# USER ASSISTANT AGENT DEVELOPER MANUAL

**WP5**

| | |
|---|---|
| Document Filename: | **KWF-WP5-UAA-IISAS-v1.0-SoftwareDeveloperManual** |
| Work package: | **WP5** |
| Partner(s): | **IISAS** |
| Lead Partner: | **IISAS** |
| Document classification: | **PUBLIC** |

Abstract: This document provides a developer manual for User Assistant Agent K-WfGrid component.

## Delivery Slip

| | Name | Partner | Date | Signature |
|---|---|---|---|---|
| **From** | Michal Laclavik | IISAS | 28/09/2006 | |
| **Verified by** | Piotr Nowakowski | CYFRONET | 11/10/2006 | |
| **Approved by** | Steffen Unger | FIRST | 11/10/2006 | |

## Document Log

| Version | Date | Summary of changes | Author |
|---|---|---|---|
| 0.1 | 28-Nov-05 | Draft version | Michal Laclavik |
| 0.2 | 22-Dec-05 | Updated version including recommendation from internal review | Michal Laclavik |
| 0.3 | 30-Aug-06 | Final software developer manual | Michal Laclavik |
| 0.4 | 03-Oct-06 | Final software developer manual update | Michal Laclavik |
| 1.0 | 11-Oct-06 | QA check | Piotr Nowakowski |

# CONTENTS

## COPYRIGHT NOTICE



This research is partly funded by the European Commission IST-2002-511385 Project “K-Wf Grid”.

# 1. INTRODUCTION

The main role of User Assistant Agent (UAA) is to assist user with relevant knowledge/suggestions, which are applicable on current user situation. In UAA, we understand experience though notes entered by user. Such notes are displayed to the users in same or similar context. Thus UAA helps users in collaboration and knowledge sharing. Notes can be generated also automatically (e.g. past workflow and result notes generated by KAA-WXA). Notes can also contain reference to other resources such as workflows, results or images.

The other role of UAA is problem definition where user can describe his/her problem typing free text and relevant semantic description of problem is detected and processed by other K-Wf Grid components to build and execute workflows for given user problem.

For more information on functionality see user manual.

UAA is built on EMBET architecture developed in Institute of Informatics, SAS. This architecture is being extended during K-Wf Grid project and sometimes when we refer to EMBET or UAA this can be identical.

EMBET architecture is built and extended the way that it can be reused and it is quite independent of other K-Wf Grid components.

## 1.1. ABBREVIATIONS AND ACRONYMS

EMBET – Platform for Collaborating and Knowledge Sharing via User Asistance. Abreviations comes from "**E**xperience **M**anagement **b**as**e**d on **T**ext Notes"

UAA – User Assistant Agent sometimes referenced as EMBET

GOM – Grid Organizational Memory sometimes referenced as Memory

GWES – Grid Workflow Execution Service

KAA – Knowledge Assimilation Agent; See KAA manuals for more details

KAA-WXA – Workflow XML Analyzer is part of KAA which process past workflows to find knowledge resulting to Result and Workflow Notes.

## 1.2. REFERENCES AND SOURCE CODE

The full source of EMBET and EMBET GUI presented in this document are available on K-Wf Grid's central CVS repository. The JavaDoc documentation can be build out of source code using ant build file.

# 2. IMPLEMENTATION STRUCTURE

## 2.1. SOFTWARE USE CASES

UML use case diagram (Figure 1) as well as text description of use cases is the content of this chapter.

