DELIVERABLE

Project Acronym: iTranslate4

Grant Agreement number: 250405

Project Title: Internet Translators for all European Languages

---

1.2 Central Communication Server

Revision: version 1

Authors:

Péter Kundráth (MorphoLogic Számítástechnikai Kft., MOR)
Kovács Miklós (Nyelvtudományi Intézet – Magyar Tudományos Akadémia, RIL)

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	Public	X
C	Confidential, only for members of the consortium and the Commission Services	

Table of contents:

Nature: prototype

1. Architecture and components

We decided to divide the whole system into two main parts: the CSA and the GUI. CSA stands for the Central Server Application (also named as Central Communication Server in this document), it aggregates the translations from different translator services (TS). The GUI handles the user interaction and communicates with CSA.
We planned to wrap these pieces into a lightweight web server which can alter routes and ports to show these services as one application.

2. Central Communication Server

For CSA we chose the Java Enterprise specification as the foundation of the system. As JavaEE applications can be deployed on all application servers implementing the specification, we choose the JBoss application server, but GlassFish server is constantly monitored for proper operation.
The implementation uses the following specifications from the Java EE collection, and some additional tools:

1 JBoss 6 Application Server
- 1.1 Servlet - web-tier component accepting user HTTP requests, processing them, making decisions on which TS API to use. The operationIDs are propagated back to the GUI.
- 1.2 JMS Queue - Java Messaging Service-based message queuing facility to store messages representing translation requests internal to the gateway
- 1.3 EJB3 - Enterprise Java Beans implementing the business logic of the gateway
- 1.4 MDB - Message-Driven Beans process the translation requests and start communication with the TS APIs. MDB operation mode allows effective load-balancing of TSes
- 1.5 Hibernate - component responsible for ORM (Object-Relational Mapping)

2 Hypersonic SQL Database - lightweight, in-memory database implemented in Java. It can be easily replaced thanks to Hibernate's capabilities.

- Apache Ant - the Java synonym to the C-world's make tool. Mainly used for building and packaging the application, multi-server deployments can easily be performed with this tool.

3. Details of the implementation

CSA basics

The CSA can be built using "ant deploy" command in terminal. The deploy properties are in the **build.properties** file. It contains the path to the JBoss application server.

Hypersonic SQL (HSQL) is used to persist the data, it resides in the **database** folder. To start the database you must run this script in a terminal window: **database/start_db**. There is a very simple

database client for HSQL, it can be used to browse the stored data. You can start it by running this script from a terminal window: **database/start_dbmanager.** (*connection url: jdbc:hsqldb:hsql://localhost:1234/itranslate, user and password: jboss/jboss*).

The **hibernate.cfg.xml** (configuration options of the database connection and connector between the HSQL and the application) and the **ServiceConfig.json** (JMS queue) are automatically loaded when the application is deployed. *Note: the database must be started first.*

