FAST AND ADVANCED STORYBOARD TOOLS

FP7-ICT-2007-1-216048

http://fast.morfeo-project.eu

Deliverable D4.3.2

Mechanisms for Gadget-Service Connections and Gadget Functionality

Ismael Rivera, NUIG

Date: 26/02/2010

FAST is partially funded by the E.C. (grant code: FP7-ICT-2007-1-216048).

Version History

Rev. No.	Date	Author (Partner)	Change description
1.0	26.02.2010	Ismael Rivera (NUIG)	Final version ready for external review

Executive Summary

This deliverable exposes several mechanisms which will allow the connection and interaction between end-user's interfaces to third-party back-end services.

These back-end services cannot be directly used; hence they need to be encapsulated in what in the FAST platform is called Resource Adapters. The application of semantics to the back-end, through the corresponding Resource Adapters, and front-end building blocks assures a powerful instrument in the task of building new gadgets, improving the search, and enhancing the connection among the different building blocks which compose a gadget.

Therefore, the focus of this deliverable is to define how these wrappers will be constructed to allow the FAST platform exploiting web services [Alonso et al., 2003] such as RESTful web services, SOAP-based web services and semantic web services through WSMO, and define mechanisms to connect them within the gadgets.

Document Summary

Code	FP7-ICT-2007-1-216048	**Acronym**	FAST
Full title	Fast and Advanced Storyboard Tools		
URL	`http://fast.morfeo-project.eu`		
Project officer	Annalisa Bogliolo		

Deliverable	**Number**	D4.3.2	**Name**	Mechanisms for Gadget-Service Connections and Gadget Functionality
Work package	**Number**	4	**Name**	Visual composition of screen-flow resources and interoperability with back-end Web Services

Delivery data	**Due date**	28/02/2009	**Submitted**	27/02/2009
Status			final	
Dissemination Level	Public ☒ / Consortium ☐			
Short description of contents	D4.3.2 is...			
Authors	Ismael Rivera, NUIG			
Deliverable Owner	Ismael Rivera, NUIG		**email**	ismael.rivera@deri.org
(Partner)			**phone**	+353 91 ******
Keywords	FAST, web services, WSDL, REST, SOAP, WSMO			

Table of contents

1 Introduction

1.1 Goal and Scope

The objective of this deliverable is the analysis and development of mechanisms to facilitate the connection between screen-flow gadgets and underlying Web services, based on front-end user requirements and on the semantic descriptions of the service wrappers, and to develop these service wrappers from of the Web services' APIs and formal descriptions.

1.2 Structure of the Document

This deliverable is structured as follows. Section 1 states the goal, scope and structure of the document. Section 2 describes what a service wrapper is, how the platform is able to interact with web services, how they these wrappers are build and then discovered to be reused in any gadget. Finally, Section 3 defines the service wrapper tool built to facilitate FAST users to create Resource Adapters of RESTful web services.

2 Web Services Wrapping

In the context of FAST philosophy, a service is a software element or system, often deployed within enterprise boundaries, designed to support interoperable Machine-to-Machine interaction over a network (e.g. Web Services, Databases, CORBA or RPC interfaces...). These services have to be used by the gadgets, but there are several complications to be resolved in order to communicate the front-end (i.e. the gadget user interface and logic) with those services, such as the disparity of interfaces and invocation mechanisms. Another issue is the complexity for an end-user to interact with those services. Hence, this interaction will be shift to a user-interaction paradigm through a user-friendly graphical interface (service front-end), allowing both humans and machines to interact with the services through a uniform fashion.

Therefore, a complex gadget in terms of FAST is aimed to provide a functional access from a graphical user interface to a set of services (SOAP or REST-based Web services) and data sources (Atom/RSS feeds). These services and data sources need to be modelled and encapsulated within the platform in order to allow the discovery and use of them by other components (i.e. forms and operators). This is called Resource Adapter in the FAST platform. Figure 1 illustrates their position and interaction with the rest of the building blocks within a gadget.

Once a brief background has been shown, the following sections explain how a user, in this case a resource developer, can build Resource Adapters, and how another user, a screen developer for instance, will discover and connect them while creating a screen. Hence, these three phases are called: building, discovery and connection.