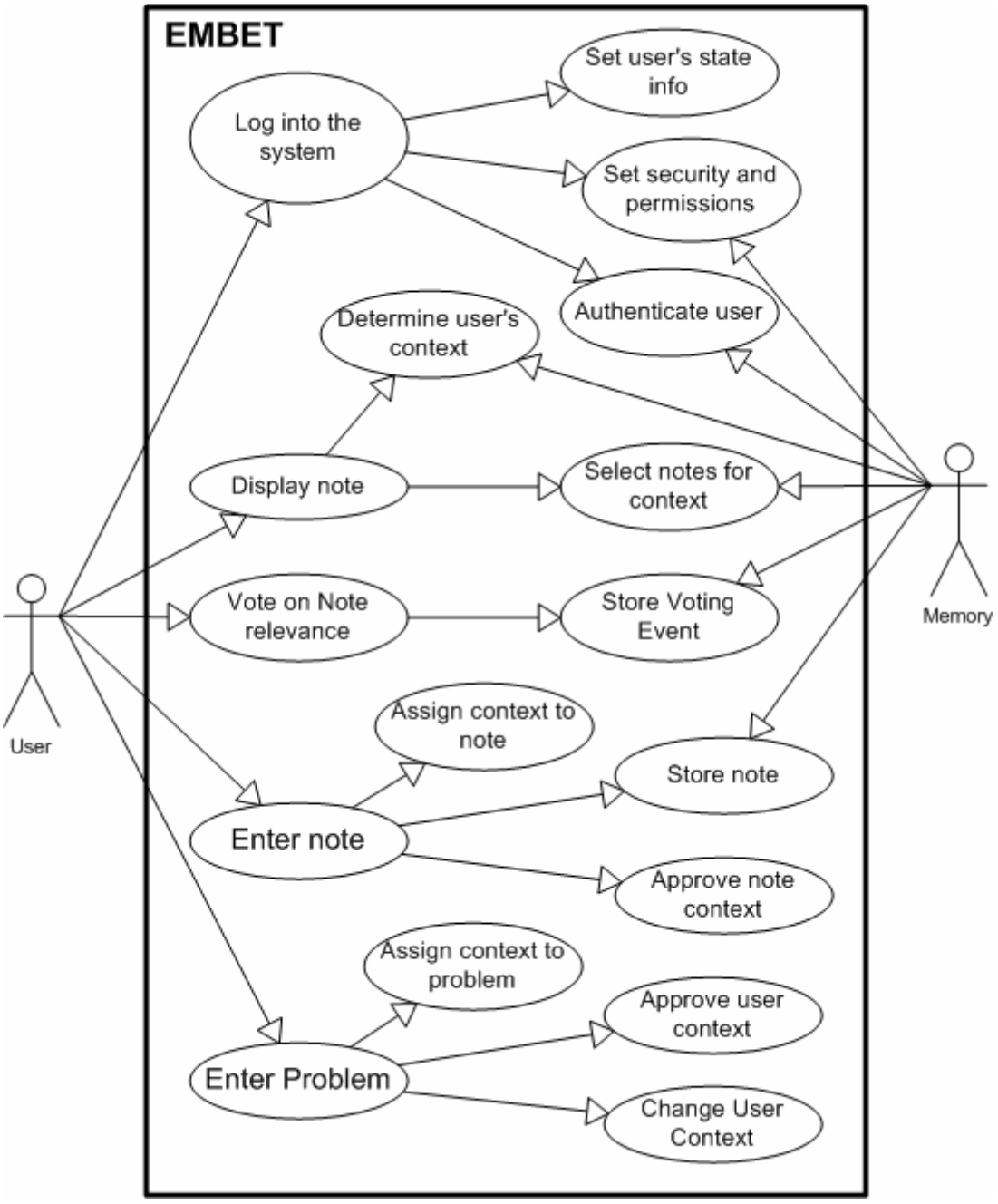


*Figure 1: UAA UML use cases diagram*

### 2.1.1. UML Use Case 1 – Log into the System

**ACTORS:** User, Memory

**GOAL:** Log into the System

- PRIMARY FLOW:
- User enters and submits his login/password
- <<uses>> Authenticate a user by checking privileges in authorization database
- <<uses>> Read security settings and permissions for the user in the authorization database
- <<uses>> Set user's status information

In K-Wf Grid logging is ensured by gridsphere portal infrastructure. Users need to be defined in domain ontology as User class individuals.

