Intelligence Virtual Analyst Capability (iVAC) - Framework and Components

High-level Software Architecture Description (SAD)

Version 0.4

Prepared by :

Luckson Vilus

Fujitsu Consulting (Canada) Inc.

2000, Lebourgneuf Blvd., Office 300

Québec (Québec) G2K 0B8

PWGSC's Contract Number: W7701-135551/001/QCL

Contract Scientific Authority: Alexandre Bergeron-Guyard

(418) 844-4000 x 4107

DRDC – Valcartier Research Centre

Contract Report
DRDC-RDDC-2014-C217
July 2013

Change History

Version	Description	Author	Date
0.1	Initial version of document	Luckson Vilus	May 9, 2013
0.2	Description of main blocks of the SAD	Luckson Vilus	May 20, 2013
0.3	Modifications following internal review	Luckson Vilus	June 6, 2013
0.4	Internal review and modifications	Guy Michaud	June 7, 2013
0.5	Modifications related to Sprint 3 delivery	Luckson Vilus	January 20, 2014
0.6	Modifications related to Sprint 4 delivery	Luckson Vilus	March 31, 2014

Issuing Organization

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Identification

This document is entitled "High-level Software Architecture Description (SAD)". It is produced by Fujitsu Consulting (Canada) Inc. for Defence R&D Canada – Valcartier, under contract "Intelligence Virtual Analyst Capability (iVAC) - Framework and Components", contract number W7701-135551.

1.2 Scope

This document presents and documents the overall architecture of the Intelligence Virtual Analyst Capability (iVAC) Prototype. It describes major building blocks, software components and services that collaborate to deliver the iVAC capabilities. The approach and design patterns used to build the system are alighted in this document.

1.3 Background

TBC

1.4 Document Overview

This document is divided into following main sections:

• Solution Overview	Presenting an overview of the solution, its context, the main objectives that guide its construction and the main use cases supported by the system;
• Construction Strategy	Presenting the strategy and approach that guide the construction of the iVAC system;
• Use Cases	Presenting the use cases supported by the iVAC Software System;
• Software System Architecture	Presenting the software components that are part of the iVAC Software System;
• Information Model	Presenting the information manipulated by the iVAC Software System or exchanged between the iVAC and its users and/or other systems with which it interacts.
• Deployment Strategy	Presenting the deployment model of the system and the strategy that guides such a deployment.
• Development Plan	Presenting the prototype development plan.
• Test Plan	Presenting the test plan for validating that the prototype meets requirements.

2 Solution Overview

2.1 Description

The Intelligence Virtual Analyst Capability (iVAC) is one of the multiple initiatives of the DRDC Valcartier, a Research and Development (R&D) center located in Quebec City, in the domain of Intelligence and Information and related areas. It can be seen as "a computerized software assistant supporting the intelligence analysts in sense making tasks, while being ultimately capable of taking on autonomous analytical tasks in concert with other analysts (virtual or human)" [1] .

2.2 Goals

The main goal of the project is to develop a multi-modal interactive spoken dialog system that can assist a user, mainly an analyst, in his daily duties. The system must have Speech Recognition and Speech Synthesis capabilities in order to be able to interact with the users in a very natural way like it would be in a human-to-human interaction. To be effective, the system must provide accurate information to the user in an acceptable time frame.

At the end, a fully functional iVAC system needs to be able to:

- Execute commands coming from different input sources:
- Automatic Spoken Language Recognition engine;
- Form-based command engine (graphical user interface);
- Command line features;
- Use non-verbal features like
  - Facial Interpretation;
  - Gestural movement analysis.
  - Etc.
- Communicate results to users by different means:
  - Speech synthesis (Text-to-speech) engine;
  - Form-based or graphical user interface;
  - Text line;
  - Maps, images and pointing features.

2.3 Objectives

The main objectives pursuit by creating the iVAC is to develop a software system that integrates those capabilities:

- Managing Context;
- Acquiring Information and Knowledge;
- Timely Informing, Communicating and Interacting with User;
- Learning User and Task Models;
- Monitoring, Scheduling, Managing and Evaluating Activities;
- Supporting Complex Intelligence Tasks;
- Interacting with Humans and Other Systems;

Context Model

In this section we present the system context diagram which highlights the main systems and/or services with witch the iVAC system interact in order to deliver its capabilities. The users interacting with the system are also presented.

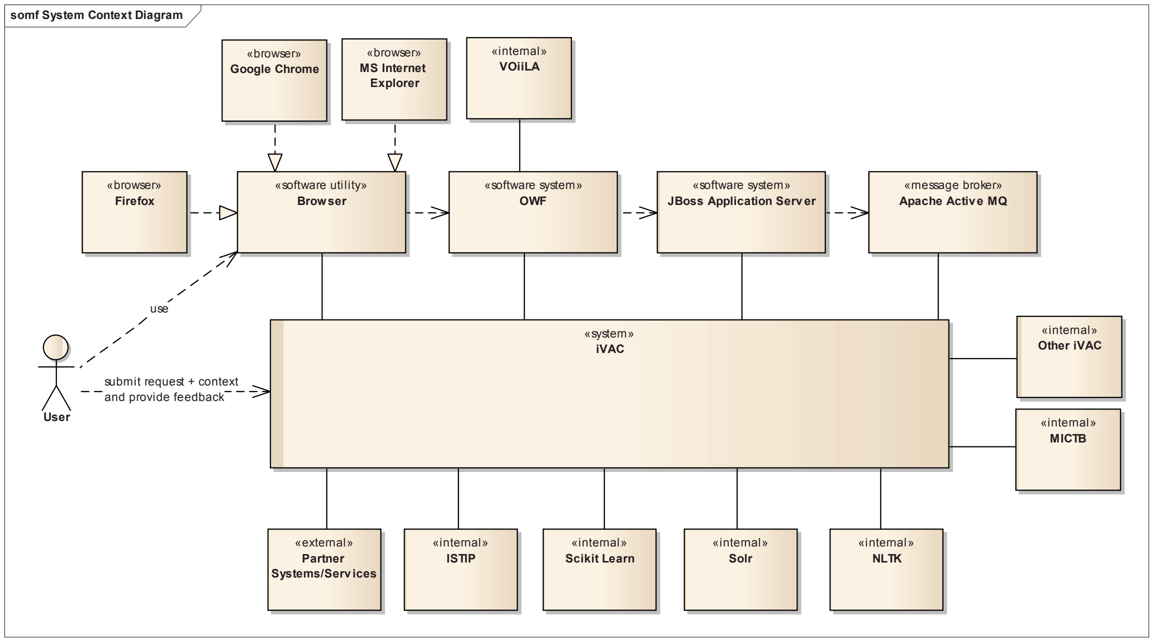

somf System Context Diagram
«browser» Google Chrome
«browser» MS Internet Explorer
«internal» VOiiLA
«browser» Firefox
«software utility» Browser
«software system» OWF
«software system» JBoss Application Server
«message broker» Apache Active MQ
use
«system» iVAC
«internal» Other iVAC
«internal» MICTB
User
submit request + context and provide feedback
«external» Partner Systems/Services
«internal» ISTIP
«internal» Scikit Learn
«internal» Solr
«internal» NLTK

Figure 1: Solution Architecture Context

As depicted in the diagram above, the iVAC system communicates with those users and system and /or services:

- **The user**: submits request and provides context to the system and gets relevant answers according to his context from the system. This interaction will be done either by voice (Speech Recognition and Speech Synthesis capabilities) and/or through the iVAC visual (UI[1]) components;
    - A browser which can be either Google Chrome, Firefox or Microsoft Internet Explorer;
- **ISTIP**: the Intelligence Software and Technology Integration Platform is a SOA-based services, data banks and applications. In some cases, the iVAC communicates with services hosted in the ISTIP infrastructure to gather information in order to be able to response to a user request. This integration or communication is done through well-defined service interfaces;
- **Document Processing System:** provides document processing and document retrieval capabilities to the iVAC system. Because document processing is a time-consuming process, the integration with such service is made by means of asynchronous communication patterns and protocols. JMS[2]

[1] UI= User Interface
[2] JMS = Java Message Service

and AMQP[3] coupled with NIO[4] are used to bring reliability to the communication between the iVAC and that software systems;

- **VOiiLA:** the Visionary Overarching Interaction Interface Layer for the Analyst is web-based user interface that provides human-computer interaction in order to exploit the ISTIP services. Ideally, the iVAC UI components need to be able to be integrated to VOiiLA and the iVAC must be able to use VOiiLA components. To make this possible, the iVAC visual components must be web-based. These components are hosted in Ozone Widget Framework(OWF);
- **JBoss:** iVAC uses JBoss as its Java EE Application Server that provides the needed low level services, multi-threading, transaction management and integration capabilities based on the capacity of JEE;
- **Apache Active MQ:** ActiveMQ acts as a message broker that brings more asynchronous and integration capabilities by means of the implementation of all the integration patterns defined in the «Enterprise Integration Patterns» book[5];
- **Scikit Learn:** this software component brings Machine Learning capability to the iVAC system;
- **NLTK:** the «Natural Language Toolkit» is a set of tools that give to the iVAC the capability to process human natural language;
- **Apache Solr:** is the popular, blazing fast open source enterprise search platform from the Apache Lucene project. Its major features include powerful full-text search, hit highlighting, faceted search, near real-time indexing, dynamic clustering, database integration, rich document (e.g., Word, PDF) handling, and geospatial search. Solr is highly reliable, scalable and fault tolerant, providing distributed indexing, replication and load-balanced querying, automated failover and recovery, centralized configuration and more. Solr powers the search and navigation features to the iVAC system;
- **Other iVAC:** the iVAC must be able to communicate with other iVAC in its environment. This communication can happen by mean of the iVAC own services and capabilities;
- **MICTB:** the Multi-Intelligence Capability Test Bed is an environment to conduct testing and evaluation of multi-intelligence capabilities. The iVAC system must be able to integrate or communicate with this environment.

[3] AMQP = Advanced Message Queuing Protocol
[4] NIO = New Input/output (I/O)
[5] Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Gregor Hohpe and Bobby Woolf, Addison-Wesley Professional, 1 edition (October 20, 2003) http://www.eaipatterns.com/

2.3.1 Interaction overview

This section presents a high level view of the interaction between the iVAC and its environment.

