Enabling an Intelligent Natural Language Based Hub for the Deployment of Advanced Semantically Enriched Multi-channel Mass-scale Online Public Services

Project Number IST-2002-507967

D73 – Final Guidelines
Version: 4
29/03/2007
ABSTRACT
This deliverable is a final document devoted to provide, within the context of a general description of the project, user-friendly guidelines able to describe the HOPS platform, its functionalities, services and usability.
This document has been drawn up as a result of the analysis of the previous deliverables, namely D22 and 23, D31 and D51 and 52

KEYWORDS
WP7, general project description, conclusions, platform guidelines (features, use)
# TABLE OF CONTENTS

1 INTRODUCTION ..................................................................................................................4

2 THE PROJECT DESCRIPTION ............................................................................................6
   2.1 BACKGROUND ............................................................................................................6
   2.2 PURPOSE, OBJECTIVES, CONSORTIUM ROLES AND ACTIVITIES .........................6
   2.3 THE PROJECT RESULT: THE HOPS PLATFORM .............................................................8
   2.4 THE VALIDATION OF THE PLATFORM: THE TRIALS .................................................8

3 FINAL GUIDELINES .............................................................................................................10
   3.1 THE PLATFORM MAIN COMPONENTS .........................................................................10
       3.1.1 Platform layers model ..........................................................................................10
   3.2 FUNCTIONALITIES ......................................................................................................12
       3.2.1 Functionalities that the technological infrastructure must support: ...12
       3.2.2 Functionalities that the toolkit must support .......................................................12
   3.3 REQUIREMENTS .........................................................................................................13
       3.3.1 Hardware and software specifications .................................................................13
       3.3.2 User requirements ................................................................................................14
   3.4 TECHNOLOGIES .........................................................................................................15
       3.4.1 HOPS software architecture ..............................................................................15

4 CONCLUSIONS ..................................................................................................................18

5 REFERENCES ..................................................................................................................19

6 APPENDIX A – USER GUIDE ...........................................................................................20
1 INTRODUCTION

This document outlines the guidelines related to the HOPS platform. It is based on the final results assessment carried out at different level by the project consortium (trail evaluation, user requirements needs, stakeholders interaction, etc).

The document is a derivative of two previous deliverables: D22 and 23, D31 and D51 and 52.

D22 and 23 presented the general framework for the approach for the successful integration of user requirements\(^1\), and a first set of user requirements that defined some sample services to be developed and demonstrated within the lifetime of the project.

D22 presented the N-Services framework, which defined the steps that need to be taken to generate and deploy new information and transactional services. During this analysis the bottom-line user requirements began to take shape with a first set of requirements for sample services emerging from the results of the first prototype. The analysis made it clear that it was necessary to refocus on user requirements and refine them so that they would drive the development of the second prototype, improving the sample services.

D23 closes the specification cycle with section 1 providing a clear definition of the system based on the platform model, the service creation framework, the sample service used in the iterative approach and the specification of the bottom-line platform requirements.

Section 2 contains the specification of the platform user requirements, an analysis of the driving user requirements, the definition of a layer model-based platform, the main platform features explained, and a list of the bottom-line platform requirements from top to bottom.

Section 3 presents a refined specification of the N-Services use case model from which the service creation framework was assembled. This section is complemented by the remaining use cases that relate to the N-Services available in the annexes of D22. The refinement of the sample services stated in section 4 complements the specification of the sample services in D22. These requirements mainly address the needs, as expressed by users, to have a more flexible service, to be able to better understand open utterances of the users, and with a dialogue system that is less system-driven. Finally, the specification of the bottom-line platform requirements gathers in section 5 the specification of the key components and functionalities, which have been previously stated in section 2.

D51 and 52 are related to WP5 focused on the onsite local development and the deployment of trails.

D51 provides a description of the activities needed to deploy the HOPS platform within each of the three cities involved in the project.

---

\(^1\) A costumized version of the Rational Unified Process (RUP)
D52 provides a technical description of the process carried on to integrate the results of the first prototype in each city.
2 THE PROJECT DESCRIPTION

HOPS-Enabling an Intelligent Natural Language Based Hub for the Deployment of Advanced Semantically Enriched Multi-channel Mass-scale Online Public Services- is a three-year project funded by the Sixth Framework Programme for Research and Technological Development- FP6. The project was coordinated by the Municipality of Barcelona and it started on 1st January of 2004. It has involved 13 Partners from 4 Countries, including 3 Universities, 3 Industrial partners, 3 Local integrators, 3 Cities and 1 Innovation center.

