to return to when errors occur
- define local forwards (forward tag)

- Declarations

Tell the Controller where to physically locate the referenced ActionForm

- Global forwards

Links to other pages that are referenced in other jsps or Actions.

Provide a level of indirection, so that a path change only has to be updated in one place.

Example: `<forward name="next" path="/forwardedPage.jsp" />`

A typical Struts- config example follows:

```
<struts-config>
  <!-- Data Sources -->
  <data-sources>
  </data-sources>

  <!-- Form Beans -->
  <form-beans>
    <form-bean type="struts.SubmitForm" name="SubmitForm"
               className="org.apache.struts.action.ActionFormBean">
    </form-bean>
  </form-beans>

  <!-- Global Exceptions -->
  <global-exceptions>
  </global-exceptions>

  <!-- Global Forwards -->
  <global-forwards>
  </global-forwards>

  <!-- Action Mappings -->
  <action-mappings>
    <action name="SubmitForm" path="/submit" scope="session">
```

Actions class

The goal of the Action class is to process the request and then return an ActionForward object identifying the view JSP to forward to. The Action class is a wrapper around the business logic. Its purpose is to translate the HttpServletRequest to the business logic.

- The Action controls the flow and not the logic of the application.
- Place business logic in a separate package or EJB, to allow flexibility and reuse.
- The purpose of the Action is to 'transform' the interface of a class into another interface the clients expect.

- Actions act as the adapter between the web/HTTP layer and the business logic layer. The Action class follows the Adapter design pattern and lets classes work together that couldn't otherwise because of incompatibility interface.
- For the Action class to be used, it needs to be subclassed and its process() method overwritten.

3.2.2 MODEL COMPONENTS

Model components are user defined. These can be any Java beans. They must have a public empty constructor as well as getters and setters for all the data elements.

ActionForm Bean (or Helper Bean)

- Extends the ActionForm class
- Create one for each input form in the application
- For every request parameter whose name corresponds to the name of a property in the bean, the corresponding setter method will be called
- The updated ActionForm bean will be passed to the Action Class perform() method when it is called, making these values immediately available
- Multiple requests can be mapped UserActionForm.
- ActionForms can contain validation code.
- The struts-config.xml file controls which HTML form request maps to which ActionForm.
- It is used by the JSP to collect data from the user and populate the form fields.

Framework includes custom tags that facilitate populating form fields from a form bean.

- They are used by the Action object to validate the form input.

```
public ActionErrors validate(ActionMapping mapping, HttpServletRequest request)
```

A form does not necessarily map to a single JSP. You should note that an "input form", in the sense discussed here, does not necessarily correspond to a single JSP page in the user interface. A form may be a series of page and Struts encourages the developer to define a single ActionForm bean that contains properties for all of the fields, no matter which page the field is actually displayed on. Likewise, the various pages of the same form should all be submitted to the same Action Class.

3.2.3 VIEW COMPONENTS

View components are mainly Java Server Pages, accompanied by HTML, JavaScript, Style sheets, Struts Tags and Custom Tags, Resource Bundles, etc.

3.2.4 STRUTS class and sequence diagram

To complete this short description of STRUTS, in the following figures the class and sequence diagrams of STRUTS are given.

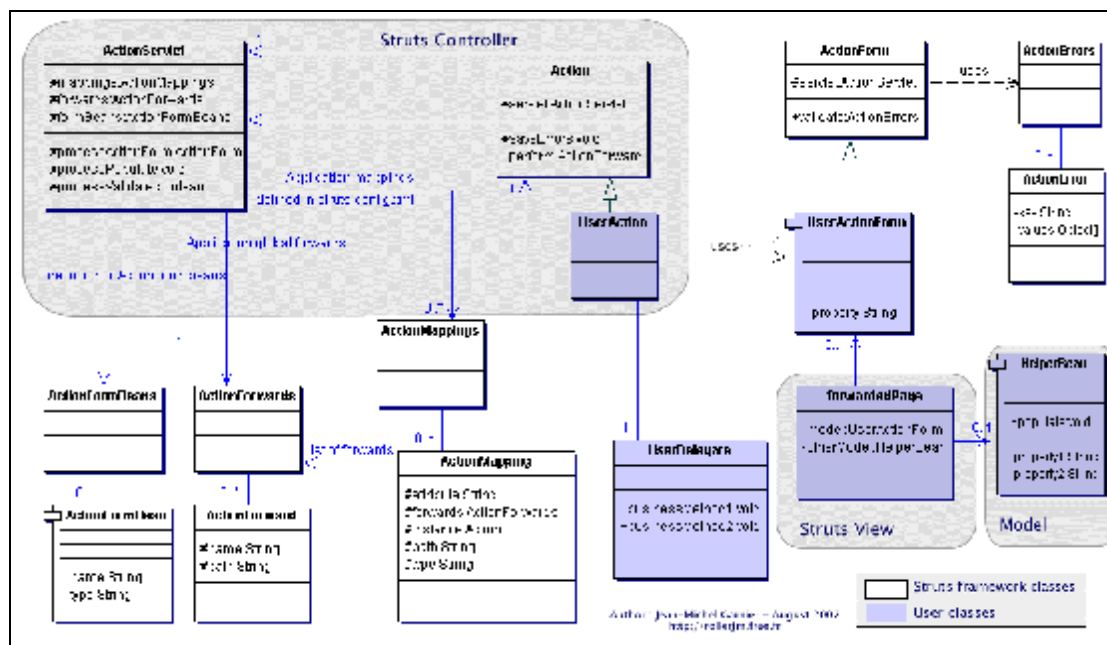


Figure 3 STRUTS class diagram

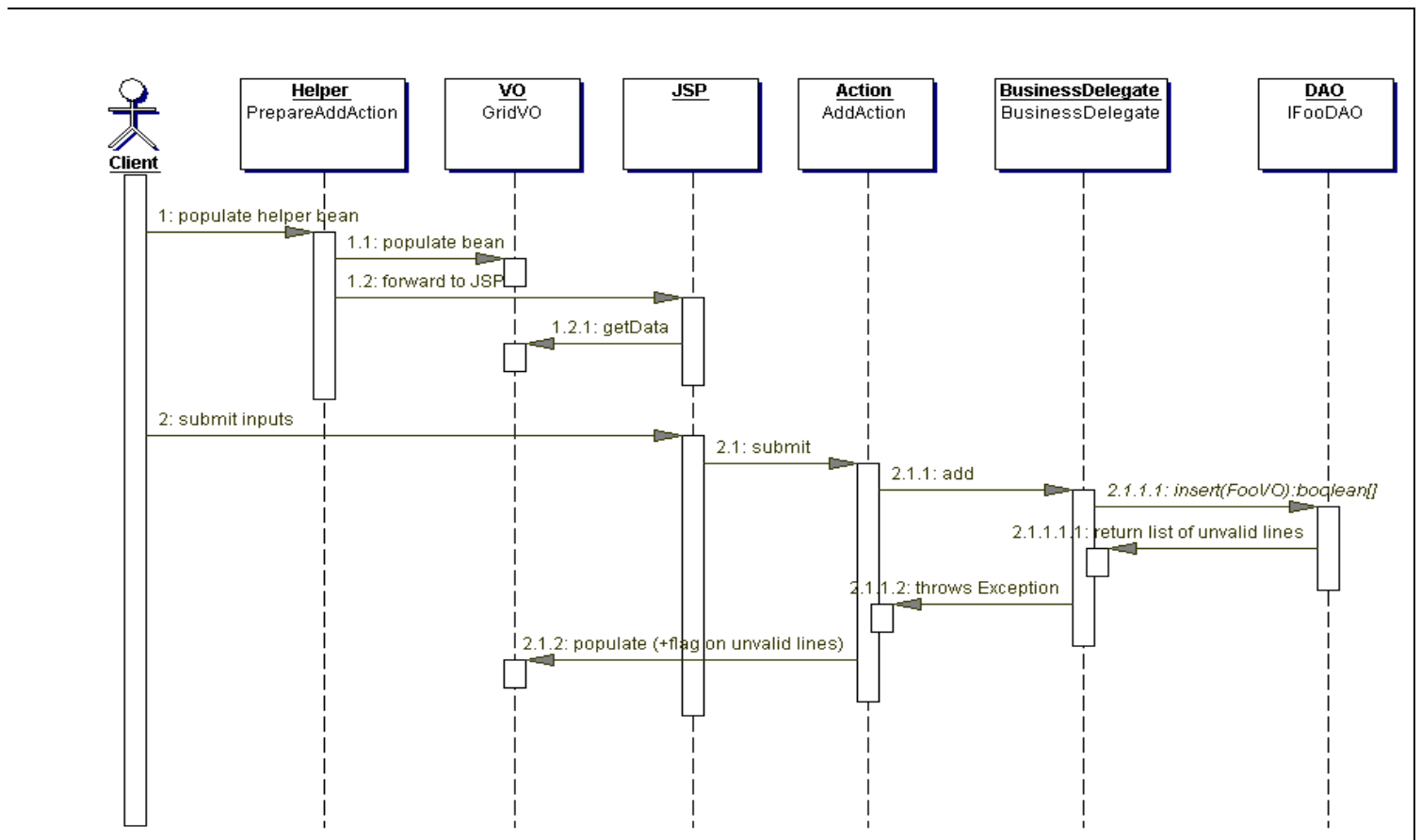


Figure 4 STRUTS sequence diagram

3.3 STRUTS in the context of the Integrator

The aforementioned short analysis of the STRUTS framework shows the components needed for STRUTS to operate. This in the context of the integrator implies that a number of files have to be generated, compiled and finally deployed. These activities are performed automatically by the integrator after processing of the XML[XML] files that contain all the relevant service descriptions (TSEs, TSEGroups, Forms etc.). More analytically but still in a high level view the following tasks are required by the integrator.

TASK #0: GENERATE XML Doc Beans

Use Castor [castor] to parse the XML schemas collection that describe the SmartGov e-forms application to generate a collection a java class (XML Doc Beans)

TASK #1: GENERATE CONTROLER COMPONENTS

Action Class

The Action class is created and used by ActionServlet. It is a wrapper around Business logic to use Action, subclass and overwrite the perform() method.

- Method perform processes the HTTP request, and creates the corresponding HTTP response. Returns an ActionForward instance describing where and how control should be forwarded.
- Method saveErrors saves the specified error messages keys into the appropriate request

[generate validation script blocks and attached them to the Action class]

Struts Configuration file

- Collect all required information to generate the configuration file
- Generate configuration file according to application requirements (form beans, action mappings blocks)

[Validate configuration file]

TASK #2: GENERATE MODEL COMPONENTS

Action Form Class

- retrieve main form description XML
- parse XML file and identify TSEs
- retrieve XML documents of included TSEs
- generate a single java class that extends ActionForm
- populate the class with all the required getters and setters

TASK #3: GENERATE VIEW COMPONENTS

JSPs

- retrieve main form description XML and referenced TSE XML files
- construct a single XML form
- retrieve XHTML form description
- introspect XHTML, XML form descriptions to locate associated TSEs (XML Doc Beans)

- perform transformations on XHTML elements into Struts JSP tags (Xalan, XSLT)[Xalan]

Resource files

- introspect XML form descriptions to locate resource related data (XML Doc Beans)
- construct resource files

TASK #4: GENERATE VIEW COMPONENTS

- generate the application deployment descriptor web.xml
- create an Ant script that puts all the files together and creates the application WAR file.

3.4 Integrator Input

This section describes the input that the Integrator requires in order to generate a service. Some of the parts included in this section also exist within the previous deliverable D51-D61, but are also included here for completeness purposes.

3.4.1 Building modules

The description of a SmartGov service can be logically divided in different layers of abstraction. These are depicted in the following diagram along with the name of the corresponding element (these layers are described in more detail in the following sections).

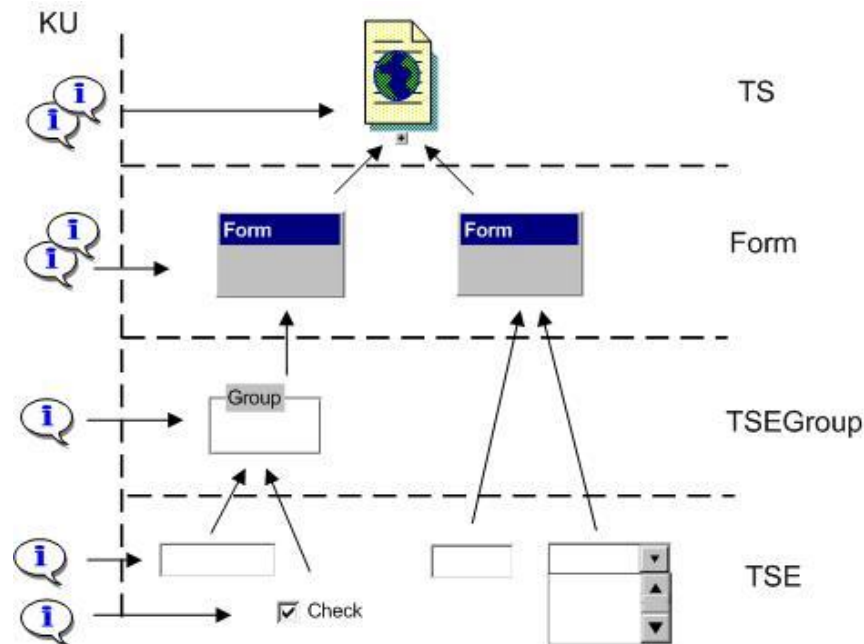


Figure 5 SmartGov service description layers

There is one descriptor per service at the top level. A SmartGov service is composed of a set of forms, one following the other as they are presented to the end-user. Each form can contain both stand-alone controls as well as controls logically grouped together. At the lowest level, a Transaction Service Element (TSE) is the most basic module of a service. A TSE may correspond to a real data field in a table or may be a temporary, calculated register.

All elements of the service can have any number of associated Knowledge Units (KUs). These elements are intended to play the role of structured, online help, during the service's usage.

3.4.2 Taxonomies

The SmartGov platform also defines an additional level of abstraction, over the ones described in the previous section. These are called service taxonomies and are depicted in the following figure.¹

¹ The concepts described in this section are transparent to the Integrator's implementation, as agreed with project partners. They are only here for completeness.

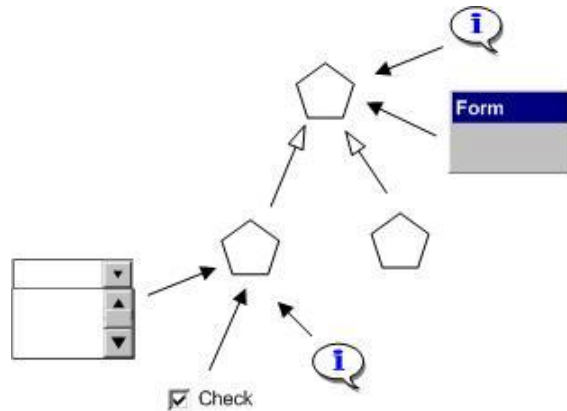


Figure 6 Taxonomies and Service elements

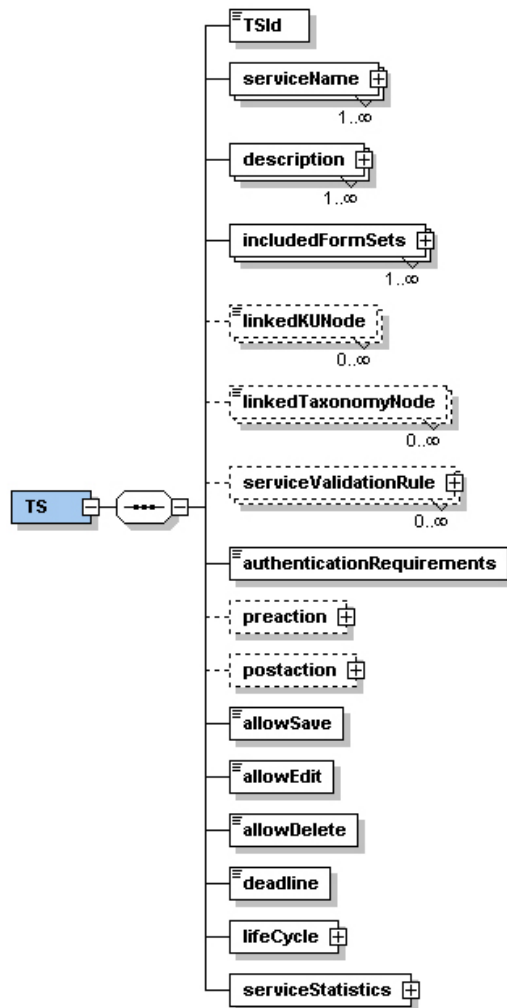
SmartGov defines hierarchies of taxonomy nodes to which service elements are attached. These taxonomies are used to logically group service concepts and elements. This is useful as it gives a better overview of a particular element's (e.g. TSE) functionality and semantics. These relations are intended for design-time and do not affect the Integrator.

3.4.3 Service elements

The following sections contain a more detailed description of the actual service description elements, as are defined in the various schemas. Boxes with solid borders denote mandatory elements, while dotted borders denote optional elements. Elements with a small cross icon next to them denote composite elements, containing more children, hidden for simplicity. Each section describes only those fields of importance to the Integrator.

3.4.3.1 TS

TS is the top-level element in the service description hierarchy. There is only one TS defined per SmartGov service. It defines a service as a set of forms, viewed/executed by the user in sequence. The execution sequence may not necessarily be sequential and may depend on certain values (not) being submitted.



TS element

TSId: a unique id for this service

serviceName: a short locale-sensitive text, suitable for a title or a similar caption

description: locale-sensitive text, providing a more extensive description of the service. This may be included in the initial page of the service

includedFormSets: SmartGov services can behave differently depending on the end-user's platform of communication (e.g. Web, WAP). Each set contains a number of forms and the name of the associated platform. The "connection" with the Form elements is done by using their unique ids (see below).

linkedKUNode: the unique ids of KUs associated with the service, in general. These can include detailed descriptions of the service, tutorials, etc.

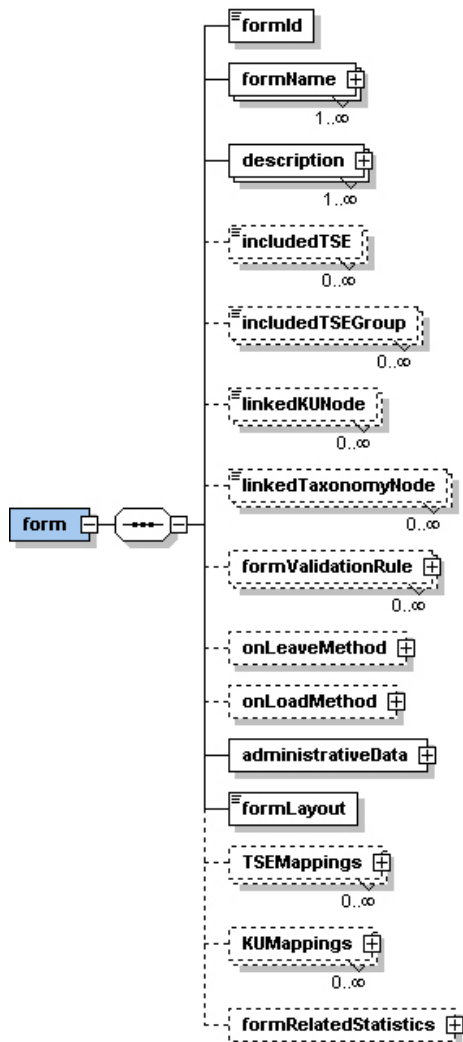
serviceValidationRule: logic rules applied to the service as a whole. A validation rule contains a piece of code, either native or pseudo-code (SmartGov expressions) that must hold true, before the service can be considered successfully completed. If a rule fails, it defines locale-sensitive error messages.

deadline: a final date, after which the service will not be available to end-users

serviceStatistics: a set of metrics that need to be monitored by the service implementation. These are defined as boolean fields.

3.4.3.2 Form

A Form element is the logical representation of an actual web form. A SmartGov form is composed of individual TSEs and TSEGroups, containing more TSEs. The Form may also have associated KUs and validation rules applicable to one or more of its internal controls. The actual implementation is located in a separate xHTML file.