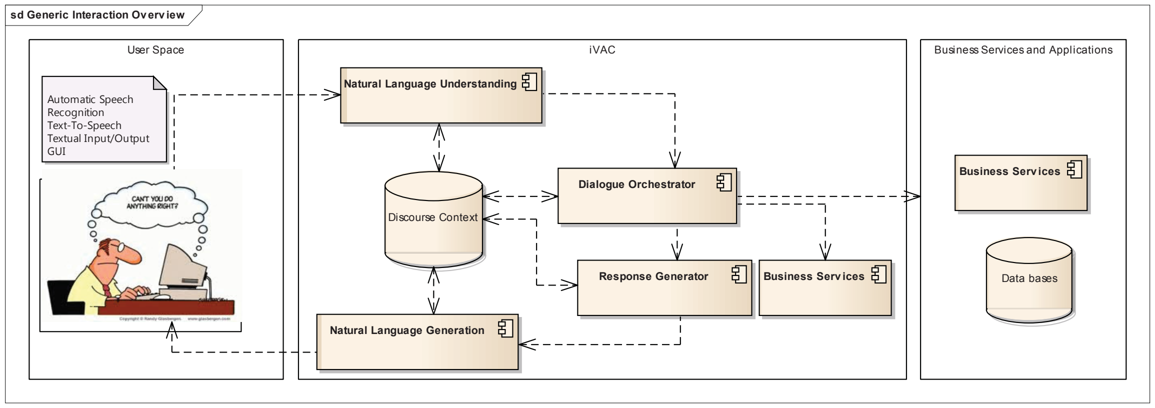

sd Generic Interaction Overview
User Space
Automatic Speech Recognition
Text-To-Speech
Textual Input/Output
GUI
CAN'T YOU DO ANYTHING RIGHT?
iVAC
Natural Language Understanding
Discourse Context
Dialogue Orchestrator
Response Generator
Business Services
Natural Language Generation
Business Services and Applications
Business Services
Data bases

Figure 2: High level Interaction overview

As depicted in the diagram above, the interaction with the iVAC system happens as fallow:

- The user submits a request to system in natural language;
- The «Natural Language Understanding» component translates the user voice into a command;
- The «Dialogue Orchestrator» receives the user request and then calling the necessary services to process that command and receives results from the calling services;
- The «Dialogue Orchestrator» updates the Discourse context;
- The «Dialogue Orchestrator» calls the «Response Generator» to produces a response from the received results;
- The produced response is forwarded to the «Natural Language Generator» in order to produce a pronounceable version the response;
- The response is pronounced to the user and displayed to user's output devices known by the system.

2.4 System Architecture

The iVAC is as interactive spoken dialog system capable to handle human-computer interaction in a natural fashion. In such a system, a user asks question to the system, in natural language, and then, the system, according to the user's context, gathers information in order to answer to the user questions.

The architecture of the iVAC is like traditional client-server architecture, a three-tier one, in which there are:

- **The iVAC UI:** this tier materializes the human-computer interaction. Beside the UI components, an avatar, a graphical image that represents a person, is hosted in this layer. Speech recognition and speech synthesis functionalities are provided by this tier;
- **The Dialogue Orchestrator:** it represents an essential part of system, and its main task is to determine, at a certain moment, based on the application domain, the user context, and the user's last utterance, which is the next action the system must perform, and, if possible, to predict the contents of the next user utterance. As such, it manages interaction between the user and the system backend holding the system business services;
- **iVAC Core Services:** this tier holds all iVAC business services or components. The intelligence of the iVAC resides in this layer. Without the software component or services of this layer, it would be impossible for the system to answer any user questions.

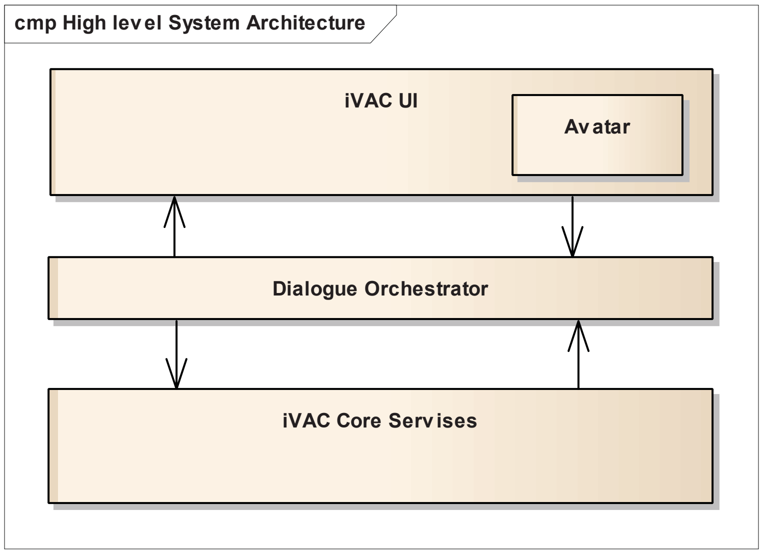


Figure 3: High level architecture

3 Construction Strategy

This section presents the strategy and the approach to develop the iVAC software system. It combines software architecture best practises with some well-chosen design patterns to build a system that conforms to the DRDC standards.

3.1 Approach

As decided in the first report (see reference [2] for more details), a SOA-based service approach is used to develop the system. According to this approach, each iVAC component will be developed by following SOA principles, standards and patterns. To guaranty an acceptable response time, a clustered architecture is used to deploy the system. For time-consuming tasks, asynchronous communication pattern is used to carry them. By doing this, the system will be more responsive.

Java Enterprise Edition version 6 (JEE 6) is the preferred development language. Because of that, each iVAC's component will provide a local view, a remote view and a web service view. Doing this, this iVAC services will be able to integrate any applications even if they are not developed in the same programming language than the iVAC services.

Some components will be developed in .Net C# (a windows-based application, because of Nuance Dragon).

The NLU components will be developed in Python (NLTK[6] library used to develop those components needs Python Interpreter). A service adapter will be developed to integrate this service.

Web technologies will be used to develop the iVAC visual components to make it possible to be integrated into VOiiLA and to be able to integrate VOiiLA widget components.

Service adapters will be used to integrate any system or service that does not provide a suitable service interface.

[6] NLTK = Natural Language Toolkit

4 Use Cases

This section present and overview of the business and system use cases that are supported by the iVAC Software System.

4.1 Business Use Cases

The iVAC support the following business use cases:

- **Process User Request**: it is an abstract general use case that supports the processing of requests from the user. This use case is materialized by some more specialized use cases that deliver the actual processing capability:
    - **Retrieve documents**: This use case processes user requests for retrieving documents;
    - **Support threat analysis**: this use case processes user requests for supporting threat analysis tasks;
    - **Perform threat analysis task**: this use case processes user requests about performing threat analysis tasks.
- **Manage Context:** supports all basic capabilities related to user’s context;
- **Manage Feedback:** allows the user to provide feedback either on the request submitted to the system or on the response getting from the system.

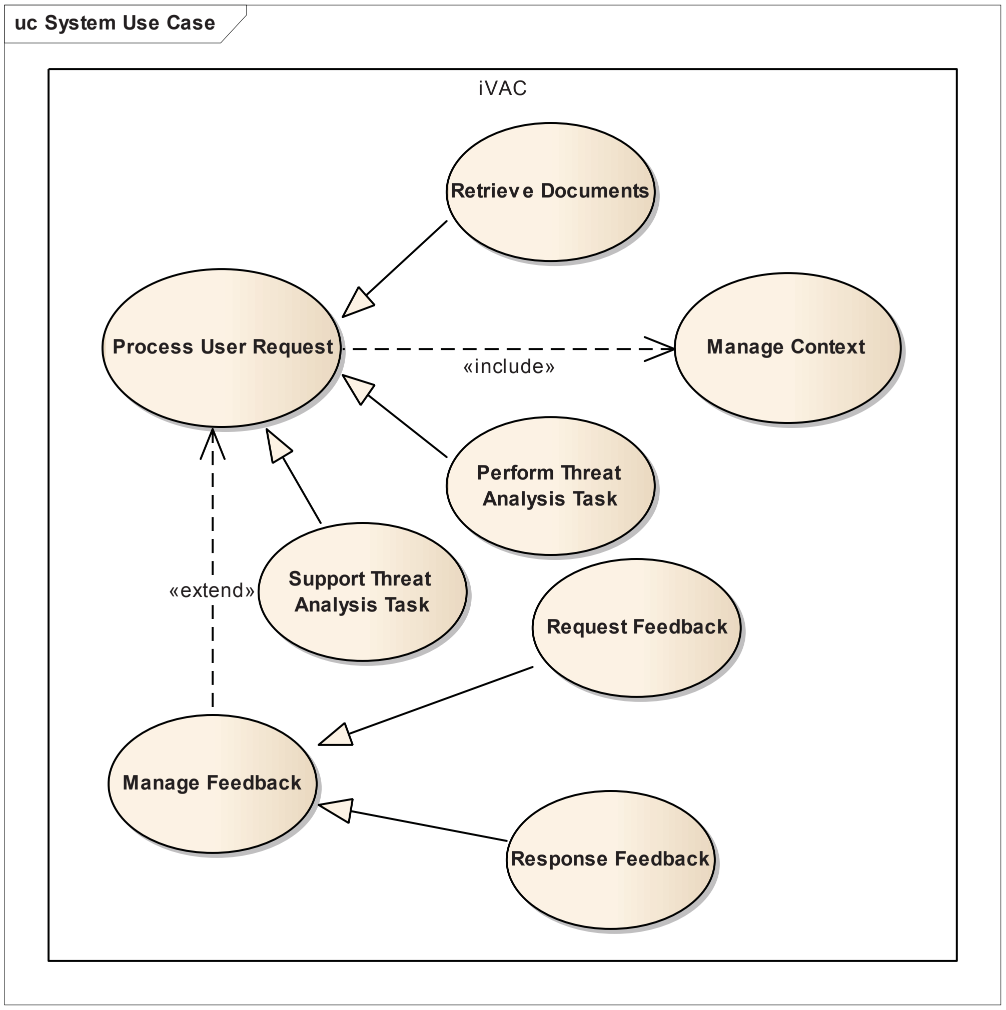


Figure 4: iVAC Business Use Cases

Note that the Process User Request use case includes the Manage Context use case, since the system must be aware of the user's context. The Process User Request use case can be extended by the Manage Feedback use case when the user is providing feedback to the results of its request.