The main goal was to provide wider and easier access to e-Government services reducing the gap between what can be achieved with voice interaction with respect to what can be done with text interaction. To achieve this goal, the project focused on the deployment of advanced ICT enabled “voice-front-end public platforms” enabling access of European citizens to their nearest Public Administration. The overall architecture also supported additional non-voice channels, having access to the Natural Language Processing and Semantic Web dimensions. In order to validate its results, during the last two years, HOPS produced fully functional prototypes tested and validated by the three Local Authorities involved in the project.

2.1 BACKGROUND

The HOPS background leverages on statistics demonstrating that citizen prefer to contact organizations by telephone: sixty per cent of telephone calls are requests for routine information or involve simple transactions, while only 30% calls involve processing of public services. Added to this, citizens form an opinion about administration efficiency based on how their telephone call is handled.

In Europe call centres solution are largely adopted, but their labour costs are often too much expensive to be afforded. The better solution seems to be integrating interaction self-service channels (web and telephone) and human agents.

The Natural Language Systems was chosen by the HOPS consortium as a good approach to self-service because it could make current e-government services, that presently suffer from with important performance shortcomings, more efficient. Additionally, the voice mode improves web usability, especially for people with different types of impairments (visual, physical or cognitive).

Other advantages of the language modes are that they are user-friendly and easy to use, they can support different types of interactions, they are able to tackle complex questions whose answer will require processing information from different sources and they are adaptable to different types of users (different languages, skills, ages and cultural sensitivity). As far as the deployment of voice channels related to on line public service is concerned, the HOPS challenge was to add new fully functional advanced technologies making it possible to deliver automated services without losing quality and further enhancing the current functionalities. The overall architecture was also able to support additional non-voice channels, as it accesses the Natural Language Processing and Semantic Web dimensions.

2.2 PURPOSE, OBJECTIVES, CONSORTIUM ROLES AND ACTIVITIES

As stated in the accepted and contracted proposal, the HOPS objective was to achieve the availability of a multi-channel, multi-modal and multi-device advanced hub that
could be integrated with the supporting systems and applications, thus enabling the launch of advanced and fully interactive Online Public Services. The aim was to enhance these particular functionalities, defining a kind of advanced front-end (hub) using the operational integration of innovative technologies and reconsidering the current system’s architecture and related organizational processes.

**Technological and research partners** performed the role of researchers, developers and system integrators. Their research was based on the integration of natural language technologies with semantic web technologies in the delivery of information and Online Public Services. In order to perform their tasks, an initial survey of state-of-the-art technologies and last cutting edge research in the field of voice portals, voiceXML, natural language processing and semantic web technologies was conducted at the beginning of the project.

Architecture Specification definition combined with Integration activity was conducted in each of the three main technological components: voice portals, natural language processing and semantic web technologies.

The partners involved were finally able to provide a prototype of the HOPS platform. Moreover, a platform prototype validation was performed using real pilots by the **three Public Administrations** involved in the project.

During the project, efforts were organised into seven different workpackages, namely: management, user requirements, enabling technologies, development and integration, on-site development and trials, evaluation and quality assessment and dissemination and exploitation. During the review process a change occurred: the European Commission established a 4 months extension (amending the duration from 36 to 40 months) and the end of project was set on 30th April 2007.

<table>
<thead>
<tr>
<th>Workpackage No²</th>
<th>Workpackage title</th>
<th>Start month³</th>
<th>End month³</th>
<th>Amendment</th>
</tr>
</thead>
<tbody>
<tr>
<td>WP 1</td>
<td>Management</td>
<td>0</td>
<td>35</td>
<td>End month: 40</td>
</tr>
<tr>
<td>WP 2</td>
<td>User Requirements</td>
<td>0</td>
<td>27</td>
<td>End month: 40</td>
</tr>
<tr>
<td>WP 3</td>
<td>Enabling Technologies</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WP 4</td>
<td>Implementation &amp; Integration</td>
<td>5</td>
<td>32</td>
<td>End month: 40</td>
</tr>
<tr>
<td>WP 5</td>
<td>On-site deploying and trials</td>
<td>12</td>
<td>35</td>
<td>End month: 40</td>
</tr>
<tr>
<td>WP 6</td>
<td>Evaluation &amp; Quality Assessment</td>
<td>0</td>
<td>35</td>
<td>End month: 40</td>
</tr>
<tr>
<td>WP 7</td>
<td>Dissemination</td>
<td>0</td>
<td>35</td>
<td>End month: 40</td>
</tr>
<tr>
<td>WP 8</td>
<td>Exploitation</td>
<td>18</td>
<td>35</td>
<td>End month: 40</td>
</tr>
</tbody>
</table>