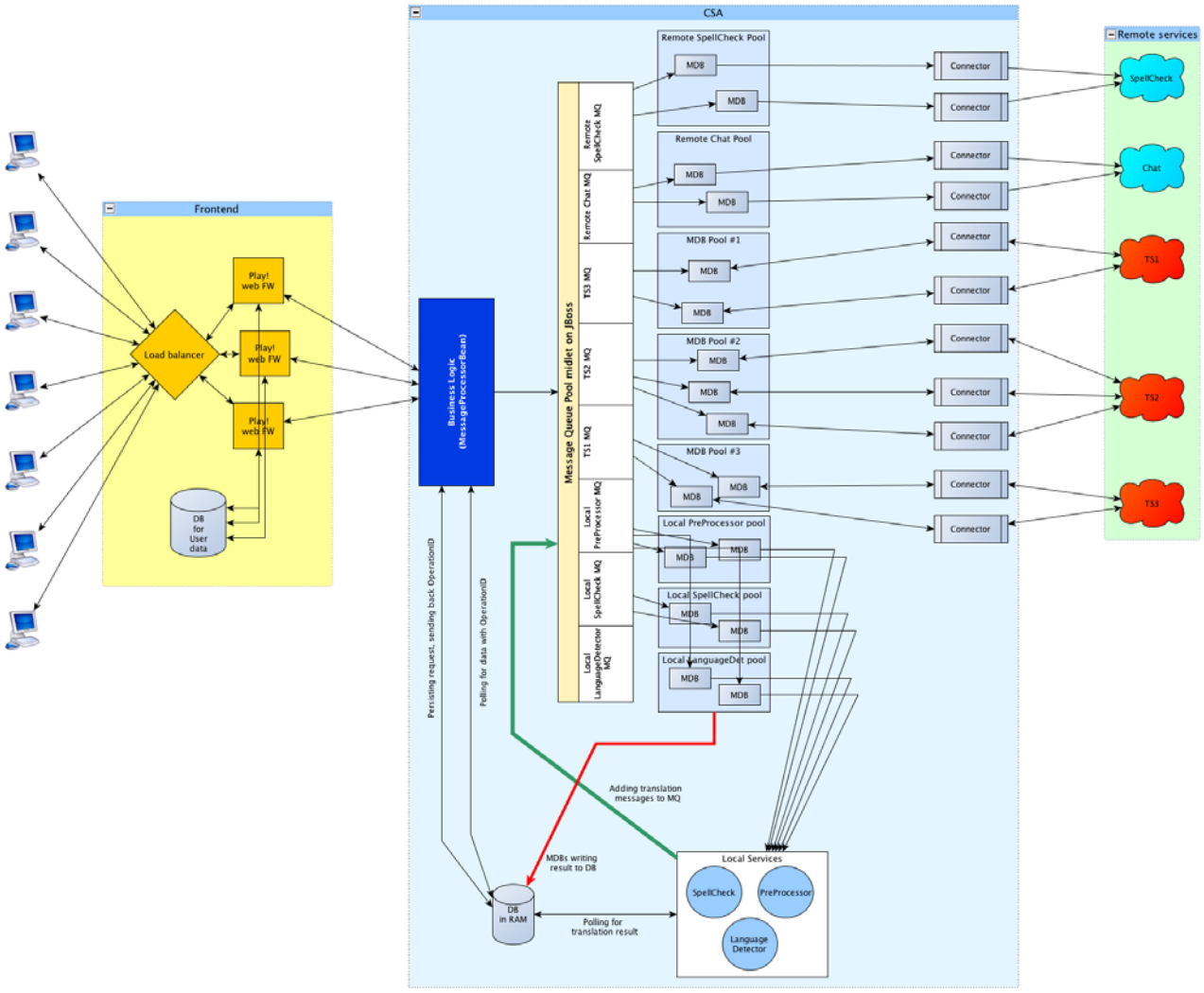
The source code files for the servlets and the beans are in the **src** folder in separate sub-folders.

Message Queue

Based on our concept the application opens one queue for one TS. At the end of every queue there are many MDBs (Message Driven Beans).

When a translation request comes it is processed by the **CSA Servlet**, it persists the data then sends it to the **MessageProcessor Bean**. This bean is responsible for driving the messages to the correct queue. After selecting the proper message queue, the MessageProcessor Bean calls the **QueueHandler Bean**. This bean puts the message to the appropriate queue. The **QueueProcessor Bean** gets the message and tries to connect to the TS and sends the text to it. At last the response of the TS is stored in the database. If an error rises during the process the MessageProcessor can decide to put the message back to the queue.

System diagram



REVISION HISTORY AND STATEMENT OF ORIGINALITY

Revision History

Revision	Date	Author	Organisation	Description
V1	31.03.2011	Péter Kundráth, Kovács Miklós	MOR/RIL	A working prototype of the central server application managing the communication with translation services. This prototype is used internally to begin testing the integration and the application itself

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.