4.2 System Use Cases

This section presents the use cases supported by the system according the user point view.

4.2.1 Process User Request

As stated in the section 4.1 the «Process User Request» is an abstract use case that is specialised in «Retrieve Documents» use case, «Support threat analysis task» use case and «Perform threat analysis task» use case.

This business use case is broken down into the following functional or system use cases:

- ***Submit Request***: Let the user submit a request to the system;

- ***Refine Request***: in case of misunderstanding, the user can refine a submitted request by providing more detail. The refined request is then submitted to the system;
- ***Rewrite Request*:** According to some rules and the knowledge the system gets about the user, the system rewrites the user's request before processing without the intervention of the user;
- ***Process user speech*:** The speech recognizer subsystem converts the user's speech sequences to text. The text representing the user' speech is then submitted to the system;
- ***Convert user speech to command*:** This process takes the text representing the user's speech and converts it to an executable command or query;
- ***Pronounce responses*:** after processing a request, the system sends the response to the user. By receiving the response, the speech synthesis subsystem pronounces the response to the user. To make this possible, the system rely on the «Generate natural language» capability to convert the response in the format required by the Speech Synthesis subsystem;
- ***Make suggestion*:** Knowing the user preferences, task and model combining with the user request, the system can make some suggestions to user. The machine learning capability is used for that purpose;
- **Learn User Model:** the system learns the model of the user with which it cooperates in order to better serve him;
- **Learn User Task Model**: the system learns the task of the user with which it interacts in order to better serve him;
- **Update User Model:** In a request-response interaction, the system can update the user model. This update results in a better understanding of the user model occurring during a learning process;
- **Update User Task Model:** In a request-response interaction, the system can update the user model. This update results in a better understanding of the user task model occurring during a learning process.

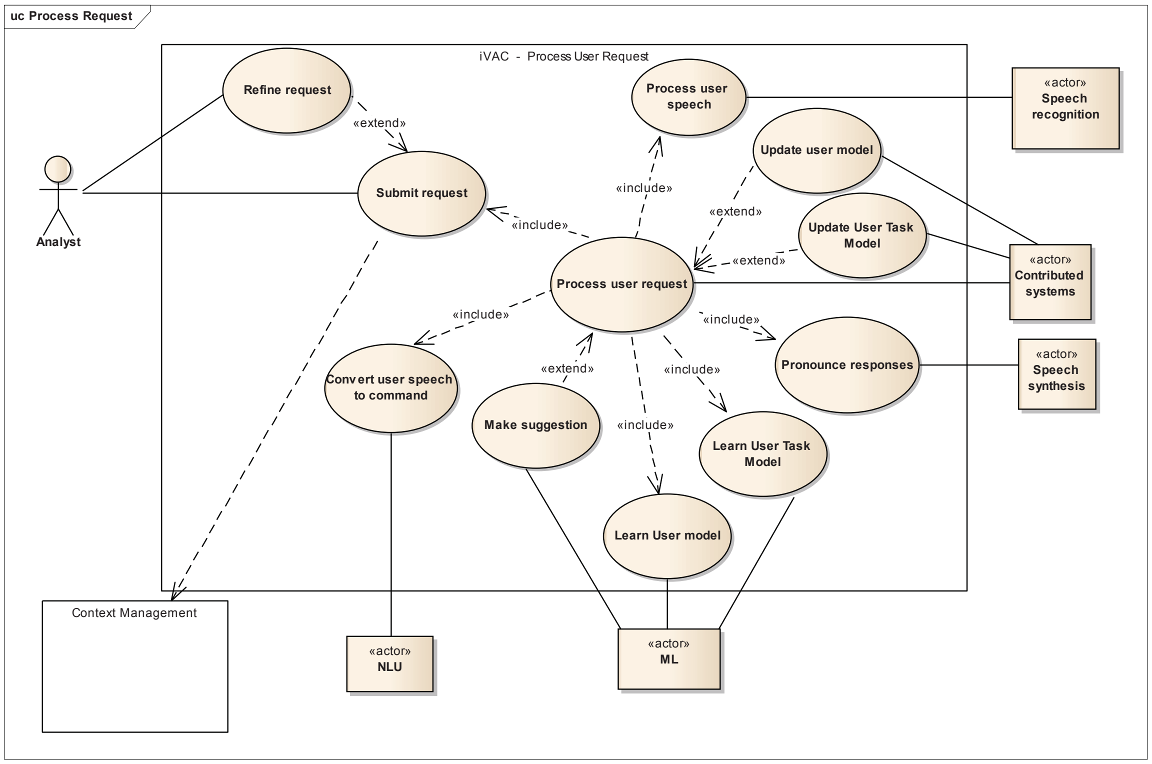


Figure 5: Process User Request Use Cases

4.2.2 Manage Context

The «Manage Context» use case supports those basic capabilities:

- ***Provide context*****:** The user provides or specifies his context.
- ***Refine context*****:** The user refines his context in order to facilitate the system learning.

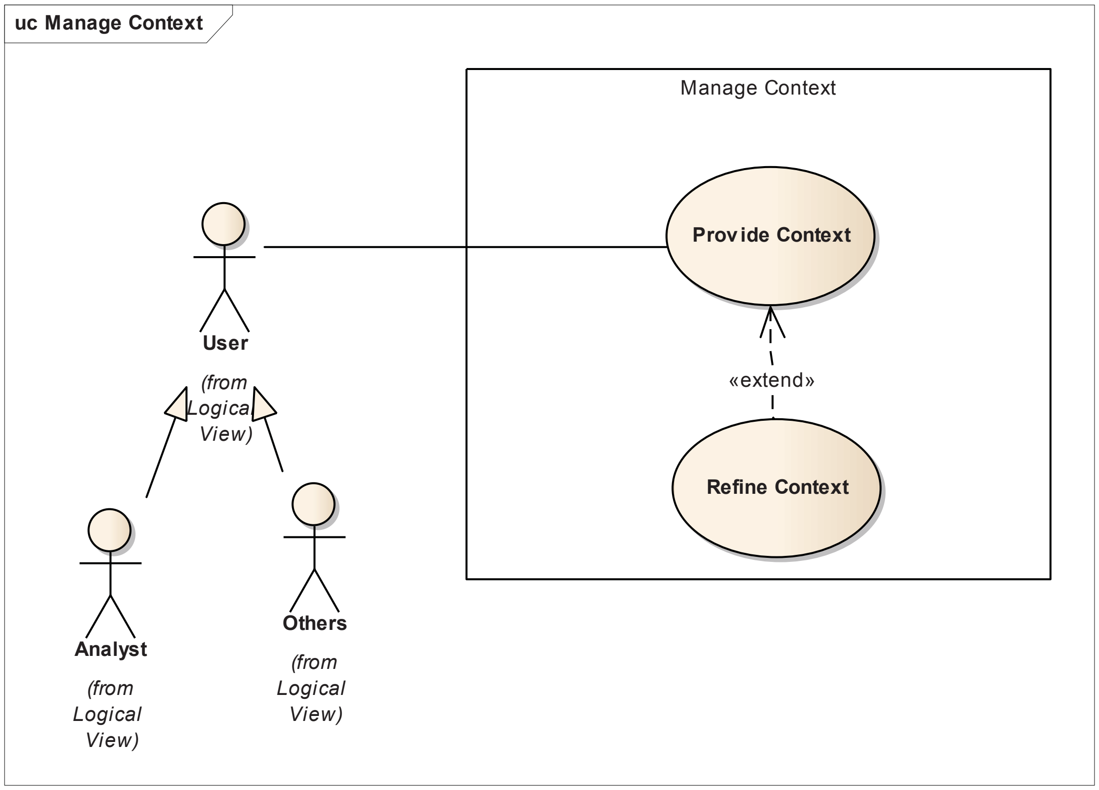


Figure 6: Manage Context Use Cases

4.2.3 Manage Feedback

The «Manage Feedback» use cases allow the user to provide feedback either on the request submitted to the system or on the response getting from the system. It can be broken down in the following system use cases

- ***Provide Feedback*****:** The user provides feedback to the system. This use case is a generic one. It is realized by:;
  - ***Provide Feedback on Request*****:** The user provides feedback to the system about a submitted request. The feedback improves the knowledge of the system about the goal of the user and by the way improves the quality of the response given by the system;
  - ***Provide Feedback on Response:*** The user provides feedback to the system about a response coming from the system for a given request. The feedback improves the knowledge of the system about the preferences of the user, his goal and by the way improves the quality of the response given by the system. This use case is an abstract one; it is realised by those more specialised ones:

- ***Select Response*****:** The user selects responses from the list returned by the system. He indicates which of the responses are relevant.
- ***Rank Response*****:** The user can rank the responses, endorse a specific response, select the most relevant response;
- ***Reject response*****:** the user provide feedback on the response and reject the response provided by the system;
- ***Save Response*****:** The user save the response for further use.

