A Multi-Actor Agnostic Platform for Web Services Agile Development and Deployment

Patrick Fontaine, Lamia Ferres, Bruno Bonnin, Philippe Larvet

Alcatel CIT, Research and Innovation, Route de Nozay, 91461 Marcoussis, France

Email: Patrick.Fontaine@alcatel.fr, Lamia.Ferres@alcatel.fr, Bruno.Bonnin@alcatel.fr, Philippe.Larvet@alcatel.fr

Abstract

A new service can be developed as an orchestrated composition of existing web services. Such aggregations of services are called "business processes". This paper describes SAMPAN (Simple & Agile Method and Platform for service AggregatioN), a multi-actor and agnostic platform that helps an analyst or a developer to produce and deploy "business processes" by the way of agile development. The user provides the use cases of the new service to be developed under the form of a semi-formal text or a graphical description, that invokes the legacy services and defines their dependencies and the way they are composed and orchestrated. From this textual or graphical description, SAMPAN generates 100% of the BPEL, C# or Java source code of the new service, and deploys this code onto a specific delivery platform (Microsoft IIS, Apache/Axis, BPEL Server). SAMPAN implements the MDA (Model-Driven Architecture) approach, through an original derivation of several fundamental concepts : masked internal object model, model transformations with embedded object modeling pattern, code generation patterns and deployment patterns.

Keywords

web service, business process, agile development, orchestration, service composition, service aggregation, MDA, code generation patterns, deployment patterns

Communities' Topics addressed

Collaboration@Work: New collaboration approaches, Community based Collaborative Workplaces

1. Problematic of Web Services Development

Web services, as stateless and contextless pieces of software, accessible from any point of Internet, are more and more suitable and convenient to build light and reusable applications.

Globally, from the point of view of their internal complexity, web services (WS) can be divided in two families : *elementary* WS and *composite* WS.

The elementary ones provide a basic service, comparable to mathematical libraries, and contain a low level of data transformation, embedded in few algorithms ; for example, translation services are elementary WS.

In the contrary, the composite WS, that we will call here *business processes,* are able to provide a high level service, and contain several levels of data accesses and transformations, given by the cooperation of several elementary services. Such a composite service can be called an *orchestrated* service ; for example, reservation services or secured-payment services are composite orchestrated WS.

If elementary web services can be built and relatively easily deployed with standard environments like Apache/Axis or .NET platforms, it could be interesting to have at one's disposal a powerful environment to build and deploy *composite* WS, made by an orchestrated aggregation of existing services.

One problem addressed by this desired environment is to express the aggregation of the legacy services and the *orchestration* of these services, their interaction and the way they have to run to reach their objective and provide the final service.

Several aggregation techniques exist today, and even if the industry is not yet agree on a common language, there are two languages that are considered as complementary:

- WSBPEL (Web Services Business Process Execution Language) or BPEL: it describes the interactions between web services, including business logic and order of the interactions
- WS-CDL (Web Services Choreography Description Language): it describes the messages exchanged between web services, including order and constraints on these exchanges.

Like BPEL, we focuses on the description of the orchestration. But BPEL is not the only way to describe a business logic: Java or C# can also be used.

Another problem addressed by this intended environment is to deploy easily the final built composite service, in order to put it at the disposal of final users.

These two problems are solved in SAMPAN, a multi-actor and agnostic Simple & Agile Method and Platform for service AggregatioN and deployment, through the different principles and concepts it embeds.

2 Applied Principles and Concepts

SAMPAN implements some basic principles and concepts in order to facilitate building and deployment of composite web services.

1. The web services built with SAMPAN are *business processes*, i.e. orchestrated aggregations of elementary legacy web services.

2. The *use cases* that describe the detailed functioning of the future business process, in terms of interactions between operations of existing services, can be expressed under the form of *semi-formal texts*, as well as *graphical* descriptions.

