Technologies for Building Intelligent Web Applications based on JULIA Toolkit

Dmitri Soshnikov
Department of Computational Mathematics and Programming,
Moscow Aviation Institute (Technical University)
Moscow, Russia
dmitri@soshnikov.com

Abstract

The paper outlines different approaches for building intelligent web applications, i.e. web applications that use some sort of logical reasoning. In particular, the use of JULIA toolkit, based on frame knowledge representation with forward- and backward-chaining production rules is considered. In the first part, some simple technologies for remote consultation are presented, measures for increasing reasoning performance over slow network connections are discussed, such as rules-on-demand loading, and several real-world applications are outlined. Finally, the complex technology, which we call Intelligent Active Server Pages (IASP) is presented, in which web application is viewed as a collection of interoperating intelligent web pages that contain knowledge in the form of frame subhierarchies, which can interoperate through standard HTTP protocol with other web pages, either intelligent or normal dynamic pages constructed using traditional tools like ASP or PHP. Together with other features of JULIA Toolkit, such as seamless integration of data from relational databases into reasoning process, ability to use passive knowledge repositories, integration of traditional objects including COM- and CORBA-objects give such a paradigm for constructing intelligent web applications the unsurpassed flexibility and power in designing small and large-scale sites and distributed intelligent applications.

Proceedings of the 3rd International Workshop on Computer Science and Information Technologies CSIT'2001
Ufa, Yangantau, Russia, 2001

1 Introduction

The growing popularity of Internet today is dominated by **web applications** — interactive web sites, that exhibit the functionality generally available in normal software applications: accessing large databases (search and retrieval), entering data from different locations (applications for distributed data collection), automating groupware tasks and workflow (intranet applications) and so on. Moreover, some areas for web applications emerge which do not have equivalents in the software part, a good examples being community and e-commerce web sites.

The majority of web applications is created using classical imperative programming techniques, and traditional programming languages, the most popular being Visual Basic (VBScript, used in Microsoft Active Server Pages technology), Perl, PHP and JScript (Microsoft Server-side version of JavaScript). Most of the languages used are typical procedural languages, and are not even fully object-oriented. Since web applications generally use database backends, those languages have some sort of database access API, and the ability to add and use additional third-party modules (in the form of DLLs and COM objects on Windows platform, or some sort of object-oriented extensions like CPAN Modules in Perl) that extend the core functionality of the language.

One way of enhancing the functionality of web applications is to use alternative programming paradigm, like functional, logic or true object-oriented programming. For example, in LogicWeb [1], logic programming and the extension to Prolog is used for adding complex behaviours to the World Wide Web. More straightforward example of using logic programming to construct web application can be found in AMZI Prolog [6] and related projects.

Another system called Zope [2] utilizes object-oriented approach to the WWW, where each web page is viewed as an object, and the whole site therefore consists of interoperating objects, which can be either written in Python or other programming language and added as a module, or constructed from other objects using proprietary DTML language (which is essentially an HTML extension) and loaded from the persistent Z Object Database.

This paper presents another approach to building web applications using frame-based knowledge representation and production rules. As it was discussed in [4], frame-based representation allows to integrate into one frame hierarchy standard frames with forward- and backward-chaining production rules, database extensionals and system frames for accessing relational database tables seamlessly during the inference process, and objects from object-oriented languages (including CORBA- and COM-objects) with imperative algorithmic functionality. Thus, discussed frame-based model successfully integrates different programming, reasoning and knowledge representation paradigms. It also presents the basis for creating distributed reasoning systems, where parts of the frame hierarchy are located on different network hosts.

Presented approach considers web application as a collection of interoperating intelligent web pages that contain knowledge in the form of frame subhierarchies, which can interoperate through standard HTTP protocol with other web pages, either intelligent or normal dynamic pages constructed using traditional tools like ASP or PHP. Since the implementation is based on the JULIA toolkit, it allows seamless integration with relational databases and CORBA objects, as well as using remote knowledge repositories and other features described in [4].

However, practical implementation of the presented approach requires special software to be installed on the web server. Also, using such powerful tool for simple applications would not be appropriate. Therefore, we begin by describing more simple and traditional approaches to remote consultations and their practical applications, moving towards more general approach. Also, we discuss some additional measures for optimising the performance of reasoning applications over slow network connections.

2 Overview of JULIA Toolkit

When implementing any kind of knowledge-based applications, the developer is faced with the options of building custom inference engine with some domain-specific knowledge representation, or using existing tool with suitable knowledge representation and inference capability. Among different tools for bulding knowledge-based systems (an overview of many modern tools can be found in [5]), only some can be integrated as a reasoning module into larger software systems with different user interface, the most notable being AMZI Prolog [6], CLIPS [7], and Jess [8].

The most open system in the mentioned list is Jess, a subset of very popular CLIPS system, which uses the same language, and is implemented as a set of Java classes, available in source code. Jess supports Rete algorithm for implementing forward-chaining inference, and the ability to extend the functionality of the system with additional functions written in Java. For remote consultations over the Internet it can be used on the client side as an applet (although the size of the code has lately grown too large to be comfortably used in this manner), or from the server-side using Servlet or any other similar technology.