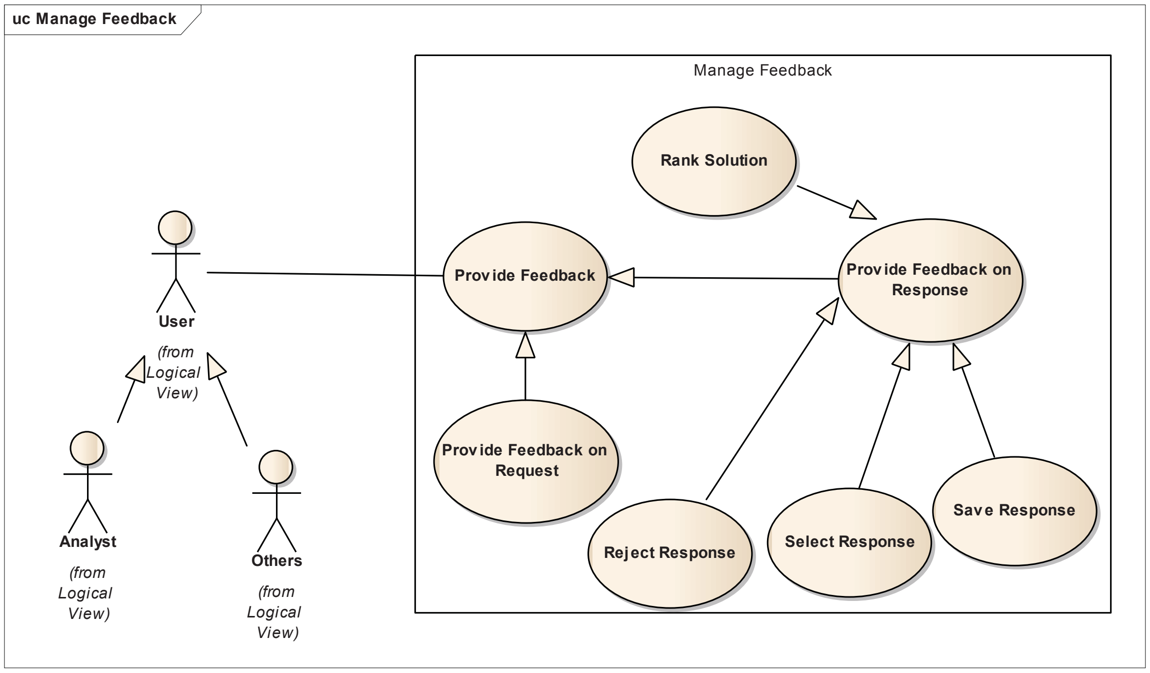


Figure 7: Manage Feedback Use Cases

5 Software System Architecture

This section presents the software architecture of the iVAC system. It presents the software components that compose the iVAC system and those with which the iVAC interacts to deliver its capabilities and services.

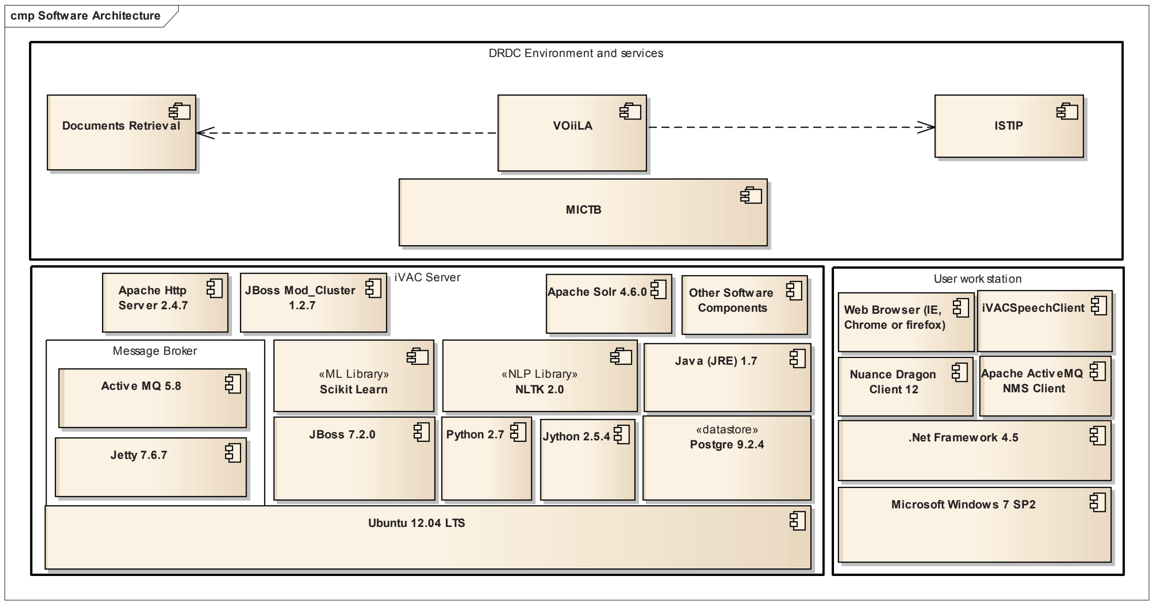


Figure 8: Software architecture

5.1 Description

The software architecture of the iVAC is composed of the following software systems:

1. **On the server:**

- **Machine learning library (to be specified):** this software library holds the computer learning capability of the iVAC. The «ML library to be mentioned explicitly» was chosen to bring machine learning capability to the iVAC;
- **Natural Language Processing (NLP) library:** for the iVAC, the «NLTK 2.0» [3] is used as the natural language processing library of choice;
- **Python 2.7**: Python is an interpreted language designed to speed up development time and rapid prototyping [4]. The Python interpreter is required by the NLTK components;
- **Jython 2.5.4**: Jython is Python for Java platform. It will be used to create and support a service adapter to integrate the NLU components;
- **Ubuntu 12.04 LTS:** Ubuntu Server is a Unix-like operating system [5]. It is the server operating system of choice of the DRDC. It is used on the server side of the iVAC system;
- **JBoss 7.1.1:** JBoss 7 is a fully certified Java enterprise edition 6 server application [6]. It is used to host the iVAC business component. Because it is a certified Java EE application server, it will provide key low level features to the system such as:

  - Multithreading capability;
  - Transaction management;
  - Session management;
  - Security management
  - Integration capability via Web services, messaging, database connector, Java connector Architecture;
  - Etc.
- **Apache Active MQ 5.8:** Active MQ is a popular and robust open source message broker and integration platform [7]. It supports many cross language clients and protocol [8] . It implements the full stack of patterns within the "Enterprise Integration Patterns" [9] via Apache Camel Library [10];
- **Jetty 9:** Jetty is an open source Java Servlet Container. As such, it provides web server and servlet hosting capability. It is the Web server that hosts the management console of Active MQ. Beside those core services, it support some integration features such as:
  - SPDY [11]
  - Web Sockets [12]
  - JMX (Java Management Extension) [13]
  - OSGi (Open Service Gateway initiative) [14]
  - JNDI (Java Naming and Directory Interface) [15]
  - AJP (Apache JServ Protocol) [16]
  - Etc.
- **Java Runtime Environment 1.7**: Java Runtime environment is an indispensable peace of software to run any Java-based programs. As such environment software systems like JBoss, Active MQ and Tomcat or Jetty cannot be run without Java RE;
- **Apache Solr** 4.6.0: is the popular, blazing fast open source enterprise search platform from the Apache Lucene project. Its major features include powerful full-text search, hit highlighting, faceted search, near real-time indexing, dynamic clustering, database integration, rich document (e.g., Word, PDF) handling, and geospatial search. Solr is highly reliable, scalable and fault tolerant, providing distributed indexing, replication and load-balanced querying, automated failover and recovery, centralized configuration and more. Solr powers the search and navigation features of many of the world's largest internet sites. As such it powers the search and navigation features to iVAC system;
- **Apache HTTP Server 2.4.6:** the Apache HTTP Server coupled with the JBoss Mod_Cluster 1.2.7 acts like a load balancer for the iVAC cluster. Its main purposes are to distribute the load between the nodes that form the iVAC Server Cluster

2. **On the user's computer**

- **Microsoft Windows 7 Professional**: Microsoft Windows 7 is the client version of Microsoft Windows operating system. The desktop version of that operating system will be deployed on the personal computer (PC) and/or Laptop of the users of the iVAC. It is the required operating system to run Nuance Dragon Client;
- **Microsoft** .Net Framework 4.5: .Net Framework is an execution environment for Microsoft .Net components, tools and framework. It will be deploy on both the client workstation and the Microsoft Windows Server;
- **Nuance Dragon Client 12:** Nuance Dragon is a software system with several capabilities among them: Speech Recognition and Speech Synthesis [17]. This software is mainly used for the mentioned capabilities;
- **ActiveMQ NMS Client 2.0**: ActiveMQ NMS client is a .Net client [18] that communicates with the ActiveMQ Message Broker using OpenWire[7].

[7] OpenWire is a cross language wire protocol to allow native access to ActiveMQ from a number of different languages and platforms; https://cwiki.apache.org/confluence/display/ACTIVEMQ/OpenWire

5.2 Business Services and Components

5.2.1 Functional View

This section presents a functional view of the iVAC system. Here is an overview that exhibits the main components of the system and their interactions.