3. The development approach is *agnostic*: the use cases descriptions are mainly functional and do not contain any constraint concerning the final implementation language.

4. The service construction mechanism embedded in SAMPAN is based on the OMG's Model-Driven Architecture (MDA) approach, through several steps of model transformations, resulting in a final phase of automatic code generation.

5. The original semi-formal or graphical use cases are translated by SAMPAN, through an *object modeling pattern,* into an *internal object model*, that is transparent and masked to the user, and that underwites the functional cohesion of the future business process as well as the good operation of MDA model transformations.

6. The MDA transformations, whose the result is the final deployed business process, are based on *code generation patterns*, in order to build the final compilable and executable code from the internal object model, and on *deployment patterns,* in order to deploy the final service on the right server. These patterns are SAMPAN-embedded and selected by the user, who can choose by this way the final implementation language and destination of the service.

3 Using SAMPAN with an Example

Let us detail these main concepts and principles through the use of SAMPAN to develop a simple business process example.

Let us suppose we have, in our web service repository, several elementary services:

- a messaging service, which is able to send SMS, Faxes or e-mails to a given recipient;
- a directory service, that is a general service whose the role is to get some specific data belonging to a user, like his e-mail address, or the list of his buddies or preferred contacts, etc.;
- a user preference service, that manages the preferences associated to a given user.

3.1 Specifying the New Service

With these elementary components, let us suppose now we want to build a new service, that a first actor, an analyst, describes primarily in a fuzzy way, with the following text:

The name of the new service is: Broadcast Message. With this new service, the user wants to broadcast a message to several receivers, according to their messaging preferences. The new service uses some existing services but, at this level, I, analyst, ignore their exact names…

Description of the new service: the user gives his name, and the new service gets his corresponding contact list. Then, for each contact of this list, the new service gets the type of messaging corresponding to the contact, and according to this type, it sends a SMS, a Fax or an e-mail to the contact.

Then, the analyst has two ways for designing his new service, either the graphical way or the textual way. With SAMPAN the users can use their preferred one or both, thanks to a bridge between these two worlds.

3.2 Designing the New Service in a Graphical Way

Starting with this simple text, the analyst can use SAMPAN to design graphically a first level of description of the new service, seen as a *business process*. The analyst splits the description given by the text into several processing steps, and he designs these steps with the Business Process Designer of SAMPAN (BP Designer).

The organization of the new service in terms of steps and the interactions between these steps represent the original way with whom SAMPAN solves the problem of the internal orchestration of the business process.