### 2.1.2. UML Use Case 2 – Problem/context specification

**ACTORS:** User

**GOAL:** Define user context

PRIMARY FLOW:

- A user wants to start a new task so he needs to specify his context if it is not already detected by external systems.
- A user enters a textual problem description
- <<uses>> UAA core tries to determine what is the context of such problem and detects an application related objects/elements relevant to text problem description
- <<uses>> A user inspects and approves the problem context proposed by UAA Core, possibly modifies the context and approves the new user context
- <<uses>> Change User context according to approved problem context

### 2.1.3. UML Use Case 3 - Display a Note for a Context

**ACTORS:** User, Memory

**GOAL:** Display relevant notes (experience) in user context

PRIMARY FLOW:

- User context had changed
    - Because changing of context was detected by external applications
    - Because a user described his/her problem (see previous Problem specification use case)
- <<uses>> Determine User's Context from the Memory
- <<uses>> UAA Core Selects Notes for the current User Context
- UAA GUI Displays the Notes for the Context

### 2.1.4. UML Use Case 4 – Voting on Notes

**ACTORS:** User, Memory

**GOAL:** Voting on displayed notes (experience) relevance

PRIMARY FLOW:

- A user reads notes for his/her problem context and thinks that some of displayed notes are not relevant or are very relevant
- A user votes on notes relevance
- <<uses>> UAA Core store the Note/Relevance pair to Memory which is used in future note display

### 2.1.5. UML Use Case 5 - Submit a Note to a Context

**ACTORS:** User, Memory

**GOAL:** Attach a note (experience) to application related context

PRIMARY FLOW:

- A user has finished an action and potentially has finding experience which he wants to share with other users
- A user enters a note
- <<uses>> UAA core tries to determine what is the context for which the User wants to write a note
- <<uses>> A user inspects and approves the context proposed by UAA Core, possibly modifies the context and approves the context
- <<uses>> Store Note submits the Note/Context pair to Memory

## 2.2. SOFTWARE COMPONENT MODEL

Architecture of UAA consists of 3 main elements:

- Core of the system
- Graphical User Interface (GUI)
- System Memory (GOM)

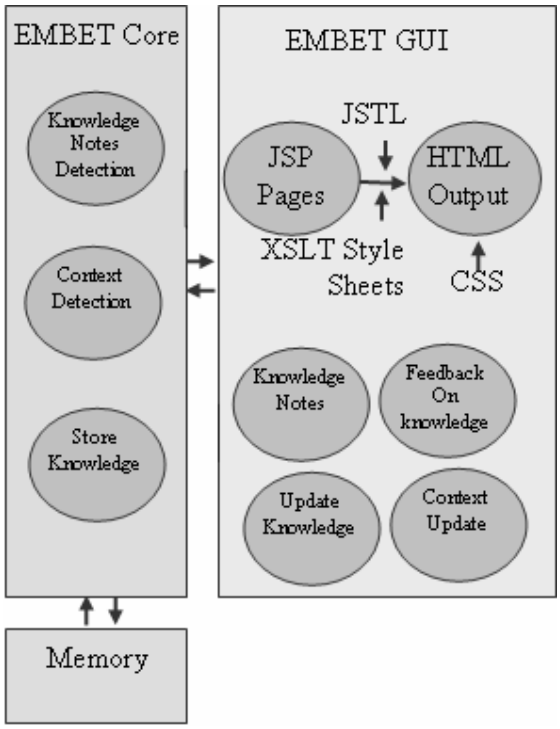


*Figure 2: UAA Architecture*

**UAA Core** provides the main functionality of UAA. It determines a User context and searches for the best knowledge (in a form of text notes) in its Memory. The knowledge is subsequently sent through XML-RPC or SOAP to UAA GUI. When a user enters a note, the UAA Core processes the note and determines the context of it from the current user's context and the context detected in the note. When the user confirms the context, the UAA Core stores the note and user's feedback. The core also handles a user the state (context).

**UAA GUI** visualizes the knowledge and the user's context information to the user. Furthermore it informs the UAA core about user context changes. The context can be reported also directly to the UAA core from external systems (e.g. from workflow systems, received emails, or file system monitors). UAA GUI visualizes knowledge based on XML transformation to HTML through XSL Templates processing. Moreover UAA GUI has an infrastructure for a note submission and context visualization. It further provides a user with feedback (voting) on knowledge relevance. In addition, it contains a user interface for knowledge management by experts where an expert can change a note and its context.