Form element

formId: unique id of this form, in the context of this service

formName: a locale-sensitive name that may be used as a title

description: a small, locale-sensitive paragraph with some info on the contents of the form

includedTSE: ids of TSE elements located on the form, not part of any TSEGroup

includedTseGroup: ids of TSEGroup elements, located on this form

linkedKUNode: unique ids of KUs related with this form

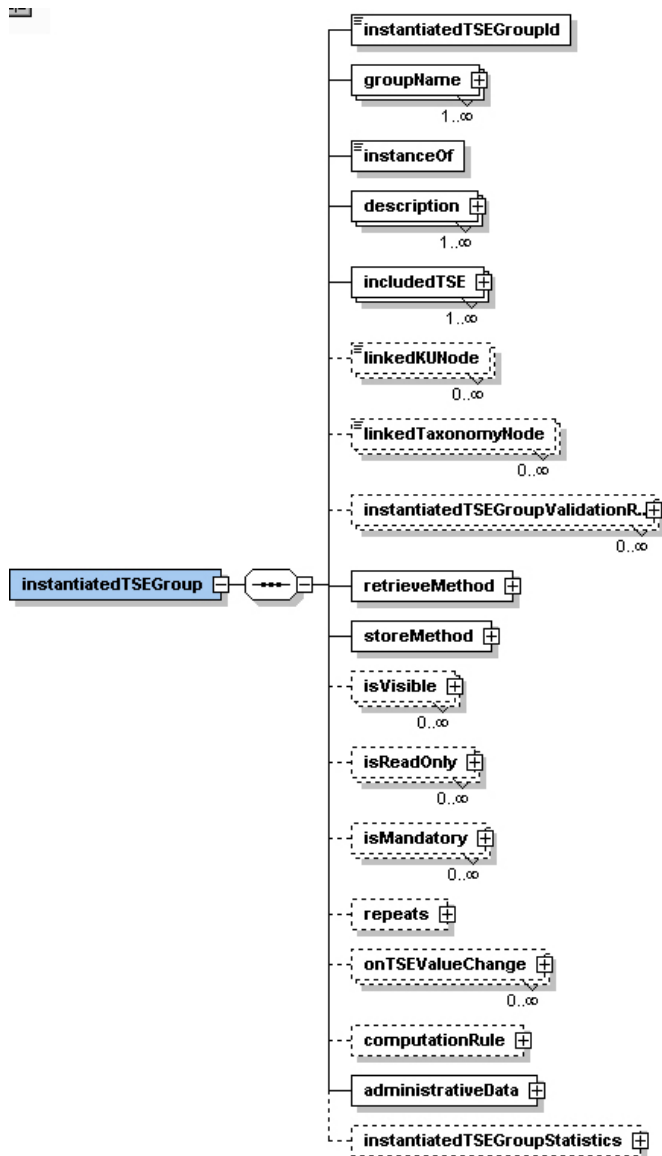
formValidationRule: one or more rules applied during data validation

formLayout: URI (relative or absolute) to the external xHTML file containing the implementation of the form

formRelatedStatistics: a set of Boolean indicating which metrics the service should maintain for this form

3.4.3.3 TSEGroup

A TSEGroup is a logical group of controls and does not necessarily correspond to a visual control (e.g. panel). Apart from the simple case of grouping a set of static controls, a TSEGroup can be used to group together repeatable controls. One such example, is rows in a data grid. In this case, a group represents a row in the grid.



tseGroupId: unique id in the context of the service

groupName: locale-sensitive name that may be used a caption for the whole group of controls

description: locale-sensitive

includedTse: unique ids of individual TSEs contained in the group

linkedKUNode: ids of KUs associated with this group

tseGroupValidationRule: validation rule applicable to all or part of the controls in the group

retrieveMethod: a piece of code used to initialize values in controls of the group

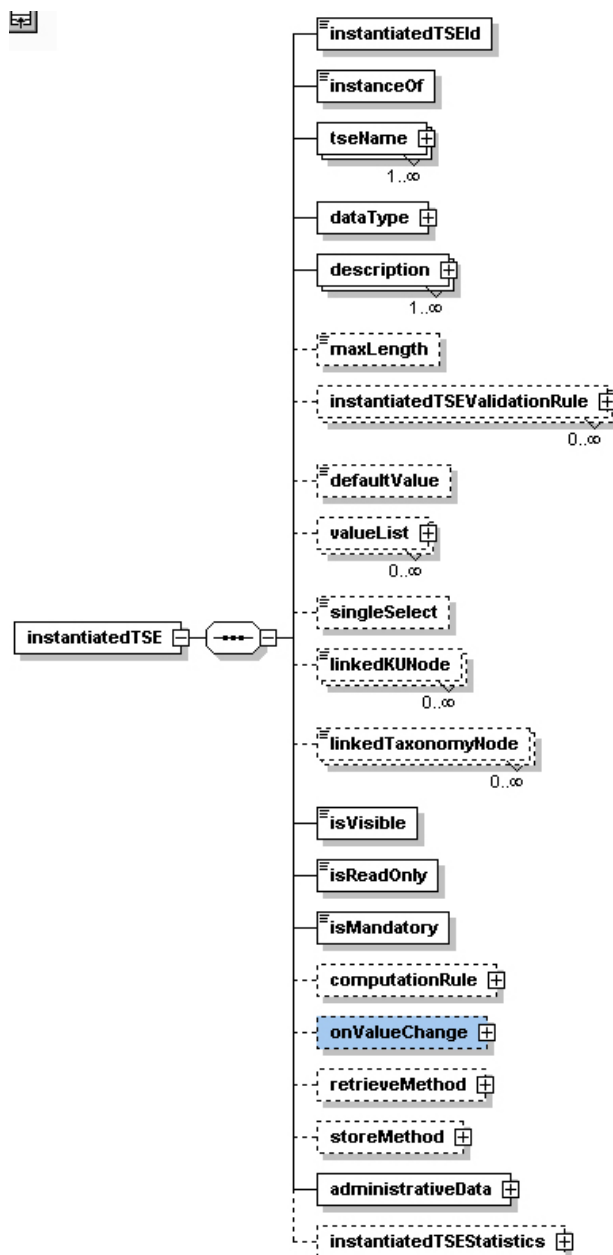
storeMethod: code for storage of control values after validation has been performed

repeats: information on how, if necessary, this group of controls should be repeated

tseGroupStatistics: metrics to monitor for controls in the group

3.4.3.4 TSE

TSE element represents an individual control in a form or a group. The controls data type can be one of the pre-defined in the SmartGov schemas or it can be a custom, user-defined data type. The actual control corresponding to a TSE may require manual user input or it may be rendered as a pick-list of pre-defined values. Data entered are subject to validation.



TSE element

tseId: unique id in the context of this service

tseName: locale-sensitive name, to be used as the control's caption

dataType: may be custom or one of the pre-defined ones

description: locale-sensitive, usually not displayed

maxLength: maximum length in characters, if a textual field

tseValidationRule: one or more rules applied during the validation of the data

defaultValue: initial value of the field

valueList: if this field is existent, the control is rendered as a pick-list. Each entry is composed of has a locale-sensitive label and the actual value.

singleSelect: whether the select list will be rendered as a combo-box or a multi-select list; ignored if *valueList* is not present

linkedKuNode: one or more KUs associated with this control

isVisible: boolean

isReadOnly: boolean

isMandatory: boolean

computationRule: if the value of the field is dynamically computed, this field gives the rule to follow

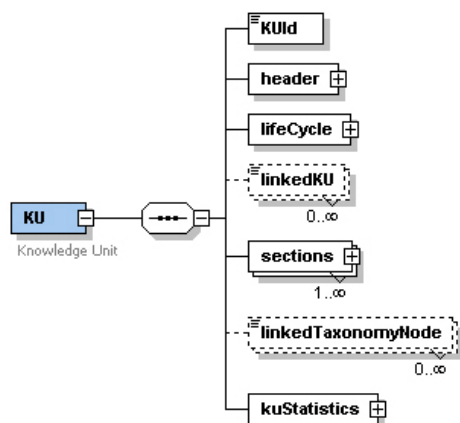
onValueChange: code to execute upon value change

tseStatistics: statistics to monitor for this control

3.4.3.5 KU

A KU is a help item associated with a service element (TS, Form, TSE, TSEGroup).

The actual content may vary from a few lines to whole paragraphs and sections.



KUIid: unique id in the context of the service

header: locale-sensitive title of this article

sections: denote paragraphs of locale-sensitive content

3.4.4 Functionality

A SmartGov service is described in a set of XML files conforming to pre-defined schemas. There is one such schema for each one of the aforementioned elements. This means that for each “instance” of a particular element there is one different file produced by the designer front-end.

These XML files are interdependent and form a logical hierarchy. One such hypothetical hierarchy is depicted in the following figure.

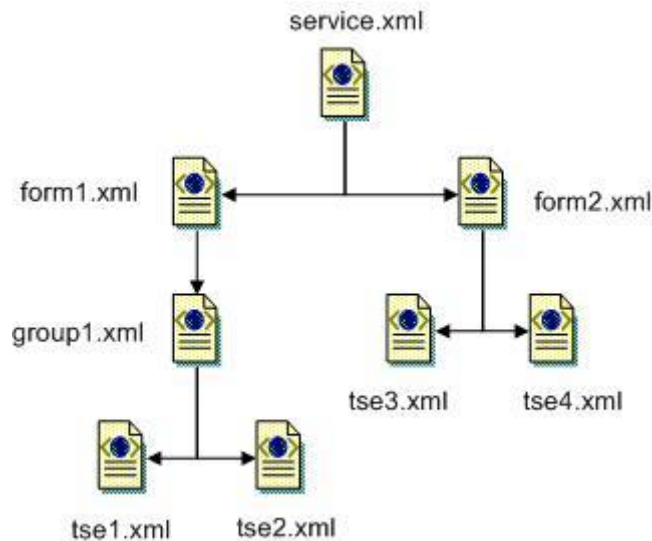


Figure 7 Example of a set of service description files

The relations between the different files resemble foreign key relations in a relational schema. The id field of each element plays the role of the foreign key in these connections. This is the reason why all ids within a service have to be unique. The relations are not “hard”, i.e. imposed and checked by the XML Schema specification or some other mechanism but, instead, they are assumed to be valid. It is up to the Integrator to read the id reference, resolve it and locate the appropriate file.

The XML files will be stored in an XML-aware persistent storage mechanism the XML repository which is described in deliverable D51-D61. This mechanism will accept XPath expressions and return XML documents and will ease the task of resolving element ids into actual XML documents

3.4.5 Establishing links between the form visual elements and SmartGov semantic elements

A form participating in a SmartGov service essentially combines two facets:

1. the visual part, comprising of XHTML elements
2. the semantic part, consisting of links to SmartGov objects, such as KUs, TSEs, TSE groups etc.

These two facets must be integrated in a way that is (a) easy and intuitive for the domain experts to use, with basic only technical skills and (b) is possible to be sequentially processed in order to produce the final service forms, together with the accompanying code. Moreover, it is highly desirable to produce high-quality forms, in order to make the service attractive to the users of its target group.

Taking these facts into account, the SmartGov project has specified a procedure for extending one of the most popular HTML editors, namely DreamWeaver MX, to allow for the integration step to be performed easily by domain experts that only have basic skills in HTML page editing. According to this procedure, domain experts use DreamWeaver MX to specify the associations between visual elements of the XHTML forms and SmartGov platform items. Domain experts select through a click-and-drag procedure the visual elements and then select the associated SmartGov item through intuitive dialog boxes. When these selections have been made, DreamWeaver MX formulates a proper *custom tag* that uniquely identifies the SmartGov platform item, and embeds this tag into the XHTML code. Upon service generation the integrator module recognises these custom tags and arranges for retrieving the information pertaining to the relevant SmartGov platform items from the SKDB and appropriately enhancing form functionality. The procedure, which will be documented in detail in the user manual, comprises of the following steps:

1. the IT staff exports SmartGov items (KUs, TSEs, TSE groups etc) from the SKDB into appropriately formatted XML files
2. The files generated in step 1 are installed in predefined locations, in order to be accessible by the DreamWeaver MX environment. This step is also performed by the IT staff.
3. the domain experts establish the links between the visual XHTML entities and the semantic items of the SmartGov platform by highlighting first the desired XHTML entities and then selecting the SmartGov item that the highlighted elements should be linked to. XHTML entity highlighting is performed through the standard "click-and-drag" methodology of window-based environments, whereas the selection of the

SmartGov platform items is performed via a tree-structured index that may correspond to the organisational taxonomy that has been entered in the SmartGov platform or, alternatively, to the Service/form set/form hierarchy which is used by the SmartGov development environment. It is also possible that both selection paths may co-exist, and the users can make use of the one more suited to their preferences.

A sample screenshot of the procedure for link establishment is illustrated in Figure 8. Full documentation for this procedure will be provided in the User's Manual.

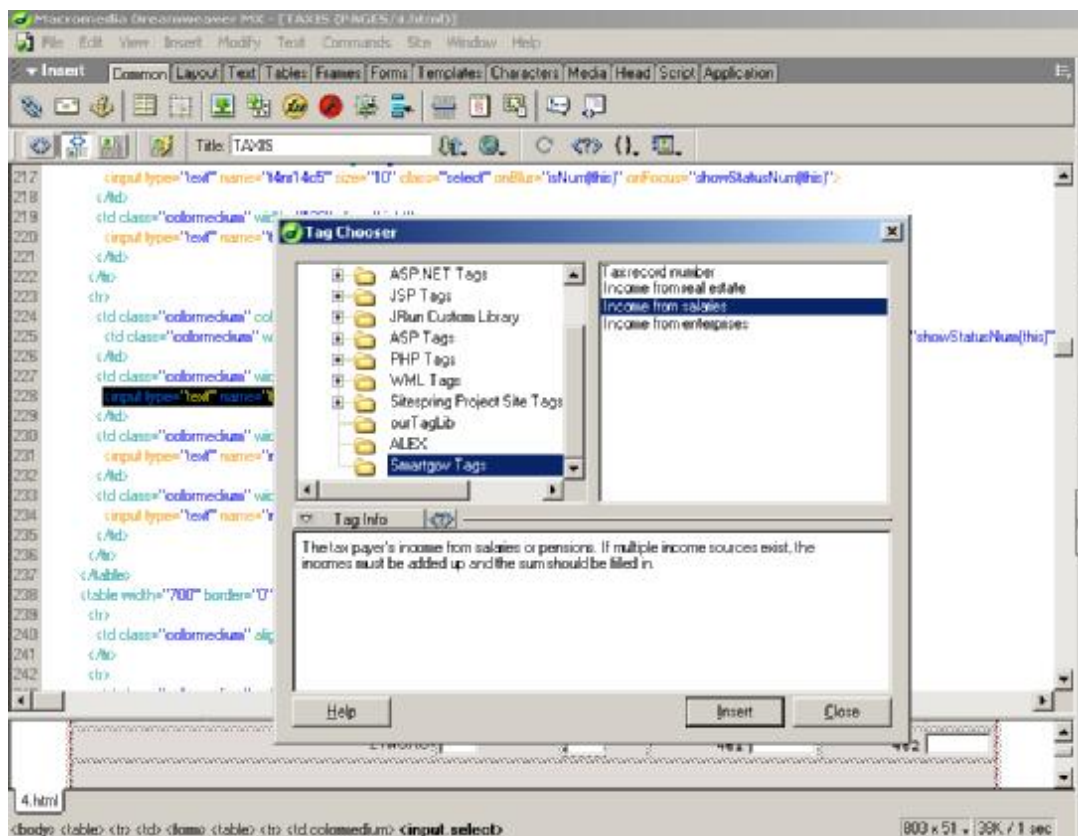


Figure 8 - Link establishment between visual and semantic entities

3.5 Integrator Processing layers

As the creation of the actual service from the descriptors is a complex task, it seemed logical to decouple as much as possible the different logical steps of the service building process. These steps were identified to be the following²:

² Other tasks as compiling code files, copying and deploying to a production server are well covered by Ant, they are not considered here.

Descriptor parsing & In-memory representation

Service element processing logic

Production of necessary files

The following paragraphs cover the different logical components of the Integrator, as they respectively cater for the previous steps. The procedure is also depicted in Figure 9.

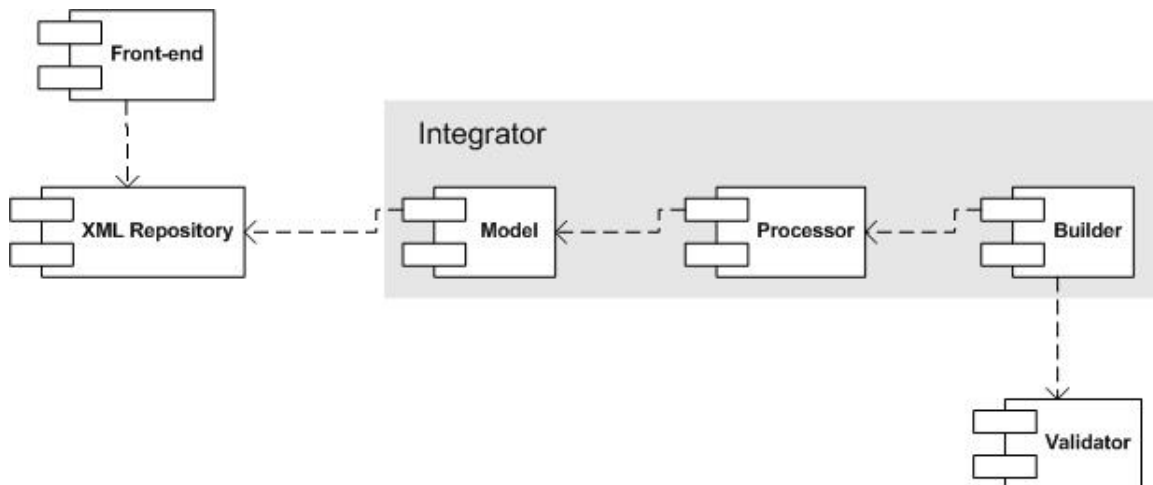


Figure 9 Integrator Layers

3.5.1 Model

As mentioned previously, the service itself is described by a set of XML files that conform to some pre-defined schemas. The existence of these schemas allows the usage of Castor to automatically generate de-serialization classes that will take care of transforming service elements into objects.

However, as the service is not described in one XML file but rather in several “softly” linked files the Integrator component is required to support these “foreign key” relations. For this reason, the model layer, apart from Castor-generated objects, is also required to provide some additional classes that will help complete the object graph that will correspond to the service element graph.

These latter classes are wrappers around Castor-objects. The only additional fields introduced are there to implement object composition relations.

3.5.2 Processor

The intermediate layer between the service model and the actual creation of the necessary service files is conventionally called Processor.

This layer contains the logic that determines which parts of the service object model, created in the previous layer, are processed and by which objects. Classes contained in this component are the place to add/modify logic concerning exactly which service elements are supported and what kind of files are produced.

By convention, this component is also the controller of the serialization process, propagating the request to contained processors and file builders (see next section).

3.5.3 Builder

The final layer in the processing sequence is the file-building layer. Classes of this logical component deal with the creation of the final electronic service files. For this reason, there are classes dealing with the creation of JSP pages, HTML files, Java code files, resource bundles etc. The correlation between model elements and file builder objects is performed at the Processor level.

3.6 Integrator Implementation

3.6.1 Packages

The following diagram shows the different packages of the Integrator component and the interdependencies among them.

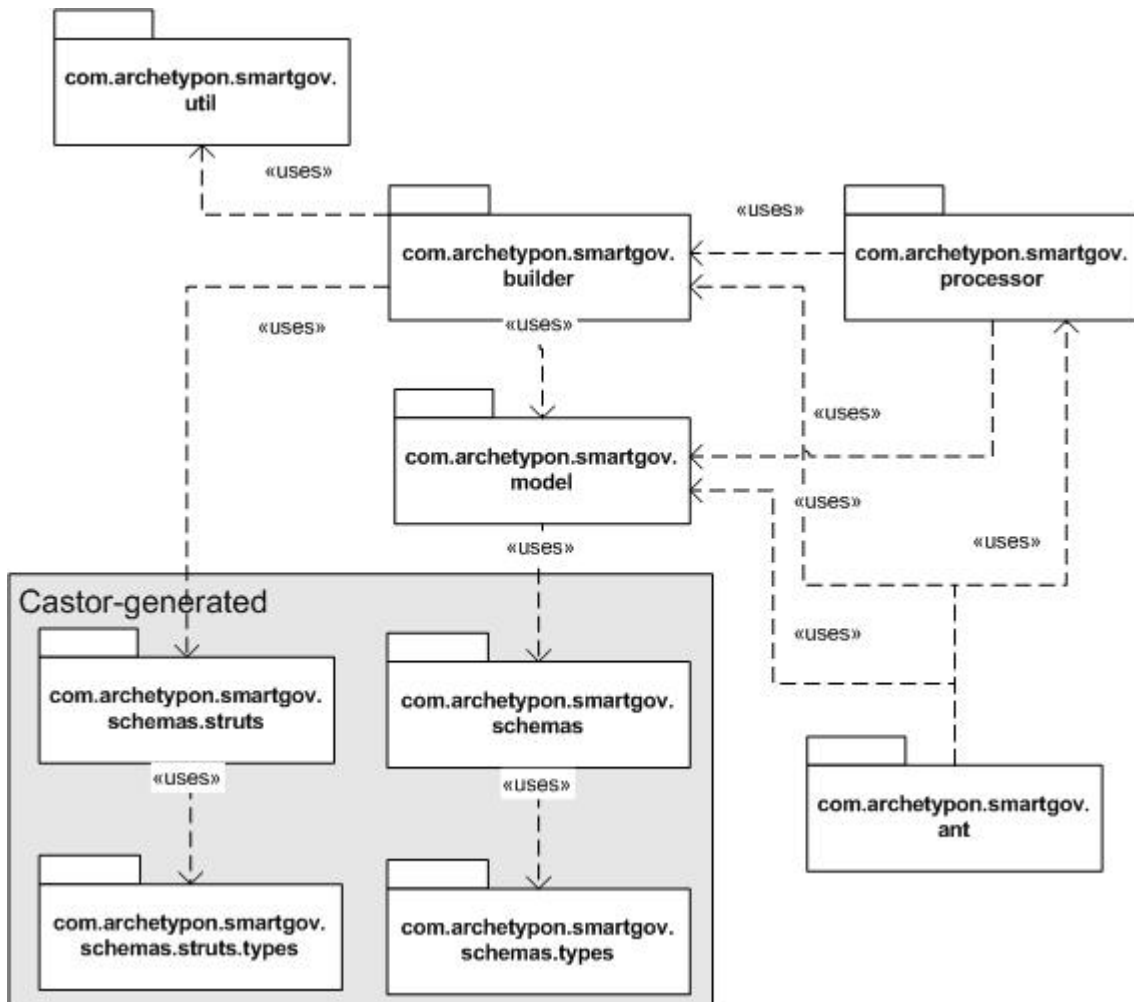


Figure 10 Integrator packages and interdependencies

Below is a short description of each package. For brevity, only the last part of the package name is visible, as indicated in the diagram.