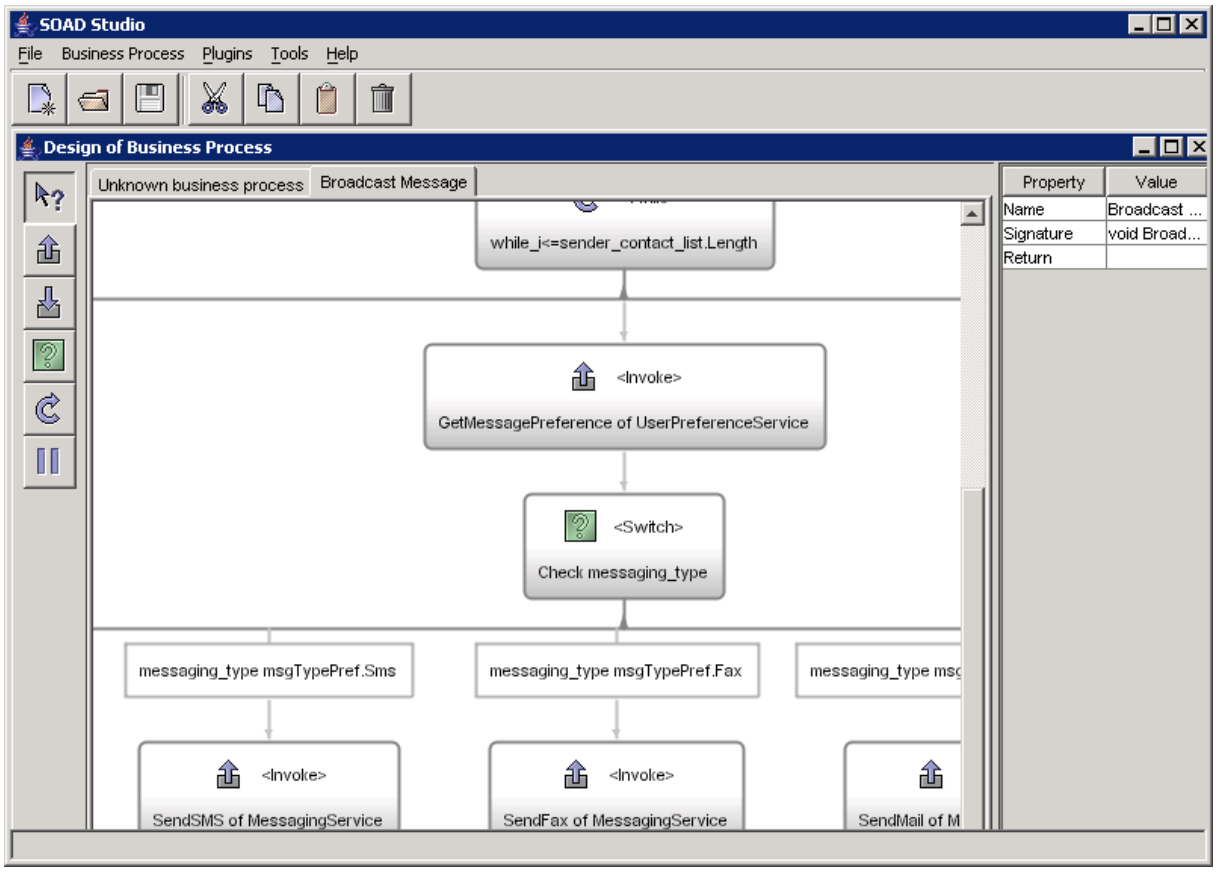

Figure 1 : Graphical view of the new service inside BP Designer

The result is a graphical first view of the new service (see Figure 1).

BP Designer can provide a formal representation of this view, under the form of a "BPXML" text – an XML representation of the business process (see Figure 2).

```
<?xml version="1.0" encoding="Cp1252"?>
<businessprocess
  name="Broadcast Message"
  signature="void BroadcastMessage(String sender_name, String message)"
  return="">
  <invoke
    name="GetEmailAddress of DirectoryService"
    asynchronous="false"
    wsdl="http://..."
    url=""
    operation="DirectoryService.GetEmailAddress"
    param_in_type="String"
    param_in_value="sender_name"
    param_out_type="String"
    param_out_name="sender_email_address"
    id="0" />
  <invoke
    name="GetMyContactList of DirectoryService"
    asynchronous="false"
    wsdl="http://..."
    url=""
```

```
    operation="DirectoryService.GetMyContactList"
    param_in_type="String"
    param_in_value="sender_name"
    param_out_type="String"
    param_out_name="sender_contact_list"
    id="1" />
  <while
    name="while_i&lt;=sender_contact_list.Length"
    initialisation=""
    condition="i&lt;=sender_contact_list.Length"
    step=""
    id="2">
    <invoke
      name="GetMessagePreference of UserPreferenceService"
      asynchronous="false"
      wsdl="http://..."
      url=""
      operation="UserPreferenceService.GetMessagePreference"
      param_in_type="String"
      param_in_value="sender_contact_list[i].Name"
      param_out_type="String"
      param_out_name="messaging_type"
      id="3" />
    <switch
      name="Check messaging_type"
      id="4">
      <case
        name="messaging_type msgTypePref.Sms"
        condition="messaging_type == msgTypePref.Sms"
        id="5">
        <…/>
      </case>
    </switch>
  </while>
</businessprocess>
```