However, Jess supports only limited object-oriented features (object-oriented features found in CLIPS are omitted), does not provide effective support for backwards chaining[1], and apart from the ability to be extended by arbitrary Java functions it does not provide out-of-the-box integration with other systems, like relational databases or distributed objects. Moreover, the nature of Rete algorithm and CLIPS-like language makes it difficut to adopt distributed inference paradigm to this architecture.

To overcome those difficulties, a system called **JULIA** (Java Universal Library for Intelligent Applications) has been developed, which is based on the principles presented in [4]. JULIA is essentially a set of Java classes that give the user an open API which can be used by any Java application (and also by non-Java applications through simplified interfaces exposed by CORBA). Frame paradigm with production rules for representing dynamic knowledge, on which the whole system is based, provides natural integration with arbitrary Java objects and JavaBeans, relational databases accessible over JDBC, distributed CORBA- and COM-objects. Built-in functionality can also be extended by writing custom functions in Java, and providing arbitrary rule selection strategies to customise the inference process to best suite the needs of the

[1] In the latest versions support for backward chaining is announced, although it is merely an attempt to provide some language support for simulating backward-chaining inference on forward-chaining system

problem domain.

Original knowledge is formulated by using specific knoledge representation language. Source file containing frame definitions and production rules is then translated into a number of frames, which form what we call a **frame world**. Frame world can be serialized at any time and stored in a self-contained file, which then can be loaded with one command for performing logical inference.

The basic units of knowledge representation are frames, which can be of different types: standard frame (`DefFrame`) that is responsible for inference and most of other system functionality, Java Class Frame (`JavaClassFrame`) for representing external Java classes as frames in the frame model and other frames for accessing components such as JavaBeans and COM-objects, frames and frame extensionals for database access (`DBFrame` and `DBFrameClass`), remote frames for accessing remote instances of JULIA worlds over different protocols (CORBA, XML-RPC, RMI). Custom frame types can be introduces by extending `Frame` or `StoreFrame` base classes depending on the functionality needed[2].

All frames follow the same interface, which specifies operations for accessing values of frame slots. Most of the system functionality is exhibited by standard frames, which contain actions assigned to their slots. All production rules in the source file are translated into either `OnGetAction`s, which are executed when the value of the slot is to be determined, or `OnSetAction`s, which are fired when the value is assigned to the slot. Backward-chaining rules (which contain only one concluding assignment) are directly translated into `OnGetAction`s for the corresponding slot, while forward-chaining rule is translated into an instance of `OnSetAction`, which is referenced from all slots in its premises, thus organizing the form of simplified Rete network. All actions are also pre-processed in such a way that they can be invoked from the frames inherited from the one they belong to, so that they can be applied during inference for all frames descending from the one the rules have been defined for. Also, the order in which actions are performed is controlled by an instance of `RuleSelectionStrategy` class, therefore allowing very flexible[3] control of the conflict resolution strategies both during forward and backward chaining. In addition to actions, slots and standard frames can also contain constraints (facet constraints, as well as constraints in the form of arbitrary expressions), which prevent certain valies from being assigned to slots thus causing backtracking.

While frame world is a collection of frames stored in one serialized file, it can also reference frames in another frame world, i.e., located on another JULIA-based server, or running on the same maching in another thread. Functionality of inter-operating frame worlds resembles **blackboard architecture**[13], but in a way it is more powerful, because rules defined in one frame world can propagate into another when a frame is inherited from another frame located in different frame worlds. However, in all cases only values of slots are being exchanged, and not the actual rules. Such complex inter-operation of frame worlds is achieved by creating **proxy frames**, which forward requests via one of the supported protocols.

Other JULIA objects, such as individual frames with attached actions, or even individual actions can also be serialized (or alternatively represented in open XML form) and then loaded into base frame world on-demand, or using special functions.

3 Using the Toolkit for Remote Consultations

The simplest form of knowledge sharing in computer networks is **remote consultation**, in which the knowledge located on one computer is used by another network node to solve the problem. There are two major types of remote consultations depending on where the inference process takes place: server-side inference (thin client model), and client-side inference (thick client model). Different aspects of implementing inference on the server and on the client side have been discussed in [3]. Here we will outline the aspects of technical implementation using JULIA toolkit.

3.1 Remote consultation with server-side inference

The most interesting case is when some questions are asked to the user during the inference process (which is typical when backward chaining is used). The problem of implementing server-side remote consultation in this case lies in the fact that such inference process cannot be separated into request-response series typical for web applications. This is because HTTP protocol is stateless, thus making it difficult to store the state of the system, which has quite complicated nature, during the inference process. Also, the architecture of the JULIA toolkit is such that question-asking routine is always called as a callback procedure, i.e. the execution of the library cannot be stopped in between request and response, which is the way most of the web

[2] `Frame` class exhibits only operations for accessing and assigning the value of slots, while `StoreFrame` adds the functionality of storing the values in memory, while still allowing to obtain them from any other source

[3] In addition to a set of standard selection strategies (first rule fires first, random selection strategy, limiting the number of firing rules, combining different selection strategies), custom rule selection strategies can be developed in Java

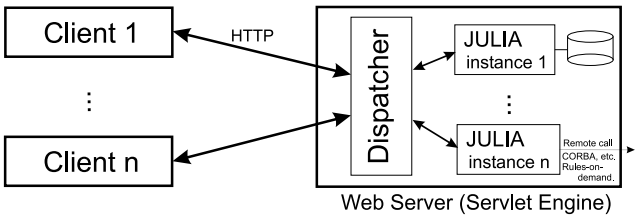

Figure 1: Server-side inference

server-side technologies operate, including CGI, ASP, and others.