model: along with packages *schemas* and *schemas.types* corresponds to the Model logical component, discussed in the previous section.

processor: classes in this package form the Processor component

builder: corresponds to the Model component

util: contains utility classes, e.g. to abstract file I/O actions

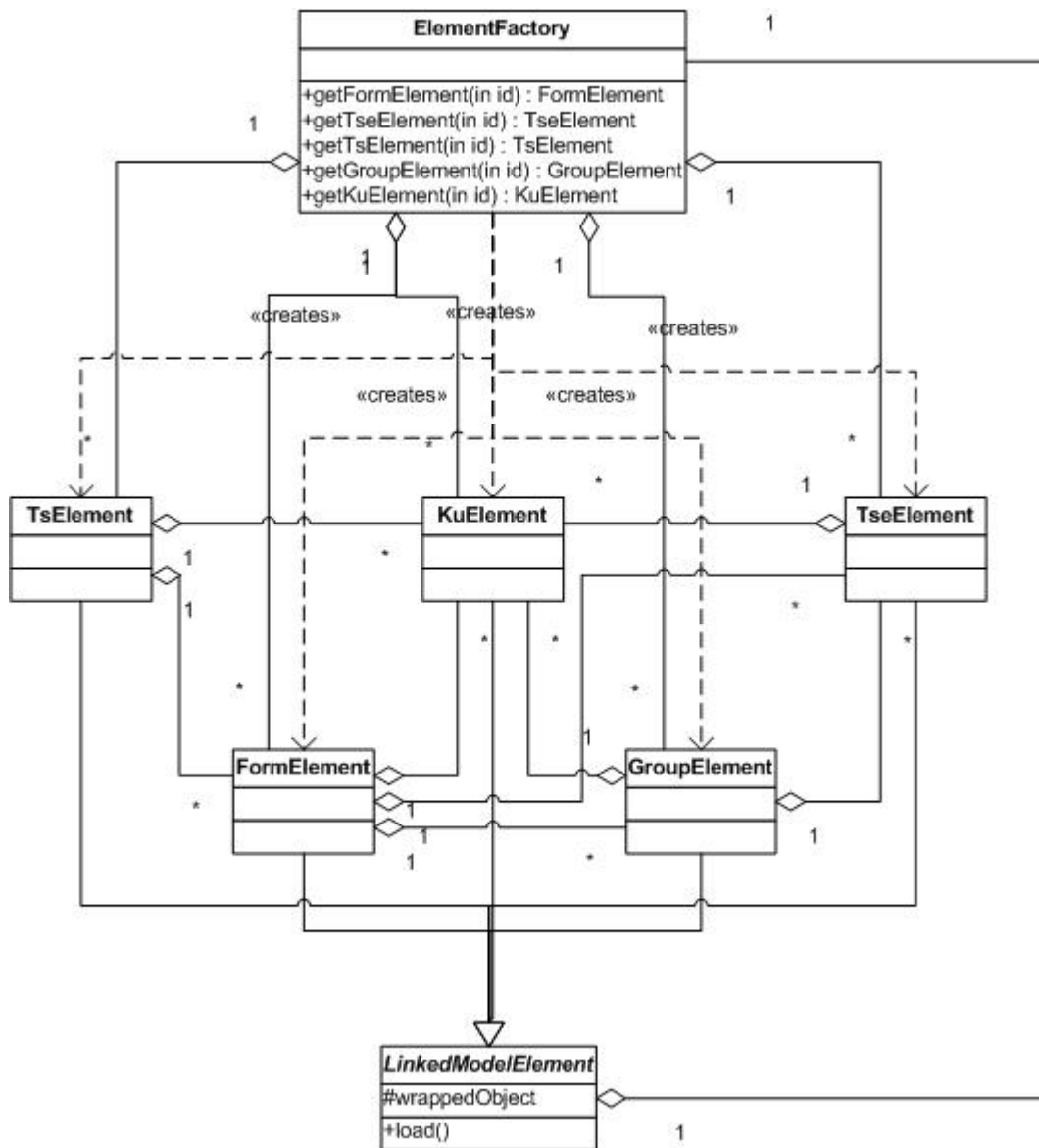
ant: contains Ant task classes that can be used as entry-points in the Integrator process

schemas.struts: along with *schemas.struts.types* this package is automatically generated by Castor to aid in the creation of the Struts configuration file.

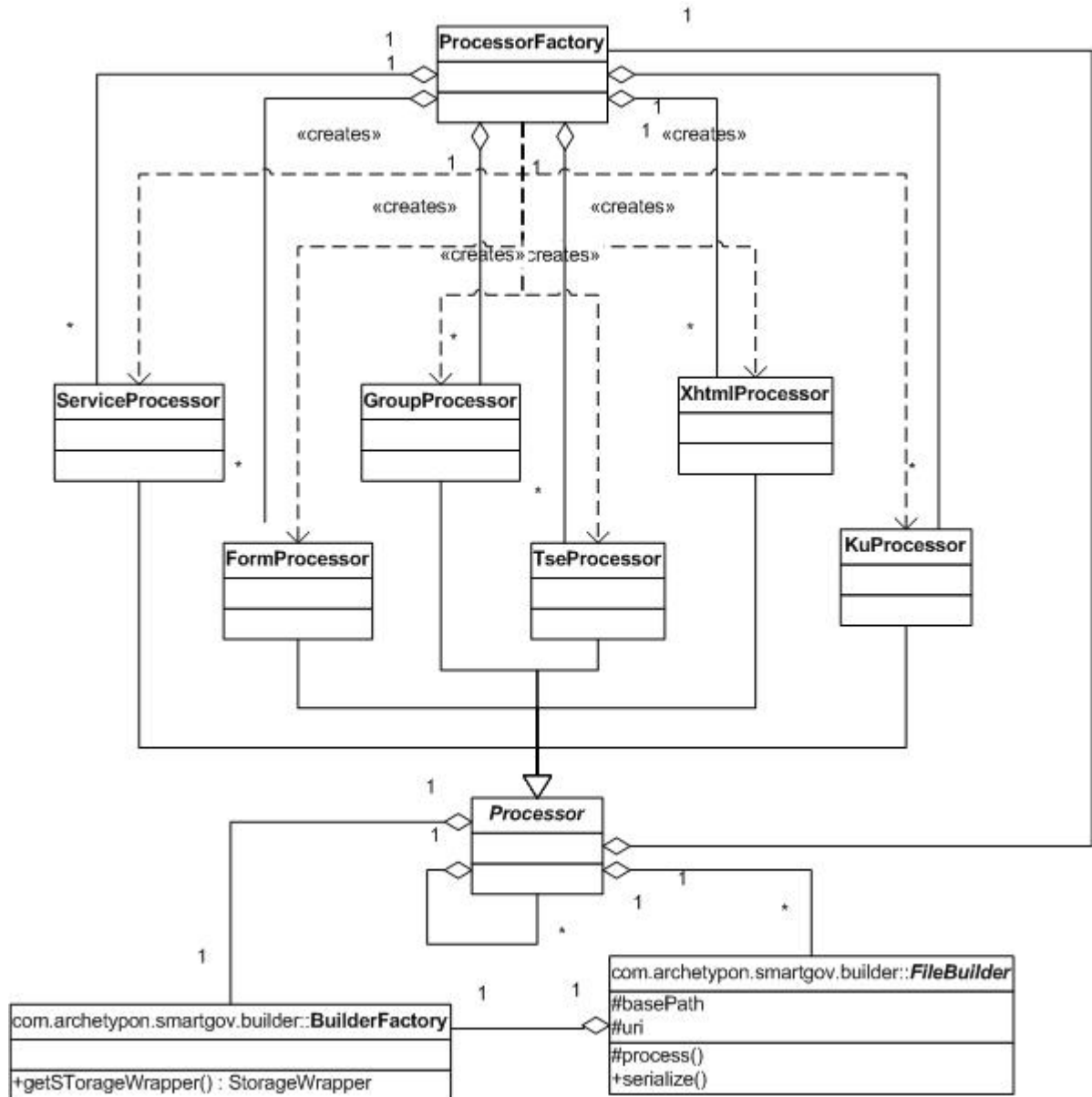
3.6.2 Class diagrams

There has been an effort to follow a consistent design pattern during the implementation of the Integrator. From the previous paragraphs, it becomes obvious that the unique service element identifier is a basic concept in the definition of a service. So, it seems natural to re-use this concept in the object model. For this reason, it was decided to adopt the factory pattern with singleton objects, based on the elements' unique ids. For this reason, almost all Integrator classes are created by a factory method that takes at least one parameter: the unique element id this object corresponds to. Factory objects maintain internal object maps, based on the provided ids. If a requested id already has an object assigned to it, this object is returned.

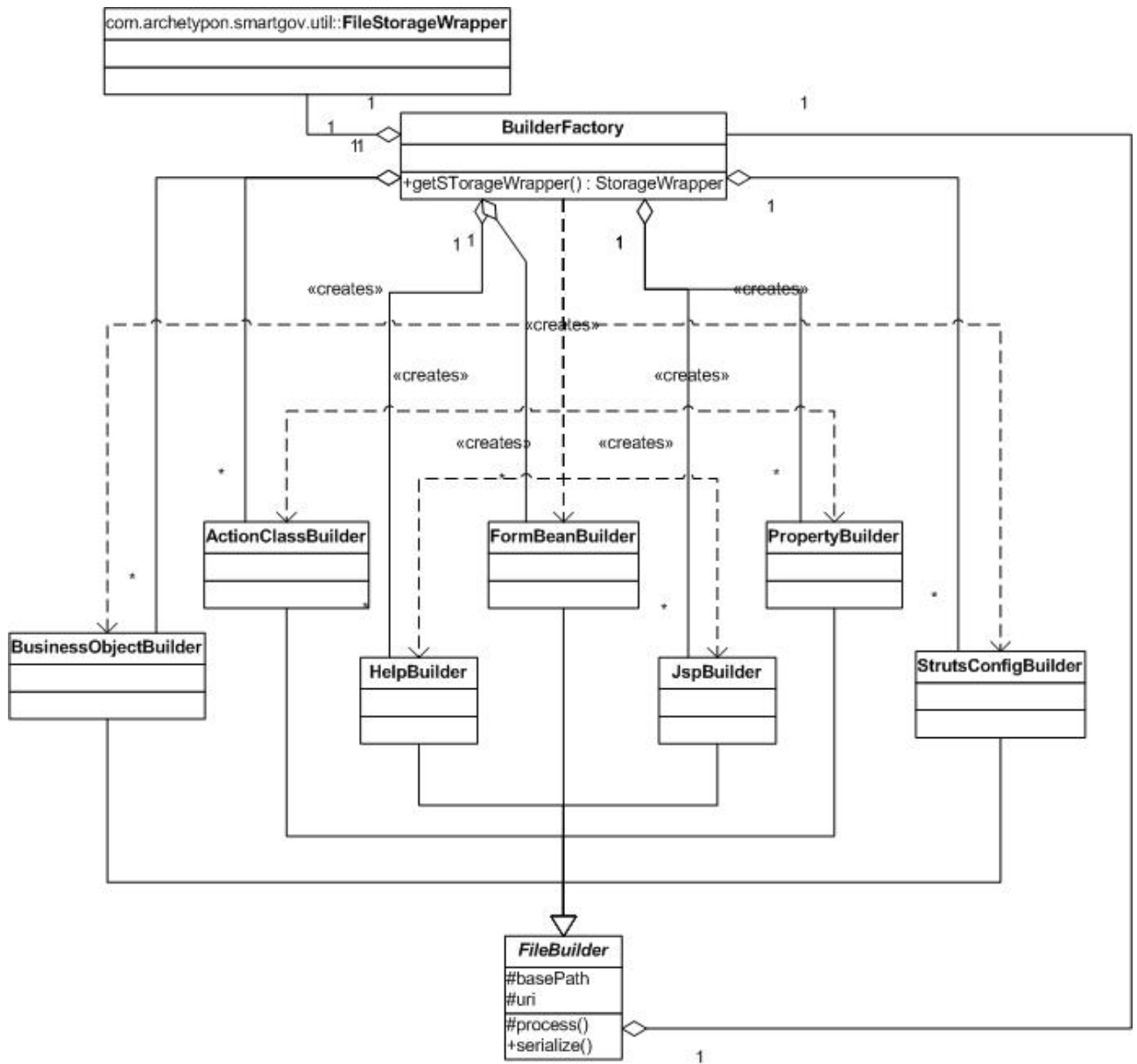
The following diagrams show the classes contained in each of the previously described packages, as well as the interdependencies among them.



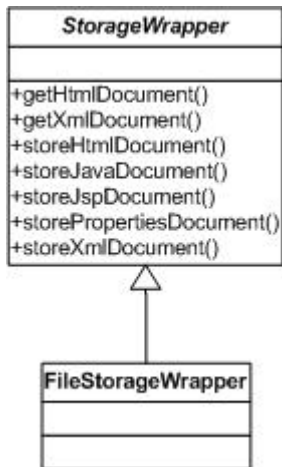
Package com.archetypon.smartgov.model



Package com.archetypon.smartgov.processor



Package `com.archetypon.smartgov.builder`



Package `com.archetypon.smartgov.util`

3.6.3 Sequence diagrams

The following sequence diagrams are here as an example of the internal functionality of the system.

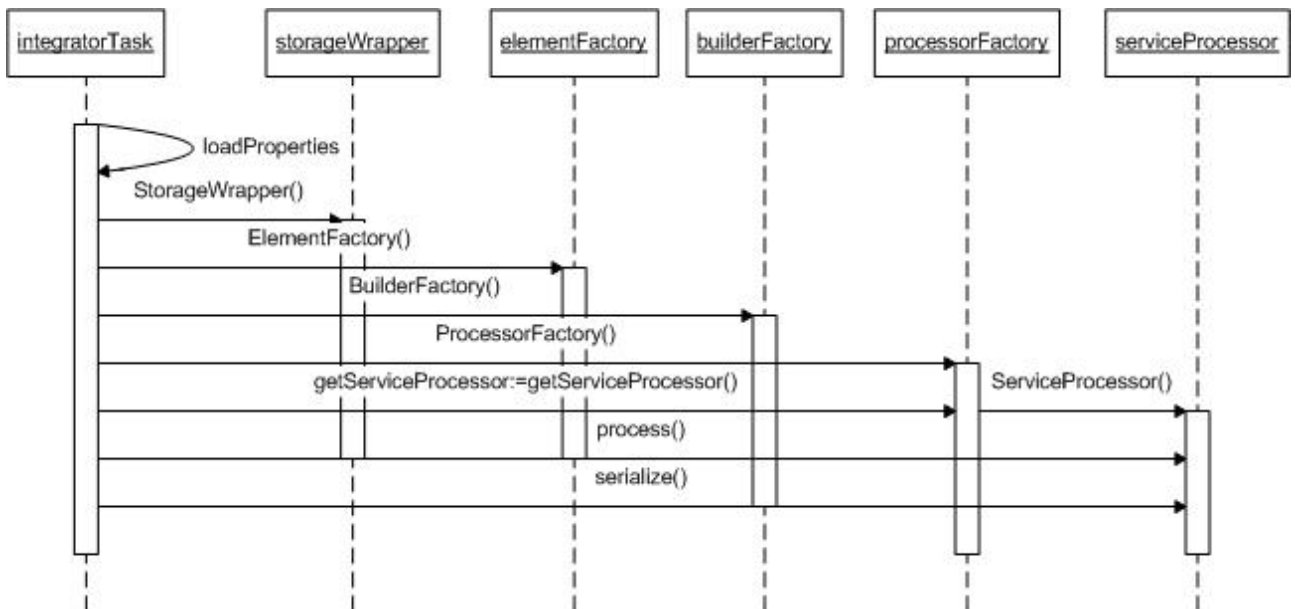


Figure 11 Process launching sequence diagram

The *IntegratorTask* loads externally provided properties and creates all the necessary factory objects. It invokes the *process()* method of the *ServiceProcessor* object for the appropriate service name and, finally, calls *serialize()* which makes all created file builders to dump their contents.

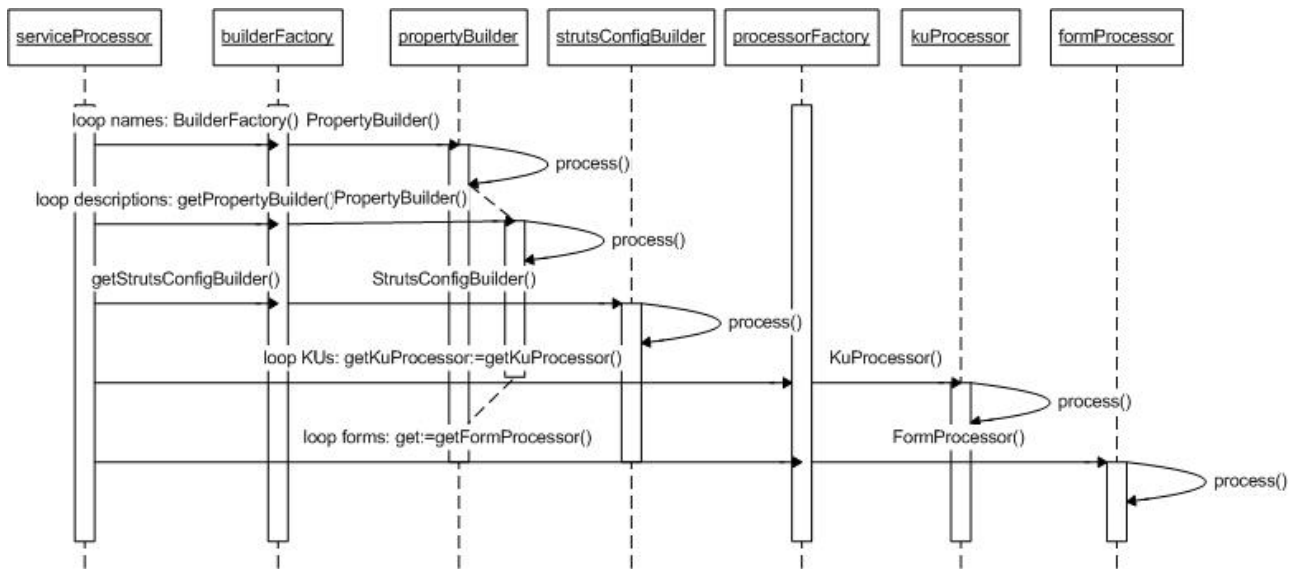


Figure 12 TS element processing sequence diagram

The *ServiceProcessor* object processes all contained XML elements of the TS element and creates the necessary *Processor* and *FileBuilder* objects, invoking their *process()* method.

3.7 Integrator Main Sub-components/Tasks

3.7.1 Help JSP

To provide support for the KUs at runtime, all KUs are transformed into HTML files with a minimal formatting. These files are to be displayed, via inclusion, by a single JSP page. The files are saved with names KUID_language.html. KUID corresponds to the id of the KU file while language to the languages that this KU exists.

This JSP will take the id of the KU as a request parameter (e.g. <http://...?kuid=...>) and displays the proper HTML file, according to the user's preferred language. This is done by examining the accepted languages header of the request and trying to match it with a file. If the file is not existent, the language en is used as a fallback value.

3.7.2 Integrator Ant Task

The functionality of the Integrator component is wrapped inside an Ant task. The task will be provided with the required properties that will be passed to the Integrator's classes.

3.7.2.1 Compilation Task

All Java source files produced by the Integrator are compiled. This process is performed by the corresponding Ant task.

The task will use the output directories of the Integrator task.

3.7.2.2 Deployment Task

This is the final task (or set of tasks) invoked in the launching Ant script. Their "duty" is to deploy all files produced by the Integrator and the compilation task to the appropriate application server directory

3.7.3 Data Storage

After all forms have been submitted and the data successfully validated, the values need to be stored in some way, for future reference.

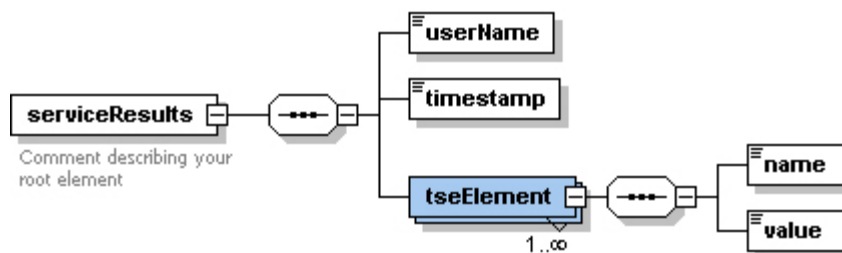
It was initially foreseen that value storage will be performed in a DB server. However, since

the created service will communicate with the back-end agent sending XML documents of the values

the service needs to be as light-weight and decoupled as possible

it has been decided to adopt an XML-based persistence approach. Storage in a DBMS is still possible through the SmartGov agent, which may be used to forward an XML document to the Information Interchange Gateway, where it will be analysed and stored in a DBMS by a pluggable module, specifically written for this task.

The XML storage file follows a simple schema, which is depicted in the following figure



Castor generates a set of classes, able to operate against XML files conforming to this schema. A small Ant script (`data_storage.xml`) is created in the `/scripts`

directory, performing exactly this operation. All generated Castor classes shall be under package `com.archetypon.smartgov.service.data`.