2.1 Building

2.1.1 REST-based Web Services

REST is a term to describe an architecture style, not a standard, of networked systems. The acronym REST stands for Representational State Transfer. REST-based or RESTful web services [Fielding, 2000] are created identifying all of the conceptual entities or resources which want to be exposed as services. Those entities or resources should be nouns, not verbs (orders, tickets, etc.). Then, the interaction with those resources is made by convention using HTTP verbs such

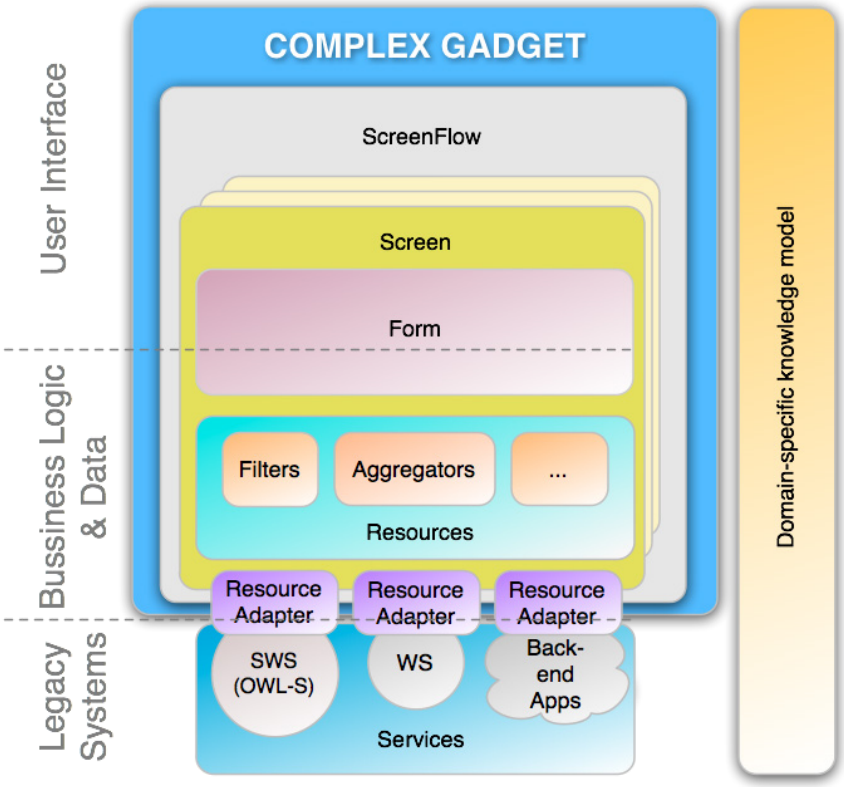

Figure 1: Complex Gadget Architecture

as GET, POST, PUT or DELETE, in order to retrieve, create, modify or delete them.

REST is lightweight (not a lot of extra xml markup, human readable results), but unlike SOAP-based web services, which have a standard vocabulary, and commonly used, to describe the web service interface through WSDL, RESTful web services are currently not formally described most of the times. For a service consumer to understand the context and content of the data that must be sent to and received from the service, both the service consumer and service producer must have an out-of-band agreement. This takes the form of documentation, sample code, and an API that the service provider publishes for developers to use. For example, the many web-based services available from Google, Yahoo, Flickr, Amazon, and so on have accompanying artefacts describing how to consume the services. This style of documenting REST-based web services is fine for use by developers, but it averts tools from programmatically consuming such services and generating artefacts specific to programming languages, as a web service described using WSDL allows. Nevertheless, Web Application Description Language (WADL) attempts to resolve some of these issues by providing a means to describe services in terms of schemas, HTTP methods, and the request or response structures exchanged, but this language is not widely adopted yet by RESTful web service developers.

To permit a high number of REST-based web services to be integrated to the FAST platform, the approach taken is from a manual development perspective. There is no need of a formal document such as WADL defining the web service, for this reason, building service wrappers for these services involve the correct understanding of the service by a human being and a tool to facilitate this task. This tool is explained in detail in Section 3.