**UAA Core - UAA GUI interface** is used for an XML data exchange between UAA Core and UAA GUI. The Interface will be based on the SOAP protocol where both components act as web services; currently we use the XML-RPC protocol for an XML message exchange.

**Interface to Memory** is used for information and knowledge extraction and storage. It is based on RDF/OWL data manipulation using Jena API, which UAA Core uses to extract and store knowledge. In K-Wf Grid project GOM is used as knowledge memory.

Experience is represented by text notes, an unstructured text, entered by a user. For the context and environment modelling we use ontology, thus we use a Protégé ontology editor for ontology based

modelling. Ontology is stored and managed in the Web Ontology Language (OWL). The Jena Semantic Web Library is used for knowledge manipulation and knowledge storing. The Java technology is used for developing the system and user Interface is based on the JSP technology. The XSL templates are used to transform XML generated from OWL to displayed HTML. Since the Java technology is chosen as background for the UAA, a choice of the web server – Jakarta Tomcat and implementation middleware is reasonable. XML is a widely used language for web development; it is regarded as a universal format for structured documents and data on the Web. Therefore in UAA XML is used in various forms/degrees from description of ontology to the communication content language. The XML/XSLT technology permits visualization of XML documents and display of information presented to the user by UAA. The JSTL is a native Java library for XML processing.

## 2.3. DETAILED IMPLEMENTATION MODEL

In this chapter we provide description of developed software libraries to be used for knowledge management in the UAA system.

Library consists of the following main classes (Figure 3): Ontology, Memory, Anotate, DetectContext, EventHandler and Core.

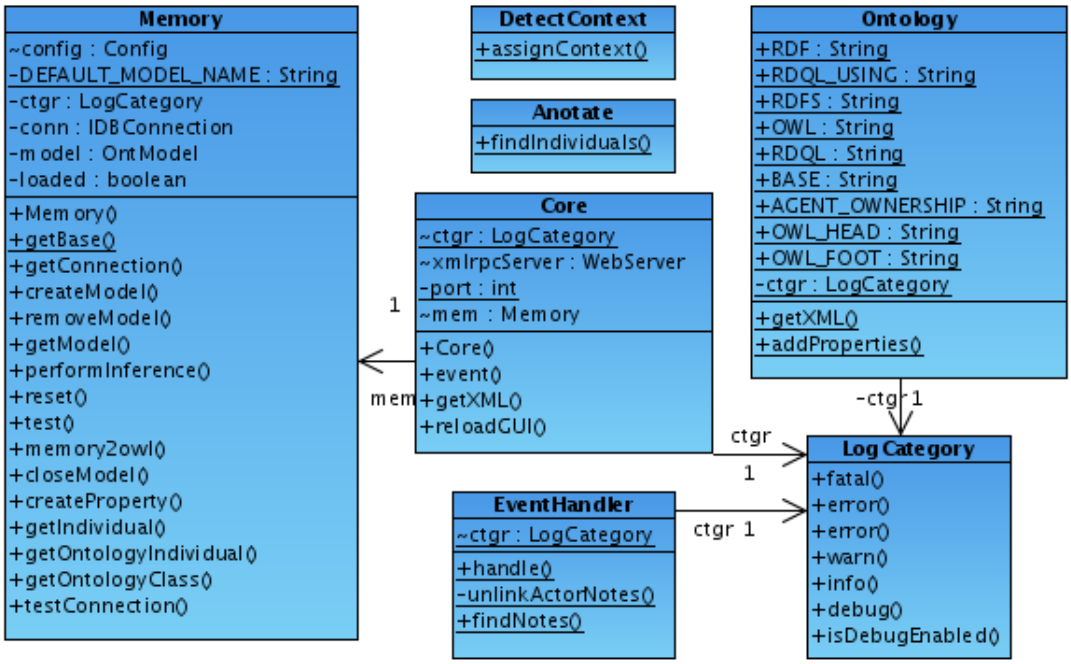


*Figure 3: UML Class Diagram of Main UAA classes*

The class Ontology (Figure 3) has basic constants related to OWL ontology and the used ontology model.