Abstract class `GenericActionForm` extends `ActionForm` with one abstract method `public Enumeration getAllTs()` that returns an enumeration of the classes private members

`ActionFormBean` extends from abstract `GenericActionForm` and implements `getAllTs`

Action casts `ActionForm` parameter with `genericActionForm` and calls `getAllTs` to get the `tseElements` in an instance of `ServiceResults`(castor generated class). Then it populates `userName` and `timestamp` and serializes the file.

By convention, the created XML file is stored in folder `<tomcat>/webapps/<service name>/data` and is named `<timestamp>.xml`, where `timestamp` is the long integer representation of the current `java.util.Date`.

The first *Action* invoked in the series of forms, calls method *ServiceHandler.create* to create a new object if not present.

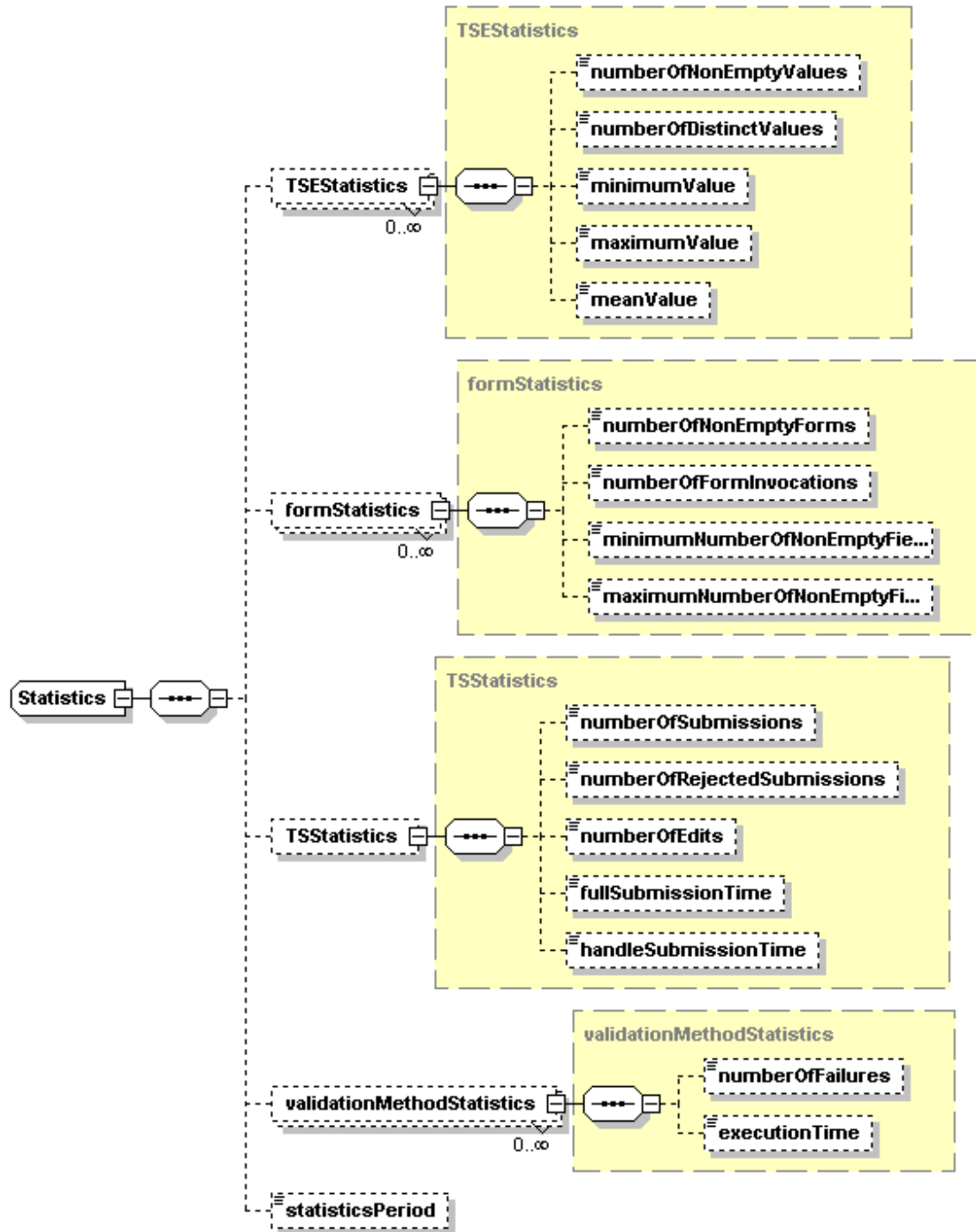
The last *Action* invoked in the sequence of forms, calls method *ServiceHandler.storeData* to save the information.

At run-time the user of the service will be able to retrieve his submitted data and edit them. For this purpose a call is made to the SmartGov agent, which returns the appropriate XML document that was used for storing the data of the specific user. Then this document is processed and its values are inserted within the `ActionForm` bean of the service. Thus the user is presented with all the form controls filled with the values that he previously submitted and he is able to edit them.

3.7.4 Statistics Storage

According to the SmartGov specification, the dynamically created service must be able to monitor some quantitative measures about the service's usage. These statistics are defined in almost all service element types.

Statistics are persistently stored inside an XML file that complies to the following schema



Castor is used to generate classes that can handle XML files conformant to this schema. An Ant script is created to perform this task (statistics.xml). Generated classes are added to package `com.archetypon.smartgov.service.statistics`.

The relevant logic of the dynamically created business object (*TsBusinessObject*) exposes the following public static methods

formSubmitted(String sessionId) Called every time an Action's perform is called; increases the corresponding counter in the table by one

formRejected(String sessionId) Called in the validate method of the ActionForm class; increases the corresponding counter by one

submissionStarted(String sessionId) Called in the perform method of the first form of the service, before any other code. If there is none, creates an entry in a Map object using the session as key and the timestamp as value.

submissionEnded(String sessionId) Called in the perform method of the last form, after all validation code. If there is an entry in the Map, it subtracts the two values and calculates the difference, storing it in the Map. If there is no entry, it ignores the call.

correctionStarted(String sessionId) Called whenever a validation method of an Action or a Business object has located an error. If there is no entry in the Map, one is created with a boolean flag set to true and a timestamp. If there already an entry with the flag set to true, it is ignored. If the flag is false, the flag is set to true and the timestamp is updated

correctionEnded(String sessionId) Called in every perform method. If there is no such entry, it is ignored. If there is an entry with the flag set to false, it is ignored. If there is an entry with the flag set to true, the two timestamps are subtracted, the flag is set to false and the result is added to the previous total correction time.

saveStatistics(String sessionId) Called in the perform method of the last form in the sequence, after all other validation code. If there is a relevant entry it is serialized to the file, otherwise it is ignored.

Calls to the previous methods are present in all relevant objects. Depending on which service definition fields are set to true, the corresponding methods of the service business object are "filled" with code or are left empty.

The file that statistics are serialized is named `serviceStatistics.xml` and is serialized in folder `<Tomcat>/webapps/<service name>/data` (in relative terms, in folder `data`, under the current working directory). Serialization (the call to *saveStatistics*) happens in the last *Action* that is invoked in the series of forms.

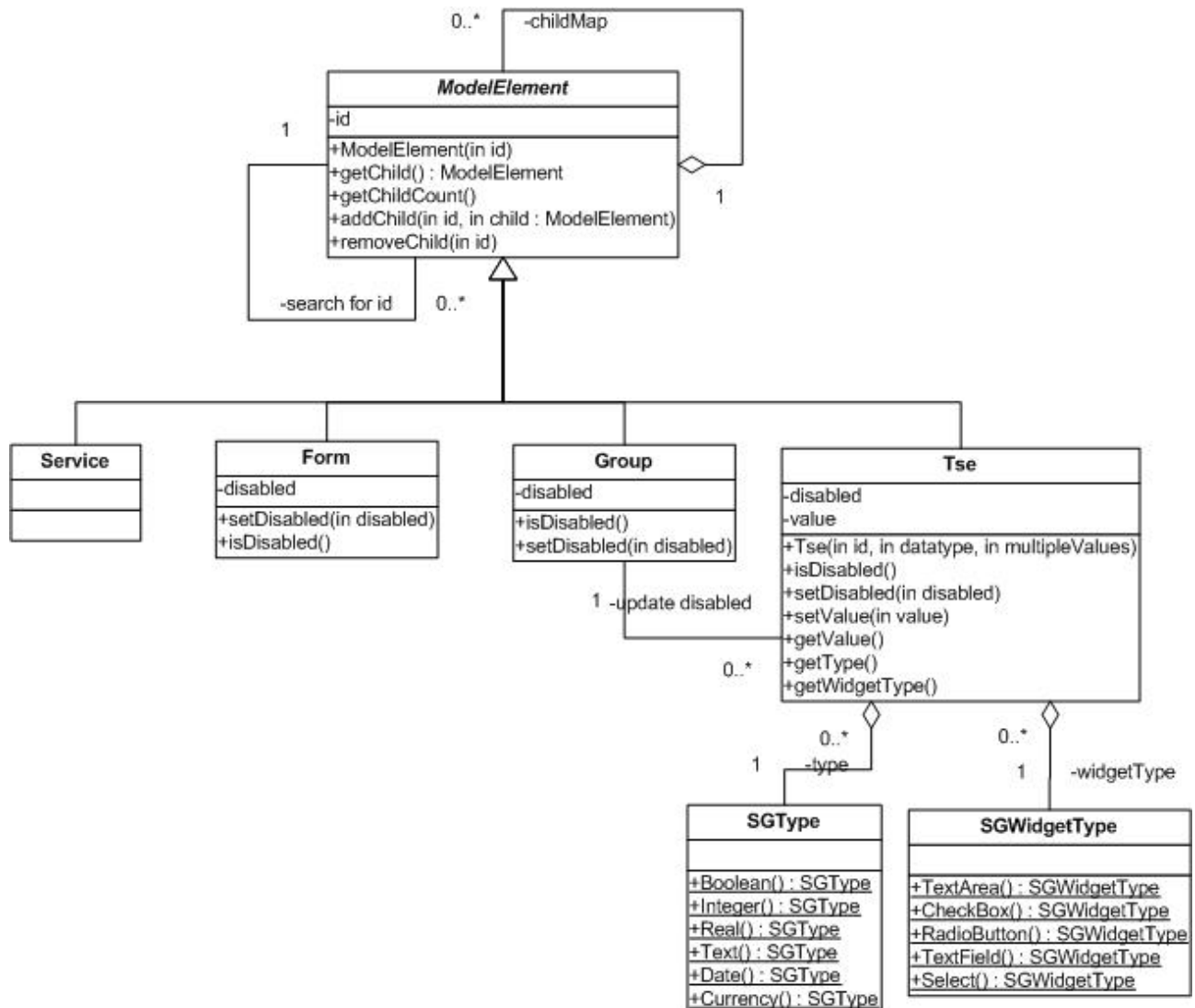
At run-time the statistics of a service will be available to managers by the respective functionality of the Front-end tool. To achieve that the XML file presented here will be transformed by an XSLT file into HTML and presented to the users.

3.7.5 Design Time Service Model

The Translator component needs a way to query the structure of the service during service integration (not run-time). This can be useful, for example, to check if a given element id specified in a rule is valid or to get the data-type of a particular TSE. For this reason, a lightweight representation model of the service is required, providing the ability to query its structure.

To minimize new components and re-use as much as possible, it seems logical to adapt the already-created model. To be able to conform to the new requirements

some changes to the original model will have to be done. These are shown in the following diagram.



Method *public Object getChild(String id)* is recursive. If the id is not found in *childMap*, then all elements in the Map are queried until the id is found and the object returned. If the object is not found, it returns null. There should be explicit test cases with non-existent structure queries (i.e. non-existent ids).

Methods *public void setDisabled(boolean disabled)* of *Group* are propagating³: if the *Group* has entries in the *childMap* their respective methods are called with the same argument. The value field of class *Tse* is of type *java.lang.Object*. It is up to the calling class to determine how to cast the returned object, based on the *datatype* field.

³ Providing information on whether a field is disabled as well as providing propagating methods is beyond the requirements discussed with UoA. However, this functionality is already present and only minor adaptations are required.

Class *TSE* also requires an *int datatype* and a *boolean multipleValues* field in its constructor. The first field is one of the values defined in Castor-generated class *com.archetypon.smartgov.schemas.types.BuiltInDataType* (e.g. *TEXT_TYPE*). The second one is true if the TSE element has multiple values. The first parameter is used to determine the datatype of the TSE element and populate the *datatype* and *widgetType* fields with instances of the corresponding classes. The only exception is if the second parameter is true: in that case, the widget type becomes *SGWidgetType.Select()*.

The *SGType* and *SGWidgetType* classes provide static factory methods that return singleton instances to be used in comparisons.

The class structure above is generic. The instantiated model needs to reflect exactly the fields and state of the service that is being created. This means that during the service parsing and processing phase, this structure will have to be created and be available during processing.

A new class is created, named *ServiceModelBuilder*. It takes a *com.archetypon.smartgov.model.TsElement* as an argument in its *process()* method and instantiates the above model appropriately. The class also has a getter method named *public Service getServiceModel()*, returning the inner variable named *private Service serviceModel*. Its *serialize()* method is overridden and does nothing.

The *BuilderFactory* is modified. A new field named *private ServiceModelBuilder serviceModelBuilder* is added. Also, a new factory method *public ServiceModelBuilder getServiceModelBuilder(<no args>)* is added. This method returns a singleton object, i.e. it first checks if *serviceModelBuilder* exists and if necessary creates it before returning it.

The logic of *ServiceProcessor* is modified. It creates a child builder of type *ServiceModelBuilder* and passes it an object of type *com.archetypon.smartgov.model.TsElement* in its *process()* method. Care should be given to create and invoke this builder before any other processor or builder is called, so as the model is created and available when it is required.

4 The SmartGovLang Language

The SmartGovLang language is a compact, yet powerful language, specially designed to allow domain experts to express validation checks and active behaviour in the context of electronic services. The design goals for the SmartGovLang language include the following:

1. the SmartGovLang language should be simple enough to be usable by domain experts, who do not necessarily have programming skills
2. the SmartGovLang language should cover various aspects of active behaviour needed in the context of electronic services, including:
 - a. modelling of validation checks which should hold for the data elements entered by the user, with appropriate issuing of error messages when validation checks fail.
 - b. ability to emit alert messages to the user when certain conditions hold; these situation do not correspond to errors that will preclude the user to submit the data, but rather typical omissions or misinterpretations that are detected.
 - c. provisions for displaying informational messages that guide the user in the process of filling in and submitting the electronic document.
 - d. capability to enable/disable certain form fields, depending on the values entered in other fields (e.g. when "marital status" is set to "Single", fields related to spouse's data should not accept input).
3. The SmartGovLang language should be able to model validation checks and active behaviour in various scopes, such as single transaction service element, transaction service element group, form and transaction service.
4. the SmartGovLang language should be easily mapped to an intuitive user interface, facilitating the work of domain experts.
5. the SmartGovLang should *not* necessarily be able to express *all* possible validation checks and interactions. This should lead to a complete, general-purpose programming language which would only be usable by programmers. Rather, it should be able to easily express the most common validation checks and allow for usage of "native" executable code when the required checks or behaviour cannot be modelled.
6. the SmartGovLang language should be translatable to languages that can be executed in the environments of service clients (typically web browsers) and in

the environment of service providers (typically web servers). Especially for the case of service clients, it is desirable that the generated code remains small and compact to minimise download time.

7. it is desirable that the SmartGovLang language is extensible. Extensions at this level cannot be performed by domain experts, and the involvement of IT staff will be required.

In the following paragraphs, the SmartGovLang language is specified and the mechanisms for translating SmartGovLang expressions to executable languages are described.

4.1 Specification of the SmartGovLang language

The SmartGovLang language provides two broad categories of syntactic constructs, namely *full rules* and *compact rules*. A full rule comprises of two parts, the *condition* and the *action*, where:

- *condition* is a logical expression evaluating to *true* or *false*; the *condition* may reference one or more form elements (i.e. pieces of data that may be entered by the user) as well as global context information (e.g. the login name of the user, the current state of the document etc). It is also possible for the conditions of the SmartGovLang language to use a number of built-in functions.
- *action* consists of one or more tasks that should be carried out when the condition evaluates to *true*.

The two parts of a SmartGovLang rule are separated by the => character sequence ("equal" sign, directly followed by the "greater than" sign).

Compact rules provide shorthands for full rule, aiming to facilitate the work of domain experts, who will be thus able to express validation checks intuitively and efficiently. Compact rules cover the most commonly used validation checks, allowing for emission of error and warning messages. Compact rules do not cover active behaviour, which requires the usage of full rules. Full rules and compact rules are described in detail in the following paragraphs, whereas a BNF notation for the SmartGov rule grammar is provided in Appendix A.

4.1.1 Specification of full rules

In the following paragraphs the specification of the SmartGovLang full rules is described.

4.1.1.1 The condition part

The condition part of a SmartGovLang rule states the prerequisites for the corresponding action part to be executed. The condition part is a logical (boolean) expression, which must evaluate to *true* or *false*. Logical expressions may be:

1. *simple*, consisting of a single term only, or
2. *complex*, comprising of multiple simple expressions combined using the *AND* and *OR* logical operators. A simple expression may also be negated using the *NOT* logical operator, and parentheses may be used to specify arbitrary groupings of expression combinations.

A *simple expression* is effectively a comparison between two quantities using one of the relational operators =, <> (not equal), <, <=, > and >=. Each of the compared quantities may be

1. *a constant* (numeric –e.g. 10, -2.4- or an alphanumeric literal enclosed in quotes –e.g. "MARRIED", "GREECE").
2. *a form element* within the current scope (e.g. within the current TSE group or the current form).
3. *a global context value*, e.g. the state of the current document (submitted, pending, finalised etc). A list of context values will be provided.
4. *a function* accepting one or more parameters, with one of them typically being a form element or a global context value. For a list of available functions see appendix B.
5. a combination of the above, through arithmetic operators (+, -, *, /, %) and function composition (i.e. a parameter to some function is the result of another function).

A special case of a simple expression is the literal *true* (not enclosed in quotes), which may be used for unconditional execution of the corresponding action part of the rule.

4.1.1.2 The action part

The action part of a SmartGovLang full rule is a sequence of one or more actions that will be performed when the respective condition evaluates to *true*. The specification of each action is terminated using the semicolon character (;) and the allowable specifications are as follows:

1. *errorMessage(message)*. Marks an error and arranges for the presentation of a suitable error message to the user, which is provided as the parameter within the parentheses. In the presence of marked errors the user may not be able to

proceed to the next stage of the transaction service, until the errors are corrected.

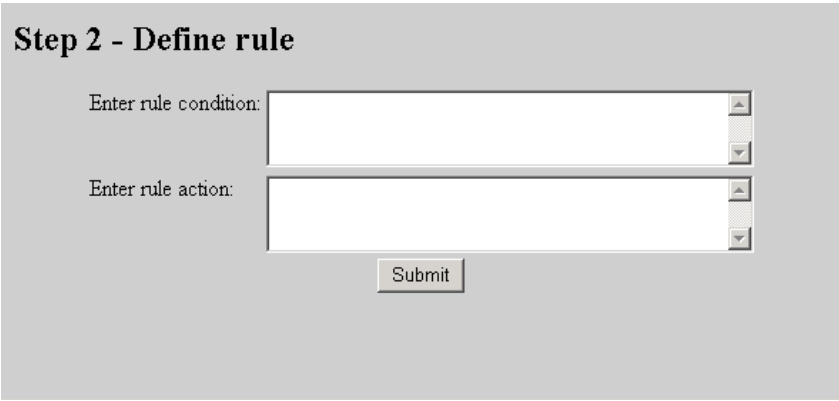
2. *warningMessage(message)*. Arranges for the presentation of a message to the user, which is provided as the parameter within the parentheses. The purpose of the message is to draw the user's attention to a specific issue and does not preclude the user to proceed to the next stage of the transaction service.
3. *informationMessage(message)*. Arranges for the presentation of an informational message to the user, which is provided as the parameter within the parentheses. The purpose of the message is to inform the user of the current transaction service state and/or to guide the user along the various stages of the transaction service.
4. *disableField(fieldId)*. Disallows the user to enter values to the designated form field, typically because of values entered in some other form field. For instance, if the user enters "Single" in the marital status field, the field labelled as "Spouse name" should not accept input.
5. *enableField(fieldId)*. Allows the user to enter values to the designated field.
6. *setField(fieldId, value)*. Sets the designated field to the specified value.
7. *setFocus(fieldId)*. Moves the input focus to the designated field. This action has no effect when the validation check is executed at the back-end, since no notion of focus is applicable in this context.

Since the SmartGov platform targets multilingual environments, it is necessary to provide the ability to specify messages (error, warning and informational) in multiple languages. A number of options are available for supporting this feature:

1. provide an appropriate interface through which the rules will be entered; the interface will then prompt for each message in all supported languages.
2. require that a *message identifier*, rather than the message itself, will be entered in actions related to messages. In a separate procedure the correspondence between the message identifier and the messages in the different languages will be specified.
3. impose an internal structure to the message, allowing it to host specification of the message in all languages. In such an approach, the message specification would actually be list of pairs (*languageId, messageText*), properly separated, and the translating module will arrange for mapping this representation to appropriate code structures. In such a case the error message for *Wrong data* in

English, Greek and Spanish could be entered as “EN::Wrong data|EL::Λάθος δεδομένα|ES::Datos incorrectos”, with the vertical bars separating representations in different languages and the double colons separating the language designation from the actual error message within a single pair.