The proposed solution to this problem is to start the inference process for each new user session concurrently in a separate process or thread, and have individual processes servicing user requests communicate with the inference process (see Fig.1). Java Servlet technology is ideal for implementing such mechanism, because it natively supports threads, and the lifetime of the servlet code corresponds to the web server running time, thus some code can continue executing in between requests.

When web server is started, servlet initialization code is executed, which creates empty session table and loads the corresponding JULIA frame world from the external serialized file. When new request is encountered, the servlet creates a new instance of Frame world by incomplete clone operation[4], records the reference to the created world in the session table, and starts inference. As soon as the question to be asked to the user is encountered during the inference, the inference process is suspended, and the qustion text is returned to the user[5]. When the user answers the question, and the answer is sent to the server in subsequent request from the same session, the value of the answer is returned to the inference thread, which continues execution until another question is encountered, or the result is obtained.

Each arriving requiest has the associated session identifier, which is used to match the request with the reference to the running thread of inference process using the session table. Thus, one servlet can service requests from multiple users simultaneously, and the actual inference processes for each request run concurrently in separate threads. When the session is abandoned by the user (which happens, for example, when Internet connection is dropped), it is destroyed after a certain timeout period by servlet control process, which executes in a separate thread through the whole period of servlet lifetime.

It has to be noted that server-side inference can be successfully applied to creating web interfaces for complex systems using distributed knowledgebases with distributed inference, or in cases where database access is desired. Since all inference takes place on the server, JULIA frame world instances can access databases or remote JULIA-based servers seamlessly to the user, provided necessary tools and protocols are installed on the server. User only accesses the site using standard HTTP protocol.

3.2 Using server-side inference in intelligent adaptive testing

Described approach has been applied for creating intelligent application for adaptive testing[6]. While most of the adaptive testing approaches use numerical and statistical methods for estimating the knowledge of the individual being tested and to adjust the difficulty level of further questions, in intelligent adaptive testing an expert system is used to build the internal model and estimate the level of the individual's knowledge. The expert system is based on the expert knowledge of teachers and tutors in the subject, and allows to represent such knowledge in much more flexible manner, then using just difficulty level and topical categorization for questions. As a consequence, the system only asks the number of questions it needs to obtain the clear picture of the testee's knowledge (not a fixed number of questions like in most of the existing systems), and difficulty level of further questions is adjusted according to the principle of gathering more information while asking less questions. To make tests more random for different users, the knowledgebase uses random rule selection strategies, draws limited number of questions from large question pool, and also uses dynamic question generation by custom algorithms written in Java, or specified using simple template language.

As a result the framework for intelligent adaptive testing has been developed, which can be applied in different areas by developing different knowledgebases. The system is now introduced into probation use on the Faculty of Applied Mathematics and Physics of Moscow Aviation Institute for testing students in courses of Computer Science and Logic Programming. More detailed description of the system can be found in [9].

[4]New instance of frame world should have different set of slot values, but the same actions, therefore some references in two worlds can point to the same object instances in memory.

[5]To have the consistent and rich web design, specified HTML template is applied to the question text, and the resulting HTML code is returned. Alternatively, servlet can be used through SSI, in which case the HTML design is encapsulated into the calling page, and servlet returns simple question string without processing.

[6]The implementation of intelligent adaptive testing framework has been developed in cooperation with Olga Malkina, graduate student of Moscow State Aviation Technical University

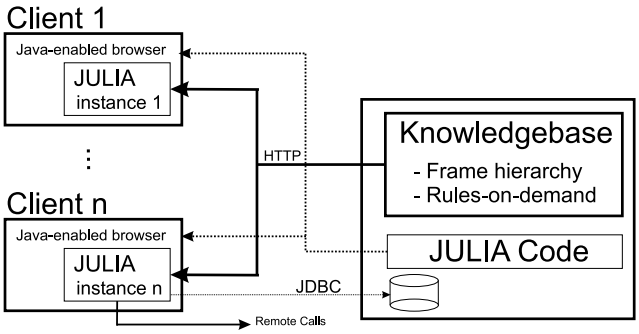

Figure 2: Client-side inference

3.3 Client-side inference

An alternative to server-side inference is client-side inference, where JULIA Toolkit is used as Java applet, and inference takes place separately on each client (see Fig.2). In this case knowledgebase (frame hierarchy and production rules) also has to be downloaded to the client prior to starting inference, which can create significant network traffic load. However, for relatively small projects the load is not too high: full JULIA class library is only about 130 Kb large, and can be stripped down to around 70-80 Kb by removing unnecessary functionality (unused add-ons, rule selection strategies, database access components, frame definition language parser, remote-access frames, etc.). Further minimization of knowledge-base size can be achieved by implementing more space-consuming serialization algorithms, and using on-the-fly compression/decompression.