The class Memory (Figure 3) has a functionality to load, store and manipulate the knowledge model based on an RDF or an OWL. The Jena library is used to this manipulation. The memory can load an OWL file developed in Protégé which becomes an ontology model of the knowledge memory. The

memory can be stored in a database back-end such as MySQL, other RDBS supported by Jena or into the OWL file. It is used to connect UAA with GOM.

The class Anotate (Figure 3) contain functionality to detect ontology elements in plain text based on the Knowledge model loaded by Memory.

The class DetectContext (Figure 3) use Anotate functionality to assign found ontology elements to Note individual as context.

The class EventHandler (Figure 3) can handle defined events, manipulate them and update knowledge model according to needs. Algorithms for updating actors' context $f_C$ and resources $f_R$ are covered in this class as well. Events are based on Event ontology class defined in the knowledge model.

The class Core (Figure 3) is the main class, which runs the UAA system. It also covers XML-RPC server and methods which can be called via XML-RPC. The event() method is used to communicate events in RDF/OWL from external systems or UAA GUI (see next chapter). The getXML() method is used by UAA GUI to retrieve data about the user knowledge model or other information and knowledge stored in the UAA Memory. The reloadGUI() method is used by UAA GUI to detect weather GUI need to be reloaded.

The developed library is built on previously developed AgentOWL library and classes such as Ontology and Memory are identical. The important part is the AgentOWL knowledge model which is described in following publications available at http://laclavik.net/ website:

- Michal Laclavik, Marian Babik, Zoltan Balogh, Ladislav Hluchy: AgentOWL: Semantic Knowledge Model and Agent Architecture; In Computing and Informatics. Vol. 25, no. 5 (2006), p. XX-YY. ISSN 1335-9150, Chapter 2 --in print--
- Michal Laclavik, Marian Babik, Zoltan Balogh, Emil Gatial, Ladislav Hluchy: Semantic Knowledge Model and Architecture for Agents in Discrete Environments; In: Proc. of ECAI 2006 Conference, 28 August - 1 September 2006, Riva del Garda, Italy
- Michal Laclavik: Experience Management based on Ontology and Text Notes (The EMBET System); Work submitted for the RNDr. degree; Ustav Informatiky, Prirodovedecka fakulta, Univerzita Pavla Jozefa Safarika v Kosiciach, September 2005; Defended 10th February 2006

## 2.4. PRODUCT INTERFACES

This chapter describes Interface between UAA Core and UAA GUI or external systems. It shows mainly an example of main Event types, which are communicated via XML-RPC method event().

All events have a following structure: an action performed by an actor on resource in particular context.

Note Adding Event is sent by UAA GUI when a new note is added by the user. In the table bellow a concrete example can be seen: note “MM5 is not a good model for Bratislava in September” is added by “misos” user, action is “aAdd”. This event is sent using XML-RPC method event() of Core class.

```
<Event rdf:ID="event20050903232243">
    <actor rdf:resource="#misos"/>
    <resource>
      <Note rdf:ID="noteTest">
        <label>
          MM5 is not good model for Bratislava in September
        </label>
      </Note>
    </resource>
    <action rdf:resource="#aAdd"/>
  </Event>
```

When a note is submitted, UAA core calls EventHandler and DetectContext and returns an XML model of Note with assigned detected context. Returned XML model is shown in the table below.

```
<Note ID="noteTest">
  <label>MM5 is not good service for Bratislava area in September</label>
  <hasAuthor>
    <User ID="misos">
      <label>Michal Laclavik</label>
    </User>
  </hasAuthor>
  <hasTime>03.09.2005 23:34:40</hasTime>
  <context>
    <Month ID="mSeptember">
      <keyword>fall</keyword>>
      <label>September</label>
    </Month>
  </context>
  <context>
    <Class ID="Location">
      <comment>Geograficke miesto</comment>
    </Class>
  </context>
  <context>
    <MeterologyService ID="serviceMM5">
      <label>MM5 Meterology service</label>
      <keyword>MM5</keyword>
    </MeterologyService>
  </context>
  <context>
    <Capital ID="locBA">
      <hasRegion>
        <Region ID="reg02">
          <label>Bratislavsky Kraj</label>
        </Region>
      </hasRegion>
      <label>Bratislava</label>
      <latitude>48.15</latitude>
      <longitude>17.116667</longitude>
      <hasRegion>
        <Country ID="countrySlovakia">
          <hasRegion>
            <Region ID="regionEurope">
            </Region>
          </hasRegion>
          <label>Slovakia</label>
        </Country>
      </hasRegion>
    </Capital>
  </context>
</Note>
```