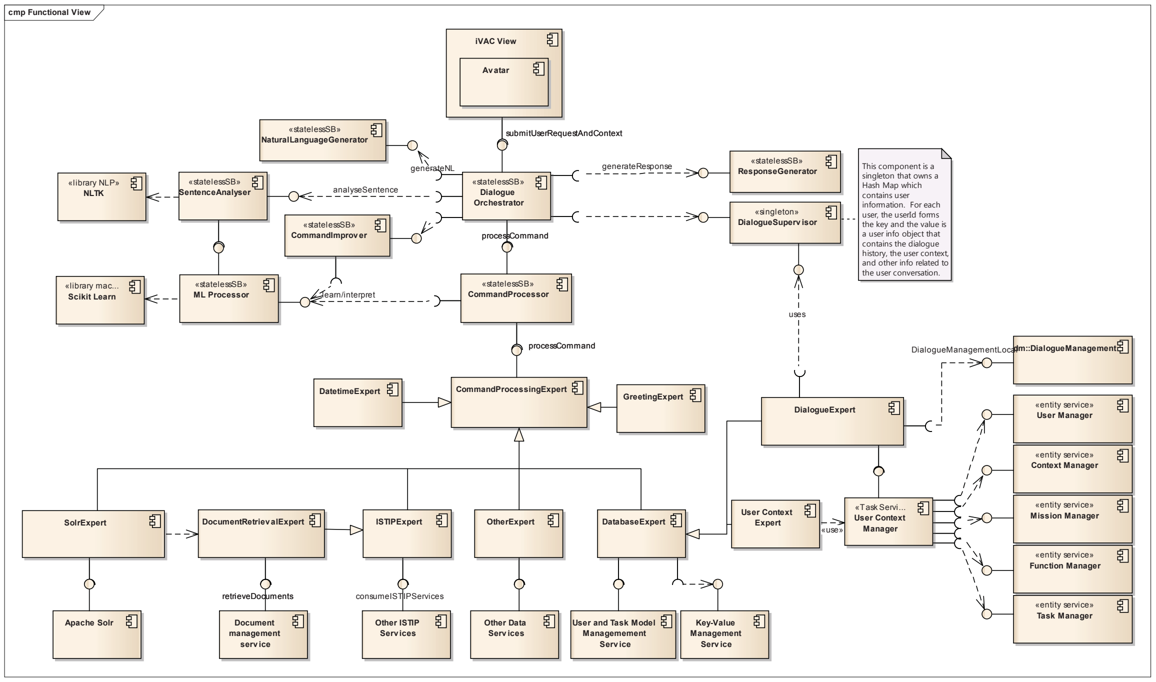

cmp Functional View
iVAC View
Avatar
submitUserRequestAndContext
«statelessSB» NaturalLanguageGenerator
generateNL
«library NLP» NLTK
«statelessSB» SentenceAnalyser
analyseSentence
«statelessSB» Dialogue Orchestrator
«statelessSB» CommandImprover
processCommand
generateResponse
«statelessSB» ResponseGenerator
«singleton» DialogueSupervisor
This component is a singleton that owns a Hash Map which contains user information. For each user, the userId forms the key and the value is a user info object that contains the dialogue history, the user context, and other info related to the user conversation.
«library mac... Scikit Learn
«statelessSB» ML Processor
learn/interpret
«statelessSB» CommandProcessor
processCommand
uses
DialogueManagementLocal
dm::DialogueManagement
DatetimeExpert
CommandProcessingExpert
GreetingExpert
DialogueExpert
«entity service» User Manager
«entity service» Context Manager
«entity service» Mission Manager
«entity service» Function Manager
«entity service» Task Manager
SolrExpert
DocumentRetrievalExpert
ISTIPExpert
OtherExpert
DatabaseExpert
User Context Expert
«use»
«Task Servi... User Context Manager
retrieveDocuments
consumeISTIPServices
Apache Solr
Document management service
Other ISTIP Services
Other Data Services
User and Task Model Managemement Service
Key-Value Management Service

Figure 9: iVAC's Functional view

As depicted in the functional architecture above, the system contains those main functional components:

- **iVAC view**: let the user interact with the system. This interaction can be made directly through the iVAC visual components (a web-based application) or through the iVAC speech capabilities provided by Nuance Dragon 12 Client (a windows-based application) [19]:
  - **Speech Recognition:** this component brings the capability to recognize speech (natural language discourse) to the system. Its task is to convert the user's spoken utterances to text.
  - **Speech synthesis:** this component gives the system the capability to translate text to speech (vocalize response the user). Its main task is to communicate the utterance generated by the response generator to the human dialogue participant using text-to-speech (TTS) technology;
- **Dialogue Orchestrator:** a central part of the system. It coordinates the activity of all components of the dialogue system, controls the dialogue flow, maintains a representation of current state of the dialogue and communicates with external applications. It acts as a mid-man between the multiple components that composing the system letting them to function as a whole.

It is the dialogue orchestrator which maps user commands to business functions and convert responses in such a way that it can be vocalized to the user;

- **Sentence Analyser (NLU[8] capable):** this component gives to the system the capability to understand user request made in a human spoken language. This component performs an analysis of the recognized utterances and produces a command that can be used by the dialogue orchestrator to issue request to core business functions. This component relies on the NLP Library to deliver its capabilities.
- **Natural Language Processing (NLP) Library[9]:** third party libraries that bring the capability to process natural language utterance to the system. The NLU component relies on such libraries to be able to understand user request;
- **Command Processor:** the command processor has a very good knowledge all the business functions that can be called to process a particular command. Then when it gets a command it forward it to right expert capable to process it;
- **Command Processing Expert:** This is an abstract command processing component. It provides a well-defined interface to the specialized expert. The actual command execution task is done by one of the specialized command processing experts :
  - **ISTIPExpert:** for tasks or commands related to services hosted in the ISTIP infrastructure. Depending of the type of the command, this component call the right ISTIP service to process the command;
  - **DocumentExpert:** for documents retrieval and/or processing commands. It relies on document processing system or services to process the command;
  - **GreetingExpert:** to greet the user, initialise the user session context;
  - **DateTimeExpert:** for date and time-related command;
  - **DataBaseExpert:** for querying database systems. It issues data query to its known databases. It relies on the interfaces and/or services provided by those database system;
  - **UserContextExpert:** for managing user context related tasks;
  - **SolrExpert:** for retrieving documents and/or similar user context in the Solr indexation system and by using Solr advanced search capabilities;
  - Other expert according the systems or services that need to be called to process a particular command;
  - Etc.
- **Response Generator:** the main purpose of the Response Generator is to aggregate all the responses returned by the command processing experts and select the most accurate response to send to the user. The nature of the request combined with the user context and task model guides the response selection process.

[8] Natural Language Understanding
[9] Either NLTK or OpenNLP can be used develop this component. In the case of NLTK, an adapter will be used to encapsulate this functionality

- **Natural Language Generator:** the task of the natural language generation module is to produce a textual utterance from an internal representation of the response delivered to the dialogue orchestrator. This component takes a response (text) and generates natural language format from it in order for the Speech Synthesis component to be able to pronounce the answer to the user;
- **Command Improver:** this component is used to improve a user command. To do this, the command Improver uses the user context, the user task model and the dialogue history. This component is called by the Dialogue Orchestrator when it needs to get a more accurate command to be able to call core business functions correctly;
- **Discourse History Manager:** manage the history of the dialogue between the user and the system. It takes care of asked questions, their status and the responses provided to them;
- **Mission Manager:** this component provides entity services for managing mission related activities;
- **Task Manager:** this component provides entity services for managing task;
- **Function Manager:** this component provides entity services for managing user functions;
- **User Manager:** this component provides entity services for managing users;
- **Context Manager:** this component provides entity services for managing user context;
- **ML Processor:** this command brings machine learning capability to the system. It depends on machine learning library to deliver its capabilities.
- **ML Library[10]:** third party libraries used for the machine learning capability. The machine learning processor relies on such libraries. Depending on the nature of the machine learning library, an adaptor will be used to integrate it in the system.

[10] The ML Library is not known at this time (date: 2013-06-06)

5.2.2 User - iVAC Interaction Overview

The purpose of the iVAC system, as already mentioned, is to assist the user in his daily dirties. The interaction between a user and the iVAC occurs in the following steps:

1. At the beginning, the system is waiting for user request;
2. The user submits a request;
3. The system analyses the request to see if it is a command or query;
4. If the user request is a command, the system executes the command:

- In case of an exit command, the system ends the conversation and cancel the user session;
- If it is a greeting command, the system greets the user.

5. If the user request is a query, the system determines if it is a follow-up query

- If the query is not a follow-up query and it is well understood, the system processes the query and returns the result to the user;
- If the query is an elliptic fallow-up query, the system asks the user to reformulate his query. A maximum attempts is defined to protect the system against infinite query reformation;
- If the query is an anaphoric follow-up, the system tries to reformulate it by replacing references with name entities already discovered in previous query and/or responses:
  - If multiple reformulations are found, the system asks the user which query he wants to ask and execute this query;
  - If one reformulation is found, the system executes that query and returns the response to the user;
  - If the system can't reformulate the query, it asks the user to reformulate the query. A maximum attempts is defined to protect the system against infinite query reformation:
    - In this case, if the maximum authorized attempt is reached, the system asks the user for a new request.