Since the specification of full rules is performed entirely in textual manner (and mainly by IT staff), the first approach is considered inappropriate, since it would replace straightforward text input with a multi-stage point-and-click procedure. The second approach also requires two distinct stages with no direct link between them, thus it is considered counter-intuitive and error prone (some message identifiers could be left undefined, or some updates could be missed). Thus, the third alternative is opted for, which allows for pure textual input and keeps all definitions gathered in a single place. A full rule may be entered as two text portions, accounting for the condition part and the action part, as illustrated in the following figure:



Step 2 - Define rule

Enter rule condition:

Enter rule action:

Figure 13 – Sample interface for defining full rules

4.1.2 Specification of compact rules

Compact rules are effectively shorthands that may be used instead of full rules, allowing domain experts to express common validation checks in an efficient manner. Moreover, compact rules have been designed to facilitate their entry through intuitive, form-based user interfaces, removing the need for knowledge of any syntactic constructs. A compact rule of the SmartGovLang language consists of three parts:

1. the *severity designation*, classifying the validation check as *information*, *warning* or *error*. Warnings only emit alert messages to the service user, while errors may prevent the user from moving on to further stages within the transaction service.
2. the *message* which will be displayed to the service user in the case that the validation check fails.
3. the actual validation check, which may take one of the following forms:
 - a. *between(fieldId, lowerLimit, upperLimit)*. The value of the designated field is required to be between the lower limit and the upper limit (both limits are inclusive). This form is equivalent to the full rule
 COND: (fieldId >= lowerLimit) and (fieldId <= upperLimit)
 ACT: notification_message(validationMessage)⁴
 - b. *requires(fieldId1, fieldId2)*. If a value is entered in the field *fieldId1* then a value should be also entered in the field *fieldId2* (i.e. it should not be left blank). This form is equivalent to the full rule
 COND: (length(fieldId) > 0) and (length(fieldId) > 0)
 ACT: notification_message(validationMessage)
 - c. *precludes(fieldId1,fieldId2)*. If a value is entered in the field *fieldId1* then no value should be provided for field *fieldId2* (i.e. it should be left blank). This form is equivalent to the full rule
 COND: (length(fieldId) > 0) and (length(fieldId) = 0)
 ACT: notification_message(validationMessage)
 - d. *requiresMulti(fieldId1, fieldIdList2)*. If a value is entered in the field *fieldId1* then at least one of the fields referenced in *fieldIdList2* should be filled in. This form is equivalent to the full rule
 COND: (length(fieldId) > 0) and ((length(fieldIdList[1]) > 0) or (length(fieldIdList[2]) > 0) or ...)
 ACT: notification_message(validationMessage)
 - e. *checkRow(fieldIdList)*. This check fails if a value has been entered for some of the fields referenced in the field list while some other fields remain blank. In other words, it is required that either *all* fields have a value or none of them has. This validation check is useful for checking the completeness of table rows. This form is equivalent to the full rule

⁴ notification_message will map to either error_message or warning_message, depending on whether the severity of the check is set to *error* or *warning*. validation_message is an appropriately bundled version of the rule message (item 2).

```
COND: (length(concatenate(fieldIdList[1], fieldIdList[2], ...) = 0)
or (length(fieldIdList[1] > 0) and (length(fieldIdList[2] > 0) and
...)
```

```
ACT: notification_message(validationMessage)
```

- f. *checkRelation(fieldId1, operator, fieldId2, c)*, where *operator* is one of the relational operators ($=$, \neq , $>$, \geq , $<$, \leq) and *c* is a constant value. This rule checks whether the condition *fieldId1 operator fieldId2 * c* holds; if not the associated message is displayed as a warning or error message. This form is equivalent to the full rule

```
COND: (fieldId1 operator fieldId2 * c)
```

```
ACT: notification_message(validationMessage)
```

The compact rules may easily be entered and edited through an intuitive user interface through which the domain experts will only specify the check type and pick the fields that participate in the validation check. A screenshot of such an interface is presented in the following figure.

Step 2 - Enter the validation rule

Field should be between and

If field has a value then field must have a value as well.

If field has a value then at least one of the fields must have a value as well.

If field has a value then field must **not** have a value.

The user should provide a value for all of the following fields or for none of them

Figure 14 – Sample interface for defining compact rules

4.2 Translation of the SmartGovLang language

The SmartGovLang language should be translated, during service instantiation, to the languages that will implement the back-end functionality (i.e. the functionality at the organisational system offering the electronic service) and, where appropriate, the front-end functionality (i.e. the functionality at the devices employed by the users to access the electronic services). According to the SmartGov platform specifications, the back-end functionality will be implemented using the Java language whereas the most prominent dissemination channel will be the Web, therefore translation to the Javascript language, implemented in most web browsers, will be supported. The code generator architecture will remain open to accommodate translations to other languages (e.g. PHP, ColdFusion etc).

The translation of SmartGovLang rules to the target language will be handled by a distinct software module, the SmartGovLang translator. The SmartGovLang translator will constitute of an abstract class, which will have one concrete interface for each of the languages that translation is required for. The software module that needs to generate concrete code will request the creation of an appropriate object from a factory, specifying the language to which translation is required. The factory will then return an object, whose methods will be invoked to implement the translation.

In the context of each such invocation the following information must be provided:

1. an object describing the SmartGovLang rule that needs to be translated.
2. an object describing the form fields that are valid in the context of the validation checks. This object should implement the following functionality (class and method names are indicative only in this stage):

➤ *Field getField(String FieldId) throws SGNonExistentField;*

This method reports whether the designated field identifier is valid in the context of the rule. If the field identifier is valid, an object of type *Field* describing the field properties is returned; if the field is not valid, the *SGnonExistentField* exception is thrown.

➤ *Enumeration getFields()*

This method returns an enumeration (list) of the fields accessible through the scope; if no fields are accessible, then the return enumeration is empty.

The *Field* type object that is returned by the previous method should provide interfaces that will allow it to be queried, *at least* for

- (a) the name via which the generated code may reference the element (for example the field *fld100* may be referenced as *textbox2* in the HTML context and as *userInput.getObject("fld100")* in the context of the server-side Java code)
- (b) its expected data type (numeric, alphanumeric, date etc) and
- (c) the client-side widget that is used for allowing the user to edit its value (e.g. text box, check-box, radio button). This is required for generating code to be used at the client side, since different widgets at the client side require different handling to obtain or set their values.

According to the above, the interfaces that need to be supported by the Field object are:

- *String getName(String langId);*
 - *SGType getType(String langId);*
 - *SGWidgetType getWidget(String languageId);*
3. an object describing the global context variables that may be referenced within the expression. This object should be able to report whether an identifier belongs to the current context value through a method
- *SGContextVariable getContextVariable(String ctx) throws SGINvalidContext;*
- (its functionality is similar to the functionality of the *getField* method described in (2) above, while the *SGContextVariable* object should support the following two methods:
- *String getName(String langId);*
 - *SGType getType(String langId);*
- reporting the name via which the generated code may reference the global context element and its expected type, respectively.

An alternative to passing these parameters upon invocation is the provision of a suitable class with static methods, which will implement the same functionality.

As a result of such an invocation, the translator object will return the concrete code in the requested language or throw a suitable exception in the event of an error (e.g. malformed syntax or usage of an unknown identifier).

In more detail, the process of translating the validation checks is as follows:

1. The integrator reads from the XML repository the XML representations of the validation checks⁵ and constructs appropriate Java objects. These objects are

⁵ The XML schema for the validation checks is described in section 4.2.1

instances of either the `SGFullRule` or the `SGCompactRule` class, which are both extend the `SGRule` abstract class.

2. The integrator uses the `SGLangTranslatorFactory` class to create one or more instances of the `SGLangTranslator` class. The `SGLangTranslatorFactory` class provides two static methods, namely `createClientTranslator` and `CreateServerTranslator` for generating code running on the client side and the server side, respectively. Both methods accept a `String`-typed parameter indicating the target language and return an object of type `SGLangTranslator`.
3. For each validation rule either in compact or full format, the integrator invokes the `translate` instance method of the `SGLangTranslator` class, using the `SGLangTranslator` objects returned in step (2). In other words the `SGLangTranslator` class implements the following two methods:

```
SGTranslatedCode translate(SGFullRule fr);  
SGTranslatedCode translate(SGCompactRule cr);
```

4. Upon successful completion, the `translate` methods return an `SGTranslatedCode` object, which contains the following elements (with the appropriate get methods, documented in section 4.2.2):
 - i. a `String`-typed element named `concreteCode` which is the result of the translation to the target language.
 - ii. an element of type `SGMultilingualMessages` named `messages`, which contains the messages associated with the validation rule. For compact rules, these messages are directly extracted from the related `SGCompactRule` object; for full rules, these are derived from the parameters to the `errorMessage`, `warningMessage` and `informationMessage` action elements. The `SGMultilingualMessages` interface implements the following methods, facilitating access to the actual messages:

```
Enumeration getMessageIds(void);6  
Enumeration getLangIds(void);  
String getMessage(String messageId, String langId);
```

The integrator may invoke the `getMessages` and `getLangIds` methods of the `SGMultilingualMessages` interface to retrieve the message identifiers and the language identifiers used within a validation check. Then, for each combination of a message identifier and language identifier, the `getMessage`

⁶ The Enumeration-typed results of these methods may be treated as strings.

method may be invoked to retrieve the relevant message in the specific language.

Figure 15 illustrates an example of SmartGovLang rules translation into Java code, for server side processing and Javascript code, for client-side execution. Translation results from multiple rules are simply executed sequentially.

SmartGovLang rule: COND: (isMarriedFld = 0) and (wifeSurnameFld <> '') ACTION: errorMessage('EN::In order to complete the wife's name, the "Married" indication must be checked'); setfocus(isMarriedFld);
Java Translation if ((this.isMarriedFld == 0) && (!(this.wifeSurnameFld.equals('')))) { // errorMessages is a global context variable holding all validation // error messages for a single submission. "m1254" is the id if // the validation message, which is assigned at service generation // time. The message in the appropriate locale will be extracted // from the service resources and presented to the user. this.errorMessages.add("m1254"); // setfocus is not meaningful in the back-end }
Javascript Translation if ((form1.isMarriedFld.value == 0) && !(form1.wifeSurnameFld != '')) { // the appropriate message for the user locale has been selected // and been planted into the code by the dissemination server alert("In order to complete the wife's name, the 'Married' indication must be checked"); form1.isMarriedFld.focus(); // terminate checks execution, inhibit further actions (link traversal // or form submission) return false; }

Figure 15 - Translating a SmartGov rule to Java and Javascript

4.2.1 Storage of the SmartGovLang language

In the following paragraphs the XML schema for storing validation checks within the SmartGov repository is documented. The XML schema includes provisions for storing validation checks expressed in SmartGovLang (both in compact and full form), but additionally checks in some native language. The conventions followed in the XML schema is as follows:

1. A validation check defines (a) a designation whether the check should be performed at the back-end only or both at the front-end and at the back-end and (b) the actual validation rule.
2. the validation rule may be expressed:
 - a. as a compact SmartGovLang rule, in one of the forms listed in 4.1.2. In this case the rule will be complemented by (i) a *severity designation* indicating whether the validation rule should be followed mandatorily (*error level* severity) or provisionally (*warning level* severity) and (ii) the message that should be emitted if the validation check fails. The error message is a multilingual resource.
 - b. as a full SmartGovLang rule. In this case, in particular, the error messages and the severity are disregarded, since this information is contained within the rule.
 - c. as a piece of native code (e.g. java, JavaScript, etc). If the rule should only be checked at the back-end, the implementers need to provide a Java implementation; if the rule should be checked both at the front-end and back-end, a JavaScript implementation should be provided as well. The XML schema includes provisions for accommodating checks in other languages as well, to support system extensibility. Rules expressed in this form should adhere to the guidelines provided for using native languages within the SmartGov platform.

The XML schema for storing SmartGovLang rules follows.

```

<xs:complexType name="nativeCodeFragment">
  <xs:sequence>
    <xs:element name="langId" type="xs:string"/>
    <xs:element name="usefulFor">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="front end"/>
          <xs:enumeration value="back end"/>
          <xs:enumeration value="front end and back end"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <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="SGbetweenCheck">
  <xs:sequence>
    <xs:element name="fieldId" type="xs:string"/>
    <xs:element name="lowerBound" type="xs:string" minOccurs="0"/>
    <xs:element name="upperBound" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="SGrequiresCheck">
  <xs:sequence>
    <xs:element name="fieldId" type="xs:string"/>
    <xs:element name="requiredFieldId" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

```

```

<xs:complexType name="SGprecludesCheck">
  <xs:sequence>
    <xs:element name="fieldId" type="xs:string"/>
    <xs:element name="precludedFieldId" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="SGrequiresMultiCheck">
  <xs:sequence>
    <xs:element name="fieldId" type="xs:string"/>
    <xs:element name="requiredFieldId" type="xs:string"
      minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="SGrowCheck">
  <xs:sequence>
    <xs:element name="fieldId" type="xs:string"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

```

```

<xs:complexType name="SGrelationCheck">
  <xs:sequence>
    <xs:element name="fieldId1" type="xs:string"/>
    <xs:element name="relationalOperator">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="="/>
          <xs:enumeration value="&lt;"/>
          <xs:enumeration value="&lt;="/>
          <xs:enumeration value="&gt;"/>
          <xs:enumeration value="&gt;="/>
          <xs:enumeration value="&lt;&gt;"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="fieldId2" type="xs:string"/>
    <xs:element name="constant" type="xs:string"/>
  </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="requiresMultiCheck"
        type="SGrequiresMultiCheck"/>
      <xs:element name="rowCheck" type="SGrowCheck"/>
      <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>

<xs:complexType name="fullSmartGovLang">
  <xs:sequence>
    <xs:element name="condition" type="xs:string"/>
    <xs:element name="action" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="smartGovLangCode">
  <xs:choice>
    <xs:element name="compactRule" type="compactSmartGovLang"/>
    <xs:element name="fullRule" type="fullSmartGovLang"/>
  </xs:choice>
</xs:complexType>

```

```

<xs:complexType name="validationMethod">
  <xs:sequence>
    <xs:element name="ruleId" type="xs:string"/>
    <xs:element name="code" type="method"/>
    <xs:element name="validateAt">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="back end"/>
          <xs:enumeration value="front end and back end"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="statistics" type="validationMethodStatistics"
      minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="method">
  <xs:sequence>
    <xs:element name="description" type="multilingualText"/>
    <xs:choice>
      <xs:element name="smartgovLangCheck" type="smartGovLangCode"/>
      <xs:element name="nativeLangCheck" type="nativeCodeFragment"
        maxOccurs="unbounded"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>

```

4.2.2 API for the SmartGovLang language translator

Translating SmartGovLang rules into solid code requires a SGLangTranslator object, that is different depending on the target environment. Creating the appropriate object is done by means of the SGLangTranslatorFactory class, which provides static “factory” methods. In Figure 16, the class diagram that describes the translator interface is presented. Two different creation methods are provided: one for client-side translators and one for server-side translators. The only languages that will be supported under the scope of the project are “Java-Struts” for the server side and “Javascript-Struts” for the client.

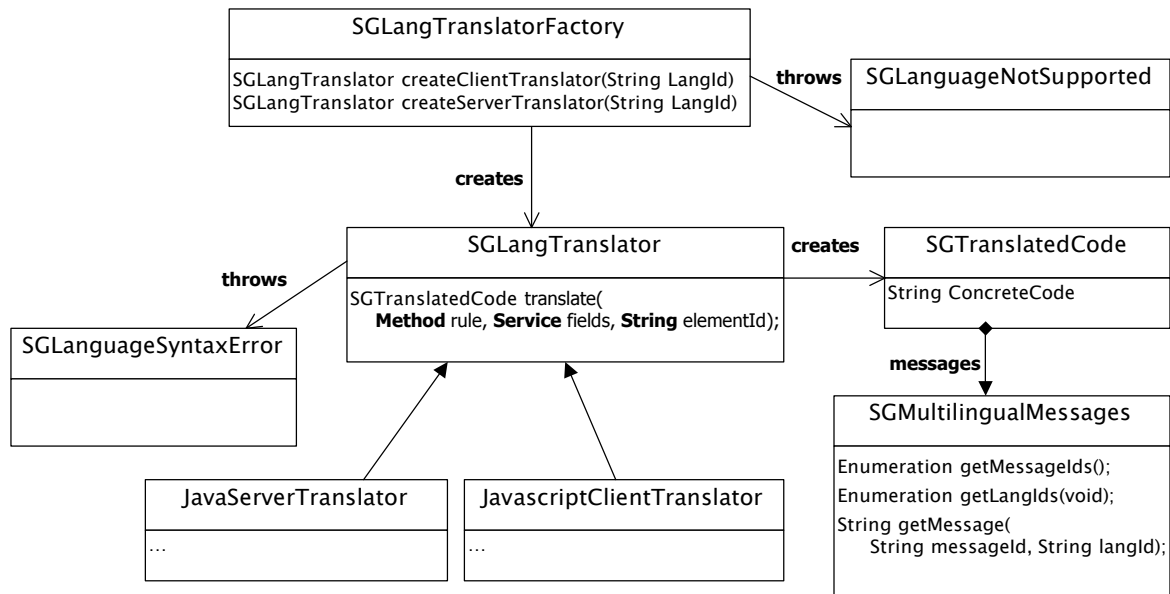


Figure 16 The SmartGovLang translator interface.

A translator translates a rule, represented by a Castor object⁷. The rule is executed in the general context described by service and is a part of the element with the specified id. All translators use the SGLangTranslator interface, which provides a translate method for the translation. Upon translation error, an SGLanguageSyntaxError exception is thrown, with a descriptive message; otherwise, a SGTranslated object is created with the translated code and an object providing an enumeration over all internationalized messages if any.

⁷ <http://www.castor.org>

5 SmartGov Agent – Information Interchange Gateway

The SmartGov Agent and the Information Interchange Gateway are the SmartGov platform components that provide facilities for communication between the service delivery environment and the organisational information system (or other, third-party systems). More specifically, the SmartGov Agent arranges for collecting requests from the services running within the Service Delivery Environment. For each such request the SmartGov Agent examines its configuration files to determine which are the methods that may be employed to satisfy the request. The directly supported methods are:

1. forward the request to the Information Interchange Gateway. Transferring may be performed via standard TCP/IP or via Secure Socket Layer.
2. storage of the request to a database.
3. storage of the request to an OS file.

The SmartGov Agent provides resilience against transient failures, such as temporary lack of communication with the Information Interchange Gateway, inability to communicate with the database (e.g. because the database server is rebooting) etc. This is implemented by storing such requests in the *Pending Actions Queue*, and by retrying the relevant methods after a certain time period.

When the Information Interchange Gateway receives a request (case 1 above), it inspects its configuration files that describe how the specific request should be served. The options available for serving a request are:

1. execution of a Java method.
2. execution of an OS-level program.
3. storage of the request to a database
4. storage of the request to an OS file.