Note context detected in a submitted note is displayed as a check list to a user and the user reselects relevant context and submit this context. Event “confirm note context” is generated by UAA GUI and sent to UAA core as an event (example bellow). In this example a user selected as relevant context the MM5 service and Bratislava location.

```
<Event rdf:ID="event20050903232406">
  <actor rdf:resource="#misos"/>
  <action rdf:resource="#aConfirm"/>
  <resource rdf:resource="#noteTest"/>
  <context rdf:resource="#serviceMM5"/>
  <context rdf:resource="#locBA"/>
</Event>
```

A user can change context by typing text (see scenarios on problem definition in user manual) or when an external system will send “change user context” event. Example of such event can be seen bellow. In this example context is changed for service MM5 and Bratislava location.

```
<Event rdf:ID="event20050903232345">
  <actor rdf:resource="#misos"/>
  <action rdf:resource="#aChange"/>
  <resource rdf:resource="#misos"/>
  <context rdf:resource="#serviceMM5"/>
  <context rdf:resource="#locBA"/>
</Event>
```

If context is changed this way, algorithm for updating actor's/user's context $f_C$ is executed. Also resources $f_R$ updating algorithm is executed and, thus relevant notes for new context are detected and assigned to an actor/user model. In the table below XML of the actor/user model is shown. In this example above the described note is shown as well as MM5 service and Bratislava location context is assigned. This XML model can be accessed using XML-RPC using getXML() method.

```
<User ID="misos">
  <context>
    <Location ID="locBA">
      <label>Bratislava</label>
    </Location>
  </context>
  <resource>
    <Note ID="noteTest">
      <label>
        MM5 is not good service for Bratislava area in September
      </label>
      <context>
        <Location ID="locBA">
          <label>Bratislava</label>
        </Location>
      </context>
      <context>
        <MeterologyService ID="serviceMM5">
          <label>MM5 Meterology service</label>
          <keyword>rain</keyword>
          <keyword>meterology</keyword>
          <keyword>MM5</keyword>
          <keyword>weather</keyword>
        </MeterologyService>
      </context>
    </Note>
  </resource>
  <context>
    <MeterologyService ID="serviceMM5">
      <label>MM5 Meterology service</label>
      <keyword>rain</keyword>
      <keyword>meterology</keyword>
      <keyword>MM5</keyword>
      <keyword>weather</keyword>
    </MeterologyService>
  </context>
  <label>Michal Laclavik</label>
</User>
```

Interface contains other events regarding voting on notes and others but this functionality is still in the development stage. Events described above are implemented and functioning.

# 3. SOFTWARE TESTING

User Assistant Agent can be tested by running appropriate test classes for all important functionalities. Such test classes are created using JUnit . Semantic Annotation, sending red workflows to GWES and others can be tested using such test classes.

UAA core can be tested using web based testing framework (Figure 4). Using this PHP based framework, events which simulate events from UAA GUI or other components can be tested. Also changes of user model can be seen on left side.