Figure 2 : BPXML source corresponding to the new business process

3.3 Designing the New Service in a Textual Way

The analyst could also describe the new service under the form of a semi-formal text containing the use case and scenario of the new service. At this step, he can express requirements for the new service in an unperfect, fuzzy agnostic way:

```
Use Cases for Messaging Business Process

Actors
The actors are the message Sender and the Receiver.

Use cases
Use cases impacting the message Sender Actor
```

```
Use case:     Broadcast Message
Service:      Messaging Business Process
Actor:        The message Sender
Description:  The Sender wants to broadcast a message to several Receivers.

The new service uses some existing services, but at this step, they are not precisely
known.

SCENARIOS

Scenario for the new service: Broadcast Message
* The new service gets the sender's contact list from the user's name.
* For each contact of this list:
* get the type of messaging corresponding to the contact
* according to the messaging type:
* if Sms:
*             send a SMS to the contact
* if Fax:
*             send a Fax to the contact
* if Email
*             send an e-mail to the contact
```

Figure 3 : First level of semi-formal text (CNL) describing the Use Case & Scenario of the new service

This semi-formal description is another original way with whom SAMPAN solves the problem of orchestration: instead of using a standard language like BPEL, that is not human-readable, SAMPAN proposes to use a kind of natural language, with some specific and not numerous accepted constraints, that is human-writeable and readable. We call this language *Constrained Natural Language* (CNL).

SAMPAN is able to reverse this semi-formal description into BPXML, to be read directly by BP Designer and appear under a graphical form, and inversely, BP Designer can translate BPXML into a semi-formal text.

BP Designer can also provide a BPEL representation of the new service (see Fig.4 below), made by transformation from BPXML or semi-formal text.

```
<?xml version="1.0" encoding="Cp1252"?>
<process name="Broadcast Message" targetNamespace="http://alcatel.com/business-process/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:abp="http://alcatel.com/business-process/">
  <partnerLinks>
    <partnerLink name="Broadcast_MessagePL" partnerLinkType="abp:Broadcast_MessagePLT"
myRole="Broadcast_Message" />
    <partnerLink name="DirectoryServicePL" partnerLinkType="abp:DirectoryServicePLT"
partnerRole="DirectoryService" myRole="DirectoryServiceRequester" />
  </partnerLinks>
  <variables>
    <variable name="Broadcast_Message_Args" messageType="abp:BroadcastMessageRequest" />
    <variable name="Broadcast_Message_Return" messageType="abp:BroadcastMessageResponse" />
    <variable name="invoke_0_Args" messageType="abp:GetEmailAddressSoapIn" />
    <variable name="invoke_0_Return" messageType="abp:GetEmailAddressSoapOut" />
  </variables>
  <faultHandlers />
  <sequence>
```

```
    <receive partnerLink="Broadcast_MessagePL" portType="abp:Broadcast_MessagePT"
operation="BroadcastMessage" variable="Broadcast_Message_Args" />
    <sequence>
      <assign>
        <copy>
          <from
expression="bpws:getVariableData(&apos;Broadcast_Message_Args&apos;,&apos;sender_name&apos;)" />
          <to variable="invoke_0_Args" part="name" />
        </copy>
      </assign>
      <invoke partnerLink="DirectoryServicePL" portType="abp:DirectoryServiceSoap"
operation="GetEmailAddress" inputVariable="invoke_0_Args" outputVariable="invoke_0_Return" />
    </sequence>
    <reply partnerLink="Broadcast_MessagePL" portType="abp:Broadcast_MessagePT"
operation="BroadcastMessage" variable="Broadcast_Message_Return" />
  </sequence>
</process>
```

Figure 4 : BPEL description of the new service

When the analyst has introduced in SAMPAN the first level of description for the new service, a second actor – the development engineer – can now come after him and detail the description.