² Workpackage number: WP 1 – WP n.
³ Relative start date for the work in the specific workpackages, month 0 marking the start of the project, and all other start dates being relative to this start date.
⁴ Relative end date, month 0 marking the start of the project, and all end dates being relative to this start date.
2.3 The Project Result: The HOPS Platform

The HOPS Platform is a set of services working together to provide new or adapted existent services to the user through different channels as voice or text, based on a modular approach to grammar development, dialogue management, and natural language understanding.

The HOPS platform provides two different interfaces to interact with its users: an interface based on voice and other interface based on text. Both interfaces share the use of Natural Language Processing to adapt the way to present the information to the user which is provided by the system and vice versa.

These interfaces have some specific characteristics related to their own nature. Whereas the voice interface is more accessible from the user point of view (phone vs computer), the text interface can hold a higher number of concurrent conversations.

2.4 The Validation of the Platform: The Trials

This paragraph provides a brief description of the activities needed to deploy the HOPS platform within each of the three cities involved in the project.

The deployment process was an essential phase of the project aiming at moving from a prototype version of the platform built in a laboratory, to a pilot deployed inside a real environment.

In the first trial cycles the local integrators started from their backend systems, and then built up the whole infrastructure needed to host the platform from there. This preliminary activity required an initial phase of analysis in order to identify the hardware and software requirements, as well as the most suitable configurations to host the first prototype release, taking into account the possible use of the services in the final trial.

Simultaneously, local integrators and technical partners moved forward with an analysis and the development of the adaptation of some of the first prototype components, in particular the linguistic resources, in order to have them fit within the local environments and their language constraints.

The final piece of work focused on the refinement of the services through specific tests in order to identify any errors in grammars and prompts or in the dialogue flow itself.

From a technological point of view, the general aim of the trial was to integrate the natural language processing technologies and the semantic web only by adding a suitable interface or pre-processor.

The convergence among different technologies and the efforts to make them work together would allow the partners to validate the assumption that a systemic integration of a voice portal, natural language processing and semantic web technologies can facilitate Public Administrations to obtain automated, high-quality and fully flexible ways of deploying new public services.

The three EU municipalities, Barcelona, Turin and London Borough of Camden municipalities, tested the prototypes in different conditions. The City of Turin in collaboration with its local partners, Loquendo, CSI and CSP decided to integrate in the first prototype architecture a component that was initially foreseen to be used only in following prototype, to support the mixed initiative in the dialogue with the user.

This component, named Spoken SubDialogue System, was installed inside the server hosting VoxNauta. The City of Turin decided to use the VOIP network as phone...
interface. This choice was suggested by the possibility to avoid the high costs of telephone board and in particular to the fact to use the VOIP network installed inside Turin Municipality.

The integration of the first pilot in Barcelona was based on an existing solid backend in the Barcelona municipality. The Barcelona backend was currently offering Internet and telephone (via operators) services to citizens. For the first prototype on-site deployment some changes were introduced in order to guarantee the adaptation of the prototype to the backend systems.

The Camden installation did not use a telephone board. The Instead Asterisk 1.0.7 Voip client was used and installed on a Debian Sarge (3.1) Linux server. This routes called to Voxnauta which in turn talked to the HOPS instance. The platform was then able to talk to the Camden Aplaws instance, querying its Oracle database to return Events information for the Cultural Agenda dialogues.
3 FINAL GUIDELINES

This chapter aims at providing general guidelines in order to exploit the HOPS platform, by defining the IT infrastructure and its applicability within different public authorities contexts.

The lack of solutions allowing local authorities to enhance the usability and accessibility of online public services using new channels is the main reason to investigate the HOPS platform applicability. The platform enables an intelligent natural language based hub for the deployment of advanced semantically enriched multi-channel mass-scale online public services.

3.1 THE PLATFORM MAIN COMPONENTS

The HOPS Platform is a set of services working together to provide new or adapted existent services to the user through different channels as voice or text, based on a modular approach to grammar development, dialogue management, and natural language understanding.

The HOPS platform basically consists of the technical infrastructure and a toolkit. The toolkit facilitates the creation, management and monitoring of the e-government services deployed using the platform.