For larger knowledgebases further optimization in network traffic and distribution of network load among the whole inference process can be achieved by **rules-on-demand** and further by **frames-on-demand loading**. It can be noted [3] that during backward inference only some production rules are used in each case, and the majority of rules are not necessary. The same is true about frame hierarchy: while it describes the general model of the problem domain, in any given case only some sub-hierarchy is used.

Therefore, it makes sense to load rules corresponding to any given slot only when they are actually required to be used in inference. It is achieved by using a rule stub instead of the actual rule in the frame hierarchy, which loads the body of the rule from the serialized file only when the rule is required during inference. Creation of rules-on-demand serialized files is automated by JULIA parser: when rules-on-demand flag is on, all following rules are created as rules-on-demand, i.e. rule stubs are inserted into frame hierarchy, and corresponding serialized files for each rule are created on disk. While the whole knowledgebase occupies slightly more space (some extra space is taken by rule stubs), it minimizes network traffic during each individual consultation.

For simple rule selection strategies (first rule first, random, etc.), the rules are loaded only when they are fired, i.e. when their premises need to be calculated, while more complex rule selection strategies where the rule premises need to be known before conflict resolution require all rules for a given slot to be pre-loaded before slot value resolution. Regardless of this, rule-on-demand loading requires only frame hierarchy to be pre-loaded before inference begins, and rules are then dynamically loaded during the inference process.

Even more minimization of network traffic for some problems can be achieved by frames-on-demand loading. Each frame in the source knowledgebase file can be declared as loaded on demand, in which case frame stub is inserted into the hierarchy, and the actual serialized frame file is loaded only when some operations are required on the given frame. For some problems where more or less all frames in the model are somehow used in the consultation (for example those requiring frame matches for object sub-classing based on constraints) this may not give any significant benefit, while in other cases (for example, when category of the object is determined on the basis of rules, and then only corresponding frame sub-hierarchy is used) it allows to reduce the amount of knowledgebase transferred quite significantly. In addition, frames-on-demand loading can be used in conjunction with rules-on-demand, thus making the system separated into a number of small serialized files, which are loaded as needed.

3.4 Implementation of web site promotion expert system using client-side inference

Client-side inference has been used in the implementation of an expert system for planning promotion campaigns for web sites and optimizing web site for promotion[7]. The expert system uses backward inference, and is capable of optimizing different promotion steps under given financial constraints based on user's anwers to a large set of questions. In addition, the system uses a specific plug-in written in Java, which gathers keywords from different web sites in a specified category, and optimizes them to create a unique set of keywords for the given resource, which can significantly improve the position of the resource in search engines listings. Since this process requires downloading and analyzing a number of third-party web pages, it has been decided to implement the system on the client side, so that all network traffic goes through the client directly. To minimize network traffic to the

[7]The implementation of the system and the development of the knowledgebase has been done is cooperation with Irina Krasteleva, graduate student of Moscow State Aviation Technical University

server, rule-on-demand and frame-on-demand loading is used where possible.

System prototype is located at `http://promoweb.shwarsico.com`. The user interface of the system is basically that of the Java applet, and is primarily intended for asking questions to the user and showing the results of the reasoning for a particular case.

3.5 Tutoring system based on client-side inference and applet-browser interaction

One disadvantage of the system described in previous section is that the user interface is determined by the Java applet, and thus it would be quite difficult to use expressive formatting in the system-user dialogue (because some sort of rendering engine would have to be implemented in the applet itself).

One of the solutions to this problem when using client-side reasoning is to use applet-browser interaction, and to open arbitrary HTML files in the browser, while keeping the reasoning applet running either invisibly, or being responsible for some subset of the user interface. This approach is being applied in development of complex knowledge-based tutoring system, which combines intelligent adaptive testing technology described above with the hypertext course materials enhanced by intelligent browsing capability. In the process of completing initial entry-level test, the model of individual's knowledge on the subject is constructed, which contains the areas of weak knowledge that require improvements. Then, in the process of backward-chaining reasoning the user is asked a series of questions that help target the weakness, and then he is redirected to the corresponding section of the hypertext course. An immediate problem or one of the test questions on that section is also offered, and the ability or inability to solve it allows to correct the user's knowledge model to reflect the changes. Also, some straightforward browsing paths are provided through the course without initial test, like chapter-by-chapter browsing with intermediate questions, that form the model of the user's understanding while studying through the hypertext material.

At any time, the current user understanding model in the form of frame hierarchy with some slots filled in can be saved as a serialized file for later use. When the course is finished, the final test is performed, which then allows to give the estimate on the effectiveness of the education by comparing initial test results with the final one. The comparison is only approximate, as the actual questions may differ, but it is possible to draw very informative conclusions by comparing some variables responsible for user knowledge representation in different subject areas.

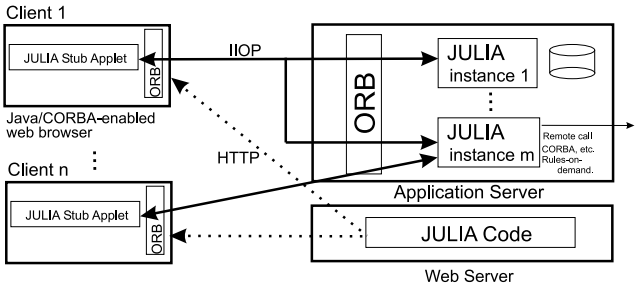

Figure 3: Three-tier model

The same knowledgebase developed for one subject can be used with minimal modifications in the server-side inference scheme. However, the client-side implementation allows placing such a system on any web server into public use, while server-side solutions are more suitable for implementations involving e-commerce soltions or in-house testing.