2.1.2 SOAP Web Services

SOAP-based web Services or "Heavyweight Web Services" use Extensible Markup Language (XML) [Bray et al., 2006] messages that follow the Simple Object Access Protocol (SOAP) standard [Group, 2003] and have been popular with traditional enterprise, usually relying on HTTP for message negotiation and transmission. In such services, there is often a machine-readable description of the operations offered by the service written in a Web Services Description Language (WSDL) document. Hence, the advantage of using WSDL is that it can be programmatically processed.

Shortly, a WSDL definition of a service, regarding the WSDL 2.0 specification [Chinnici et al., 2007], will contain the following information:

Interfaces A set of Interface components describing sequences of messages that a service sends and/or receives.

Bindings A set of Binding components describing concrete message formats and transmission protocols which may be used to define the endpoints.

Services A set of Service components describing a set of endpoints at which a particular deployed implementation of the service is provided.

Element declarations A set of Element Declaration components defining the name and content model of the element information items such as that defined by an XML Schema global element declaration.

Type definitions A set of Type Definition components defining the content model of the element information items such as that defined by an XML Schema global type definition.

From a specific WSDL definition, a set of methods or operations can be easily extracted which encloses a set of inputs and outputs. These operations would be transformed into actions, and the inputs and outputs would be used to define Resource Adapter pre/postconditions. With the use of this information along with the XML Schema defining the types, a Resource Adapter can be semantically defined.

Moreover, there are several frameworks to facilitate the programmatic use of these formal definitions of web services. These frameworks shall be use to transform a WSDL definition into an executable Resource Adapter ready to use inside a gadget.

2.1.3 WSMO Web Services

Semantic web services bring a number of advantages in the creation of the resources adapters over classic web services. Formal and semantic descriptions of web services allow mechanisms to raise its exploitation in a more automatic way. In this section, some general notions of the Web Service Modelling Ontology (WSMO) will be explained and how FAST can make use of it.

In a few words, WSMO provides means to describe all relevant aspects of semantic web services

in a unified manner. A web service description in WSMO consists of five sub-components: non-functional properties, imported ontologies, used mediators, a capability and interfaces. However, we will focus on the capabilities and the interfaces since they are components which will make possible the integration of these services in FAST.

Capabilities and interfaces are the two types of Web Service description in WSMO. The capabilities describe the different functions of WSMO, while the interfaces specify:

1. How to communicate with a web service in order to avail of its functionality. This is called Choreography.
2. How the functionality of a web service is enabled by interacting with other Web Services. This is called Orchestration.

A web service in WSMO defines one and only one capability. The capability of a web service defines its functionality in terms of pre/postconditions, assumptions and effects. A web service capability is defined by specifying the following elements: non-functional properties, imported ontologies, used mediators, shared variables, precondition, postcondition, assumption, and effect.

Basically, the web service will offers to a client a postcondition when some conditions are met in the information space (precondition). Effects and assumptions can stay out of this study, since they will not be taken into account by any of the building blocks of a gadget.

NEED TO BE FINISHED

2.2 Discovering

In the building phase, every Resource Adapter is created sharing a common structure for screen components, such as forms, operators and resources. Hence, it will have a set of actions which will contain a set of preconditions to satisfy in order to be executed, and after its execution, it may produce any of the conditions of their postconditions.

For this reason, a screen developer during the screen development phase is able to reuse these screen components which are already stored in the catalogue, establishing which the screen's pre/postconditions are, and then the system will suggest several screen components which may satisfy these pre/postconditions. In a first step, the system will suggest screen components which have any of the preconditions which can be connected to any of the screen preconditions (the

connection will be represented by an internal pipe), and the same would happen for the postconditions. Once a screen component is inserted into the screen, the system will take its definition into consideration to recommend new screen components which can be connected to the ones the screen is composed of.

In these bases, the mechanism is relatively similar to the screen recommender, used to find the best screens to create a screen flow. Once more, it is based on the pre/postconditions of a set of building blocks.

3 RESTful Web Services Wrapper Tool

In FAST, the wrapper tool will be in charge of the Resource Adapters' building phase. At the moment, just supports RESTful web services. The building phase for this type of web services is done in two steps: a first step will be in charge of the construction of a service request and a second step will analyse the response got from the execution of the service, allowing the extraction of facts contained in that response and mapping them to domain-specific concepts from the ontologies used within FAST by any building block.

