Flang and Web Programming

Andrei Mantsivoda

Irkutsk State University,
664003 Irkutsk, Russia
andrei@baikal.ru
http://www.baikal.ru/andrei

Abstract. Flang is a logic language augmented with functional and object-oriented features. Flang is intended for the development of large and scalable Web programming projects. Components of Flang are tuned on processing of semi-structured data. Flang can be applied in various fields of Web programming on both the server and client side. Close connections with Java, on which the Flang system is built, allows incorporation of a huge number of tools developed within the Java community.

1 Introduction

Flang is a relatively old, although hardly known language [1], [2]. A new version of it came from practice. It appeared when we developed a large knowledge management system. A lot of information in this system was presented in XML, and we needed quite sophisticated services to manipulate this data. Our attempts to apply well-known means and libraries (like XSLT, JDOM, etc.) failed due to their inappropriateness. Then we recalled Flang and decided to revive it and augment it with special tools for semi-structured data processing. We augmented the language in a straightforward way, worrying only about simplicity and naturalness of added tools (from our point of view, of course). What we wanted to achieve was to build a methodologically transparent language which would allow us to easily develop and debug data domains, services and methods.

Flang can be considered a superstructure over two technologies: logic programming and object oriented programming. In Flang object-oriented tools are used for explicit construction of data domains as hierarchical systems of classes. These classes are provided with methods which are defined in a functional-logic style.

Flang is focused on Web programming applications. We believe that development of logic programming tools for semi-structured data processing can have significant impact on the Web programming technologies. LP techniques can process HTML and XML documents efficiently and do this in a uniform manner in different Web programming fields (in which barely consistent technologies, like XSLT, JSP, PHP, JDOM, etc., are currently used).

In Flang, XML and HTML documents are treated as mathematical terms. Understanding semi-structured objects as terms allows us to apply to them all the power of the functional and logic means. We found it very important to support in the

language the standard appearance of XML and HTML expressions. This has brought faster development and debugging of Flang programs and, we hope, better chances for acceptance of the language by a wider circle of Web programmers. Trial versions of Flang which we have been developing during the last two years show that this approach allows efficient processing of semi-structured documents. Flang was successfully applied to the following types of tasks:

- Transformations of XML and HTML documents into other XML and HTML documents (the scope of XSLT applications) and into other data formats (executable codes, texts, database records, multimedia);
- Development and support of dynamic and interactive Web resources on the server side (the scope of servlet technologies, ASP, etc);
- Development of compound projects combining tasks of semi-structured data processing, advanced data and knowledge analysis, automated data generation, etc.

The Flang's core is a functional-logic 'engine' which determines its procedural semantics. This small and fast engine runs in the environment of 'the outer world'. The outer world is described and manipulated by Flang's object-oriented tools. A feature of Flang is that it uses atoms for presentation of object attributes (fields). That is, in Flang atoms denote attributes of objects. The logic engine works with all data that comes from the outer world. The engine's communications with the outer world are established through the unification procedure and atoms values. A Flang program can change the outer world by changing attributes. Unification in Flang is more general than that in Prolog, because it takes into account all the variety of types within the class system. The Flang unification conception is considered in more detail below.

2 Definitions in Flang

Consider examples of simple Flang definitions. Let us start with a functional definition of the `append` function

```
append([], X) :- X;
append([X|Y], Z) :- [X|append(Y, Z)];
```

Here, like in Prolog, [X|Y] denotes a list with head X and tail Y. A function definition in Flang consists of the list of rules having the form

```
F:- P1,…,Pk,G;
```

Rules in Flang are interpreted as follows: if `P1,…,Pk` hold then `F` equals `G`. The procedural semantics of Flang includes Prolog's. In particular, backtracking plays an important role in Flang. Let's make a query to the Flang system:

```
?- append([1,2,3], [4,5,6]);
    [1,2,3,4,5,6]
```