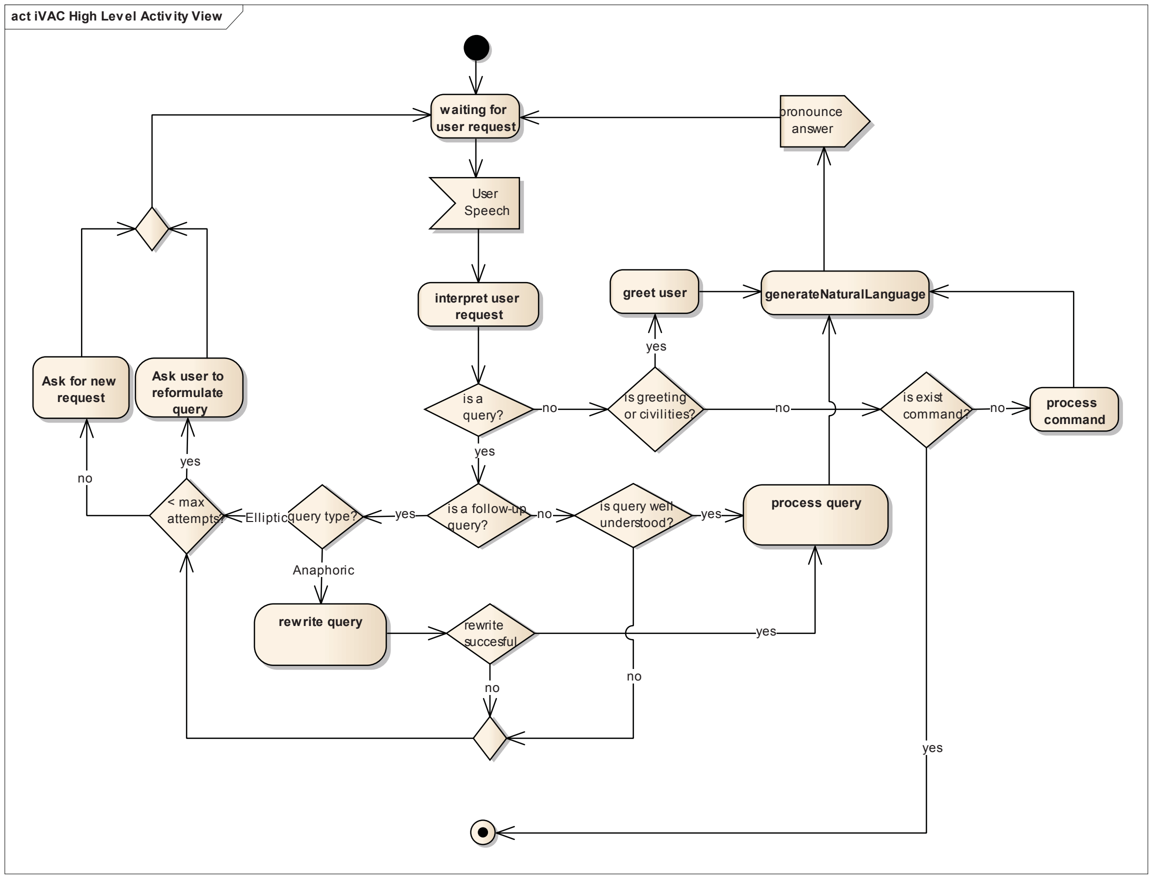
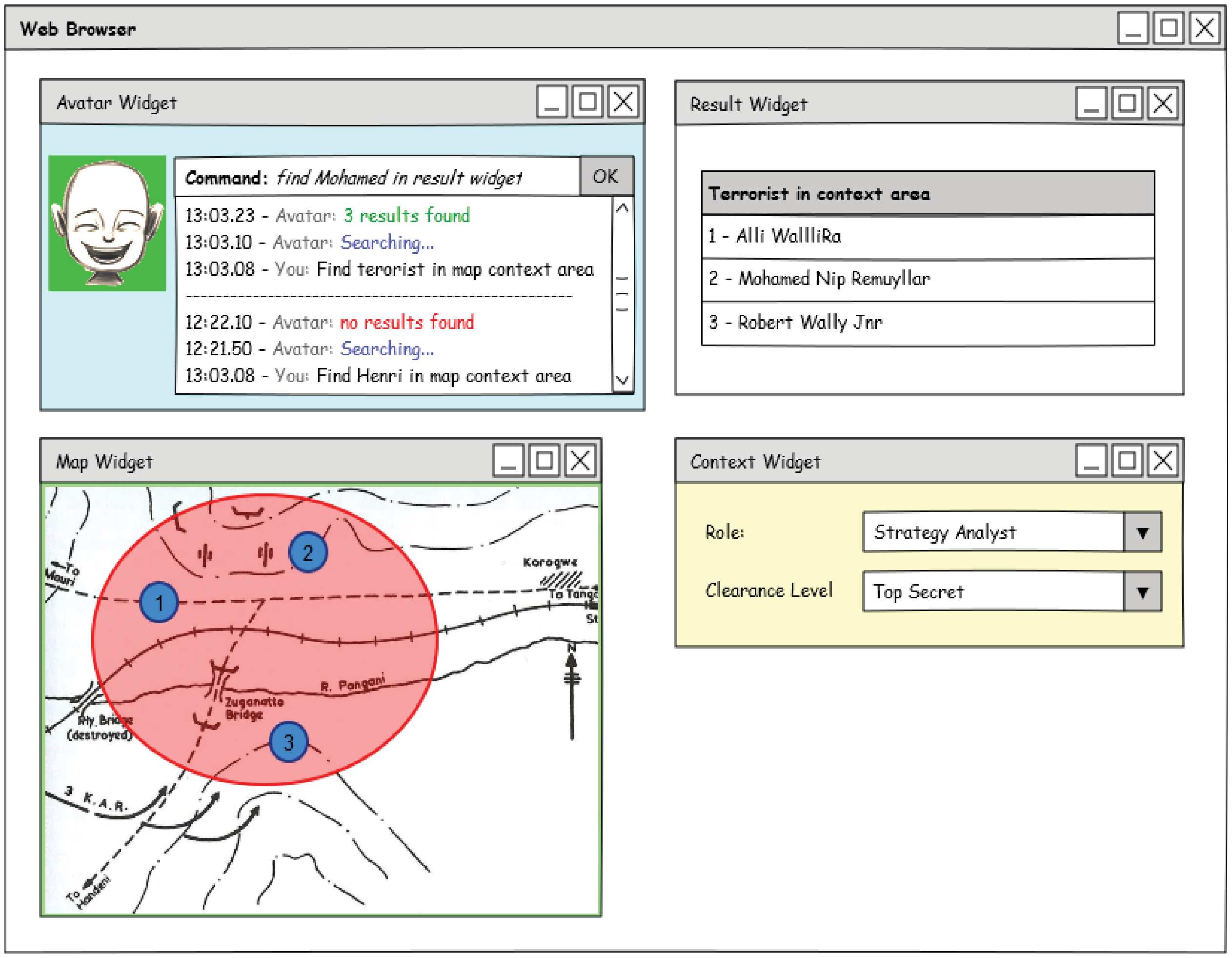


Figure 10: iVAC - Dynamic view

5.3 User Interface Components

This section describes the strategy and patterns used to develop the iVAC visual components.

5.3.1 General Structure of the User Interface



Command: *find Mohamed in result widget*

13:03.23 - Avatar: 3 results found
13:03.10 - Avatar: Searching...
13:03.08 - You: Find terorist in map context area

12:22.10 - Avatar: no results found
12:21.50 - Avatar: Searching...
13:03.08 - You: Find Henri in map context area

Terrorist in context area
1 - Alli WallliRa
2 - Mohamed Nip Remuyllar
3 - Robert Wally Jnr

Role: Strategy Analyst

Clearance Level: Top Secret

Figure 11: Structure of the UI

As depicted in Figure 11, the user interface of the iVAC is a container of widgets organized as follows:

- A main window, like a dashboard, where other iVAC visual components will be displayed. It contains:
  - A chat box in which the conversation log is displayed. This widget contains also:
    - A text box where text representing the user utterance, as translated by the speech recognition module, will appear. This text box can be used by the user to submit his requests directly to the system;

    - Beside the text box, a command button. This button is used when the user wants to submit a text request to the system. The user enters his request in the text box and then presses the button to submit his request to the system;
    - The avatar;
- Other visual components such as:
    - a widget to display the results
    - the analysis components;
    - etc.

5.3.2 Patterns Overview

Building a robust, scalable, reliable, maintainable system requires the use of some established and well-known design patterns.

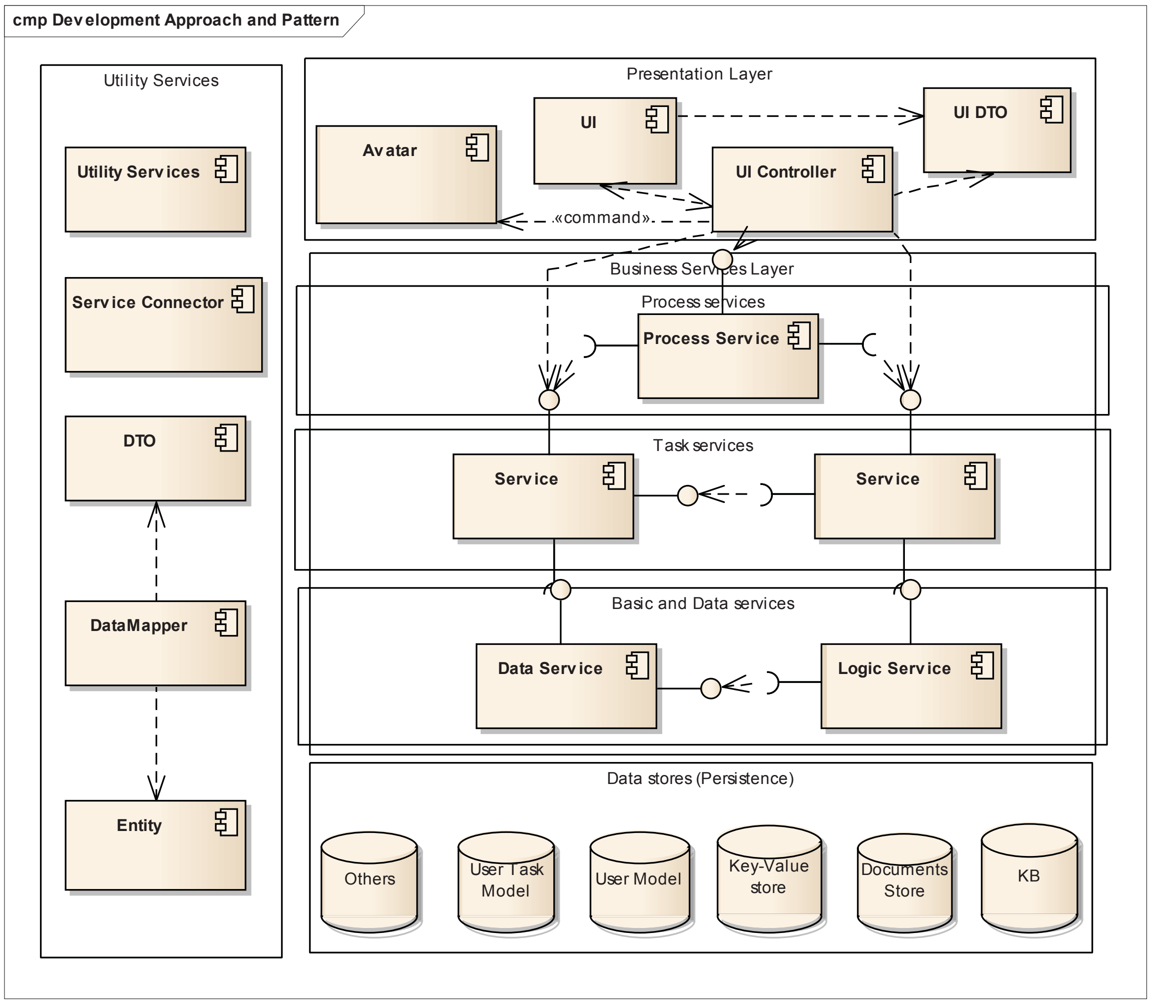


Figure 12: Development Strategies and Design Patterns

- **Model View Controller (MVC):** The widgets are built using the MCV design pattern. This pattern separates the representation of data from the user's interaction with it:
  - ***View*:** The View is responsible to organise the data and to interact with the user using GUI controls (lists, buttons, etc.). The View uses the Model to get or set data. In the context of the iVAC, an Avatar will also be part of the view;
  - ***Controller*:** The controller interprets the GUI interactions with the user, informing the Model and/or the View to change as appropriate;
  - ***Model (UI DTO)*:** The widget uses a Model that is a lightweight representation of the data it requires. The Model uses the JavaScript Object Notation (JSON) format. The classes that are part of the Model and

are organized based on the widget needs only. This lightweight representation ensures the responsiveness of the user interface.