3.6 CORBA-based three-tier model

Finally, let us mention the three-tier model which is considered to be one of the most popular and progressive approaches in the implementation of modern client-server systems[11]. In this model, distributed system consists of the client part implemented as an applet (which runs a limited subset of JULIA responsible for user interface and maintaining a small subset of frame hierarchy responsible for a particular case), and JULIA server (which contains main part of the frame hierarchy with most of the production rules) communicating through any remote protocol (CORBA, XML-RPC or Java RMI — see Fig.3). In this model, several clients can access one reasoning server, given that all problem-specific knowledge is accumulated on the client-side part of the frame hierarchy. Alternatively, different server-side frame hierarchies can be created for different clients, in which case client can only be responsible for calling JULIA server remotely, with all problem-specific static knowledge being stored on the server side. In such model, remote databases can be accessed both from the server or from the client (using CORBA or JDBC protocols), as seems more appropriate from the point of view of implementation.

4 Intelligent Active Server Pages

4.1 Overview and related technologies

Techniques discussed in the previous section can be successfully applied for remote consultations, i.e. for knowledge exchange between two network nodes: web server and client, with inference process taking place on one of the nodes. The three-tier model can in principle be used for cooperative inference between different servers communicating over RMI or IIOP, and loading additional knowledge from serialized files.

However, in each case the source knowledgebase has to be converted into searialized form, and CORBA- or RMI-based JULIA server has to be started on each of the servers.

In this chapter we would discuss more complex technology that we would call Intelligent Active Server Pages (IASP), which uses HTTP for communication between network nodes, and allows to construct web pages that incorporate knowledge into HTML design much in the same way as traditional Active Server Pages (ASP) Technology is used to mix HTML and programming code in traditional imperative languages (Visual Basic, JScript, Perl, etc.). IASP can be viewed as a similar paradigm for mixing HTML and declarative knowledge, represented in the form of frame model and accompanying production rules. In addition, such Intelligent Active Pages would be intended not only for producing HTML output to the user, but could also be invoked non-interactively from another pages, thus creating distributed frame hierarchy with distributed inference. Other ways of exchanging knowledge between network nodes, implemented in JULIA toolkit, can also be used in IASP, giving the developer the most advanced functionality in creating complex intelligent systems distributed over the net, utilizing either of HTTP, XMP-RPC, RMI or CORBA/IIOP protocols for communication.

The idea of presenting the web as a collection of inter-operating intelligent pages is not new. [1] describes the system called LogicWeb, where each web page is viewed as a logic program, which can be used in cooperation with other pages to achieve complex reasoning task. An extension to Prolog syntax and to the logic programming paradigm is presented, and the system is implemented, with LogicWeb engine running on one machine, and logic programs from other pages loaded on demand from the net. Processing of each complex request results in the growing collection of logic programs loaded into the system from the web, which interoperate according to introduced concept of context switching to produce the final result. All logical inference takes place on one client computer that runs an instance of LogicWeb system, which includes Prolog Interpreter and specialized accompanying software.

The approach presented in this paper is in a way similar to LogicWeb, but uses frame knowledge representation and inference in the set of production rules as the underlying paradigm for representing intelligence. Each page is viewed as a separate frame world, which can either be executed on a remote computer (this returning the result of its execution), or downloaded and included into local frame hierarchy. To achieve more complex interoperability between frame worlds located on different computers, more powerful communication

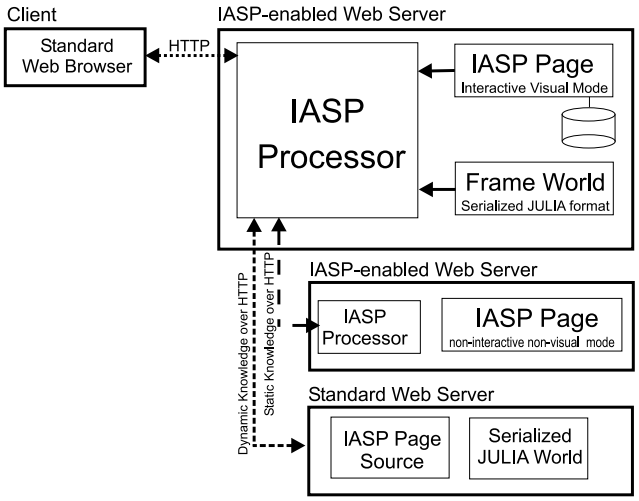

Figure 4: Intelligent Active Server Pages Architecture

protocol is needed, like CORBA or RMI. However, the functionality offered by simple HTTP interoperability turns out to be powerful enough for most applications.