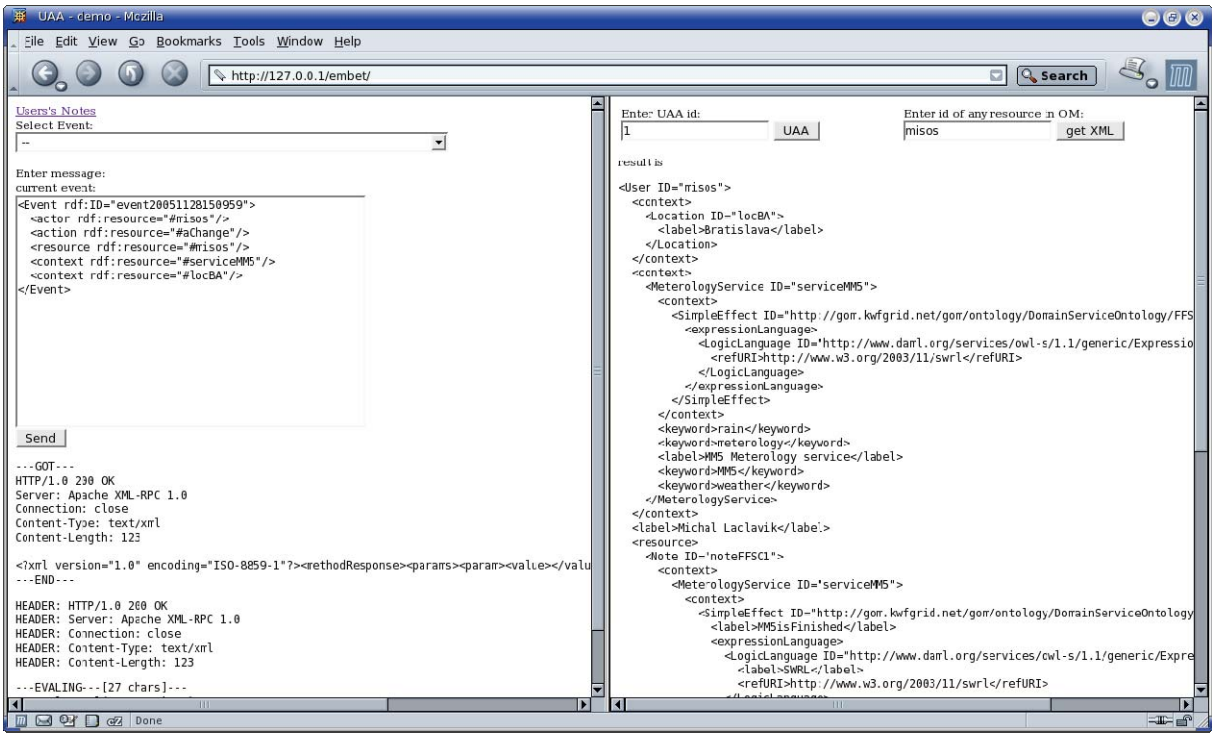


*Figure 4: Web based testing framework of UAA core.*

## 3.1. KNOWN BUGS

- Always type at least 3 words when problem or note is specified because otherwise entered problem/note will be matched by semantic annotation as well.
- Reimplementation of Annotation algorithm is needed because now it is case sensitive which match only some results.
- Reimplementation of Context matching algorithm is needed since similarity of context is not supported. This will have influence on voting and note/knowledge ordering mechanism.

# 4. CONTACT INFORMATION AND CREDITS

**Michal Laclavik**

- Author of EMBET architecture
- Core implementation
- Memory implementation & integration with GOM
- 2nd version of GUI implementation
- Ontea - Semantic based annotation

Email: laclavik.ui@savba.sk; Web: http://laclavik.net/

**Emil Gatial**

- Author of First GUI implementation
- Reloading of GUI using AJAX technologies
- First implementation of annotation algorithm
- consultations

Email: emil.gatial@savba.sk

**Zoltan Balogh**

- GUI related and other consultations

**Martin Seleng**

- Semantic Annotation (Ontea) success rate measuring and evaluation.

**Martin Maliska, Branislan Simo**

- Java programming and related consultations

**Ondrej Habala, Ladislav Hluchy**

- General consultations
- Thanks for pushing to work on UAA ☺

**Bartek Kryza, Marian Babik, Jacek Kitowski**

- Ontology, Knowledge description, semantic web consultations
- GOM integration

**K- Wf Grid & Pellucid Project Consortium**

- Ideas
- Evaluations
- Consultations

## 5. THE LICENSE AGREEMENT

GPL License

See http://www.gnu.org/licenses/gpl.txt for more details.