The technological infrastructure includes the server types, phone boards, phone and internet network connection, operative systems used, functional architectural components and the interconnection elements between them. This technological infrastructure is perfectly defined at a technical level, both within the documents related to the prototypes developed and within the documents explaining the deployment of the pilots in each city. This deliverable does not aim at specifying the technical infrastructure needed, which should actually be left to developers, but rather the functionalities that the technological infrastructure must support to have a fully operative and useful platform.

The toolkit is the set of tools that enables users to use the platform not the sample services. The toolkit not only consists of the software components, but rather, as will be further detailed below, the reference framework, linguistic resources libraries or service creation process specifications are part of it.

3.1.1 PLATFORM LAYERS MODEL

Although the platform definition quoted above is useful to roughly understand what the platform model that we have adopted is, it is not enough to have a clear overall vision of the user requirements.

A layer platform model provides the final user with a clear vision of the grade of specialisation, diversity and dependencies of components that compose the platform and the stakeholders involved in the development or the platform functionalities, thus making it easier to identify what the basic platform user requirements are.

Each layer of the model represents a set of tools and the users related to this layer. Layers are sorted by complexity of use of the tools and the grade of user specialization. Layers exist within a service creation framework that states the stakeholders involved and identifies the stages in the creation process. This framework known as N-Services has been defined in deliverable D22, the user requirements refinements.
For instance, the upper layer contains a description of the features of the tools used to manage the platform by the final user, whilst the bottom layer specifies the technological infrastructure with respect to the user requirements. The layers stated below are drawn from an analysis of the driving users requirements and they gather the basic platform user requirements.

3.1.1.1 PLATFORM MANAGEMENT TOOLS
The Platform management tools are specially designed for the final user who will be the platform administrator. The platform administrator will be in charge of the daily maintenance and monitoring of the platform. The HOPS monitor will be the tool that will facilitate most of the maintenance tasks.

3.1.1.2 SERVICE CREATION TOOLS
A service domain expert is the person who will define the type of deployed service and will actively participate in the service definition and creation process. In order to do that he or she will need a tool to create the services. The tool will be the service modeler that will allow the creation and/or specifying of any adaptable resource that had been previously created. The service selector will facilitate the task to deploy new services.

3.1.1.3 RESOURCES CREATION TOOLS
The resources creation tools could be independent tools or embedded functionalities of any component that will be used by the Resources domain expert to create within a reasonable time the linguistic resources that will be part of the service dialogs. This set of tools or functionalities will leverage the reusability and extensibility of the deployed services. For instance any sort of tool to create adaptable grammars for changing contents is required for informational services.

3.1.1.4 ADAPTABLE RESOURCES
All the services deployed by means of the HOPS platform are based on a fixed infrastructure and a set of adaptable resources specific for each local authority. The adaptable resources are the linguistic resources related with a specific domain, the implementation of API designed to integrate HOPS platform and municipal legacies, the dialogue structure, some specific domain ontology. The adaptable resources must be multilingual, multi-channel, reusable, open, extensible, ensure a good quality of service, and also must interoperate with the backend systems. A bottom-line requirement extracted from the platform user requirements is required by the essential dialog components that will leverage the extensibility and minimize the cost of creating new services. When a new resource is needed, either the Resources domain expert or the Backend domain expert will be required.

3.1.1.5 TECHNOLOGICAL INFRASTRUCTURE
From the point of view of the user, the platform must support both a phone and text channel, providing citizens with a quality of service as defined in the driving user requirements. The platform facilities must be scalable to adapt the platform facilities to
several parameters such as transferring citizen requests, which will be different in each city.
The HOPS platform consists of a technical infrastructure and a toolkit. The technological infrastructure includes the server types, phone boards, phone and internet network connection, operative systems used, functional architectural components and the interconnection elements between them. This technological infrastructure is perfectly defined at a technical level, both within the documents related to the prototypes developed and within the documents explaining the deployment of the pilots in each city. The toolkit is the set of tools that enables user to use the platform not the sample services. Toolkit is not only composed of the software components and, as we will detail below, the reference framework, linguistic resources libraries or service creation process specifications are part of it. The toolkit facilitates the creation, management and monitoring of the e-government services deployed using the platform.

3.2 FUNCTIONALITIES
This paragraph describes functionalities that the technological infrastructure and the toolkit must support to have a fully operative and useful platform.