4.2 IASP format and translation rules

Each IASP page contains some or all of the following sections, which can also be mixed:

- **Visual representation** contains HTML code for displaying the page, which may include variables or expressions in the Frame definition language embraced by <%= and %>.
- **Knowledge representation**, surrounded by <% and %>, which is the arbitrary code in frame definition language that is included into frame world representation of the page.
- **Dialog template**, surrounded by <%> and </%>, that define HTML design templates of intermediate pages used during backward chaining inference in the interactive mode (see below).

The suggested implementation (see Fig.4) uses IASP processor located on each IASP-enabled web server. When IASP page (i.e. a page containing some production rules or frame definitions in addition to normal HTML) is requested from the server, IASP processor:

- creates the corresponding frame world by parsing the IASP file and producing frame hierarchy,
- fills values of some slots in the model according to the HTTP variables passed to the page using either GET or POST method,
- evaluates the resulting frame world to obtain the result. Note, that the evaluation can result in

some other frame worlds being downloaded from the web, or in other more complex activities.

The following principles are used for creating frame world from the IASP page:

- Knowledge representation sections are directly interpreted and compiled into the internal representation according to the standard parsing algorithm used in JULIA toolkit. Optional %> ... <% construction can be used to mix complex HTML constructions in string constants, for example

```
IF diagnosis='flu' THEN goal=%>
 <H1>You have got flu!</H1>
 <P><B>Caution:</B> Flu is a kind of
   infection which is easily transmitted....
 </P><% ;
```

- Visual representation containing of the mixture of HTML elements and FMDL expressions is transformed into one string expression assigned to the `Response.Body` variable. For example, the following fragment:

```
<HTML><BODY>
 <H1>The result is
  <%=MainFrame.Result1 + MainFrame.Result2%>
 </H1>
 </BODY></HTML>
```

  is equivalent to the following FMDL statement:

```
SET Response.Body=
 '<HTML><BODY><H1>The result is '+
  (MainFrame.Result1+MainFrame.Result2)+
 '</H1></BODY></HTML>'
```

- Dialog template is basically stored as is in the `Runtime.DialogTemplate` slot.

4.3 IASP Page Invokation

When the frame world is created for a given page, all variables passed to it using either GET or POST method are stored as corresponding slot values of `Request` frame. The parameter can either be passed as a normal value generated by HTML form, in which case the corresponding slot will be of simple `SCALAR STRING` or `LIST STRING` type[8], or they can be encoded as XML VALUEs according to JULIA conventions, which is automatically recognized and can result in values of any specific complex type being passed. All invokations of IASP pages from other pages or programs in non-interactive manner normally use XML representation to preserve slot value types.

In addition, `Runtime.Method` is set to either GET or POST according to the request method used, `Runtime.Mode` is set according to the page invokation mode, and `Runtime.Type` is set according to the invokation type (see below).

The process of setting the slot values according to input variables occurs after the frame world has been fully created, and that is where forward-chaining inference can take place. `Runtime` slot values are assigned first, so that forward-chaining rules assigned to `Request` slots can use the information on type and mode of invokation.

4.4 Invokation mode and type

IASP page is normally invoked either by the user (directly by specifying the URL, or from HTML form, resulting in some HTML output to be displayed in the browser), or by another IASP processor or automated web system. In the first case (so-called **visual invokation**), visual representation of the page is obtained by calculating the value of `Response.Body` variable, which is then returned to the client. In the second case (**non-visual invokation**), the HTML visual representation part of the page is ignored, and the name of the slot to calculate should be passed via `goal` HTTP variable while calling the script. In this case IASP processor obtains the value of the specified slot, and returns it to the client in the XML representation[9]. Thus, the same page can be used both by the visual HTML client (web browser), and by non-visual agents. Some pages, however, can be explicitly designed for non-visual invokations, in which case visual representation part would be missing. The type of invokation (visual or non-visual) is contained in `Runtime.Type`, and is determined by the presence of `goal` HTTP request variable.

When a page is invoked, and the value of the goal is being obtained, some further questions may arise as a result of backward chaining inference. If the knowledgebase contains `ASK` statements for asking such questions to the client, the page can return a question instead of the final answer, using `Runtime.DialogTemplate`[10]. When the user answers the question, the result is passed back to IASP proces-

[8] Singular parameters will be represented as strings, while multiple parameters with the same name, like in `<SELECT MULTIPLE>`, would be represented as a list of strings.

[9] For convenience of building web applications, if the value being returned begins with `<HTML>`, the value is returnes as is, and is displayed to the client as HTML page. This allows to construct pages that result in HTML output even when invoked non-visually.

[10] When the page is invoked non-visually, the question is returned in the XML-form suitable for automated processing

	interactive	non-interactive
visual	Typical invokation by the user when backward chaining is used, and the inference is driven by questions-answers dialogue	Invokation from the web form, when the user fills in all initial values, and gets the result of the inference in one cycle. Excellent for simple reasoning applications with few initial facts.
non-visual	Rarely used due to complexity of implementation – typically is replaced by CORBA/IIOP interoperability.	Used in automated invokation from other IASP page by providing all initial values, and obtains the result in one cycle.

Table 1: Summary of invokation modes and types

sor, which associates it with previously created JULIA frame world (in the same manner as discussed in section 3.1) and proceeds with inference. This mode of invokation which results in the dialog with the client is called **interactive mode**.