At this step, the activity of the engineer consists of adding technical details, to allow a correct code generation: exact names of the services, signatures and parameters, etc.

These details can be inserted in SAMPAN in a graphical way, with BP Designer, as well as in a textual way, expressed in semi-formal text.

3.4 Checking the Consistency of the Scenario

Once these details are inserted, a consistency check can be ordered in SAMPAN, by comparing the signatures and parameters selected by the development engineer with the original WSDL descriptions of the legacy repository services. A specific SAMPAN command allows to find and memorize the WSDL descriptions of the services required by the scenario for the new service.

3.5 Internal Object Model of the Scenario

The central key of SAMPAN mechanism consists in embedded text-to-formal-language and WSDL-to-formal language transformation patterns (see one of these patterns Fig. 6 above) whose the role is to translate the semi-formal text of the use case and scenario, as well as the WSDL legacy services descriptions, into a formal, internal object model on which the further operations of check consistency, code generation, etc. will be processed.

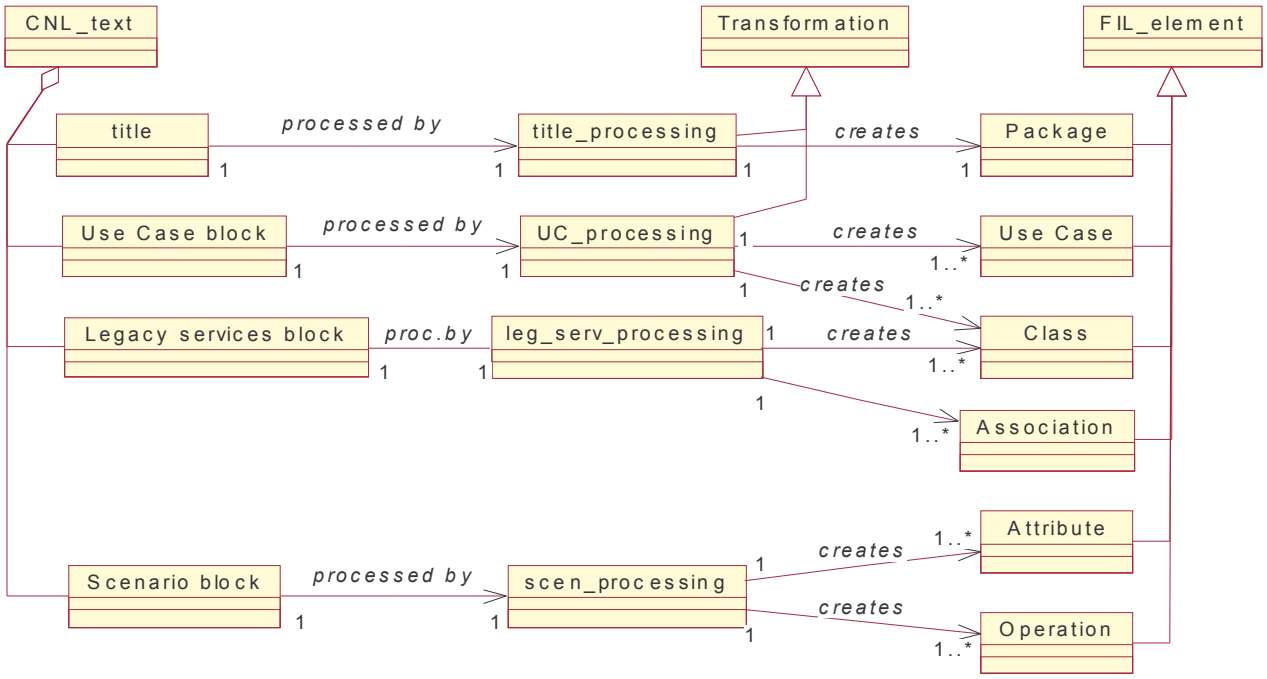

Figure 6 - Model of the embedded text-to-Formal Internal Language (FIL) transformation pattern