3.2.1 FUNCTIONALITIES THAT THE TECHNOLOGICAL INFRASTRUCTURE MUST SUPPORT:
The HOPS platform can be divided into three layers depending on its main role in the system:
? Frontend services
? Dialogue and application management services (Middle Layer services)
? Backend services

3.2.2 FUNCTIONALITIES THAT THE TOOLKIT MUST SUPPORT
The bottom line platform user requirements are the tools that must be created to guarantee the exploitability of the system at the end of the project. These tools and methodology will permit the final user, for instance Citizen Information Bureau staff, to minimize time and human resources costs when implementing a new service.

Service monitor
The Platform management tools are specially designed for the final user who will be the platform administrator. The platform administrator will be in charge of the daily maintenance and monitoring of the platform. The HOPS monitor will be the tool that will facilitate most of the maintenance tasks.

Service modeler
Service modeler allows the creation and/or specifying of any adaptable resource that had been previously created. The service selector will facilitate the task to deploy new services.

Resource creation tools
The Resource Creation Tools allow technical staff to create faster new resources in the form of essential component libraries.
The front-end resources
The front-end resources are adaptable resources, connectors’ libraries, web services, EJB and other configuration files that manage interactions between the system and the local authorities’ backend.

The backend
There are basically two types of interaction with the backend. The first one is the query of information about a specific topic. After the front-end has gathered all of the required inputs to interact with the backend, the platform sends data and the backend returns the information, which is processed by the front-end to be finally delivered to the final user.

The second one is related to the transactional services. The final user uses the front-end to ask for a service and fill a form, the system sends this form to the local authority application, which processes this query and informs the HOPS system of the acceptance of the service. After the acceptance, the transaction is realized by the municipal application.

3.3 REQUIREMENTS

3.3.1 HARDWARE AND SOFTWARE SPECIFICATIONS
The HOPS platform, from the hardware point of view, has some requirements mainly related to VoxNauta, the enhanced voice platform environment. The hardware configuration for the server hosting VoxNauta can be highly dependent on several factors such as the number of channels, the number of languages and voices the services will be using, as well as the size and complexity of services and grammars.
From the software point of view, the platform presents the following requirements:
- SO for server hosting VoxNauta WIN 2000 server
- SO for hops platform Anyone
- JVM 1.4 or later
- Web server Anyone
- Application server (needed for 1st prototype) Tomcat (last version available)

### 3.3.2 User Requirements

As quoted in D21, starting from the background high-level requirements stated in the Technical Annex, the Local Authorities have agreed on a set of global requirements that have lead the development of the HOPS solution. The following list describes this set of driving requirements.

- **Multi-channel approach**
  HOPS is a solution that aims to be accessible through different channels. Voice access through the telephone channel is obviously one of the most important access modes because of its high availability and accessibility by citizens under many different conditions. However this is not the only channel that the project has to implement. Other access means, such as written text, must be supported by the HOPS solution.

- **Reusability**
  The features of the HOPS solution must be able to be used beyond the demonstration services that will be actually deployed during the project. This idea leads to a conceptualisation of HOPS offering features that will allow cities to build their own applications.

- **Openness**
  All components must offer the possibility to be called upon by other software systems. APIs to exploit all components must be available with proper documentation.

- **Extensibility**
  It must be easy and efficient to add or change any of the components necessary to build a public service:
  - Linguistic resources
  - Domain and business rules
  - Access channels
  - Knowledge repositories
  - Backend systems

- **Quality of service**
  The services developed in the HOPS Platform should be able to keep the current high level of citizen satisfaction.⁵

- **Interoperability with the cities’ backend systems**

---

⁵ According to the surveys reported in section 6 of deliverable D21
The HOPS solution should not constrain the cities to use specific backend systems. It should be compatible with the existing or intended databases, application servers and call centre components.

- **Flexible development model**
  The development model must allow the early use of the prototypes created during the development process, i.e. the sample services should be developed using the HOPS Platform whenever operational versions of the required tools are available. As already mentioned, requirements will evolve during the life of the project, and will be maintained in a RUP website. However, this short list of core high-level driving requirements is not expected to change significantly.

Service Selector functionality: Another bottom-line requirement extracted from the platform user requirements of the technological infrastructure is the Service Selector functionality that will divert the citizen phone call or internet query to the chosen service with ease.

All these components and characteristics are largely described in D 41-Architecture Specification.

### 3.4 TECHNOLOGIES

The HOPS platform architecture will not only provide support for voice channels but for non-voice (such as enhanced text interface) channels, taking advantage of the technologies already present in the system which can help obtain better performance from these channels (which can not be reached without the contribution of the specific technologies involved in the HOPS platform).