Now there are a couple of small examples of definitions with XML terms. In Flang we can define a function, the result of which is an XML document:

```
matrix(M11, M12, M21, M22) :-
  <matrix>
```

```
    <tr><td>_M11;</td><td>_M12;</td></tr>
    <tr><td>_M21;</td><td>_M22;</td></tr>
  </matrix>;
```

Here, `matrix/4` is a function which returns the matrix of four elements. The standard syntax can be used in Flang to represent XML terms. It is very useful for development of transparent and readable programs and faster debugging. As we see XML expressions can be parameterized. Logic variables (and, more general, evaluated expressions) can be inserted in XML terms in the frame of symbols '_' and ';'. This resembles XML entity references (like `<`) but without ampersands. Ampersands influenced seriously on readability of programs, so we had to replace them with '_'.

A Flang parser translates XML expressions into standard logical terms (structures) with special functors. There are seven functors which represent XML components: `xDoc/1`, `xElem/4`, `xComments/1`, `xEntityRef/1`, `xCDATA/1`, `xPI/3`, `xNS/2`. For instance, `xDoc/1` forms a document root, `xElem(Name,Attr,Cont,NS)` represents an XML element with name `Name`, attributes `Attr`, contents `Cont` and namespace `NS`. In particular, the following XML segment

```
<address type="post">
  <street>Gagarin</street>
  <bld>115</bld>
</address>
```

is represented by the term

```
xElem("address", {"type"->"post"},
    [xElem("street", {}, ["Gagarin"], _),
        xElem("bld", {}, ["115"], _)], _)
```

This representation is awkward for human eyes, but convenient for automated processing because represents an XML document in the tree form and allows application of unification and other logic and functional tools.

Consider one more example. The following definition, having as input a list of strings, aggregates them into an HTML page. The type of an HTML list is also a parameter of the function:

```
stringList(List, ListType) :-
    <html><head/>
        <body>
            <_ListType;>_sequence(List);</>
        </body>
    </html>;

sequence([]) :- [];
sequence([String | Tail]) :-
        [<li>_String;</li> | sequence(Tail)];
```

In the first rule the evaluated expression `_sequence(List);` is inserted in the XML term. Let's make a query:

```
?- stringList(["John", "Paul", "George"], "ol");
    <html><head/>
        <body>
            <ol>
               <li>John</li>
               <li>Paul</li>
               <li>George</li>
            </ol>
        </body>
    </html>;
```

In Flang programs HTML terms must follow the XML syntactic rules. But invocation of the predefined `writeHTML/1` method generates an ordinary HTML file:

```
?- stringList(["John", "Paul",
                "George"]).writeHTML("names.html");
```

3 Flang and Semi-structured Data

XML and HTML are the key data exchange formats in the Web. While HTML is focused on the graphical presentation of information, XML provides logical data structuring tuned on data domains of discourse. XML is very important because it is a de facto standard for exchanging data between automated services in the Web. For storing information XML is less efficient. Therefore XML input data is often converted by automated services into some internal forms, e.g. databases records. This information can be also converted into formats oriented to human understanding. In the latter case, XML documents must be translated into structures convenient for visual perception. Thus special methods, which have XML documents as input and return as output information in another form, are needed. On the other hand, inverse problems of dynamic generation of semi-structured data from diverse information sources are also very important, including development of dynamic and interactive Web pages, sites and portals.

Thus, input and output formats in Web programming can be diverse: HTML documents, XML documents in various vocabularies, executable codes, and database records, multimedia. Today poorly interconnected tools for processing and generation of Web resources are popular. As a rule, each technology solves a relatively narrow problem. Therefore, development of complicated projects demands involvement of several technologies. Relatively straightforward transformations of XML documents can be arranged with XSLT [15]. Unfortunately, the basic version of XSLT 1.0 implements the scheme "one input, one output". Due to incredibly fast expansion of XML around the Web, this scheme appears very restrictive. The second problem is brought by certain unbalance in the language itself. Some tasks are solved in XSLT relatively quickly, but traps are near – sometimes 'almost the same' neighboring tasks

appear surprisingly difficult. The paradigm of XSLT as an extended version of style sheets is also very restrictive because many tasks go far beyond the problems of data display. In the further version of XSLT these problems were mostly fixed. Nevertheless, it is not easy to get rid of the restrictive nature of the basic XSLT ideas.