This internal object model is expressend in a formal way and stored into the SAMPAN proprietary database:

```
// Formal Internal Object Model Language
// for the new service Broadcast Message
//-----------------------------------------
// package declarations
package Messaging Business Process
package MessagingService
package UserPreferenceService
package DirectoryService

// class declarations
// class [<package_name>]<class_name>
class [MessagingService]MessagingService
class [DirectoryService]DirectoryService
(...)

// association declarations
association MessagingBusinessProcess (1..1) uses (1..1) DirectoryService
(...)
association DirectoryService (1..1) uses (1..n) ContactInfo

// attribute declarations
attribute ContactInfo.Name:string
(...)
attribute MessagingBusinessProcess.theDirectoryService:DirectoryService
attribute MessagingBusinessProcess:type_of_messaging:String

// operation declarations
```

```
operation UserPreferenceService.GetUserProfile(userName:string):UserProfile
operation DirectoryService.GetMyContactList(username:string):ArrayOfContactInfo

// operation declaration + operation detailed description
operation  MessagingBusinessProcess.BroadcastMessage(String sender_name, String message)
display_message("The CDS user wants to broadcast a message to several Receivers")
sender_email_address = theDirectoryService.GetEmailAddress(sender_name)
sender_contact_list = theDirectoryService.GetMyContactList(sender_name)
subject_of_call = "Publishing test"
for (i<sender_contact_list.Length)
type_of_messaging = theUserPreferenceService.GetMessagePreference(sender_contact_list[i].Name)
switch (type_of_messaging)
case Sms:
theMessagingService.SendSMS(sender_email_address,sender_contact_list[i].Mobile,subject_of_call,me
ssage)
case Fax:
theMessagingService.SendFax(sender_email_address,sender_contact_list[i].Fax,subject_of_call,messa
ge)
case Email: default:
theMessagingService.SendMail(sender_email_address,sender_contact_list[i].EMail,subject_of_call,me
ssage)
end switch
end for
// end of FIL
```

Figure 7 - Internal formal object model of the business process Broadcast Message

3.6 Generating the Source Code from the Internal Model

The internal formal object model – which is masked to the user – contains all the technical details to allow a consistent code generation.

The development engineer can now select the target language (BPEL, C# or Java), and through the specific code generation embedded pattern, the internal model will be translated into the target source language.

For example, the Figure 8 below shows the generated C# source code for the new service Broadcast Message:

```
//-----------------------------------------
// C# Class Name : MessagingBP
//
// Creation Date : 11-15-2004
// Generated by SAMPAN
//-----------------------------------------
 <%@ WebService Language="c#"
Class="PoweredBy.BusinessProcess.Messaging.MessagingBP" %>

using System;
using System.Web.Services;
using PoweredBy.WebServiceProxy;

namespace PoweredBy.BusinessProcess.Messaging
{
        [WebService]
        public class MessagingBP:System.Web.Services.WebService
        {
                // Attributes
                private DirectoryService theDirectoryService;
```

```
                private MessagingService theMessagingService;
                private UserPreferenceService theUserPreferenceService;
                private String sender_email_address;
                private ContactInfo[] sender_contact_list;
                private String call_subject;
                private msgTypePref messaging_type;
                private int i;

                // Constructor
                public MessagingBP()
                {
                        theDirectoryService = new DirectoryService();
                        theMessagingService = new MessagingService();
                        theUserPreferenceService = new UserPreferenceService();
                }

                // Operations
                [WebMethod][SoapRpcMethod]
                public void BroadcastMessage(String sender_name, String message)
                {
                        sender_email_address = theDirectoryService.GetEmailAddress(sender_name);
                        sender_contact_list = theDirectoryService.GetMyContactList(sender_name);
                        call_subject = "CDS Messaging";
                        for (i=0; i<=sender_contact_list.Length; i++)
                        {
                        messaging_type =
theUserPreferenceService.GetMessagePreference(sender_contact_list[i].Name);
                        switch (messaging_type) {
                                case msgTypePref.Sms: {
        theMessagingService.SendSMS(sender_email_address,sender_contact_list[i].Mobile,call_subje
ct,message);
                                        break; }
                        (...)
                        } // end switch
                        } // end for
                } // end of operation
        } // end of class
} // end of namespace
```