3.1 Constructing service requests

As a first approach, the interaction with these services will be limited to retrieve information to feed the gadgets using simply GET requests. A service request would be assembled using a certain URL and a set of parameters. As an example, Ebay Shopping web service will be studied. The following URL is invoked to retrieve a list of items corresponding to certain search keywords:

`http://open.api.sandbox.ebay.com/shopping?appid=KasselUn-efea-4b93-9505-5dc2ef1ceecd&version=517&callname=FindItems&ItemSort=EndTime&QueryKeywords=USB&responseencoding=XML`

As you may see, the URL invoked is http://open.api.sandbox.ebay.com/shopping and the parameters used in the example are:

appid this is the application ID obtained to use the API.

version the API version.

callname in this case FindItems to search through all items in Ebay.

itemsort sorting method for the list of items.

querykeywords list of keywords.

responseencoding format of the response message obtained by the invocation of the request.

If needed, a detailed specification of the Ebay Shopping API can be found at [eBay, 2010].

The above URL for searching items in Ebay is followed by the query parameters, which take the form *argument=value*, where the arguments and values are URL encoded, and are separated by an ampersand (&). For instance, the only relevant parameter the user would need to specify is

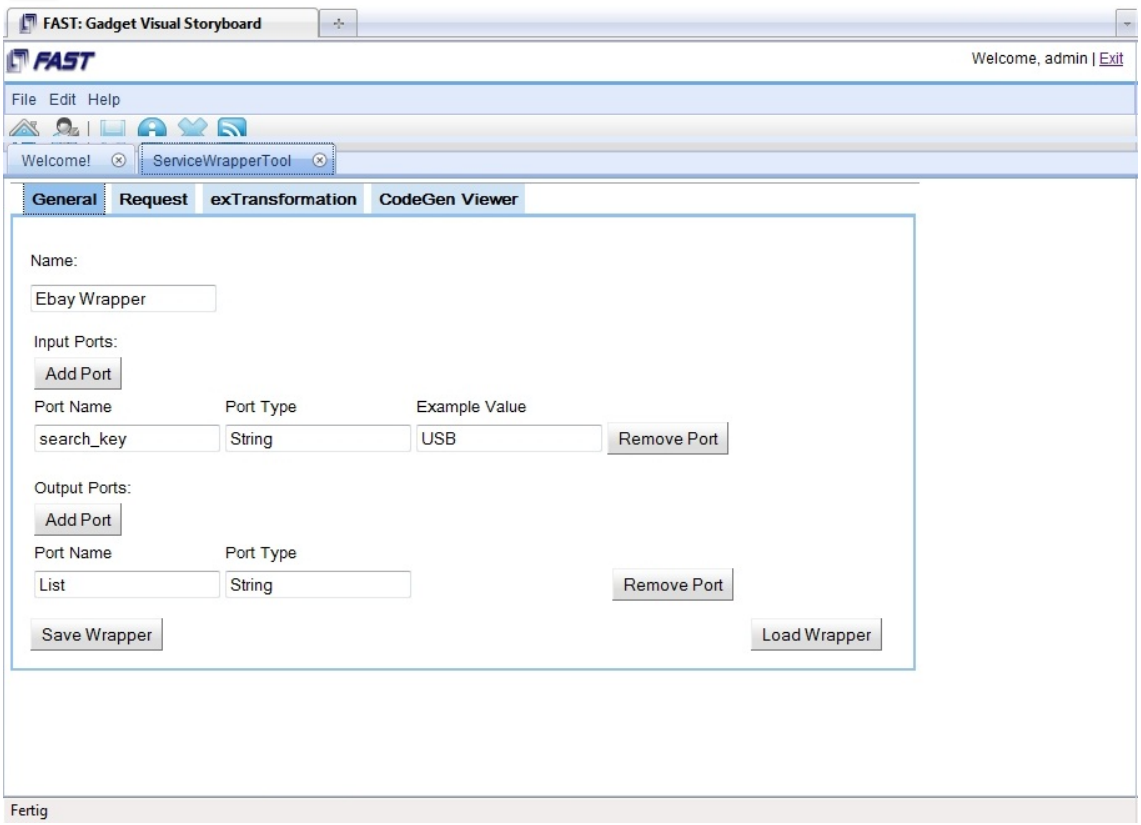

Figure 2: Configuring pre- and postconditions of a service wrapper (photoshoped screen dump)

querykeywords, thus somehow the service has to receive an input value for it, while the other parameters can be set to a default value. However, in order to develop a generic wrapper for the any service, all parameters might be set by the user.