If a task can not be solved with XSLT, we have to take a long jump towards, say, Java, a very powerful and universal language which provides considerably lower level of programming. Many Web programming technologies have been developed in Java. For instance, an important servlet technology and Java Server Pages (JSP [19]) support creation and manipulation of dynamic resources on the server side (we can also mention here similar means of ASP, PHP, etc.). Like XSLT, JSP is also focused on generation and transformation of XML and HTML documents. But there is a fundamental distinction between JSP and XSLT. Servlets are based on the idea of understanding documents as made of 'string pieces', without taking into account their internal structure. When it is necessary to handle in Java the internal structure of documents, we have to use other approaches, such as JDOM [16].

Even this brief analysis shows that we have for Web programming incoherent tools which use inconsistent representations of the same data. This is bad for the Web today and the poor basis for the Web of a new generation. In this situation logic programming augmented with object oriented tools has a good chance to show its capabilities. The conception of the Semantic Web is also significantly based on the logic means [17], and this also expands areas for declarative languages applications.

To meet these challenges Flang is based on the following principles:

1. It incorporates object oriented tools for construction and presentation of classes' hierarchies;
2. It incorporates functional-logic tools for development of methods applied to these objects;
3. It treats semi-structured (XML and HTML) documents as mathematical objects (terms) and supports their usual syntactic representation;
4. It is developed on Java technologies and provides 'on the fly' incorporation of Java methods and libraries.

We tried to develop Flang as a universal language that solves equally well Web programming tasks on both the server and client sides. We included in the language means which permit development of large, distributed and scalable Web projects. Binding Java, with its huge number of libraries and applications, gives us a strong established foundation to Flang. Representation of Web data as structured mathematical objects allows for the application of advanced means of logic and functional programming. We believe that functional and logic methods, which take into account the hierarchical nature of semi-structured information, are able to act as a basis for multi-purpose, simple and essentially new tools which integrate the key technologies of today's Web.

4 Declaring classes in Flang

Declarations of classes and objects in Flang resemble those in Java and can be considered as a simplified version of a Java scheme. This is a deliberate step which

allows development of complicated projects involving Flang and Java codes in a coherent and uniform way. Consider an example of a Flang class declaration for a well-known object:

```
class $Stack {

  private $List stack;

  $Stack() :- stack := [];
  $Stack(List) :-
        stack := [],
        addList(List);
  empty() :- stack = [];
  push(Obj) :- stack := [Obj | stack];
  pop() :-
        [Obj | Tail] = stack, !,
        stack := Tail,
        Obj;
  pop() :- $Exception("Stack is empty");

  private addList([]) :- true;
          addList([X|Y]) :- push(X), addList(Y);
}
```

Names of classes in Flang always start with the symbol '$'. The class above contains one field (attribute) denoted by the atom stack. In this field a list representing the current state of the stack is stored. Fields in Flang must be declared. Values of fields are changed by the assignment operation `atom:=value`. The declaration of `$Stack` contains also two constructors which generate objects of this class – `$Stack/0` and `$Stack/1`. The first constructor generates an empty stack. The second one generates a new empty stack and inserts in it elements from a list. This constructor uses the auxiliary method `addList/1`. `$Stack` also contains definitions of three standard stack methods – `push`, `pop` and `empty`. To demonstrate the use of this class, define a method which sums up elements of a stack.

```
sum(Stack) :- Stack.empty(), !, 0;
sum(Stack) :- X = Stack.pop(), X + sum(Stack);
?- sum($Stack([1, 2, 3, 4, 5]));
      15
```