Figure 8 – C# generated source code of the business process Broadcast Message

3.7 Testing the Generated Service

Once the source code has been generated as an ".asmx" file, for example, it can be immediately tested by the test environment embedded in SAMPAN.

The Web Service tester parses the wsdl file corresponding to the web service url (.asmx?wsdl), to retrieve the description of the service. This description (here a list of methods and their parameters) is presented in an HTML format. The user can then choose any method, enter needed parameters and invoke the method.

To be able to do this, the tester tool contains: a wsdl parser to get the description of the service, an HTML generator for the presentation, and a generic web service client for invoking operations.

3.8 Deploying the Final Service

The development engineer can now deploy the final service, by selecting the type of intended deployment, for example Microsoft IIS, Apache/Axis or BPEL Server.

Then, through the deployment embedded pattern, the final source code of the new service will be moved to the appropriate directory, and the corresponding proxies will be generated and deployed.

4. Perspectives and Work in Progress

Today, the adding of technical details to the first level of description (see Figures 1 and 3), in order to allow a right code generation, is a manual activity, made by a development engineer.

We are currently working on a mechanism that will allow to associate, semi-automatically or fully automatically, the exact names of services and signatures to the original use case description.

This association will be made by a pattern matching between the lexical trees extracted from the original text of the use case and the lexical trees associated to the legacy services in the WS repository.

For example, the original requirement expressed in the text:

The user gives his name and the new service gets his corresponding contact list

will be lexically analyzed, and the following lexical tree will be associated to the original request:

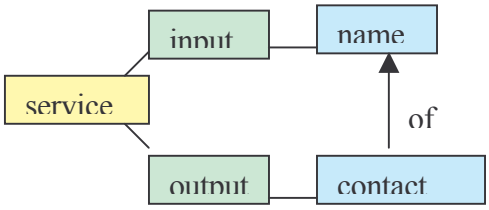

Figure 9 - Lexical tree associated to the request

Then, an automatic service discovery will be done in the service repository, to check and find the right service that could cover the request.

This implies every service in the repository has its own formal-expressed lexical tree.

The service discovery mechanism will discover, for example, the following lexical tree, associated to DirectoryService in the service repository (see Fig.10 below).

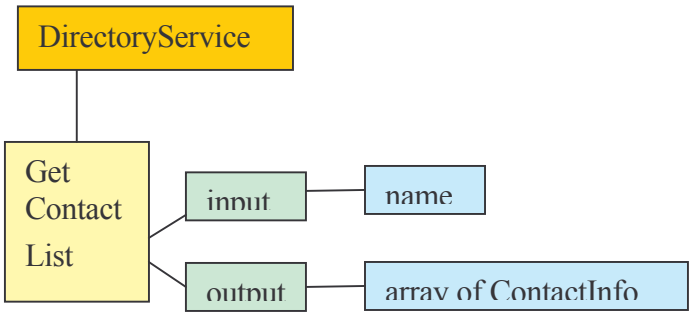

Figure 10 - Lexical tree associated to the GetContactList operation of DirectoryService

The automatic matching mechanism associated to the automatic service discovery will be then able to "recognize" the right service and to replace, in the text of the use case, the original sentence by the complete semi-formal expression of the corresponding signature, and will propose afterwards to the user to validate this replacement.

By such a step-by-step iterative process – discovery, matching, replacement, validation – the original unprecise, unperfect and fuzzy use case will be little by little transformed into a final detailed semi-formal description of a consistent business process, ready to be generated in C# or Java to become a new service.