To achieve this, service wrappers will be handled as pseudo-screens with preconditions and post-conditions (inputs and outputs). Therefore, these precondition ports might be used to determine values to parameters like query keywords coming from a building block inside the screen the service is placed or even an external screen as a screen precondition.

To define the preconditions and postconditions of a new service wrapper the FAST service wrapper tool provides a form that allows the editing of these entries, cf. Figure 2[1]. Note, we allow to edit example values for the preconditions. These example values may be used to test the service wrapper.

The service wrapper tool composes the request using a template string which will contain placeholders for precondition values. Before sending the request to the service, the placeholders are

[1]This screen dump is a combination of two different FAST tool screen dumps that show how the final tool will look like. We add a real screen dump ASAP.

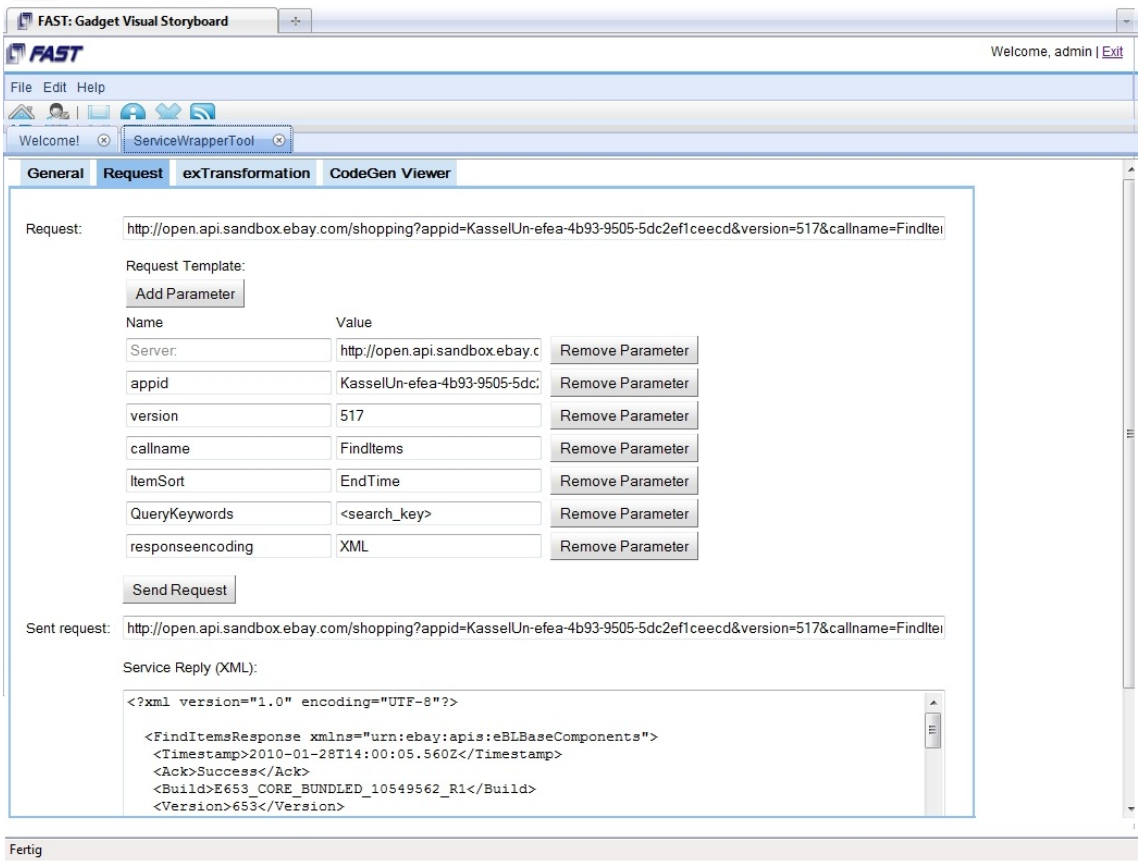

Figure 3: Constructing the service request URL and its parameters (photoshoped screen dump)

replaced with their corresponding values and the resulting URL is then ready to be sent the service request.