The essential difference between Flang and Prolog is that in Flang the role of atoms is considerably changed. Atoms in Flang are used as fields in which attributes of objects are stored. Atoms behave as imperative variables, and this essentially distinguishes them from logic variables (beginning with capital letters). At the level of methodology, it seems important to separate pure functional-logic mechanisms from schemes demanding the destructive assignment operator. Assignment in Flang is used only for setting the current state of the outer world. We can say that functional-logic methods are wrapped in an object-oriented environment which is determined by

atoms' values. The methods can change the outer world by changing the values of the attributes.

5 Object-oriented Unification

Unification is an important component of logic programming which provides a flexible and powerful scheme for passing parameters and constructing the result. Since Prolog is a type-free language, with only one basic set of elements defined by Herbrand universe, unification in Prolog is also uniform. On the other hand, in Flang a complex hierarchical system of classes is implemented. Thus, it is natural to generalize the notion of unification/matching so it can be tuned to any class of that objects. This idea can be implemented if we allow explicit definitions of the unification procedure in class declarations. One of such schemes has been incorporated in Flang.

In Flang the user can define unification procedure for any class. For this he or she should include in the class declaration a definition of the method `unify/1`. Let us demonstrate this on the `$Stack` example. We can add to the `$Stack` class declaration the following rules:

```
unify($Stack()) :-  !, empty();
unify($Stack(Top, Tail)) :-
        stack = [Top | Tail],
        stack := Tail;
```

The first rule defines unification with the empty stack, while the second one unifies an argument with a stack consisting of the top element `Top` and the rest `Tail`.

Resting on these definitions we can rewrite the method `sum/1` in the following way:

```
sum($Stack()) :- !, 0;
sum($Stack(Top, Tail)) :- Top + sum(Tail);
```

It is important that object-oriented unification does not spoil semantics of Flang, because it can be easily transformed into an equivalent program without usage of generalized unification. For instance, the definition of `sum/1` can be rewritten as follows (here `quote/1` stops evaluation of its argument making it a structure):

```
sum(St) :- St.unify(quote($Stack())), !, 0;
sum(St) :-
      St.unify(quote($Stack(Top, Tail))),
      Top + sum(Tail);
```

Moreover, object-oriented unification allows development of more compact and transparent programs, because it defines and encapsulates most standard methods of access to objects components. This explains why we call this modification of parameters passing as 'object-oriented' unification. Another advantage of generalized unification is that it specifies and declares the types of arguments. Now we are investigating methodological and practical aspects of this unification scheme.

Special unification is defined for a number of built-in Flang classes. One of the basic classes of Flang is `$Map`. It behaves like a hash table and is used, in particular, for representation of attributes in XML terms. Unification for objects of `$Map` is defined as follows. If we have, for example, the following rule

```
f({"a"->1, "b"->2 | Rest}) :- Rest;
```

with the formal parameter `{"a"->1, "b"->2 | Rest}`, this parameter is unified with maps which contain the pairs `"a"/1` and `"b"/2`. Then the variable `Rest` is instantiated with the rest of the map (a new map which contains all the pairs except `"a"/1` and `"b"/2`). Note that since `$Map` is an important and frequently used built-in class we added special syntactic representation (like for lists) to it to improve its readability.

6 Enumerations

Unlike in Prolog, in Flang the user can create many different classes which aggregate objects of other classes (such as lists, vectors, sets, tables, arrays, etc). In such a situation, enumerations provide universal and flexible methods for processing elements of these structures one by another. An indicative example here is processing of XML terms. The sequence of components of an XML term can be aggregated within different structures. Suppose, for example, that a Flang program has generated the following XML term:

```
<text> Hello, <green>world</green>!</text>
```

As a rule, contents of XML elements (such as `Hello, <green>world</green>` above) are being constructed gradually and the final structure can be a mess of different structures, something like this:

```
["Hello", [", "],
           [xElem("green", {}, ["world", ["!"]],_)]
```

Our experience shows that this is a usual situation. If we do not provide special means for processing such intricate structures, we'll have to process them "manually" all the time. On the other hand it is unwise to restrict the Flang programmer in possibilities of constructing and representing such structures. It must be up to him.