In some cases, it is undesirable to initiate a dialog, and it is only needed to see if the result can be obtained from the data provided to the page upon the request. In this case, invokation mode (interactive or non-interactive) can be explicitly specified in the **_mode_** request varialble. If `ASK` action is encountered in non-interactive mode, it fails, and backtracking is initiated. If the page does not contain any `ASK` statements, it always behaves as though invoked non-interactively, regardless of the explicit mode specification.

4.5 Interoperability of IASP pages

Different IASP pages can be used in conjunction in one web applications through the following forms of interoperability:

- **Invokation**, when one IASP page invokes another one non-visually, thus causing some logical inference to take place on another IASP server. Invokation is performed using `&IASP_Call(URL, goal)` built-in function, returning the value produced by the remote page. Also, non-IASP page can be called in the same manner, providing it returns XML-compatible JULIA Value. Pages returning arbitrary HTML output can be invoked by `&WebCall` funtion, returning the page description frame descendant from `WebPage`, that contains different elements of the page parsed into corresponding slots.

  Simple invokation cannot normally be used interactively[11], i.e. the page invoked cannot use callbacks to determine the values of the calling frame world. This, it is not possible for the dynamic knowledge to propagate through the invokation, and the page being called therefore behaves autonomously.

  One specific case of invokation is **combination**, in which different IASP pages, responsible for solving the same problem using different knowledge, are invoked simultaneously with the same initial data, and then the results returned are combined (and, possibly, post-processed to be presented to the user in required way) to form the final result to the user. Different algorithms for combination could be used, including some reasoning on the results returned.

- **Inclusion**, by means of which the knowledge contained in the remote IASP page can be downloaded, parsed and included into the current JULIA Instance, either as a separate frame world that interoperates with the original world using standard JULIA methods (side-inclusion), or as a sub- or super- frame hierarchy (sub- and super-inclusion). Also, rules-on-demand and frames-on-demand loading from remote knowledge repository can also be used with IASP.

  An interesting variant of inclusion is **multiple consecutive inclusion**, when a certain frame sub-hierarchy (consisting of one or more frames) is consecutively attached as a sub-hierarchy to different frame worlds, thus using dynamic knowledge of those worlds in turn. This provides an alternative to **blackboard architecture** [13], where different knowledgebases operate on the same static knowledge repository. Attached sub-hierarchy acts as a blackboard, and it can also contain certain knowledge of its own. The difference is that in blackboard architecture all expert systems have access to the blackboard, and the inference can take place simultaneously, while in the case of consecutive inclusion at any given time the inference takes place only in one frame world. However, using different switching algorithms it is possible to control the way in which dynamic knowledge from different frame worlds is combined.

- **Complex interoperability** using JULIA distributed features available through CORBA/IIOP protocol or similar. It allows to

[11] Additionally, explicit non-interactive invokation mode allows to ensure that no intermediate questions are returned to the original request

call another JULIA server and establish two-way reasoning session, thus creating distributed frame hierarchies in which dynamic knowledge from upper levels can propagate to the lower levels located on different computers by means of two-way exchange of slot values.

- **Database connectivity**, **interoperability with Java classes**, and **moving code interoperability**, which is achieved by means provided by JULIA toolkit.

4.6 Other features of IASP Processor

When designing large reasoning expert system with many rules and relatively small visual component, it may be inconvenient to store the page in the source IASP form, because the IASP processor would need to parse it upon each invokation. To avoid this, IASP processor also supports invokation of serialized JULIA frame worlds directly: when the requested URL points to serialized world file (with `.jsw` extension), the world is instantiated, and then slot values are filled and inference is started in the same way as when invoking IASP page.

As far as implementation is concerned, IASP processor can be implemented in one of the following ways:

- As a separate web server application, which would service standard HTTP requests to normal HTML pages in addition to providing IASP processing.
- As a Java Servlet application, which is used in conjunction with third-party web server (Apache). In this case, URL rewriting may be required to achieve smooth and clear URL naming conventions.
- As web server plug-in, which is automatically invoked by web server (Apache or Java-based servers) when pages with certain extensions or types are encountered in the request.

In addition to support of IASP technology, it is convenient to implement JULIA CORBA-based server on the same host, so that all IASP pages can also be used in response to CORBA requests in web application requiring complex interoperability. Configuration of the JULIA toolkit used in servicing requests also determines if the instance of IASP processor has access to a database, either on the same, or on different server.

4.7 Relation to other DAI paradigms

The dominating paradigm for the development of distributed knowledge-based systems is **agent** paradigm [12], where large and distributed intelligent system is viewed as a collection of smaller components called **autonomous agents**, which act cooperatively to achieve a complex task. The relation of distributed funtionality of JULIA toolkit to the agent architecture has been discussed in more detail in [4]; here we will just outline some agent-like features of the proposed IASP architecture.

The main property of agents is **autonomy**, i.e. each agent considered separately should be able to perform some task on its own. This property is also true for IASP page, which contains separate frame model, which is typically able to perform reasoning by itself, provided the initial data. Agents are capable of reacting to the changes in environment by performing certain actions. For IASP page, the environment is constituted by the set of initial request variables with which the page is invoked, and the page can then perform actions or produce result as a reaction to the environment. IASP page can inter-operate with other IASP pages, thus a set of pages can achieve more complex task, which is very similar behaviour to the society of agents.