The two first methods may return results to the Information Interchange Gateway through the return value (Java methods) or through the standard output (OS-level programs). These results are appropriately forwarded to the calling service. In the latter two cases, only a success indication is returned to the calling service. In order to implement this communication scheme, the SGA and the IIG exchange XML messages that comply to the XML DTDS illustrated in Figure 17 and Figure 18:


```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT XMLPacket (serviceName, XMLMessage, realTime, credentials)>
<!ELEMENT serviceName (#PCDATA)>
<!ELEMENT XMLMessage (#PCDATA)>
<!ELEMENT realTime (#PCDATA)>
<!ELEMENT credentials (#PCDATA)>
```

Figure 17 – DTD for SmartGov Agent to Information Interchange Gateway Messages

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT XMLPacket (replyType, replyBody)>
<!ELEMENT replyType (#PCDATA)>
<!ELEMENT replyBody (#PCDATA)>
```

Figure 18 – DTD for Information Interchange Gateway to SmartGov Agent Messages

The platform is complemented by facilities for posting notifications from the organisational information system to the service delivery environment. To this end, the Information Interchange Gateway Notification Initiator (IIG-NI) library is provided that may be included by application programs. Notifications posted from the organisational information system are received by the SmartGov Agent Notification Interceptor (SGA-NI) process running in the context of the service delivery environment. Upon reception of a notification, the SGA-NI schedules the execution of a task, which will constitute the service delivery environment's reaction to the notification. The message sent by the IIG-NI to the SGA-NI contains only the notification name, without any transformations or envelopes.

Responsible for the execution of scheduled tasks and retrying of the transiently failed requests is undertaken by two separate processes, namely the SmartGov Agent Pending Actions Queue Daemon (SGA-PAQUED) and the Information Interchange Gateway Pending Actions Queue Daemon (IIG-PAQUED), running in the context of the service delivery environment and the organisational information system, respectively.

Finally, the SmartGov platform provides a separate library, SGLogging, allowing the applications to log messages to persistent storage. Messages posted using the facilities of this library are retrieved and stored into persistent storage by a separate process, the SGLogListener.

5.1 Implementation of the SmartGov Agent – Information Interchange Gateway

The SmartGov Agent is implemented as a Java software library that is directly callable from within electronic services. Applications willing to post requests employ the `SGAppToSGAgentRequest` method of the `SGAgent`, which then –depending on the configuration– passes the responsibility for serving the request to one of the following classes: `SGAClient`, `SSLSGAClient`, `DatabaseStore`, while local file stores are handled within the `SGAgent` class. This modular architecture caters for extensibility, when new handling procedures should be added. The class diagram for the pertinent classes is illustrated in Figure 19.

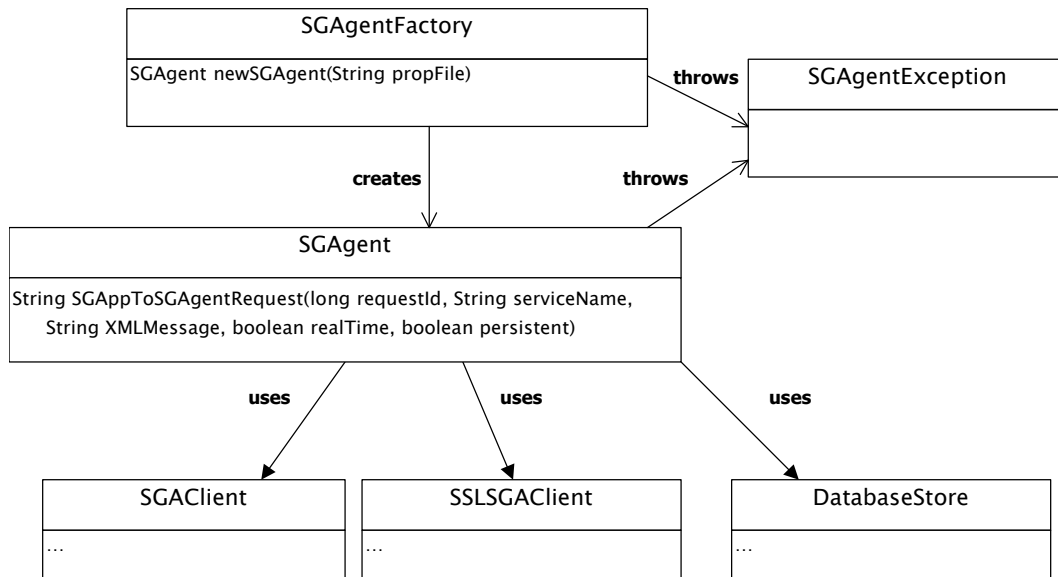


Figure 19 - Class Diagram for SmartGov Agent

Within the context of the organisational information system, when the Information Interchange Gateway receives a request, it spawns a new *execution thread* to handle it, facilitating thus parallel processing of requests. The contents of each request are first verified for syntactical correctness and completeness and then, depending on the configuration, are forwarded for actual execution to the appropriate code fragment. Concrete separation of code fragments handling each execution procedure facilitates extensibility, since only the additional code should be provided. The class diagram for the Information Interchange Gateway is depicted in Figure 20.

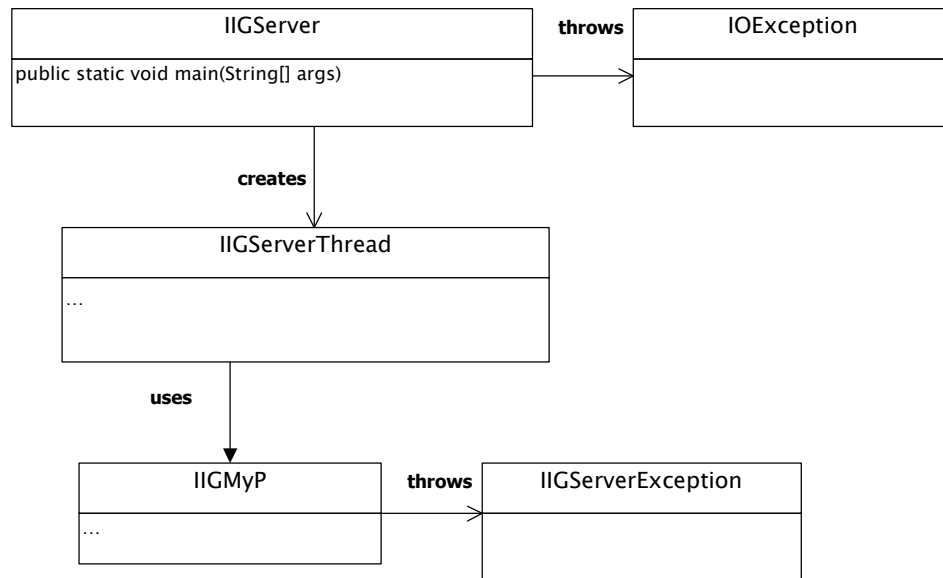


Figure 20 - Class diagram for the IIGServer

The IIGServer only handles “normal” TCP/IP communication i.e. unencrypted and unauthenticated. When security policy dictates the use of more secure mechanisms, the SSLIIGServer can be used instead, which includes support for secure communications, over the SSL communication library. The SSLIIGServer can use any of the encryption/authentication algorithms supported by the SSL library (e.g. RC4, 3DES etc), provided that the appropriate keys for the algorithms have been generated and installed in the appropriate keystore; alternatively a certification authority may be used to verify the keys. The class diagram for the SSLIIGServer is illustrated in Figure 21.

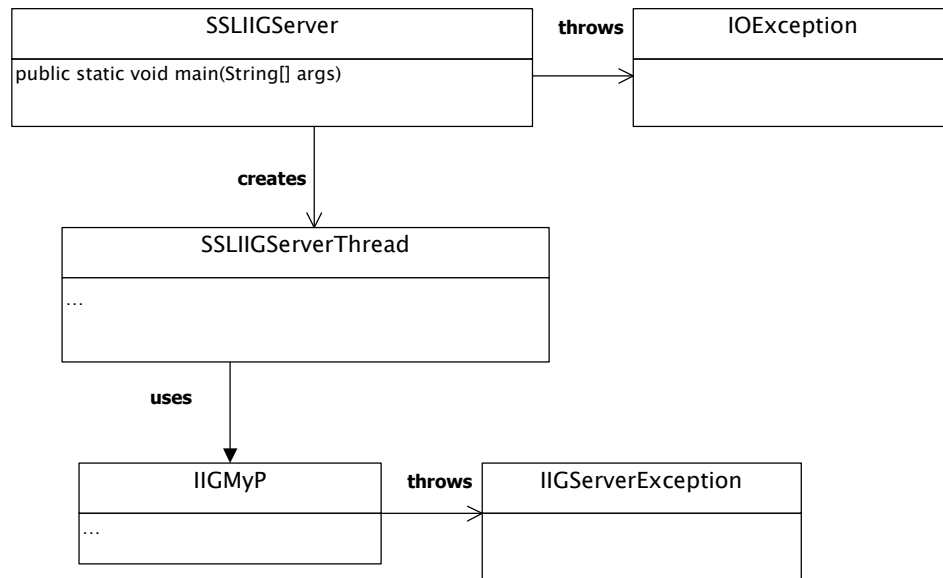


Figure 21 - Class diagram for the SSLIIGServer

Both the SmartGov Agent and the Information Interchange Gateway use the facilities of an XML parser to (a) analyse their configuration files and (b) validate and process the requests and results.

The logging facilities are provided as a Java software library that is directly callable from within electronic services or any other program running within the context of the service delivery environment or the context of the organisational information system. A service or application willing to make use of the logging facilities may either use the `SGLogging` package or use the higher-level constructs provided in the `SGUtil` class. More specifically, the `SGUtil` class implements the `logMessage` method, which creates a new `SGLogger` or retrieves the existing one, and uses it to log the given message. The class diagram for the logging facilities is depicted in Figure 22.

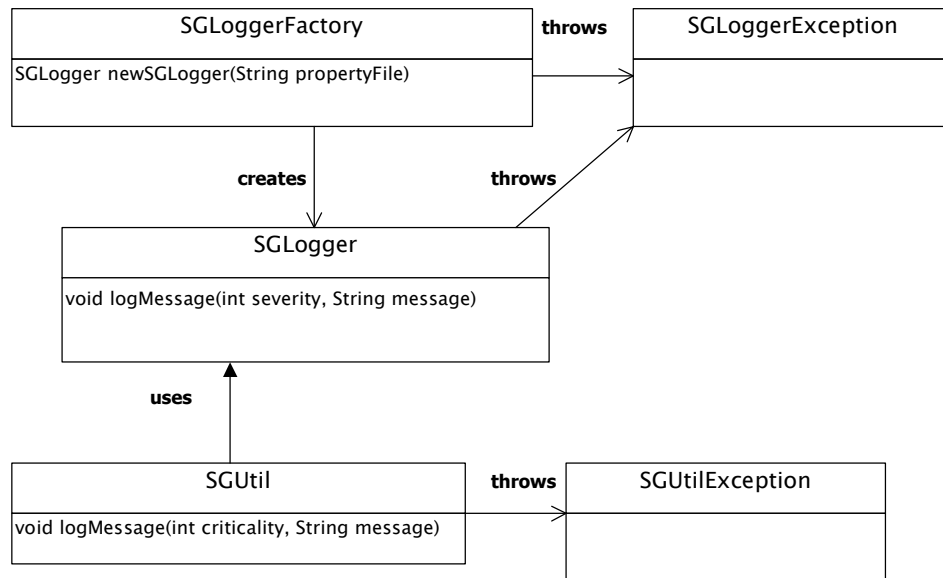


Figure 22 - Class diagram for logging facilities

The SGA-PAQUED and IIG-PAQUED are implemented as separate programs, which may be executed directly from the operating system via the Java runtime environment.

The `SGLogListener` facility is implemented as separate program, directly executable from the operating system via the Java runtime environment.

The SGA-NI facility is implemented as separate program, directly executable from the operating system via the Java runtime environment.

6 Conclusions

This deliverable presented the result of the final iteration of the development phase of the SmartGov platform and focused on the components developed within workpackage 6 i.e. the integrator, the SmartGovLang Translator and the SmartGov agents. As it was anticipated, the main design principles and functionality features as drawn in D51-61 are followed; however small amendments have been made as a result of the iterations performed within the development phase.

7 References

- [RUP] The Rational Unified Process, IBM Software Group, available at <http://www.rational.com/products/rup/index.jsp>
- [D31] State-of-the-Art and Current Situation at Public Authorities, SmartGov Project Deliverable 31, Stelios Gorilas, George Boukis, Giorgos Lepouras, Kostas Vassilakis, Akrivi Katifori, John Fraser, Heredia Larios Segundo, Rafael Canadas Martinez, Gerald Weiss, Kirstin Karasz, Spyros Argyropoulos and Hilary Coyne (May 31, 2002) available at <http://www.smartgov-project.org/index.php?category=results&langid=eng>
- [D41] User Requirements, Services and Platform Specifications, SmartGov Project Deliverable 41, Akrivi Katifori, Anna Charissi, George Lepouras, Stathis Rouvas, Costas Vassilakis, Nick Adams, John Fraser, Segundo Heredia Larios, George Boukis, Stelios Gorilas, Rafael Canadas Martinez and George Laskaridis, available (July 31, 2002) at <http://www.smartgov-project.org/index.php?category=results&langid=eng>
- [D51-61] Low-level Specifications of SmartGov Services and Applications and the Knowledge-Based Core Platform, SmartGov Project Deliverable 51-61, Stelios Gorilas, Pablo Fernandez Pardo, Tomas Pariente Lobo, Costas Vassilakis, Akrivi Katifori, Anna Charissi, George Lepouras, Stathis Rouvas, Nick Adams, John Fraser, Ann Makynthos (February 28, 2003) available at <http://www.smartgov-project.org/index.php?category=results&langid=eng>
- [Struts] The Jakarta project, The Struts Framework, available at <http://jakarta.apache.org/struts/index.html>
- [Ant] Apache foundation, The Apache Ant Project, available at <http://ant.apache.org/>
- [XML] World Wide Web Consortium, The XML Specification, available at <http://www.w3.org/xml>
- [Castor] The Exolab Group, The Castor Project, available at <http://castor.exolab.org/index.html>
- [Xalan] The Apache XML Project, Xalan, available at <http://xml.apache.org/xalan-j/>

Appendix A – SmartGovLang grammar

```
SmartGovLangRule ::= fullRule
                  | compactRule
                  ;

compactRule ::= '(' severityDesignation ',' message ',' validation ')'
            ;

severityDesignation ::= 'information'
                   | 'warning'
                   | 'error'
                   ;

message ::= multiLingualText;

multiLingualText ::= '(' multiLingualTextElements ')' ;

multiLingualTextElements ::=
    multiLingualTextElement multiLingualTextElements
    ;

multiLingualTextElement ::= '(' localeSpecifier ',' anyText ')';

validation ::= 'between' '(' fieldId ',' lowerLimit ',' upperLimit ')'
             | 'requires' '(' fieldId ',' fieldId ')'
             | 'precludes' '(' fieldId ',' fieldId ')'
             | 'requiresMulti' '(' fieldId ',' fieldIdList ')'
             | checkRow(fieldIdList)
             | 'checkRelation' '(' fieldId ',' operator ',' fieldId, ','
                numberConstant ')'
             ;

fieldIdList ::= fieldId
             | fieldId '::' fieldIdList
             ;

operator ::= '=' | '!=' | '>' | '>=' | '<' | '<=' ;

fullRule ::= condition '=>' action ;

condition ::= simpleCondition
          | '(' condition ')'
          | not '(' condition ')'
          | condition 'AND' condition
          | condition 'OR' condition
          ;

simpleCondition ::= quantity operator quantity
               | booleanFunction
               ;

quantity ::= stringConstant
         | numberConstant
         | fieldId
         | globalContextReference
         | nonBooleanFunction
         | arithmeticExpression
         ;

arithmeticExpression ::= quantity arithOp quantity ;
```



```

arithOp ::= '+' | '-' | '*' | '/' | '%' ;

booleanFunction ::= 'hasAlphabet' '(' quantity ',' quantity ')'
                  | 'matches' '(' quantity ',' quantity ')'
                  | 'startsWith' '(' quantity ',' quantity ')'
                  | 'endsWith' '(' quantity ',' quantity ')'
                  | 'isValidDate' '(' quantity ')'
                  | 'isValidDate' '(' 's=' quantity ')'
                  | 'isValidTime' '(' quantity ')'
                  ;

nonBooleanFunction ::= 'int' '(' quantity ')'
                    | 'fractional' '(' quantity ')'
                    | 'length' '(' quantity ')'
                    | 'index' '(' quantity ',' quantity ')'
                    | 'substr' '(' quantity ',' quantity
                        ',' quantity ')'
                    | 'concatenate' '(' quantity ',' quantity ')'
                    | 'date' '(' ')'
                    | 'year' '(' ')'
                    | 'month' '(' ')'
                    | 'day' '(' ')'
                    | 'weekDay' '(' ')'
                    | 'time' '(' ')'
                    | 'minute' '(' ')'
                    | 'second' '(' ')'
                    | 'count' '(' groupId ')'
                    | 'sum' '(' columnId ')'
                    | 'max' '(' columnId ')'
                    | 'min' '(' columnId ')'
                    ;

action ::= singleAction ';'
        | singleAction action
        ;

singleAction ::= errorMessage '(' multilingualText ')'
              | warningMessage '(' multilingualText ')'
              | informationMessage '(' multilingualText ')'
              | disableField '(' fieldId ')'
              | enableField '(' fieldId ')'
              | setField '(' fieldId ',' quantity ')'
              | setFocus '(' fieldId ')'
              ;

```

Appendix B - List of functions available in SmartGovLang

Function	Comments
int(x)	The integral part of value <i>x</i> , which must be arithmetic.
fractional(x)	The fractional part of value <i>x</i> , which must be arithmetic.
length(s)	The number of characters in string <i>s</i>
index(s, t)	Returns the position, in characters, numbering from 1, in string <i>s</i> where string <i>t</i> first occurs, or zero if it does not occur at all.
substr(s,m, n)	Returns the at most <i>n</i> -character substring of <i>s</i> that begins at position <i>m</i> , numbering from 1.
concatenate(s1, s2)	Returns a string whose value is string <i>s1</i> , followed by <i>s2</i> .
hasAlphabet(s, a)	Returns true if the string <i>s</i> contains only characters listed in string <i>a</i> , or false, otherwise
matches(s, e)	Returns true if the string <i>s</i> matches the pattern specified in <i>e</i> or false otherwise. This function may not be supported in all front-ends.
startsWith(s, prefix)	Equivalent to the following expression matches(substr(s,1,length(prefix)),prefix)
endsWith(s, postfix)	Equivalent to the following expression: matches(substr(s,length(s)-length(postfix),length(postfix)),postfix)
date()	Returns the current system date
isValidDate(yyyy, mm, dd) isValidDate(s="yyyy/mm/dd")	Returns true if the given date is valid, otherwise false.
isValidTime(hh, mm, ss) isValidTime(s="hh:mm:ss")	Returns true if the given time is valid, otherwise false.
year()	Returns the current system year

Function	Comments
month()	Returns the current system month
day()	Returns the current system day
weekDay()	Returns the current system week day (sun,mon,tue,...)
time()	Returns the current system time
minute()	Returns the current system minute
second()	Returns the current system second
count(repeatingGroup)	Returns the number of rows in a repeating group (table) html table? Server side checking?
sum(columnReference)	Returns the sum of the values in the designated column.
max(columnReference)	Returns the maximum of the values in the designated column.
min(columnReference)	Returns the minimum of the values in the designated column

Appendix C – Integrator JavaDocs

Package `com.archetypon.smartgov.builder`

public class **com.archetypon.smartgov.builder.StrutsConfigBuilder** extends [com.archetypon.smartgov.builder.FileBuilder](#)

Builds the struts-config.xml file of the service web application. It is a wrapper around the Castor-generated objects that correspond to the struts-config schema.

Methods

public void process(Object modelObject)

Creates the Struts configuration file from the given service definition object.

Parameters

modelObject - an object of type @see [com.archetypon.smartgov.model.TsElement](#)

See Also

[com.archetypon.smartgov.builder.FileBuilder#process\(Object\)](#)

public void serialize()

Marshals the contents of the contained Castor object to a temp String and stores the file.

public class **com.archetypon.smartgov.builder.PropertyBuilder** extends [com.archetypon.smartgov.builder.FileBuilder](#)

Creates the application's language resources. These FileBuilders are created one per model element and contain the text of all internal fields of type multilingualText. Each individual PropertyBuilder object corresponds to the multilingual content of a particular field (e.g. Form.description). For this reason, they are created using a URI following this pattern: _ (e.g. Form1_description). Each multilingual entry is stored in an internal Map using its locale as name. At the end, during serialization, each entry is appended to the corresponding properties file, named _locale.

Methods

public void serialize()

Appends properties to the respective message bundles. For each entry in the value Map, a file name is created called `_locale`. For each one, the `@see com.archetypon.smartgov.util.StorageWrapper#storePropertiesDocument(Reader, String, boolean)` method is called, and entry like the following is appended, = value.

See Also

`com.archetypon.smartgov.builder.FileBuilder#createFiles()`

public void process(Object modelObject)

Processes a multilingual entry. The content's value is placed in the Map using the locale as key.

Parameters

`modelObject` - of type `@see com.archetypon.smartgov.schemas.MultilingualText`

See Also

`com.archetypon.smartgov.builder.FileBuilder#process(Object)`

public class **com.archetypon.smartgov.builder.JspBuilder** extends [com.archetypon.smartgov.builder.FileBuilder](#)

This class is in charge of post-processing and serializing the JSPs of the service. It expects an object of type `java.lang.StringBuffer` in its process method. This buffer is used as the core for the generated JSP page, prepended with tag library definitions and directives. The generated JSP is saved inside the global destination path (`basePath`), under folder identified by global variable `jsp.path` .

Methods

public void serialize()

Stores the contents of the internal buffer as a JSP file. The path to store JSP pages, located in property "jsp.path" is prepended to the base URI used to invoke this Processor.

public void process(Object externalBuffer)

Processes the given JSP page core. The provided object is expected to be of type `java.lang.StringBuffer` . It appends the tag library declaration as defined in the iteration document

Parameters

externalBuffer - of type StringBuffer

See Also

Iteration Plan #1c, Task 12, *<i>Usage of common ActionForm in JSPs</i>*

Iteration Plan #1c, Task 05, *<i>Creation of a common ActionForm class</i>*

```
public class com.archetypon.smartgov.builder.HelpBuilder extends  
com.archetypon.smartgov.builder.FileBuilder
```

Creates one or more help files that usually corresponds to a KU object. The produced files are named uri _ contentLocale .txt and have some basic HTML formatting. The files are meant to be displayed by help.jsp .

Methods

public void process(Object modelObject)

Processes a KuElement. For each of its elements, appends the appropriate information in the corresponding StringBuffer object as formatted HTML.

Parameters

modelObject - of type @see com.archetypon.smartgov.model.KuElement

See Also

[com.archetypon.smartgov.builder.FileBuilder#process\(Object\)](#)

public void serialize()

For each locale, creates the necessary file. The naming scheme that is followed is basePath/ uri _ contentLocale . The files are stored as HTML documents by the used @see com.archetypon.smartgov.util.StorageWrapper

See Also

[com.archetypon.smartgov.builder.FileBuilder#createFile\(\)](#)