This problem can be efficiently solved with the notion of an enumeration. When defining an enumeration we abstract from concrete ways of aggregating elements. We are interested only in the order in which these elements occur in this structure. Enumerations have three basic methods – `value()`, `next()` and `exhausted()`. Method `value()` returns the current (active) element of an enumeration, `next()` returns the enumeration in which the pointer is shifted to the next element, and `exhausted()` is true if the enumeration is over.

Generators of enumerations depend on the nature of enumerated structures. For predefined Flang classes (lists and vectors) two methods – `enum()` and `enumFlat()` – are used. They differ in the way they select elements. The first one

selects elements only at the top level of the structure while `enumFlat()` tries to enumerate sublevels recursively. For instance, for the list `[1,[2,[3]]]` the method `enum()` enumerates the two elements `1` and `[2,[3]]` while `enumFlat()` enumerates 1, 2 and 3.

Consider the use of enumeration in the example of summing-up elements of an enumerated structure.

```
sum(Enum) :- Enum.exhausted(), !, 0;
sum(Enum) :- V = Enum.value(), V + sum(Enum.next());
```

Now,

```
?- sum([1, 2, 3].enum());
     6
?- sum([1, [2, [3]]].enum());
    Runtime error: argument of '+' must be integer or string

?- sum([1, [2, [3]]].flatEnum());
    6
```

Note that a special unification procedure for enumerations is also predefined. With its use the `sum/1` definition can be rewritten as follows:

```
sum($Enum()) :- !, 0;
sum($Enum(Value, Next)) :- Value + sum(Next);
```

Actually, the definition of the unification method for `$Enum` is very simple:

```
unify($Enum()) :-  exhausted();
unify($Enum(Value, Next)) :-
          Value = value(), Next = next();
```

The users can define new types (classes) of enumerations for their own classes. Flang also contains a number of predefined enumerations for XML structures. For XML terms special enumerations using XPath [18] are applied. For instance, the method `eXElem(XPathCond)` enumerates all elements of an XML term which satisfy the condition `XPathCond`. In the following example

```
XMLDoc.eXElem("//organization[@identifier='built-ins']")
```

all sub-elements of `XMLDoc` named “organization” and having attribute “identifier” assigned “built-ins” are enumerated.

7 Flang libraries

Flang is focused on development of large and scalable projects. The strict object-oriented architecture of the language allows joint work of large and distributed teams. Close interaction with Java provides access to huge quantity of libraries created in the Java community. Flang is tuned for the development of Web services, for components of Web services, as well as for reusable modules, such as menu generation modules,

interactive Web resources, forums, access to databases, etc. To put it briefly, Flang is shaped to allow consecutive accumulation, exchange and reuse of Flang modules and libraries. But for this it is necessary to have flexible tools for development of Flang libraries.

Since Flang was constructed on Java technologies, the structure of Flang libraries resembles the structure of Java libraries. In general, a Flang library is a hierarchy of catalogs which consist of

- Flang programs (Flang classes definitions), and
- Compiled Java classes (files "*.class") generated by the Flang compiler or developed manually.

The Flang system can use both the unpacked libraries and ZIP-archives.