Therefore HOPS technologies are related not only with voice but to other technologies that are used to strengthen the initial technologies to profit from them in a more efficient manner. In this respect, voice technologies like ASR -Automatic Speech Recognition-, TTS -Text to Speech- are complemented with Natural Language Process technologies and Semantic Web technologies.

#### 3.4.1 HOPS SOFTWARE ARCHITECTURE

The HOPS Platform is built on top of a service-oriented architecture providing a set of common procedures and tools to enable the suitable integration of all the parts of the platform.

The platform presents a Service-oriented architecture (SOA) that it separates the service’s implementation from its interface or contract.

All the components present in a SOA can be divided into three different entities:

- Service Providers
- Service Consumers
- Service Registry
The SOA architecture presents the following characteristics:

- Services are discoverable and dynamically bound.
- Services are self-contained and modular.
- Services are loosely coupled.
- Services have a network-addressable interface (not necessarily an IP address).
- Location transparency.
- Services are composable
- Supports self-healing.

All SOA components and characteristics are largely described in D 41-Architecture Specification.

SOA architecture is supported by FADA (Federated Advanced Directory Architecture - http://fada.sourceforge.net) technologies, which aid a system to comply with the service oriented architecture principles.

The FADA system is used primarily to store service proxies (Java Serialized Objects), providing suitable mechanisms for registering and looking up these service proxies by third parties (service providers vs. service consumers).

FADA provides its functionality by a proxy that can be located and downloaded by the means of the FADA libraries, the fada-toolkit. The toolkit provides convenience methods to perform complex operations based on the FADA proxy simpler operations.

The service proxy must implement some Java interface.

The FADA proxy can perform several operations: register, deregister and lookup are the main ones.

By registering, a service proxy is uploading onto FADA. This registration assigns some descriptive data with the proxy, in the form of service entries.

The FADA bundle contains a library which contains the needed classes to interact with the FADA infrastructure through a high level API. Also, a set of helper classes free the FADA-aware software (service provider side) to deal with the SOA specific mechanisms as the lease renewal or neighbor candidates management. This library is known as the FADA toolkit.

A third party library has been developed by a HOPS consortium partner (IT-Deusto) for integrating a J2EE web application with the HOPS Platform using a new module called ServiceManager. This component can be briefly defined as a tool to create, register and manage FADA-aware services, which are running on top of a servlet engine, into a FADA network. The ServiceManager can be seen as a container of FADA services which is deployed as a servlet in a web container. As it was committed during that review ITDeusto has added the component to the sourceforge.net (https://sourceforge.net/forum/forum.php?forum_id=479450) where it is available for the rest of the Open Source community.
All FADA components and characteristics are largely described in D 41-Architecture Specification.
4 CONCLUSIONS

This document set out final guidelines related to the HOPS platform, based on feedback and deliverables received from the entire life cycle of the project.

The results gathered and stipulated within this document express the final product description, how it is composed and how can be used. In particular it lists the complete functional specifications of the tools needed to create and manage the services. For specific details, links with other deliverables are necessary.

This document also provides an overall vision of the project and includes the complete references of the platform and its links with other deliverables.
5 REFERENCES

6 **APPENDIX A – USER GUIDE**

See the annexed paper.
Enabling an Intelligent Natural Language Based Hub for the Deployment of Advanced Semantically Enriched Multi-channel Mass-scale Online Public Services

Project Number IST-2002-507967

D7.3 Appendix A
User Guide
# TABLE OF CONTENTS

1 **INTRODUCTION** .......................................................................................................................... 3

1.1 **PREVIOUS CONCEPTS** ........................................................................................................... 3

1.2 **HOPS ATTRIBUTES AND RECOGNITION OBJECTS** .......................................................... 3

2 **LINGUISTIC RESOURCES** ......................................................................................................... 5

2.1 **SPEECH GRAMMARS** ............................................................................................................. 5

2.1.1 **Description** ........................................................................................................................ 5

2.1.2 **Function** ............................................................................................................................. 6

2.1.3 **Files** .................................................................................................................................. 6

2.1.4 **Relationship** ....................................................................................................................... 7

2.1.5 **Manipulation** ....................................................................................................................... 7

2.2 **NLPP GRAMMARS** .................................................................................................................. 8

2.2.1 **Description** ........................................................................................................................ 8

2.2.2 **Manipulation** ....................................................................................................................... 9

2.3 **ONTOLOGIES** ......................................................................................................................... 12

2.3.1 **Description** ........................................................................................................................ 12

2.3.2 **Function** ................................................................................................................................ 13