```
public class com.archetypon.smartgov.builder.FormBeanBuilder extends  
com.archetypon.smartgov.builder.FileBuilder
```

Creates a Java class to keep a form's values. The generated class has the following

characteristics: - is named formId Form - belongs to the package "form.apackage", defined by @see BuilderFactory#properties - has getters and setters for all TSEs defined in the corresponding form

Methods

public void process(Object modelObject)

Processes the given Form element as well as it's linked TSEGroups. For each included TSE it creates a variable and a pair of getter and setter methods.

Parameters

modelObject - of type @see com.archetypon.smartgov.model.FormElement

See Also

com.archetypon.smartgov.builder.FileBuilder#process(Object)

public void serialize()

Stores the class file in the basePath. It transforms the package name into the appropriate path suffixed to the basePath.

See Also

com.archetypon.smartgov.builder.FileBuilder#createFiles()

public abstract class **com.archetypon.smartgov.builder.FileBuilder**

Base class for all SmartGov dynamic file building objects.

Methods

protected abstract void process(Object modelObject)

Actual model object processing. This method must be overridden by descendant classes.

Parameters

modelObject - can be an object that is automatically generated by Castor (package com.archetypon.smartgov.schemas) or it may be a model wrapper object (package com.archetypon.smartgov.model). The descendant class should clarify this in this method's documentation.

public abstract void serialize()

File creation method. Must be overridden by descendant classes.

Fields

protected factory

parent factory object

protected uri

URI of the resource this builder is attached to

protected basePath

path where produced files should be sent to

public class **com.archetypon.smartgov.builder.BusinessObjectBuilder** extends
[com.archetypon.smartgov.builder.FileBuilder](#)

Methods

public void serialize()

See Also

[com.archetypon.smartgov.builder.FileBuilder#createFiles\(\)](#)

protected void process(Object modelObject)

See Also

[com.archetypon.smartgov.builder.FileBuilder#process\(Object\)](#)

public class **com.archetypon.smartgov.builder.BuilderFactory**

Provides factory methods that return all the required file-building classes. All objects created with this class are singleton, i.e. the class maintains an internal registry of created objects. This means that no two objects of the same type can correspond to the same URI.

Constructors

public BuilderFactory(Properties properties, StorageWrapper wrapper)

Methods

**public com.archetypon.smartgov.builder.BusinessObjectBuilder
getBusinessObjectBuilder(String uri)**

public com.archetypon.smartgov.builder.FormBeanBuilder

getFormBeanBuilder(String uri)

public com.archetypon.smartgov.builder.JspBuilder getJspBuilder(String uri)

public com.archetypon.smartgov.builder.PropertyBuilder getPropertyBuilder(String uri)

public com.archetypon.smartgov.builder.StrutsConfigBuilder getStrutsConfigBuilder(String uri)

public com.archetypon.smartgov.builder.HelpBuilder getHelpBuilder(String uri)

public java.lang.String getPath(String key)

Retrieves the path from the given property.

Parameters

key - to retrieve the value of

Returns

the value of the key, suffixed with a "/". If the property is not existent, or the properties object is null, the current directory path (./) is returned.

public java.lang.String getProperty(String key)

Retrieves the given property.

Parameters

key - to retrieve the value of

Returns

the value of the property, or null if it is not existent

public com.archetypon.smartgov.util.StorageWrapper getStorageWrapper()

Returns the storageWrapper.

Returns

StorageWrapper

public class com.archetypon.smartgov.builder.BuilderException extends

`com.archetypon.smartgov.schemas.struts.Exception`

Constructors **`public BuilderException()`**

Package com.archetypon.smartgov.process

public class **com.archetypon.smartgov.process.XhtmlProcessor** extends [com.archetypon.smartgov.process.Processor](#)

Takes care of pre-processing xHTML files associated with service forms. The class is responsible of processing the xHTML file and replacing HTML form control placeholders with Struts control tags. The placeholders are in the form of xHTML comments and contain exactly the id of the corresponding element or a known text. The correspondence between placeholders and Struts tags is the following: FORM_BEGIN -- : action="..." method="POST" >FORM_END -- : TSE_ID -- : or ELEMENT_ID _HELP -- : onclick="..." holding all KU items that are linked with the given ELEMENT_ID ELEMENT_ID _CAPTION -- : var="..." stating the name of the control The changes are applied to the original file, which is read and processed in-memory. The result is this processing is provided to an instance of @link com.archetypon.smartgov.builder.JspBuilder that is created and invoked as a child builder. and written back as a JSP file, with the same base URI.

Methods

public void process()

Loads and processes the xHTML file. First it replaces the form tags and then all the rest (TSE, KU). It then creates an instance of @link JspBuilder and "feeds" it with the results of the pre-processing.

See Also

[com.archetypon.smartgov.process.Processor#process\(\)](#)

public class **com.archetypon.smartgov.process.XhtmlProcessor.Placeholder**

Utility class. Represents an identifier (TSE or KU) located inside the file. The identifier's name is accompanied by the beginning and ending index of the whole placeholder (ID -- >)

Constructors

public XhtmlProcessor.Placeholder(String id, int beginIndex, int endIndex)

Methods

public int getBeginIndex()

public int getEndIndex()

public java.lang.String getId()

public boolean isHelp()

public boolean isCaption()

public class **com.archetypon.smartgov.process.TseProcessor** extends
[com.archetypon.smartgov.process.Processor](#)

Contains the logic applied during TSE element processing. The class currently supports the following fields: - tseName - description - valueList - linkedKUNode. The class creates the following child Processors: - @see [com.archetypon.smartgov.process.KuProcessor](#) The following builders are created as children: - @see [com.archetypon.smartgov.builder.PropertyBuilder](#)

Methods

public void process()

Main processing method.

See Also

[com.archetypon.smartgov.process.Processor#process\(\)](#)

public class **com.archetypon.smartgov.process.ServiceProcessor** extends
[com.archetypon.smartgov.process.Processor](#)

Contains the high-level logic applied during TS element processing. The class, currently, processes the following TS child fields: - name - description - includedFormSets - linkedKUNode . It creates the following types of Processors: - @see [com.archetypon.smartgov.process.FormProcessor](#) - @see [com.archetypon.smartgov.process.KuProcessor](#) It creates the following types of FileBuilders: - @see [com.archetypon.smartgov.builder.PropertyBuilder](#) - @see [com.archetypon.smartgov.builder.StrutsConfigBuilder](#)

Methods **public void process()**
Launching method. Takes care of parsing the service and launching the builders.

public class **com.archetypon.smartgov.process.ProcessorFactory**

Factory class for model @see Processors. The class offers singleton access to Processor objects via factory methods. The uniqueness of objects is guaranteed by a set of Maps where they are stored based on their URIs. Factory methods require the uri of the LinkedModelElement that the Processor will correspond to and the enclosing @see com.archetypon.smartgov.model.LinkedModelElement, if any.

Constructors **public ProcessorFactory(BuilderFactory builderFactory, ElementFactory elementFactory)**

Default constructor.

Parameters

builderFactory - will be passed to the constructors of all created Processors

elementFactory - will be used to retrieve the model elements corresponding to the given URIs

Methods **public com.archetypon.smartgov.process.ServiceProcessor
getServiceProcessor(String uri, LinkedModelElement parent)**

**public com.archetypon.smartgov.process.FormProcessor
getFormProcessor(String uri, LinkedModelElement parent)**

**public com.archetypon.smartgov.process.KuProcessor
getKuProcessor(String uri, LinkedModelElement parent)**

**public com.archetypon.smartgov.process.XhtmlProcessor
getXhtmlProcessor(String uri, LinkedModelElement parent)**

**public com.archetypon.smartgov.process.GroupProcessor
getTseGroupProcessor(String uri, LinkedModelElement parent)**

public com.archetypon.smartgov.process.TseProcessor

getTseProcessor(String uri, LinkedModelElement parent)

public **com.archetypon.smartgov.builder.BuilderFactory**
getBuilderFactory()

Returns the builderFactory.

Returns

BuilderFactory

public abstract class **com.archetypon.smartgov.process.Processor**

Base class for all model processors. Descendants of this class form the glue between the service model as defined by classes in package com.archetypon.smartgov.model and the actual file builders as defined in package com.archetypon.smartgov.builder . In other words, Processor descendants decide which model (sub)elements will be processed, but not what will happen with them. This is the "duty" of the corresponding FileBuilder descendants that are invoked.

Methods

public abstract void process()

Contains the logic of the object. Descendants need to override this method and perform whatever processing is necessary. The main idea, however, should be that this method only decides which builders are created/invoked. Some Processors may need to use field @see Processor#wasProcessed to prevent erratic behaviour due to multiple calls to this method.

public void serialize()

Forces content serialization. The default implementation causes all child FileBuilders to create the necessary files. It also "propagates" the request to all child Processors. This method is using field @see Processor#wasWritten to prevent multiple calls to this object's method. If descendant classes need a different behaviour, they should override this method.

protected final void addChildBuilder(FileBuilder builder)

Utility method. Adds a child builder to the local store.

protected final void removeChildBuilder(FileBuilder builder)

Utility method. Removes a child builder from the local store.

protected final void addChildProcessor(Processor processor)

Utility method. Adds a child processor to the local store.

protected final void removeChildProcessor(Processor processor)

Utility method. Removes a child processor from the local store.

Fields

protected factory

factory that creates other processors

protected builderFactory

factory that creates file builders

protected object

model object this processor corresponds to

protected uri

URI of the linked model element; utility field for quick access

protected parent

logically enclosing model element; may be left null. Useful if the processor needs to "backtrack" to the upper level

protected wasProcessed

indicates if the processor has already been called

protected wasWritten

indicates if the processor has forced the enclosed builder to flush the file

protected childBuilders

local storage of all created FileBuilders; to be used when a serialization command has been received

protected childProcessors

local storage of all created Processors; to be used when a serialization command should be propagated

public class **com.archetypon.smartgov.process.KuProcessor** extends
[com.archetypon.smartgov.process.Processor](#)

Contains the processing logic for the KU element. Creates the following child FileBuilders: -
@see com.archetypon.smartgov.builder.HelpBuilder

Methods **public void process()**
See Also
com.archetypon.smartgov.process.Processor#process()

public class **com.archetypon.smartgov.process.GroupProcessor** extends
[com.archetypon.smartgov.process.Processor](#)

Contains logic for processing of TSE groups. The class currently supports the following fields of the element: - groupName - description - includedTSE - linkedKUNode The class creates the following child Processors: - @see com.archetypon.smartgov.process.KuProcessor - @see com.archetypon.smartgov.process.TseProcessor It also creates the following child builders: - @see com.archetypon.smartgov.builder.PropertyBuilder

Methods **public void process()**
See Also
com.archetypon.smartgov.process.Processor#process()

public class **com.archetypon.smartgov.process.FormProcessor** extends
[com.archetypon.smartgov.process.Processor](#)

High-level logic applied during Form element processing. The class supports the following Form child fields: - name - description - includedTSE - includedTSEGroup - linkedKUNode - formLayout The class creates the following Processors as children: - @see com.archetypon.smartgov.process.KuProcessor - @see com.archetypon.smartgov.process.TseProcessor The following FileBuilders are created as

children: - @see com.archetypon.smartgov.builder.FormBeanBuilder - @see
com.archetypon.smartgov.builder.ActionClassBuilder

Methods **public void process()**

See Also

com.archetypon.smartgov.process.Processor#process()

Package com.archetypon.smartgov.util

public interface **com.archetypon.smartgov.util.StorageWrapper**

Provides an abstraction layer over the underlying storage mechanism of XML and other documents.

Methods

public java.io.Reader getXmlDocument(String uri)

Returns a Reader object from the given XML element URI.

Parameters

uri - It is up to the implementation to interpret this parameter. It may be treated as a URL, a file path etc.

Returns

Reader object corresponding to this URI. If more than one were found, it is up to to the underlying implementation to decide which one to return.

Throws

DocumentNotFoundException - if there is no file corresponding to this URI or if an error occurred.

public java.io.Reader getHtmlDocument(String uri)

Returns a Reader object from the given HTML file URI.

Parameters

uri - It is up to the implementation to interpret this parameter. It may be treated as a URL, a file path etc.

Returns

Reader object corresponding to this URI. If more than one were found, it is up to to the underlying implementation to decide which one to return.

Throws

DocumentNotFoundException - if there is no file corresponding to this URI or if an error occurred.

public void storePropertiesDocument(Reader reader, String uri, boolean append)

public void storeJavaDocument(Reader reader, String uri)

public void storeXmlDocument(Reader reader, String uri)

public void storeJspDocument(Reader reader, String uri)

public void storeHtmlDocument(Reader reader, String uri)

public void storeSqlDocument(Reader reader, String uri)

public class **com.archetypon.smartgov.util.FileStorageWrapper** implements
[com.archetypon.smartgov.util.StorageWrapper](#)

Provides an abstraction over a file system.

Constructors **public FileStorageWrapper()**

Default constructor. All consequent searches saves will expect an absolute path.

**public FileStorageWrapper(String baseLoadDirectory, String
baseSaveDirectory)**

Specifies base directories. All consequent searches saves will be treated as relevant to these, if not null.

Methods **public java.io.Reader getXmlDocument(String uri)**

Returns a Reader object from the given file URI. The file is always assumed to be in UTF-8 encoding.

Parameters

uri - It is interpreted as a file name. If the suffix is not .xml, it is assumed so. If the @see FileStorageWrapper#baseDirectory field is not null, then the uri is considered as a relative path; otherwise it is considered an absolute one.

Throws

FileNotFoundException - if there is no file corresponding to this URI

See Also

[com.archetypon.smartgov.util.StorageWrapper#getXmlDocument\(String\)](#)

public java.io.Reader getHtmlDocument(String uri)

Returns a Reader object from the given file URI.

Parameters

uri - It is interpreted as a file name. If the suffix is not .xhtml, it is assumed so. If the @see FileStorageWrapper#baseDirectory field is not null, then

the uri is considered as a relative path; otherwise it is considered an absolute one.

Throws

DocumentNotFoundException - if there is no file corresponding to this URI

See Also

`com.archetypon.smartgov.util.StorageWrapper#getXmlDocument(String)`

public void storeJavaDocument(Reader reader, String uri)

Stores Java documents in the file system.

public void storeJspDocument(Reader reader, String uri)

Stores JSP pages in the file system.

public void storePropertiesDocument(Reader reader, String uri, boolean append)

Stores property files in the file system.

Parameters

reader - to get data from

uri - to use as the file's name, appending .properties suffix

append - flag to indicate if content is to be appended at the end of an existing file

public void storeHtmlDocument(Reader reader, String uri)

Stores HTML documents in the file system. The file suffix is xhtml.

public void storeXmlDocument(Reader reader, String uri)

Stores XML documents in the file system.

public void storeSqlDocument(Reader reader, String uri)

Stores SQL documents in the file system

Package com.archetypon.smartgov.model

public class **com.archetypon.smartgov.model.TsElement** extends [com.archetypon.smartgov.model.LinkedModelElement](#)

Performs initialization of a TS element and all associated Form elements. While parsing the object's fields, the following linked elements are processed: - FormElements - KuElements. All of the linked forms are loaded into the internal formStore field and are organized per formSet . The internal store's key is the targetPlatform field of each individual formSet . The value corresponding to this key is a Vector of @see FormElements, in the order they were encountered in the service definition file. All of the linked KUs are loaded into the internal kuStore field, with their id as key. The wrapped object is of type @see com.archetypon.smartgov.schemas.TS.2

Methods

public void load()

Handles deserialization of the TS element. After the Castor object has been successfully created, all of its linked elements are in turn parsed. Events are generated for the following linked fields: - includedFormSets.formId - linkedKUNode

See Also

[com.archetypon.smartgov.loader.GenericLoader#load\(\)](#)

public java.util.Vector getFormSet(String platform)

Returns the set of FormElements associated for the particular platform.

Parameters

platform - name of the platform as defined in the relevant formSet element of the service definition file

Returns

Vector of forms or null if nothing found

public java.util.Set getAllPlatforms()

Returns all platforms this service supports.

Returns

Set of String elements (platform names)

public java.util.Set getAllKuIds()

Returns all included KU ids.

Returns

Set of Strings

public com.archetypon.smartgov.model.KuElement getKuElement(String uri)

Returns a particular @see KuElement for a given id.

Parameters

uri - of the KuElement

Returns

KuElement or null if not found

public class **com.archetypon.smartgov.model.TseGroupElement** extends [com.archetypon.smartgov.model.LinkedModelElement](#)

Handles loading of InstantiatedTSEGroup elements. During parsing, the following linked elements are also loaded: - KuElement - TseElement The wrapped object is of type @see com.archetypon.smartgov.schemas.InstantiatedTSEGroup

Methods

public void load()

Core processing method. After initializing the wrapped Castor object, it also scans the following fields for linked objects: - linkedKUNode - includedTses

See Also

com.archetypon.smartgov.loader.GenericLoader#load()

public java.util.Set getAllKuIds()

Returns all included KU ids.

Returns

Set of Strings

public com.archetypon.smartgov.model.KuElement getKuElement(String uri)

Returns a particular @see KuElement for a given id.

Parameters

uri - of the KuElement

Returns

KuElement or null if not found

public java.util.Set getAllTseIds()

Returns all included TSE ids.

Returns

Set of Strings

**public com.archetypon.smartgov.model.TseElement
getTseElement(String uri)**

Returns a particular @see TseElement for a given id.

Parameters

uri - of the TseElement

Returns

TseElement or null if not found

public java.util.Set getAllTseGroupIds()

Returns all included TSE ids.

Returns

Set of Strings

public class **com.archetypon.smartgov.model.TseElement** extends
[com.archetypon.smartgov.model.LinkedModelElement](#)

Handles initialization of InstantiatedTSE elements. During parsing the following linked elements are also parsed: - KuElement The wrapped object is of type @see [com.archetypon.smartgov.schemas.InstantiatedTSE](#)

Methods

public void load()

Core processing method.

See Also

[com.archetypon.smartgov.loader.GenericLoader#load\(\)](#)

public java.util.Set getAllKuIds()

Returns all included KU ids.

Returns

Set of Strings

public com.archetypon.smartgov.model.KuElement getKuElement(String

uri)

Returns a particular @see KuElement for a given id.

Parameters

uri - of the KuElement

Returns

KuElement or null if not found

public class **com.archetypon.smartgov.model.ModelException** extends
[java.lang.Exception](#)

Indicates an error during the loading process.

Constructors **public ModelException()**

public ModelException(String uri)

Constructor used when a URI was not found.

Parameters

uri - of the element that was not found

public ModelException(String uri, Throwable exception)

Constructor used when an error occurred.

Parameters

uri - of the element that was not found

exception - that was thrown

public ModelException(String uri, ModelException exception)

Constructor used when another ModelException occurred.

Parameters

uri - of the element that was not found

exception - that was thrown

Methods **public java.lang.String getUri()**

Returns the uri.

Returns

String

public java.lang.String getMessage()

Returns the message.

Returns

String

Fields

protected uri

protected exception

protected message

public abstract class **com.archetypon.smartgov.model.LinkedModelElement**

Base class for all inter-linked elements.

Methods

public abstract void load()

Core processing method. Uses the uri field to locate the appropriate XML resource and create the corresponding Castor object, which will be returned. Each descendant is responsible for introspecting the created Castor object(s), creating the appropriate linked objects and populating the internal storage. In addition, descendant classes should also set the wrappedObject field.

Throws

ElementInitializationException - when the initialization process fails for some reason

public java.lang.Object getWrappedObject()

Returns the wrapped, Castor-generated object.

Returns

Object see the documentation of descendant classes for the exact type of this

public com.archetypon.smartgov.model.ElementFactory getFactory()

Fields

protected uri

URI of the element to be loaded. This can be an XPath expression or a file URL. This is solely dependent on the implementation of the underlying @see

StorageWrapper

protected factory

abstraction of the underlying storage layer. It provides services to the various loaders.

protected wrappedObject

wrapped Castor-generated object

```
public class com.archetypon.smartgov.model.KuElement extends  
com.archetypon.smartgov.model.LinkedModelElement
```

Handles loading of KUNodes. The wrapped object is of type @see [com.archetypon.smartgov.schemas.KUNode](#)

Constructors **[public KuElement\(String uri, ElementFactory factory\)](#)**

Methods **[public void load\(\)](#)**
Main processing method.

See Also

[com.archetypon.smartgov.loader.GenericLoader#load\(\)](#)

```
public class com.archetypon.smartgov.model.FormElement extends  
com.archetypon.smartgov.model.LinkedModelElement
```

Handles initialization of objects related to Form element. The following linked elements are parsed and processed: - TseElement - TseGroupElement - KuElement. All of the linked objects are stored using their ids as as keys. The wrappedObject is of type @see [com.archetypon.smartgov.schemas.Form](#).

Methods **[public void load\(\)](#)**
Parses the Form element along with all linked objects. Parsing events are

generated for the following linked fields: - includedTSE - includedTSEGroup - linkedKUNode

See Also

`com.archetypon.smartgov.loader.GenericLoader#load()`

public java.util.Set getAllKuIds()

Returns all included KU ids.

Returns

Set of Strings

public com.archetypon.smartgov.model.KuElement getKuElement(String uri)

Returns a particular @see KuElement for a given id.

Parameters

uri - of the KuElement

Returns

KuElement or null if not found

public java.util.Set getAllTseIds()

Returns all included TSE ids.

Returns

Set of Strings

public com.archetypon.smartgov.model.TseElement getTseElement(String uri)

Returns a particular @see TseElement for a given id.

Parameters

uri - of the TseElement

Returns

TseElement or null if not found

public java.util.Set getAllTseGroupIds()

Returns all included TSEGroup ids.

Returns

Set of Strings

public com.archetypon.smartgov.model.TseGroupElement getTseGroupElement(String uri)

Returns a particular @see KuElement for a given id.

Parameters

uri - of the KuElement

Returns

KuElement or null if not found

```
public class com.archetypon.smartgov.model.ElementNotFoundException  
extends com.archetypon.smartgov.model.ModelException
```

Indicates that an element was not found in the element repository. Although these errors are covered by @see [com.archetypon.smartgov.model.ElementInitializationException](#), this class is necessary in cases where the previous exception does not make sense to be thrown. One such example, is the case of @see [com.archetypon.smartgov.process.Processors](#) when all model elements are supposed to be loaded and present.

Constructors **[public ElementNotFoundException\(String uri\)](#)**

[public ElementNotFoundException\(String uri, Throwable exception\)](#)

[public ElementNotFoundException\(String uri, ModelException exception\)](#)

Methods **[public java.lang.String getMessage\(\)](#)**