- ***Business Service Layer:*** A Service Layer defines a software system boundary with a layer of services that establishes a set of available operations and coordinates the system response in each operation. The Service Layer is, in itself, a layered architecture that structures and organizes the services according to SOA best practices and patterns:
    - **Data *or Entity Services*:** encapsulate and provide information service in the system. This kind of services mainly offers CRUD (Create, Read, Update and Delete) operations on the data served by the system;
    - ***Logic Services*:** provide basic logic processing or algorithmic service in the system. For example, a service to provide current date and time can be seen as a basic service;
    - ***Task Services*:** provide task centric services in the system. A task service can require services from other task, data and/or logic services to deliver its capabilities. For example, the NLU service is a task service;
    - ***Process Services*:** provide process centric services in the system. Process services orchestrate the execution of several services to response to requests coming from users or other system components. For example, the «Process user request» service is a process service.
- ***Utility services*:** provide some cross layer concerns services in the system, such as:
    - ***Data Mapper*:** A Data Mapper is used by the widget to transform the structure of the Data Transfer Object (DTO) into the Model it requires to operate. It is also used to transform back to Model into the DTO when the services have to be invoked;
    - ***Data Transfer Object* (DTO):** The services expose their data using Data Transfer Object. A DTO is an object that carries data;
    - ***Service Connector*:** The Service Connector prevents duplicating the code required to use a specific service. It prevents different widgets to duplicate the code they require to call the same service;
    - ***Logging Services*:** provide logging capabilities in the overall system;
    - ***Security Services:*** provide security management capability in the overall system.
- ***Separated Interface*:** Separated Interface is a design pattern and best practice which promotes the separation between a service interface (the service) and its implementation into two distinct software (packaged) components. This pattern facilitates loose couple between a service consumer (the client) and the service implementation. Every service built in the context of the iVAC follows that path.

5.4 Future iVAC Components

In this section, we present the strategy for integrating future iVAC components.

5.4.1 Business Components and Services

It would not be difficult to integrate any future components designed and developed for the iVAC. This integration can be done in multiple ways:

- Through the component's business interface (local or remote, preferably local for performance);
- Through web service facilities if the component provides a web service interface;
- By using messaging capability provided the message broker if the component supports messaging communication patterns;
- If the component does not provide a suitable interface and if the component does not support messaging, a service adapter will be used to integrate the component.

Even though multiple integration approaches are possible for the integration of future iVAC components, we recommend that the development of any future iVAC components follows the same construction strategy and patterns used to develop the iVAC actual components.

5.4.2 UI Components

In the future, the iVAC will need to integrate several visual components among of them the analysis components. To be integrated in the iVAC, those components need be web-based components. Because the iVAC is, by nature, a web application, it is very important that any visual component developed to integrate into the iVAC is web-based. To facilitate smooth and straight forward integration to the iVAC, those visual components should not imbed or make use of any ActiveX components.

5.5 ISTIP Components and Services

ISTIP is a SOA-based services infrastructure. Most ISTIP components and services are developed with the JEE technologies stack. Generally, those services provide a local, remote and web service view or communicate through messaging capabilities provided by the application server that hosts them. Then, integrating ISTIP components and services with the iVAC may not be challenging. The integration can be done through:

- The component's remote view (interface);
- The web service view provided by the service;
- Messaging facilities (when available).

5.6 VOiiLA Interaction Mechanisms

VOiiLA[11] is web-based application composed mostly of widgets building blocks that interact with each other by sending message using publish and subscribe mechanisms. Each widget handles and triggers events. Those events are managed by OWF[12] so that each widget is independent of each other's.

The iVAC visual components are web-based developing with the same technologies stacks than VOiiLA components. Then, the iVAC will be able to use directly VOiiLA components and the visual components developed for the iVAC system will be able to integrate into VOiiLA.

[11] VOiiLA = Visionary Overarching Interaction Interface Layer for the Analysis
[12] OWF = Ozone Widget Framework

5.7 External Components

The iVAC is designed in such a way that it will be able to integrate other components and/or services. This integration can be done in multiple ways:

- If the component provides a compliant interface, it will be used as it is integrate that component;
- If the component provides a web service interface, web service facilities will be used to integrate it;
- The messaging capabilities provided the message broker and its supported protocols can be used to integrate components that support messaging communication patterns;
- If the component does not provide a suitable interface and if the component does not support messaging, a service adapter will be used to integrate that component;
- If any of those strategies cannot be used, the Java Connector Architecture (JCA) facilities will be used to make the integration possible.

6 Information Model

This section presents the information model used to deliver the iVAC capabilities. The structure of the information exchanged between the services is also documented in this section.

TBC.

7 Deployment Strategy

7.1 Description

In its conception, the iVAC system is built as a distributed service-based SOA application based mainly on Java EE 6 technologies stack. There are two different ways such an application can be deployed:

1. Clustered deployment with collocated components architecture in which the business components (EJB) and the components (UI) are deployed on the same physical server;
2. Distributed deployment architecture.

7.1.1 Clustered Deployment with Collocated Components Architecture

A clustered architecture is an architecture in which several physical servers are linked together such a way that they work as single unit. By using a collocated architecture, the business components (EJB technology components) and the web components (UI components) are deployed on the same physical server. Together, they form that is called an Enterprise Application. This is possible because of the nature of a Java EE application server. In such deployment architecture, the web components access the business components through their local view (interface) that speed up the execution time and then the overall performance of the application.

Most Java EE applications are deployed such a way. When the load becomes important, other physical server can be added to the cluster. If a server failed, the work is dispatched to other server. The failed server can then be removed from the cluster for repair.

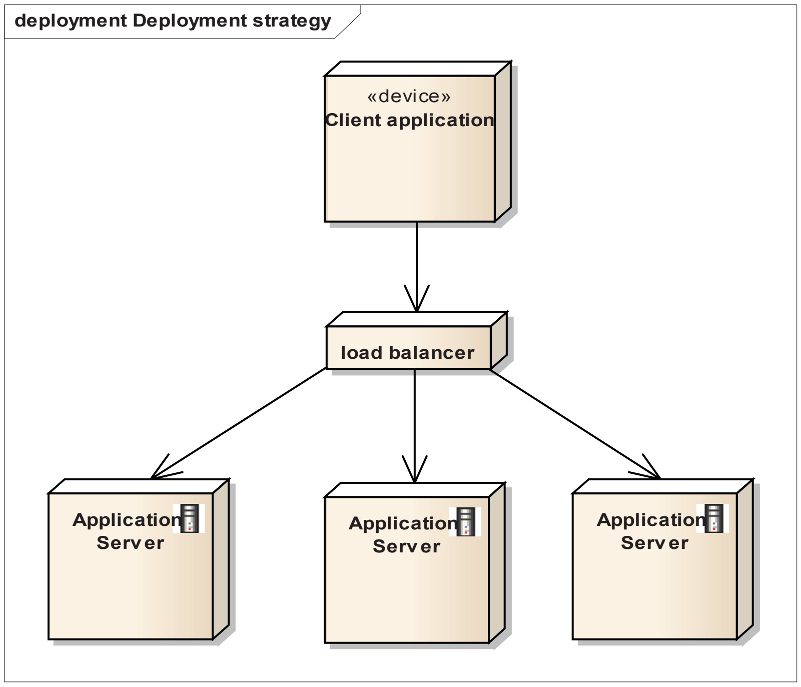


Figure 13: Java EE - Clustered deployment with distributed components architecture

7.1.2 Distributed Deployment Architecture

In distributed deployment architecture, the web components (UI) and the business components (EJB) are deployed in their respective physical server. In such deployment architecture, the web components access the business components through their remote view (interface) with the cost of a network access.

Generally, the two parts of such architecture are clustered: a cluster for the EJB components another cluster hosts the web components. This distribution architecture is the most complicated Java EE deployment architecture. Consequently, it is not used very often. It is only required for applications that need to support several thousand of concurrent users where each user can have with several active transactions. This architecture is also recommended for applications that need to be available across many countries and continents.

This distribution architecture brings more security to the overall system because each part of the system can be isolated from the other and firewall can be placed in front of each cluster. The overall performance of that system can be increased by adding more servers on each cluster.

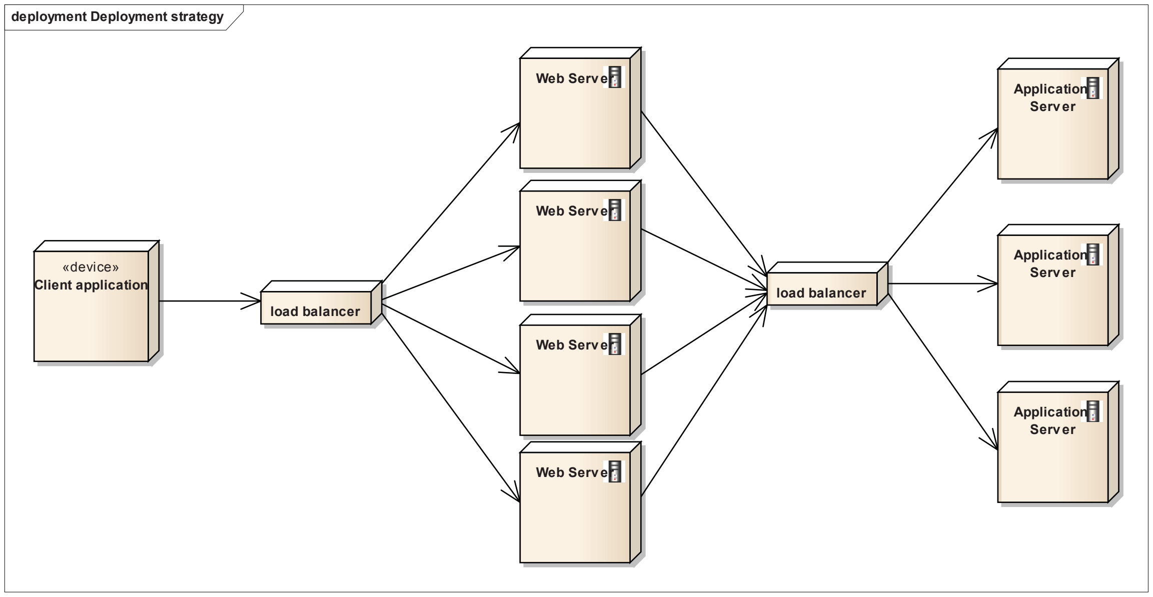


Figure 14: Java EE - Distributed deployment Architecture

7.1.3 iVAC Deployment Architecture

Considering the nature of the iVAC system and the number of concurrent users it will need to support, we have chosen the clustered deployment with collocated components architecture to deploy the system. But the system is designed such a way that it can be easily deployed in distributed deployment architecture. Each component that composes the system will offer a remote and a web service view beside the locally visible view.