Let's consider a Flang definition of a class `$Rectangle`:

```
package mylib.geometry;
import flang.$Math;

public class $Rectangle {
  $Float left;
  $Float top;
  $Float right;
  $Float bottom;

  public $Rectangle(L, R, T, B) :-
          left := L,
          right := R,
          top := T,
          bottom := B;

  public getDiagonal():-
          $Math.sqrt((R-L)*(R-L) + (B-T)*(B-T));
  public getSquare():- (R-L)*(B-T);

  public static
      getMaxSquare([], MaxSq):- !, MaxSq;
      getMaxSquare([Rect|Rest], MaxSq):-
          getMaxSquare(Rest, $Math.max(MaxSq,
          Rect.getSquare()));
}
```

A rectangular is defined by coordinates of the left-top and right-bottom points (the fields `left`, `top`, `right`, `bottom`). The package declaration says that this class is an element of the library `mylib.geometry` (and this means that the file `Rectangle.fln` must be physically stored in a directory mylib/geometry of the library catalogs hierarchy).

A Flang file can contain no more than one public class, and the name of the file should coincide with the name of this class (without symbol '$'). `$Rectangle` uses methods of a class `$Math`, access to which is declared in the import declaration.

`$Rectangle` contains dynamic methods `getDiagonal` and `getSquare` which return the length of a rectangular diagonal and its square, respectively. Static method `getMaxSquare` finds the maximal square in the list of rectangles.

8 Flang in Java

Flang can be used as an independent tool for development of various projects. On the other hand, one of the basic ideas of Flang is tight integration with technologies based on Java:

1. The Flang system is implemented in Java and can be used as a Java library. This means that development of Java projects can be powered by modules in Flang (efficient and compact XML documents processing, etc.).
2. Flang includes flexible tools for "on-the-fly" incorporation of Java modules and libraries, which can be mapped in Flang as built-in classes and methods.
3. The object-oriented framework of Flang is consistent with that of Java and can be considered as a simplified version of the latter.

Consider now how Flang can be incorporated in a Java code and (in the next section) how Java modules can be incorporated in Flang projects. It was mentioned above that Flang can be exploited as a Java library. The Flang system has a standard interface allowing interpretation of Flang programs within a Java code and retrieving the results of this interpretation. Suppose we have a Flang program news.fln which handles news data. To call and evaluate this program within a Java code we need to include in this code the following statements:

```
// import the Flang library
import ru.baikal.flang.*;

// parse a Flang program news.fln and create
// the corresponding project
FlangProject fp =
        new FlangProgect("c:/flang/news.fln");

// create an instance of the Flang interpreter for
// this Flang project
Interpreter interp = new Interpreter(fp);

//interprete the Flang query
//  ?- makeNews("c:\news\news.xml");

FlangObject result =
  interp.interprete("makeNews(\"c:\news\news.xml\");");
```

Suppose, the Flang function `makeNews(…)` generates an HTML representation of a news channel stored in news.xml. Then the variable `result` contains a tree-like Flang object, an internal representation of an HTML document. Then using special methods-converters we can get standard representations of HTML, for example,

```
Element jdomTree = result.toJDOM();
String resultString = result.toString();
```

In the first case the result is transformed into the popular JDOM format for XML documents, in the second case the result is transformed into a standard Java String which can be used, say, in a JSP code.

9 Java in Flang and Compilation of Flang Classes

Java classes can be used in Flang projects. The idea here is that Java classes and methods can be 'wrapped into' Flang classes and methods. For augmenting Flang with new modules written in Java it is not necessary to correct and recompile the code of the Flang system itself. The interface between a linked Java class and the Flang system consists of a couple of Java classes which can be quickly developed.

Since Java classes are encapsulated in the same way as Flang classes, it is not possible to distinguish whether a class was written in Java or in Flang. For instance if we define `$Stack` in Java, the behavior of objects of this class in Flang programs will be exactly the same as if they were defined in Flang (although, the Java code is faster). Moreover, now we are developing a Flang compiler which translates Flang classes into corresponding Java classes. The Flang system automatically substitutes interpreted Flang classes by functionally similar Java classes.