```
public class com.archetypon.smartgov.model.ElementInitializationException  
extends com.archetypon.smartgov.model.ModelException
```

Indicates that the initialization process of a particular element has failed.

Constructors **[public ElementInitializationException\(String uri\)](#)**

[public ElementInitializationException\(String uri, Throwable t\)](#)

Methods **[public java.lang.String getMessage\(\)](#)**

public class **com.archetypon.smartgov.model.ElementFactory**

Provides factory methods for the creation of various linked elements. The class ensures that only a single instance of each element exists. For this purpose it defines a set of internal Maps that store elements based on their URI. If an element URI is requested and this element already exists, it is returned. Elements are "populated" upon creation, by calling @see com.archetypon.smartgov.model.LinkedModelElement#load() All element getters throw an @see com.archetypon.smartgov.loader.ElementInitializationException, as defined in @see com.archetypon.smartgov.model.LinkedModelElement#load()

Constructors **public ElementFactory(StorageWrapper wrapper)**

Methods **public com.archetypon.smartgov.model.TsElement getTsElement(String uri)**

public com.archetypon.smartgov.model.FormElement getFormElement(String uri)

public com.archetypon.smartgov.model.KuElement getKuElement(String uri)

public com.archetypon.smartgov.model.TseElement getTseElement(String uri)

public com.archetypon.smartgov.model.TseGroupElement getTseGroupElement(String uri)

public com.archetypon.smartgov.util.StorageWrapper getWrapper()

Returns the wrapper.

Returns

StorageWrapper

```
public class com.archetypon.smartgov.model.DuplicateDefinitionException  
extends com.archetypon.smartgov.model.ModelException
```

Indicates that an element is included more than one times. It is usually thrown when the same element is linked to twice.

Constructors **public DuplicateDefinitionException(String uri)**

public DuplicateDefinitionException(String uri, ModelException exception)

public DuplicateDefinitionException(String uri, Throwable exception)

Methods **public java.lang.String getMessage()**

```
public class com.archetypon.smartgov.model.DocumentNotFoundException  
extends com.archetypon.smartgov.model.ModelException
```

Indicates that a requested XML document was not found or an error occurred.

Constructors **public DocumentNotFoundException(String uri)**

public DocumentNotFoundException(String uri, ModelException exception)

public DocumentNotFoundException(String uri, Throwable t)

Methods **public java.lang.String getMessage()**

Package **com.archetypon.smartgov.service.model**

public class **com.archetypon.smartgov.service.model.Tse** extends
[com.archetypon.smartgov.service.model.ModelElement](#)

Performs initialization of a Tse. Used as an abstraction of a Tse so that the validation code can operate with it without problems.

Constructors **public Tse(String id, int dataType, boolean multipleValues)**
Constructor for Tse. Checks the dataType parameter and determines the type and widgetType fields.

Methods **public boolean isDisabled()**
Returns true if Tse is disabled and false if not.

public void setDisabled(boolean disabled)
Makes Tse visible or not visible.

public java.lang.Object getValue()
Returns the value of the Tse.

public void setValue(Object value)
Sets the value of the Tse

public com.archetypon.smartgov.service.model.SGType getType()
Returns the SGType of the tse

**public com.archetypon.smartgov.service.model.SGWidgetType
getWidgetType()**
Returns the SGWidgetType of the tse

public class **com.archetypon.smartgov.service.model.TestModel**

Constructors **public TestModel()**

Methods **public static void main(String[] args)**

public class **com.archetypon.smartgov.service.model.SGWidgetType**

Provides static factory methods that return singleton instances to be used in comparisons.

Constructors **public SGWidgetType()**

Methods **public static com.archetypon.smartgov.service.model.SGWidgetType
 TextArea()**

**public static com.archetypon.smartgov.service.model.SGWidgetType
 CheckBox()**

**public static com.archetypon.smartgov.service.model.SGWidgetType
 RadioButton()**

**public static com.archetypon.smartgov.service.model.SGWidgetType
 TextField()**

**public static com.archetypon.smartgov.service.model.SGWidgetType
 Select()**

public class **com.archetypon.smartgov.service.model.SGType**

Provides static factory methods that return singleton instances to be used in comparisons.

Constructors **public SGType()**

Methods **public static com.archetypon.smartgov.service.model.SGType Boolean()**

public static com.archetypon.smartgov.service.model.SGType Integer()

public static com.archetypon.smartgov.service.model.SGType Real()

public static com.archetypon.smartgov.service.model.SGType Text()

public static com.archetypon.smartgov.service.model.SGType Date()

public static com.archetypon.smartgov.service.model.SGType Currency()

public class **com.archetypon.smartgov.service.model.Service** extends
[com.archetypon.smartgov.service.model.ModelElement](#)

Performs initialization of a service object. Used as an abstraction of the service so that the validation code can operate with it without problems.

Constructors **public Service(String id)**
Constructor for Service

public abstract class
com.archetypon.smartgov.service.model.ModelElement

Base class to be used as a superclass for Service, Form, Group and Tse subclasses. These subclasses are used as an abstraction layer so that the validation code can operate without problems with the rest of the application.

Constructors **public ModelElement(String id)**
Constructor for ModelElement

Methods **public java.util.Iterator getImmediateChildIds()**
Returns an Iterator of all id's found in the internal map. These are the immediate children of the element as the method is not recursive.

**public com.archetypon.smartgov.service.model.ModelElement
getChild(String id)**

Returns the modelElement specified with the id. This method is recursive, that is if the id is not found, all childs in the map are queried until the id is found and the object returned.

public int getChildCount()

Returns the number of the elements in the map.

public void addChild(String id, ModelElement child)

Adds a modelElement in the childMap

**public com.archetypon.smartgov.service.model.ModelElement
removeChild(String id)**

Removes a modelElement from the childMap. Returns the modelElement previously associated with specified id, or null if there was no mapping for key. (A null return can also indicate that the map previously associated null with the specified key.)

Fields **protected childMap**

public class **com.archetypon.smartgov.service.model.Group** extends
[com.archetypon.smartgov.service.model.ModelElement](#)

Performs initialization of a group of Tses. Used as an abstraction of the group so that the validation code can operate with it without problems.

Constructors **public Group(String id)**

Constructor for Group

Methods **public boolean isDisabled()**

Returns true if group is visible and false if not.

public void setDisabled(boolean disabled)

Makes group visible or not visible by updating visibility in all childs.

public class **com.archetypon.smartgov.service.model.Form** extends
[com.archetypon.smartgov.service.model.ModelElement](#)

Performs initialization of a Form object.Used as an abstraction of the form so that the validation code can operate with it without problems.

Constructors **public Form(String id)**

Constructor for Form

Methods **public boolean isDisabled()**

Returns true if form is visible or false if not.

public void setDisabled(boolean disabled)

Makes form visible or not visible.

Appendix D - SmartGovLang Translator JavaDocs

Package gr.uoa.di.smartgovlang

public class **gr.uoa.di.smartgovlang.SGTranslatedCode**

The object that contains the output of a successful translation: the translated code and the messages bundle.

Constructors **public SGTranslatedCode(
 String solidCode,
 SGMultilingualMessages messages)**

Create the object that carries the results of a successful translation.

Parameters

solidCode - The generated code string.

messages - The messages bundle.

Methods **public java.lang.String getSolidCode()**

Return the translated code.

Returns

The translated code.

public gr.uoa.di.smartgovlang.SGMultilingualMessages getMessages()

Returns the messages bundle.

Returns

The messages bundle, which can be empty, but cannot be
<code>null</code>.

public class **gr.uoa.di.smartgovlang.SGMultilingualMessages**

In the case where internationalized messages are specified as part of the rule, the translator adds those messages in an object of this class. Each message has an id and the language it is written.

Constructors **public SGMultilingualMessages()**

Creates an empty messages bundle.

Methods

public java.util Enumeration getMessageIds()

Returns all the message ids of messages stored in the messages bundle.

Returns

An enumeration of messages ids as strings.

public java.util Enumeration getLangIds()

Returns all the languages ids under which messages are stored in the messages bundle.

Returns

An enumeration of language ids as strings.

**public java.lang.String getMessage(
String messageId,
String langId)**

For the given message and language ids, only one message can be stored and is returned by this method if any, otherwise null is returned.

Parameters

messageId - The id under which the message is stored.

langId - The language id under which the message is stored.

Returns

The message stored in the messages bundle or `null`.

public class **gr.uoa.di.smartgovlang.SGLanguageSyntaxError** extends
[java.lang.Exception](#)

A class that represent all syntax errors in rules.

Constructors

**public SGLanguageSyntaxError(
String rule,
String message)**

Creates a new syntax error exception.

Parameters

rule - The rule that produced the error.

message - The syntax error message.

```
public class gr.uoa.di.smartgovlang.SGLanguageNotSupported extends  
java.lang.Exception
```

A class used to describe errors during the creation of translators.

Constructors **public SGLanguageNotSupported(
 String message)**

```
public SGLanguageNotSupported(
    String message,
    Throwable cause)
```

public class **gr.uoa.di.smartgovlang.SGLangTranslatorFactory**

A factory class providing static methods to create custom translators.

NOTE: Currently the only supported language for the client-side translator is "Javascript-Struts" and for the server-side translator "Java-Struts".

Constructors **public SGLangTranslatorFactory()**

```
Methods      public      static      gr.uoa.di.smartgovlang.SGLangTranslator
             createClientTranslator(
                 String language)
```

Creates a new translator for the client-side, that is written in the specified language.

Parameters

language - The language of the translated code.

Returns

The translator.

Throws

SGLanguageNotSupported - If the language is not supported.

```
public static gr.uoa.di.smartgovlang.SGLangTranslator  
createServerTranslator(  
    String language)
```

Creates a new translator for the server-side, that is written in the specified language.

Parameters

language - The language of the translated code.

Returns

The translator.

Throws

SGLanguageNotSupported - If the language is not supported.

Fields

public static SERVER_LANGUAGES

The supported languages for the server-side.

public static CLIENT_LANGUAGES

The supported languages for the client-side.

public abstract class **gr.uoa.di.smartgovlang.SGLangTranslator**

The abstract translator object that provides the basic functionality offered by all translators.

Constructors

**public SGLangTranslator(
String language)**

Creates a new translator for the specified language.

Parameters

language - The language of the translator.

Methods

public java.lang.String getLanguage()

Returns the translator language.

Returns

Translator language.

**public gr.uoa.di.smartgovlang.SGTranslatedCode translate(
Method rule,
Service fields,
String elementId)**

The method translates the rule as expressed by the first argument. The translation assumes the rule belongs to the element with the specified id, which in turn belongs to the general context described by the fields argument.

Parameters

rule - The castor object for the rule.

fields - The object describing the runtime fields.

context - The name of the element the rule is part of.

Returns

The translated code and internationalization messages if any.

Appendix E - Agents JavaDocs

public class **gr.uoa.di.dispatcher.dispatcher**

This class implements the pending actions queue dispatcher running on the service delivery environment. The class is executable.

Constructors **public dispatcher(
 String propFile)**

Creates a new instance of dispatcher

Parameters

propFile - The dispatcher property file.

Methods **public static void main(
 String[] args)**

Fields **public static final DEFAULT_SLEEP_TIME**

The default sleep time for the intervals during the dispatcher processing

public class **gr.uoa.di.dispatcher.dispatcherException** extends
[java.lang.Exception](#)

This class models the exceptions thrown by the dispatcher.

Constructors **public dispatcherException()**
Creates a new instance of dispatcherException

**public dispatcherException(
String message)**
Constructs a new exception instance with a given error message.

Parameters

message - The message associated with the exception.

**public dispatcherException(
Throwable nestedException)**
Constructs a new exception instance that wraps another exception instance.

Parameters

nestedException The exception to be wrapped.

public class **gr.uoa.di.dispatcherIIG.dispatcherIIG**

This class implements the pending actions queue dispatcher running on the information interchange gateway environment. The class is executable.

Constructors **public dispatcherIIG(
 String propFile)**

Creates a new instance of dispatcher

Parameters

propFile - The dispatcher property file.

Methods **public static void main(
 String[] args)**

Fields **public static final DEFAULT_SLEEP_TIME**

The default sleep time for the intervals during the dispatcher processing

```
public class gr.uoa.di.dispatcherIIG.dispatcherIIGException extends  
java.lang.Exception
```

This class models the exceptions thrown by the dispatcher.

Constructors **public dispatcherIIGException()**
Creates a new instance of dispatcherIIGException

**public dispatcherIIGException(
 String message)**
Constructs a new exception instance with a given error message.

Parameters

message - The message associated with the exception.

**public dispatcherIIGException(
 Throwable nestedException)**
Constructs a new exception instance that wraps another exception instance.

Parameters

nestedException The exception to be wrapped.

public class **gr.uoa.di.IIGNI.IIGNI**

This class may be used by programs in the IIG environment to post notifications to the service delivery environment.

Methods **public void IIGToSGAgentNotification(
 String notificationName)**

This method is used in the environment of the organisation's information system and posts a notification event to signify that some event has taken place.

Parameters

notificationName - A symbolic name for the notification event.

Returns

Nothing

Throws

IIGNIException - In case of failure of the underlying IIGNI mechanism.

public class **gr.uoa.di.IIGNI.IIGNIException** extends [java.lang.Exception](#)

This class models the exceptions thrown by the IIG notification initiator.

Constructors **public IIGNIException()**

Constructs a new exception instance.

**public IIGNIException(
 String message)**

Constructs a new exception instance with a given error message.

Parameters

message - The message associated with the exception.

**public IIGNIException(
 Throwable nestedException)**

Constructs a new exception instance that wraps another exception instance.

Parameters

nestedException The exception to be wrapped.

public class **gr.uoa.di.IIGNI.IIGNIFactory**

This is the factory for IIG-NI objects. It arranges for creating instances of IIG-NI and ascertains that only one instance is active within the program at any time. As a parameter to the constructor, the name of the property file for the IIGNI should be provided.

Constructors **public IIGNIFactory()**

Methods **public static gr.uoa.di.IIGNI.IIGNI newIIGNI(
 String propertyFile)**

Factory method that acts as a virtual constructor for IIGNI. The property file contains the information for the IIGNI

Parameters

propertyFile - The IIGNI property file

Throws

IIGNIException - If the IIGNI creation failed

public static gr.uoa.di.IIGNI.IIGNI getIIGNI()

Factory method that returns the IIGNI.

public class **gr.uoa.di.sga.SGAgent**

This class provides the functionality for posting requests from the service delivery environment to external information systems and returning the appropriate replies. Requests may be real-time (synchronous) or non-real-time (asynchronous). Asynchronous requests are further characterized as "persistent", if they are allowed to persist in case of transient failures or "non-persistent", indicating that they should be disregarded if they cannot be immediately executed.

Constructors **public SGAgent(
 String propFile)**

Default constructor for SGAgent. The propFile parameter points to the property file which contains information about the agent configuration.

Methods **public java.lang.String SGAppToSGAgentRequest(
 long requestId,
 String serviceName,
 String XMLMessage,
 boolean realTime,
 boolean persistent)**

This method is used by the SGApp to make a request for a service to a third party system using SGAgent.

Parameters

requestId - A unique request identifier that serves to characterize this request.

serviceName - A symbolic service name that the message refers to.

XMLMessage - A message that contains all information that the named serviceName requires.

realTime - A request parameter used to show if a request is real-time or not.

persistent - A request parameter used to show if a request is persistent or not.

Returns

String Reply to the request

Throws

SGAgentException - In case of failure of the underlying SGAgent mechanism.

Fields **public static final SGA_NONPERSISTENT**

Constant used if the request is not persistent

public static final SGA_PERSISTENT

Constant used if the request is persistent

public static final SGA_NONREALTIME

Constant used if the request is not real-time

public static final SGA_REALTIME

Constant used if the request is real-time

public class **gr.uoa.di.sga.SGAgentException** extends [java.lang.Exception](#)

This class models the exceptions thrown by the SmartGov agent.

Constructors **public SGAgentException()**

Constructs a new exception instance.

**public SGAgentException(
 String message)**

Constructs a new exception instance with a given error message.

Parameters

message - The message associated with the exception.

**public SGAgentException(
 Throwable nestedException)**

Constructs a new exception instance that wraps another exception instance.

Parameters

nestedException The exception to be wrapped.

public class **gr.uoa.di.sga.SGAgentFactory**

This is the factory for SGAgent objects. It arranges for creating instances of SGAgent and ascertains that only one instance is active within the program at any time.

Constructors **public SGAgentFactory()**

Constructs a new instance of SGAgentFactory.

Methods **public gr.uoa.di.sga.SGAgent newSGAgent(
 String propFile)**

Factory method that acts as a virtual constructor for SGAgent. The property file for the SGAgent is given as a parameter.

Throws

SGAgentException - If the SGAgent creation failed

public class **gr.uoa.di.SGANI.SGANIFactory**

This class implements the factory for SGANI objects. Only one static SGANI object is created and then returned by the factory when it is requested. When executed the factory creates an SGANI and starts listening for notification events sent by the IIGNI. The class is executable.

Constructors **public SGANIFactory(
 String propFile)**

Creates a new instance of SGANIactory

Parameters

propFile - The IIGNIproperty file

Methods

```
public gr.uoa.di.SGANI.SGANI newSGANI()  
Factory method that acts as a virtual constructor for SGANI.  
Throws  
SGANIException - If the SGANI creation failed  
  
public static void main(  
String[] args)
```

public class **gr.uoa.di.SGLogging.SGLogger**

This class implements a logger for events in the SmartGov platform. The SGLogger is responsible for sending a message to the SGLogListener along with a message severity code number.

Methods

**public void logMessage(
int severity,
String message)**

This method logs the event described by message with a severity indication as specified by the first parameter.

Parameters

severity - An indication of the severity of the logged event

message - The message to be sent.

Throws

SGLoggerException - In case of failure of the underlying SGLogger mechanism.

Fields

An indication of the severity of the logged event

public static final SG_LOG_EMERG

public static final SG_LOG_ALERT

public static final SG_LOG_ERROR

public static final SG_LOG_WARNING

public static final SG_LOG_INFO

public static final SG_LOG_DEBUG

public class **gr.uoa.di.SGLogging.SGLoggerException** extends
[java.lang.Exception](#)

This class models the exceptions thrown by the SGLogger.

Constructors **public SGLoggerException()**
Creates a new instance of SGLoggerException

**public SGLoggerException(
String message)**
Constructs a new exception instance with a given error message.

Parameters

message - The message associated with the exception.

**public SGLoggerException(
Throwable nestedException)**
Constructs a new exception instance that wraps another exception instance.

Parameters

nestedException The exception to be wrapped.

public class **gr.uoa.di.SGLogging.SGLoggerFactory**

```
public class gr.uoa.di.SGLogListener.SGLogListenerException extends  
java.lang.Exception
```

This class models the Exceptions thrown by SGLogListener and SGLogListenerFactory.

Constructors **[public SGLogListenerException\(\)](#)**

Creates a new instance of SGLogListenerException

**[public SGLogListenerException\(
String message\)](#)**

Constructs a new exception instance with a given error message.

Parameters

message - The message associated with the exception.

**[public SGLogListenerException\(
Throwable nestedException\)](#)**

Constructs a new exception instance that wraps another exception instance.

Parameters

nestedException The exception to be wrapped.

public class **gr.uoa.di.SGLogListener.SGLogListenerFactory**

This is the factory for SGLogListener objects. When executed, it creates a new SGLogListener and it starts listening for logging requests. As a parameter during the execution, the name of the property file for the SGLogListener should be provided. The class is executable.

Constructors **public SGLogListenerFactory()**
Creates a new instance of SGLogListenerFactory

Methods **public static void main(
 String[] args)**

public class **gr.uoa.di.SGUtil.SGUtil**

This class contains utilities that are used by the other SG packages. These are the following:

1. The loading of properties from a property file to a Property array.
2. The retrieval of a specific property from a property file.
3. The logging of messages using SGLogger

Constructors **public SGUtil()**

Creates a new instance of SGUtil

Methods **public java.lang.String getProperty(
 String propertyName,
 String propertyFile)**

This method is used to get a property from a property file.

Parameters

propertyName - The name of the property to be read

propertyFile - The property file

Throws

SGUtilException - in case of failure to read the property

**public static synchronized void logMessage(
 int criticality,
 String message)**

This method logs the message with the specified criticality. In order to do so, uses the SGLogger.

Parameters

criticality - The code name defining how critical this message is.

message - The message to be logged

Throws

SGUtilException - in case of failure to log the message

public class **gr.uoa.di.SGUtil.SGUtilException** extends [java.lang.Exception](#)

This class models the exceptions thrown by the SGUtil class.

Constructors **public SGUtilException()**

Creates a new instance of SGLoggerException

**public SGUtilException(
 String message)**

Constructs a new exception instance with a given error message.

Parameters

message - The message associated with the exception.

**public SGUtilException(
 Throwable nestedException)**

Constructs a new exception instance that wraps another exception instance.

Parameters

nestedException The exception to be wrapped.

public class **gr.uoa.di.SSLIIGServer.SSLIIGServer**

This class is used to initiate the execution of a SSL-enabled IIG-server. This is equivalent to an IIG-server, but communicates using secure-socket layer, to provide for communication encryption and message authenticity. The class is executable.

Constructors **public SSLIIGServer()**
Creates a new instance of SSLIIGServer

Methods **public static void main(
 String[] args)**