Figure 3 shows the screen to construct the service requests.

In the top input field of Figure 3, the user may drop an example http request taken e.g. form the service documentation e.g. from [eBay, 2010]. The tool analyses the example request and in the middle of the screen a form for editing the request parameters is provided. In our example, the user has connected the *QueryKeywords* parameter with the precondition *search_key* by adding a corresponding reference to the value field of that parameter. In addition, we have retrieved an access key which has been entered as value for the *appid* parameter.

Below the request paramter editing form of Figure 3, a *Send Request* button allows to validate the constructed service by sending it to the specified service address (via a server relay). Then the placeholders for preconditions are replaced with example values and the resulting http request is shown below the parameter form. In addition, the request is send and the response is shown on the bottom of that page. This gives the user a fast feedback whether the constructed request

works as desired.

3.1.1 Limitations

It is worth pointing out that currently the wrapper tool is able to construct input ports for the wrappers using just basic types. However, every building block accepts any (complex) concept as a pre/postcondition, so the wrapper tool needs to be adapted to permit any concept as input or precondition, extracting the required text to construct the request. This is planned for the third year.

3.2 Interpreting service responses

Once the service request is constructed and sent to the service provider, it will send back a response. This response message could be serialize in any format, though the most common formats used nowadays are XML or JSON among others. To continue the example started in the previous section, the response of the Ebay Shopping service will be in XML as specified in the request, which is the format supported by the wrapper tool.

3.2.1 Translation XML into Facts

Figure 4 shows the data tranformation tab of the wrapper tool.

Once the service response, in XML format, has been retrieved, the transformation tab shows it as an interactive object tree on the left side of Figure 4. To construct this interactive object, the XML document has been parsed into a DOM, and a simplified tree representation of that DOM is built up. This tree representation of the XML data is used as an input to construct transformation rules.

A transformation rule is used to analyse the XML data and to generate domain-specific facts from concepts from the ontologies used by the pre/postconditions of the different building blocks. A transformation rule is composed of three elements, cf the middle part of Figure 4. First, the *from* field indicates the XML elements to be translated by rule. These XML elementsare identified by