The scheme of introducing Java classes in Flang allows us to easily incorporate to the Flang system all the power of existing Java libraries. It supplies the user with huge number of tools. This feature makes it possible to apply Flang as an advanced module management system in which basic methods written in Java are managed on the higher level by the flexible tools of Flang.

10 Flang Server Pages

Flang is tuned for development of server applications and dynamic interactive Web resources. The following features of the language support this:

1. In Flang standard syntactic layout of XML and HTML texts is supported.
2. In methods written in Flang calls and other evaluated codes can be inserted directly into XML and HTML texts.
3. Unlike other approaches Flang supports processing of XML and HTML texts not as strings, but as hierarchical structures. That allows application for their processing of the advanced logic and functional means.
4. Flang contains tools and interfaces for integration with other server technologies (servlets, JSP, etc.)

A Flang Server Pages (FSP) technology makes it possible to develop and manipulate dynamic interactive resources. This technology allows development of analogues of servlets and Java Server Pages. This feature of Flang was approved by a number of resources developed in FSP.

A Flang Server Page is an HTML page with insertions of Flang code which is evaluated dynamically at the moment of page generation. Unlike JSP, Flang server pages are not collections of text strings but sound and correct XML structures. Thus, Flang handles HTML documents not as a bundle of text pieces, but takes into account their hierarchical nature. This essentially distinguishes the approach implemented in Flang from the servlet and similar technologies.

As an example consider a simplified but real service which dynamically generates an HTML page. This page represents news stored in the popular XML/RSS format [20]. The following is a simple example of a RSS 2.0 news document:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
  <title>Reuters: Top News</title>
  <link>http://www.reuters.com</link>
<item>
  <title>Gunmen Abduct Two Italian Aid Workers</title>
  <link>http://www.reuters.com/newsArt1005.jhtml</link>
  <pubDate>Tue, 07 Sep 2004 15:32:22 GMT</pubDate>
  <description>BAGHDAD (Reuters) - …</description>
  </item>
<item>
  <title>Congress's Analysts See Deficit</title>
  <link>http://www.reuters.com/newsArt1006.jhtml</link>
  <pubDate>Tue, 07 Sep 2004 14:34:56 GMT</pubDate>
  <description>WASHINGTON (Reuters) - …</description>
  </item>
</channel>
</rss>
```

Now define a Flang class representing RSS news:

```
package rss;

import flang.$XMLReader;

public class $RSSNews {
 private $XML news;

 public $RSSNews(File) :- news := $XMLReader(File);

 public showNews() :- showItems(news.eXElem("//item"));

 showItems($Enum()) :- !, [];
 showItems($Enum(Value, Next)) :-
     [showItem(Value) | showItems(Next)];
 showItem(XElem) :-
  <div>
   <h3 class="itemTitle">
```

```
        <a href=_XElem.getXText("//link");>
           _XElem.getXText("//title");
           </a>
        </h3>
     <p class="pubDate">_XElem.getXText("//pubDate");</p>
     <p class="itemDescr">
        _XElem.getXText("//description");
        </p>
     </div>;
}
```

This class contains only one field `news` in which an XML/RSS term is stored. The class also contains a method `showNews()` which generates a list of HTML elements for news items. Auxiliary function `showItems(Enum)` establishes a loop over news items and `showItem(XElem)` generates an HTML element representing one news item. The predefined method `getXText(XPathCond)` finds the first sub-element of an XML term meeting the condition `XPathCond` and extracts the text from it. The method `eXElem("//item")` establishes an enumeration over sub-elements "item" of this XML element.

An FSP file rssNews.fsp displaying news is very simple:

```
@page import rss.$RSSNews;
  <HTML>
    <HEAD><TITLE>Reuters: Top News</TITLE></HEAD>
    <BODY>
      _$RSSNews(
"http://www.microsite.reuters.com/rss/topNews").showNews();
    </BODY>
  </HTML>
```

If the user requests this FSPage from the server, the Flang system dynamically generates an HTML page with the fresh top news from Reuters and returns it to the user's navigator.

11 Conclusions

In this paper we have overviewed the basic components of Flang, a logic language augmented with object-oriented and functional features. There are a lot of creative projects concerning expansions of basic Prolog (see [3]-[14], and many others). Flang has its own peculiarities. The only guideline for us was the support of compact and uniform techniques for the development of large and distributed Web programming projects. It can seem that the tools incorporated in Flang are a little bit incoherent. But, actually, these tools interplay between each other in a transparent and compact way. The style of programming in Flang is simple and uniform, in particular, because all the tools are incorporated in the language straightforwardly (are not simulated by other means). Flang allows the development of uniform modules and libraries for different types of problems which are solved today with the diverse and hardly

consistent techniques. Java, on which Flang is constructed, brings about a great number of libraries, necessary to meet challenges of practice.

At this moment we are debugging the complete version of the Flang system which will be published for free access when ready. Also we are developing a compiler which allows gradual compilation of Flang programs, class by class, to Java.

References

1. A. Mantsivoda. Flang: A Functional-Logic Language//Lecture Notes in Comp.Sci., 567, Springer, Processing Declarative Knowledge (eds. H.Boley and M.M. Richter), Springer, 1992, p.257-270.
2. A. Mantsivoda, V. Petukhin. Compiling Flang//Proceedings of 2nd Russian Conference on Logic Programming, St.Petersburg, 1991, No. 592 in Lecture Notes in Computer Science, Springer, pp. 286-293, Springer-Verlag, 1992.
3. Aït-Kaci, H., Podelski, A. Towards a Meaning of LIFE? Proceedings PLILP '91, LNCS, Vol. 528, pp. 255-274, Springer-Verlag, August 1991.
4. Sannella, D. et al. A Calculus for the Construction of Modular Prolog Programs. J Logic Prog 12:147-177 (1992).
5. Baldwin, J.F., et al. Towards Soft Computing Object-Oriented Logic Programming. Proceedings of FUZZ-IEEE 2000.
6. Prolog+CG 2.0. http://prologpluscg.sourceforge.net/.
7. Peter Van Roy, Seif Haridi, Per Brand, Gert Smolka, Michael Mehl, Ralf Scheidhauer. Mobile Objects in Distributed Oz. ACM Transactions on Programming Languages and Systems, 804-851, 1997.
8. Naish, L. HTML Form Interface to NU-Prolog. http://www.cs.mu.oz.au/~lee/src/forms/..
9. D. Cabeza, M. Hermenegildo. A New Module System for Prolog. International Conference on Computational Logic, CL2000, LNAI, Num. 1861, pages 131-148, Springer-Verlag, July 2000.
10. SICStus Prolog. http://www.sics.se/ps/sicstus.html.
11. Morozov, A. Development and Application of Logical Actors Mathematical Apparatus for Logic Programming of Web Agents. ICLP'2003, LNCS 2916, Springer-Verlag, 2003, 494-495.
12. Quintus Prolog 3.5. http://www.sics.se/isl/quintus/.
13. Moss, C. Prolog++: The Power of Object-Oriented and Logic Programming. Addison-Wesley, 1994, 312p.
14. Demoen, B. Dynamic Attributes, their hProlog implementation, and a First Evaluation. Report CW350, KULeuven, 2002, 26p.
15. XSL Transformations (XSLT). Version 1.0, W3C Recommendation 16 Nov 1999.
16. JDOM. http://www.jdom.org/.
17. Berners-Lee, T., Hendler, J. Lassila, O. The Semantic Web, Scientific American, May 2001
18. XML Path Language (XPath). Version 1.0. W3C Recommendation 16 Nov 1999.
19. JavaServer Pages Technology. http://java.sun.com/products/jsp/.
20. RSS 2.0 Specification. http://blogs.law.harvard.edu/tech/rss.