Typically, several other properties ofter exhibited by different types of agents can be noted:

- **Mobile Agents** act by transferring their executable code through the net and working remotely. In IASP paradigm, *code* is represented by frame hierarchy with the attached actions that are responsible for reasoning, thus *inclusion* operation provides functionality typically exhibited by mobile agents. Since JULIA world can be serialized at any time and transferred to another network node (as it happens, for example, when multiple consecutive inclusion is used on the server-side) with its state preserved, it is also possible in some cases[12] to create learning mobile agents.

  As far as the execution of arbitrary mobile code is concerned, the extension to JULIA library can be developed that provides this functionality to Java-implemented frames, in which case the full flexibility of mobile code system will be achieved.
- **Adaptive/Learning Agents** are capable of changing their behaviour according to the knowledge gained from observing the changes in environment with time. According to the IASP architecture discussed, the state of inference (and thus all internal knowledge accumulated during page execution) is lost after invokation, thus preventing the system from being able to accumulate knowledge of its environment. To some extent this problem is solved by using persistent database frames,

[12] In the present design, JULIA world cannot be serialized in the middle of backward chaining loop

slots of which are actually stored in the external relational database, and thus retain their value from one invokation to another. Also, IASP architecture can be extended by allowing serialized frame worlds to be updated after execution, or even by creating serialized version of each IASP page which is stored from one invokation to another much like the session is stored in ASP.

However, even though proposed IASP architecture exhibits the functionality typical for multi-agent systems, there are certain differences. One of the main goals of IASP and distributed JULIA functionality was to provide the support for distributed frame hierarchies, where the knowledge located on one computer would be applied accross the network boundary to the inherited sub-hierarchy located on a different computer. Thus, complex distributed application would rather consists of non-autonomous parts in the traditional sense, but inherited parts would depend on the functionality of the parent.

5 Conclusion

In this paper the complex architecture for building intelligent web applications has been considered, which greatly extends the functionality of simple remote consultations, and bring intelligent web application closer to multi-agent systems. With this technology, the knowledge for solving complex problems can be distributed over the net in organized hierarchical or cross-linked manner, and can be used collectively to solve complex resoning problems. Knowledge located on different network nodes can be maintained independently, in the manner typical for component-based architectures. With special features like seamless database access, ability to include arbitrary Java code into the knowledgebase and extend the basic functionality, access to CORBA and EJB objects and others, IASP architecture presents very attractive set of tools for buliding intelligent web applications.

Due to the size limitations of the article, possible examples of use of IASP technology, as well as ontology-based methodologies for creating and maintaining distributed knowledgebases, have not been considered.

Acknowledgements

I would like to express my gratitudes to all graduate students of the Computational Mathematics and Programming Department of Moscow Aviation Technical University who worked with me on the projects mentioned in this paper, the department staff, in particular Zaytsev V.E. for his scientific leadership, my family for their understanding, and Ortal Saar for intelligent discussions on philosophical aspects of DAI.

References

[1] Seng Wai Loke, Adding Logic Programming Behaviour to the World Wide Web, PhD Thesis, Department of Computer Science, The University of Melbourne, Australia, 1998.

[2] Web site of Zope: `http://www.zope.org`.

[3] Soshnikov D. An Approach for Creating Distributed Intelligent Systems. In J.-C. Freytag and V. Wolfengagen, editors, *Proceedings of the 1st International Workshop on Computer Science and Information Technologies*, Moscow, Mephi Publishing, 1998. pp. 129–134.

[4] Soshnikov D. Software Toolkit for Building Distributed and Embedded Knowledge-Based Systems. In *Proceedings of the 2nd International Workshop on Computer Science and Information Technologies*, Ufa, USATU Publishers, 2000. pp. 103–111.

[5] Gavrilova T.A., Khoroshevsky V.F. Knowledge-bases and Intelligent Systems, Piter Press, St. Petersburg, 2000. (in Russian)

[6] Web site of AMZI Prolog: `http://www.amzi.com`

[7] CLIPS Web Page: `http://www.ghgcorp.com/clips/CLIPS.html`.

[8] Web site of JESS: Java Expert System Shell, `http://herzberg.ca.sandia.gov/jess/`

[9] Malkina O., Soshnikov D. Creating Interactive Systems of Adaptive Testing through the Internet using Intelligent Technologies. *In Selected Abstracts of 9th International Workshop "New Information Technologies"*, Moscow State Institute of Electronics and Mathematics, 2001. (In Russian)

[10] Krasteleva I., Soshnikov D. Promotion of Web Resources using Intelligent Technologies. *In Selected Abstracts of 9th International Workshop "New Information Technologies*, Moscow State Institute of Electronics and Mathematics, 2001. (In Russian)

[11] Orfali R., Harkey D., Edwards J., Instant CORBA, Wiley Computer Publishing, 1999.

[12] Hyacinth S. Nwana, Software Agents: An Overview, *Knowledge Engineering Review*, Vol. 11, No.3, 1996. pp. 1–40.

[13] Pfleger K., Hayes-Roth B., An Introduction to Blackboard-Style Systems Organization, KSL Technical Report KSL-98-03, Computer Science Department, Stanford University, 1997.