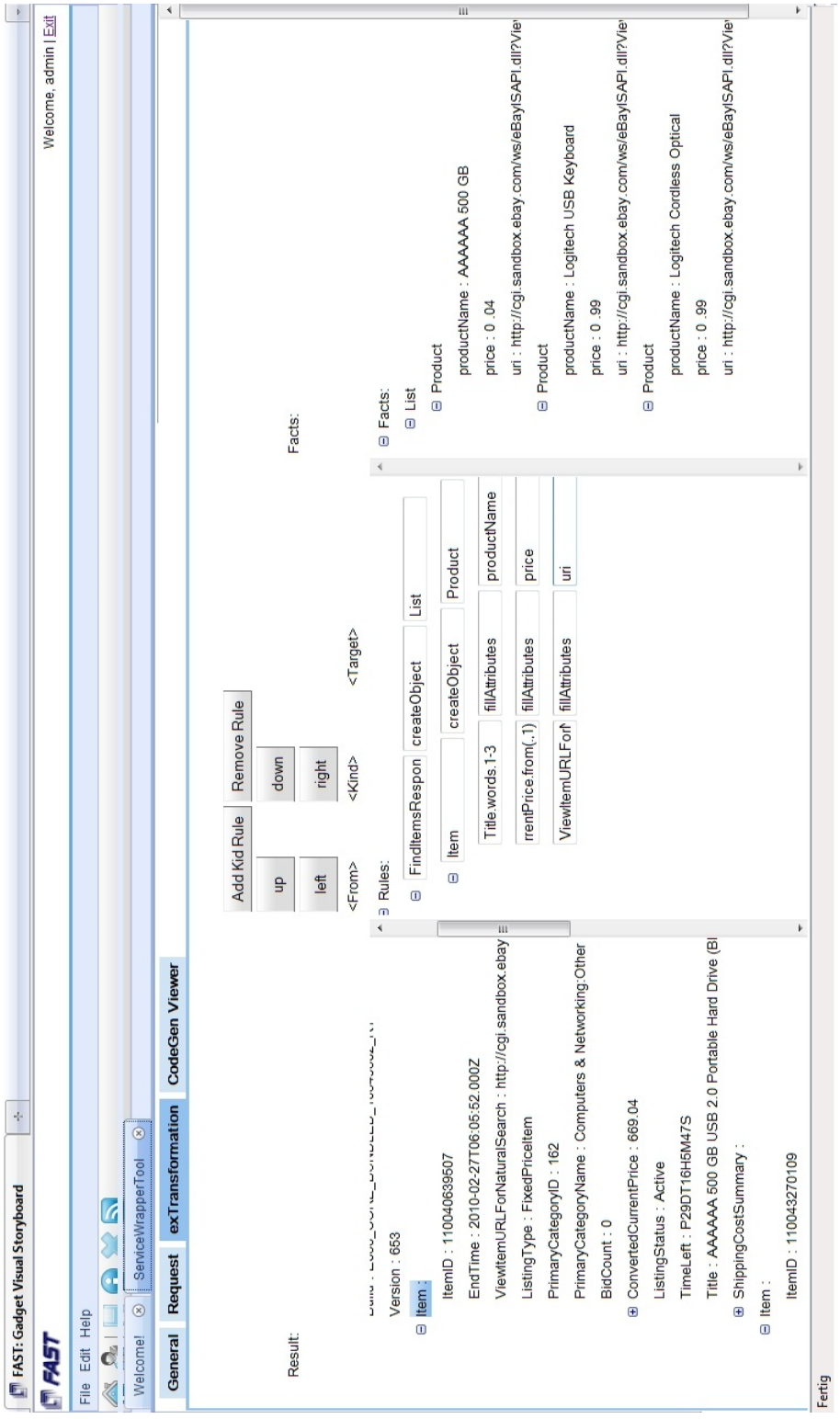

Figure 4: Interactive, rule based transforming of an XML response to FAST facts (photoshoped screen dump)

the tagname of a DOM element from the XML document. Second, the type of the rule will be set, taking one of the following values: *createObject*, *fillAttributes* or *dummy*. And third, the target of the rule specifies a certain concept or attribute, to be created or filled. A detailed explanation of the type of actions to be trigger from the transformation rules is:

action *createObject* specifies the creation of a new fact object. The type of that new fact is provided in the third compartment. In the example being explained, the root rule searches for XML elements with tagname *FindItemsResponse* and for each such element a *List* fact is created. The resulting facts are shown in a facts tree in the right of Figure 4.

action *fillAttributes* does not create a new fact but it fills the value of the attribute provided as third part of such rules. In our example, the third transformation rule searches for XML elements with tagname *Title*. Note, the rule is a sub-rule of the second rule, which generates *Product* facts. Thus, the sub-rule searches for *Title* tags only in the subtree of the XML data that has been identified by an application of the parent rule before. For example, the *Item* rule may just have been applied to the first *Item* element of the XML data. Then, the *Title* rule is applied only to the first *Item* sub-tree of the XML data and thus it will find only one *Title* element in that sub-tree (not visible in Figure 4). The value of that *Title* element is then transfered to the *productName* attribute of the corresponding *Product* fact. Actually, our *from* fields allows also to refer to parts of an XML attribute e.g. to *words* 1 through 3. It is also possible to combine constant text and elements of multiple XML tree elements.

action *dummy* does not create or modify any facts but such rules are just used to narrow the search space for their sub-rules. For example, in the Amazon case, the XML data for an item contains sections for *minimum price*, *maximum price*, and *average price*. Each such section contains the *plain price* and the *formatted price*. Thus, in the Amazon case, a rule that searches for *formatted price* elements within an *Item* element would retrieve three matches. Using a dummy rule, we may first search for *minimum price* elements and then search for *formatted price* elements within that sub-tree.