2.3.3 **Files** .................................................................................................................................. 13

2.3.4 **Relationship** ....................................................................................................................... 13

2.3.5 **Manipulation** ....................................................................................................................... 14

2.4 **LANGUAGE PATTERNS** ......................................................................................................... 16

2.4.1 **Description** ........................................................................................................................ 16

2.4.2 **Function** ................................................................................................................................ 16

2.4.3 **Files** .................................................................................................................................. 17

2.4.4 **Relationship** ....................................................................................................................... 18

2.4.5 **Manipulation** ....................................................................................................................... 18

2.5 **VOICE AND TEXT TEMPLATES** ............................................................................................ 19

2.5.1 **Description** ........................................................................................................................ 19

2.6 **FILES** .................................................................................................................................. 20

2.6.1 **Relationship** ....................................................................................................................... 20

2.6.2 **Manipulation** ....................................................................................................................... 20

2.7 **DIALOGUE PLANS** ............................................................................................................... 22

2.7.1 **Description** ........................................................................................................................ 22

2.7.2 **Files** .................................................................................................................................. 24

2.7.3 **Relationship** ....................................................................................................................... 25

2.7.4 **Manipulation** ....................................................................................................................... 25

2.8 **BACKEND CONTENTS** ............................................................................................................ 25

2.8.1 **Description** ........................................................................................................................ 25

2.8.2 **Relationship** ....................................................................................................................... 25

2.8.3 **Manipulation** ....................................................................................................................... 25

3 **VARYING OR UPDATING AN EXISTING SERVICE** ................................................................. 26

4 **CREATING A NEW SERVICE** ................................................................................................... 27
1 INTRODUCTION

When developing the HOPS project, one of the main goals was to obtain a multilingual platform with excellent capabilities on customization and service adaptation to different purposes and environments.

This section is aimed at establishing the guidelines to operate with the system. It describes all linguistic resources which play a special role and how to work with.

On the attached schema, the user can identify the various linguistic resources taking part in the process and intuitively know about how HOPS works. This graph wants to illustrate where and how each resource can influence on the different steps to follow.

Circled numbers refer to usual steps (to be ran increasingly) once the user makes an utterance.

1.1 PREVIOUS CONCEPTS

1.2 HOPS ATTRIBUTES AND RECOGNITION OBJECTS

Before we get too deep into the explanation of the services which are present in the HOPS Platform, it is necessary to define and clarify the concept of the HOPS Attribute which will often appear from now on throughout the document, as well as the related concept: the Recognition Object.

The different services that comprise the HOPS Platform share some specific information which feeds its business logics. This information is usually taken from the interaction with the user or retrieved from the backend systems. To allow this information to be shared between the disparate components that comprise the HOPS Platform, it is necessary to fix a common understandable data structure for formatting
the information. The format that expresses the translated information is based on key-value pairs where a key can become a value for another key-value pair depicting a tree (even a graph) hierarchy. So, a HOPS Attribute is an instance of applying this translation to the information retrieved from the user or the backend to be shared between the HOPS Platform services.

The HOPS Attributes have a close relationship with the ontology concepts, not in vain, the contents presented in each HOPS attribute is always referring to an ontology concept. In other words, the key and the values of the HOPS attributes are ontology concepts. In this sense, it is clear that, during the (user/backend) information translation phase, the task of creating/modifying the structure for storing the data is not only carried out, but also, the task of properly mapping the user inputs to the ontology concepts is also carried out. As it will be explained in next chapters, this last task will be handled by the Natural Language services, if the input comes from the user, or through pre-backend services if the information is retrieved from the backend systems. The following pictures represent the relationships between some ontology concepts used by the HOPS Attributes:

When it is needed to share more than one HOPS Attribute at the same time between two services, a specific data structure is used which acts as a simple container: the Recognition Object.
2 LINGUISTIC RESOURCES

2.1 SPEECH GRAMMARS

2.1.1 DESCRIPTION

A grammar is a formal way to describe the allowed words and sentences, (and eventually to extract semantic information useful for application).

The development of Speech Recognition applications is based on man machine interaction dialogues. For each point of this interaction you need a way to guess the sentences the user can say (this is mandatory to reach satisfactory recognizer performance).

Vocabulary type creates a new vocabulary grammar (that could return a single semantic value linked to main vocabulary words). It's possible (but not necessary) to add two (or more) other vocabularies (Prefix and Postfix) which can be optional and are useful if there are some words that can be said before or after principal words you want to be recognized. We call them also the CORE part and the FILLERS.