5. Conclusion

We wish to insist on the interest of the global SAMPAN approach, which is an aggregation of several elementary interesting approaches that really allow to make web services better, faster and cheaper, with a high level of quality:

- with the *business process* approach, a new service is seen as an orchestrated aggregation of several existing elementary web services;

- with the *multi-actor* approach, different actors – analyst, developer, etc. – can use SAMPAN at different detail levels;

- with the *multi-form* approach, SAMPAN can read, transform, and reverse semi-formal or fuzzy texts, as well as graphical descriptions;

- with the *agnostic* approach, the use cases and scenarios of the intended business processes are expressed in a way that is independent of the final implementation language;

- with the embedded and masked MDA approach, an internal formal and consistent object model is built, which is hidden to the user who only manipulates high-level descriptions;

- with the *agile* approach, which is a positive consequence of the MDA-embedded approach, the user only enters his high-level use cases and selects his choices, and a consistent new service is built in a few seconds under his amazed eyes…

References

BPEL Editorial Team, *BPEL Learning Guide,* February 2005, http://searchwebservices.techtarget.com/originalContent/0,289142,sid26_gci880731,00.html

BPMI, Business Process Management Initiative, *BPML, Business Process Modeling Language Specifications,* BPMI.org, 2002, http://www.bpmi.org/specifications.htm

BPTrends, *Business Processes and the OMG, An Overview,* February 2204, http://www.omg.org/bp-corner/bp-files/ The_OMG_BP_Corner_INTRO_Paper_3-2-04.pdf

Dubray Jean-Jacques, *BPML for Web services,* June 2004, in http://www.ebpml.org/bpel4ws.htm

Jishnu Mukerji, Miller Joaquin, *MDA Guide,* OMG, June 2001, http://www.omg.org/cgi-bin/doc?omg/03-06-01

Kavantzas Nickolaos, *WS-CDL, Web Service Choreography Description Language,* December 2004, http://www.ebpml.org/ws_-_cdl.htm and http://www.w3.org/TR/ws-cdl-10/

Kavantzas Nickolaos & al., *Process-centric realization of SOA : BPEL moves into the limelight,* Web Services Journal, Nov.2004, http://www.findarticles.com/p/articles/mi_m0MLV/is_11_4/ai_n7071401

Nanda Mangala & al., *Decentralized Orchestration of Composite Web Services,* IBM Research Computer Science, Innovation Matters, November 2004, http://www.research.ibm.com/compsci/project_spotlight/distributed/

Peltz Chris, *Web services orchestration, a review of emerging technologies, tools, and standards,* Hewlett-Packard Co, January 2003, http://devresource.hp.com/drc/technical_white_papers/WSOrch/WSOrchestration.pdf

Smith Howard, *BPM and MDA, Competitors, Alternatives or Complementary,* Business Process Trends, White Paper, July 2003, http://www.bptrends.com/publicationfiles/07%2D03%20WP%20BPM%20and%20MDA%20Reply%20%2D%20Smith%2Epdf

Bonnin Bruno, *SAMPAN Specifications and Architecture,* Alcatel internal publication, 2004

Bonnin Bruno, *BP Designer Specifications and Architecture,* Alcatel internal publication, 2004

Bonnin Bruno, *BPXML Specifications,* Alcatel internal publication, 2004

Ferres Lamia, *Web Service Deployer, Specifications and Architecture,* Alcatel internal publication, 2004

Ferres Lamia, *Web Service tester, Specifications and Architecture,* Alcatel internal publication, 2004

Fontaine Patrick & Larvet Philippe, *Use Case Wizard,* Alcatel internal publication, 2004

Larvet Philippe, *Constrained Natural Language,* Alcatel internal publication, 2004

Larvet Philippe, *Lexical tree-ed Web services,* Alcatel internal Publication, 2005