7.2 Deployment Model

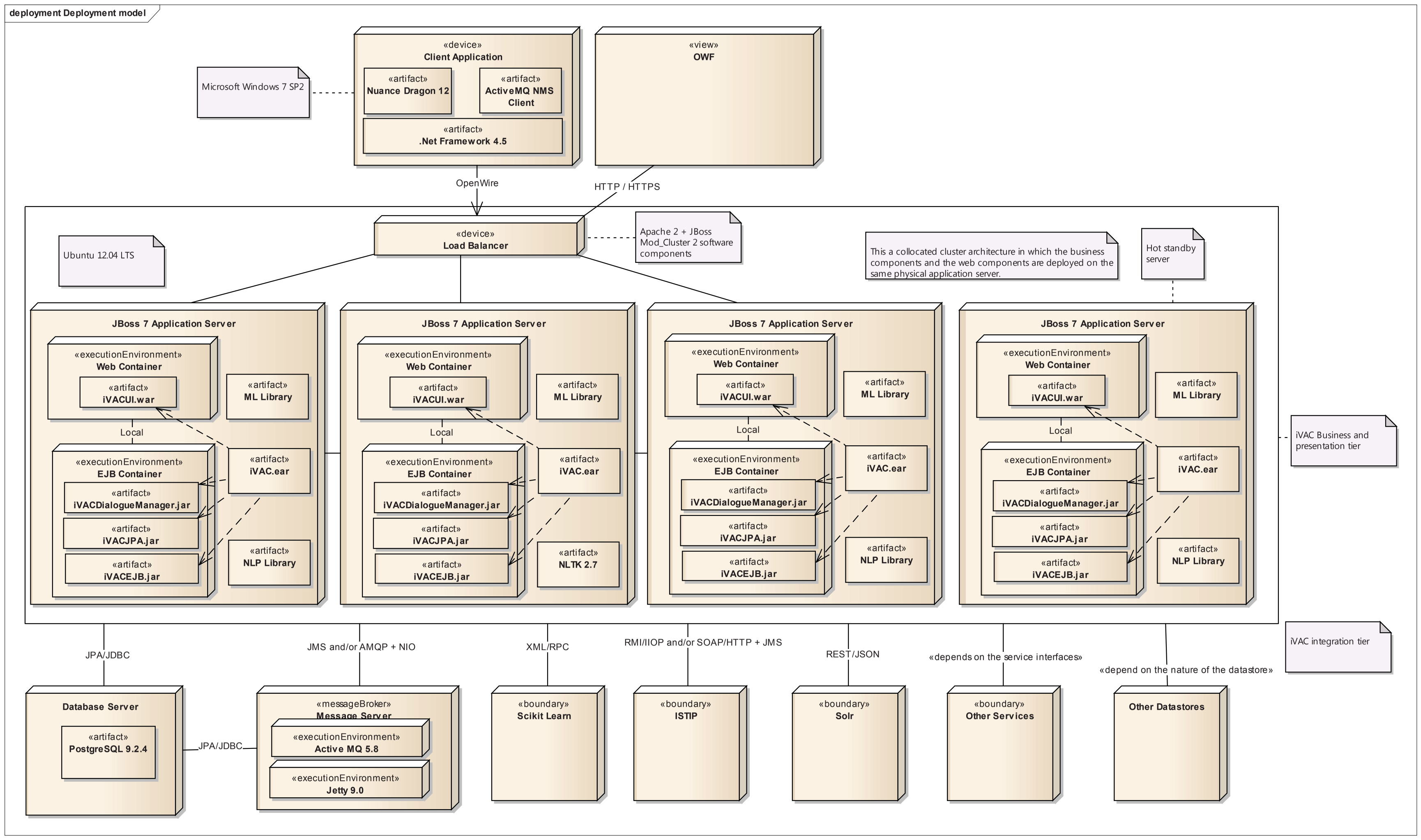


Figure 15: Deployment model

7.2.1 Description and Considerations

7.2.1.1 Server

As the deployment model suggests, we have chosen a clustered with collocated components deployment architecture in which the iVAC UI Components and the business component are hosted in the same physical computer even if each different kind of components (UI and business) will be served by different parts of the application server (the Web container and the EJB container).

A clustered architecture guaranties the server can scale enough to bring acceptable performance and low latency to the system. In case of important load, additional server computer can be added to the cluster to better service the users and lower the response time.

Even if a message broker is already embedded in the application server, we have chosen to separate the messaging concerns from the business services processing concerns within the application server by deploying a standalone message broker. By doing this, the overall system will become more reliable and more available; the performance will be better.

Ubuntu 12.04 LTS is the operating system deployed on the server.

7.2.1.2 User's Computer

On the computer of each user, the following software components will be deployed:

- Microsoft Windows 7 SP2;
- .Net Framework 4.5;
- ActiveMQ NMS Client;
- Nuance Dragon 12 Client;
- Windows-based application: this application is required because of Nuance Dragon;
- A web browser (Internet Explorer, Google Chrome, Firefox …).

8 Development Plan

This section presents the development plan for developing the main components of the iVAC prototype. The development will be performed using an agile methodology and components will be built iteratively through multiple sprints.

The following sections present a proposed set of capabilities to be developed during the first two sprints of the development phase. Other capabilities to be developed in the future are listed in the backlog section.

8.1 Sprint #1

- Preliminary web-based interface with (including their controllers):
  - Textbox for entering requests
  - Chat box
- Dialog orchestrator component.
- Sentence analysis component, using a simple bag of words approach for extracting commands.
- Command processor component.
- Document retrieval expert component. In this first iteration, it will call the Document Repository service of the ISTIP.
- Response Generator component, with basic capabilities.
- Natural Language Generator (NLG) component, with basic capabilities.

8.2 Sprint #2

- Integration of Nuance Dragon Client, if available.
- Sentence Analysis component. Addition of the analysis approach based on a grammar. Will integrate NLTK or OpenNLP external components.
- Web-based interface. Addition of the list of retrieved documents.

8.3 Backlog

- Develop additional Command Processing Experts.
- Develop required service adapters to integrate systems that require one.
- Develop the windows-based application;
- Integrate an avatar into the UI.
- Machine Learning Components.
- Dialog History Manager;
- Command Improver;
- Define user model and user task model;
- Database and Data services to save the user model

- Database and Data services to save the user Task model;
- Database and Data services to save user requests and responses if necessary;

9 Test Plan

The test plan for the development of the iVAC prototype is detailed in a separate document. See reference [20].

References

[1] "Human-computer interaction with an intelligence virtual analyst ", D. Gouin, V. Lavigne, and A. Bergeron Guyard. In Proceedings of Knowledge Systems for Coalition Operations 2012, Pensacola, FL, USA, 2012

[2] "A Survey of Available Technology and Recommendations for Building an iVAC Capability", Fujitsu Consulting (Canada) Inc., W7701-135551, 2013.

[3] NLTK website: http://nltk.org/

[4] Python website: http://www.python.org/

[5] Ubuntu Server website: http://www.ubuntu.com/server

[6] JBoss Application Server website: https://www.jboss.org/jbossas/

[7] ActiveMQ website: http://activemq.apache.org/

[8] ActiveMQ languages and protocols: http://activemq.apache.org/cross-language-clients.html

[9] Enterprise Integration Patterns: http://www.enterpriseintegrationpatterns.com/toc.html

[10] Apache Carnel library website: http://camel.apache.org/

[11] http://en.wikipedia.org/wiki/SPDY

[12] http://www.websocket.org/

[13] http://fr.wikipedia.org/wiki/JMX

[14] http://en.wikipedia.org/wiki/OSGi

[15] http://en.wikipedia.org/wiki/Java_Naming_and_Directory_Interface

[16] http://en.wikipedia.org/wiki/Apache_JServ_Protocol

[17] Nuance Dragon: http://www.nuance.com/dragon/index.htm

[18] .Net Message Service API: http://activemq.apache.org/nms/index.html

[19] Nuance Dragon 12 Client: http://www.nuance.com/for-developers/dragon/index.htm

[20] "iVAC Prototype Test Plan", Fujitsu Consulting (Canada) Inc., W7701-135551, 2013.

Annex A – Supplementary Definition Details

List of symbols/abbreviations/acronyms

AJP	Apache JServ Protocol
API	Application Programming Interface
CRUD	Create, Read, Update and Delete
DL	Dialog Manager
DRDC	Defence Research & Development Canada
DS	Dialog System
DTO	Data Transfer Object
EJB	Enterprise Java Bean
ISTIP	Intelligence Science and Technology Integration Platform
iVAC	Intelligence Virtual Analyst Capability
J2EE	Java 2 Enterprise Edition
Java EE or JEE	Java Enterprise Edition
JAX-RS	Java API for RESTful Web Services
JAX-WS	Java API for XML Web Services
JCA	Java Connector Architecture
JNDI	Java Naming and Directory Interface
JSON	JavaScript Object Notation
LTS	Long-Term Support
ML	Machine Learning
MVC	Mode View Controller
NLG	Natural Language Generation
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NLU	Natural Language Understanding
OWF	Ozone Widget Framework
R&D	Research & Development
SAD	Software Architecture Description
SIIP	Self-Improving Inference Prototype
SR	Speech Recognition
SS	Speech Synthesis
TTS	Text To Speech
UI	User Interface

UML	Unified Modeling Language
URI	Unique resource identifier
VOiiLA	Visionary Overarching Interaction Interface Layer for the Analysis
WSDL	Web Service Description Language
WS-Policy	Web Services Policy Framework specification
XML	Extensible Markup Language
XSD	XML Schema Definition

Table 1: List of symbols, abbreviations and acronyms

Glossary

Table 2: Glossary