Since FAST is storyboard oriented, the service wrapper tool follows the storyboard idea as well. Any time, a change to a transformation rule is done, the transformation process is triggered and the resulted facts tree is directly shown. This process helps the user to deal with the slightly complex semantics of the transformation rules avoiding errors or mistakes. In addition, FAST is semantic-driven, therefore, the service wrapper designer shall retrieve the domain-specific types from a FAST ontology server together with the structure of each type, i.e. together with a descrip-

tion of the attributes of each fact. Thus, the transformation rule editor is able to provide selection boxes for the target element of the rules. For a *createObject* rule, this selection box shows the fact types available for that domain. For the *fillAttributes* rules, the selection box shows the attributes of the fact type chosen in the parent rule. In addition, we may provide some analysis tool, which will help to guarantee that the facts generated by the transformation rules conform to the fact types defined in the corresponding FAST ontology. This helps to ensure that the facts generated by the designed pseudo screen will be compatible for precondition ports of subsequent filter steps and or screens.

3.3 Generating a Resource Adapter

Once the wrapping of a service has been defined and tested in the service wrapper tool, it shall generate an implementation of the desired Resource Adapter in XML, HTML, and JavaScript, ready to be deployed and executed inside a gadget. This service wrapper implementation is compliant with the formats required by the Gadget Visual Storyboard Tool (GVS) and it shall be stored inside the FAST catalogue, in order to be found and used by any user.

3.4 Limitations

The rule driven approach presented above is somewhat limited. It is deliberately restricted to such a simple rule mechanism in order to keep things simple enough for end-users. Still, the selected approach suffices for most practical and real world cases. As a more complex example, the XML data for a person may provide two different tags for the first and the last name of a person. Contrarily, a person fact which conforms to a certain ontology for that domain may provide only one *fullname* attribute that shall be filled by a concatenation of the first and the last name. To achieve this, the *from* field of that tranforamtion rule might look like: `lastname"', "‘firstname`. We are also able to do some navigation in the XML tree to follow XRef elements. For example the attribute `grandmother` could be filled using `mother.mother` in the *from* field.

However, we there are some transformations that these rules cannot perform. For example, we do not support any mathematical operations. Thus, transforming e.g. Fahrenheit into Celsius

temperatures is not supported. To cover such cases, intermediate fact formats can be used which would allow generating facts to be further processed by additional filters. Such additional filters may be realized using (hand coded) operators, since some generic operators can act as filters for aggregation and conversions of facts from multiple sources. Then, service wrappers in combination with these filter operators will allow covering these complex cases.

4 Related Work

The rule based approach of the service wrapper tool has been inspired by triple graph grammars, cf. [Schuerr, 1994, Jahnke et al., 1997].

Write about some related work such as:

Wrapping WSDL-Described Web Services as Moby Services (using SAWSDL) `http://biomoby.open-bio.org/CVS_CONTENT/moby-live/Java/docs/sawsdlServlet.html`

References

[Alonso et al., 2003] Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2003). *Web Services*. Springer.

[Bray et al., 2006] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., and Cowa, J. (2006). Extensible markup language (xml) 1.1 (second edition).

[Chinnici et al., 2007] Chinnici, R., Moreau, J.-J., Ryman, A., and Weerawarana, S. (2007). Web services description language (wsdl) version 2.0.

[eBay, 2010] eBay (2010). ebay shopping apis.

[Fielding, 2000] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine.

[Group, 2003] Group, X. P. (2003). Soap version 1.2. W3c recommendation, W3C.

[Jahnke et al., 1997] Jahnke, J.-H., Schaefer, W., and Zuendorf, A. (1997). A Design Environment for Migrating Relational to Object Oriented Database Systems. *Software Engineering and Database Technology (Dagstuhl-Seminar-Report 173)*.

[Schuerr, 1994] Schuerr, A. (1994). Specification of graph translators with triple graph grammars. In Mayr, E. W., Schmidt, G., and Tinhofer, G., editors, *WG*, volume 903 of *Lecture Notes in Computer Science*, pages 151–163. Springer.

Appendix A (Lists of Tables and Figures)

List of Tables

List of Figures