Vocabulary element is one of the basic elements. It is composed by an alternative of words. In general, if you use a vocabulary element just one of its words can be said.

For example, you can say:

\[ I \text{ want to 'or' } I'd \text{ like to} \]

You cannot say:

\[ I \text{ want I'd like to} \]

As we get further along in our quest for HOPS's grammar knowledge, the topic of complex, multi-level grammars is sure to come up: a grammar construct is simply a number of independent rulenames the reference each other, allowing a caller to speak dynamic utterances naturally that can be recognized and returned by the application. For instance, when we have the following grammar structure, it allows us quite a bit more flexibility than a usual flat-file grammar, and also offers increased recognition time when a large grammar is used:

RULE_1
  (RULE_2 music concert)
RULE_2
  (rock|classic|pop)

Looking at the above, when we reference 'RULE_1' then it allows us to also reference RULE_2 and the utterances contained there. As such, the following utterances would be considered an acceptable grammar match:

"rock music concert"
"classic music concert"
"pop music concert"
Taking this example a few more steps further could allow us to capture an entire sentence as a structure, providing that we have appropriate utterances defined for each rule, and that grammar 'collisions' (due to similar defined utterances) do not occur. As such, the robustness of using subgrammars versus a flat file grammar becomes very apparent.

2.1.2 FUNCTION

Each dialog interaction has one or more speech and/or DTMF grammars associated. A grammar specifies the input that is acceptable to the VoiceXML Interpreter at a given time. A grammar is a template that specifies combinations of words to be matched; the grammar must be written in the formalism supported by the ASR engine. The Loquendo ASR engine and, therefore, also Loquendo VoiceXML interpreter, support the following grammar formalisms:

SRGS 1.0 XML format;
SRGS 1.0 ABNF format;
JSGF 1.0 (voice grammars only. No dtmf).

Which kinds of specific grammars are there? This depend on what is intended by ‘specific grammar’. Examples of application specific grammars are:

Event’s Titles
Event’s Performers
Event’s category/thematic channel/themes
Event’s Locations Name
While grammars like:

Town’s districts
Town’s borough
Town’s addresses
are application specific, but can also be reused for other location based application (e.g. nearest Pharmacies). Reusable grammars and adapted built-in grammars have been also used in HOPS for specific tasks, e.g.:

Date
Time
Numbers and phone
Boolean
List Browsing

2.1.3 FILES

Production

Any developer can create a new grammar. The file extension is related to the grammar formalism and must be consistent with its mimetype as specified in VXML page/file: for JSGF format it is ‘<grammarname>.sjv’; for SRGS (XML) format it is ‘<grammarname>.grxml’; for SRGS (ABNF) format it is ‘<grammarname>.gram’.
Location
Grammar can be located in any folder. URL must be specified in the VXML file.
Currently: C:\Tomcat\webapps\ROOT\hops-vxml\grammars\third_prototype\BCN\

2.1.4 RELATIONSHIP
VoiceXML files: there’s a specific field pointing to corresponding grammar/s: depending on thematic or expected info to receive from user, system will provide the grammar or grammars’ reference to have into account for a successful recognition and dialogue flow. VoiceXML files are generated on Dialogue Manager – Outpu Generator, for further information see the point XX.

Used by:
ASR ()
SNLA (Spoken Natural Language Analyzer)
Both of them use them on recognition of user utterance.

Warning: it’s important to keep attribute’s name correlation in other resources.

2.1.5 MANIPULATION

Varying /adding an existing service
At anytime we can modify or add vocabulary when willing to update an existing service. An important point to consider will concern to public rules within top level service grammars that need to be checked once a new rule in an existing grammar or a new grammar is added, in order to allow the new rule (and its vocabulary) to be reached by top levels. Grammars can include also references to other grammars, either to their public root rule or their internal compounding rules: these references need to be checked too. This is not needed in case that only vocabulary elements are modified, added or deleted in a grammar rule.

Changes in grammars does not require any initialization. They can be simply copied (or also overwritten) as they are compiled by ASR “on the fly” at runtime.

When introducing a new grammar user must be aware to communicate its presence to the VoiceXML Application/Dialogue Developer (in HOPS corresponding to DM-OG). As a result of previous operations prompts and grammar’s references in the VXML pages must be up-dated and checked (in HOPS automatically generated by DM-OG).

Recommended grammars development process for any new grammar is:

Define the information items that the user should provide;
Design the prompt (to be included in VXML);
Anticipate the user responses;
Identify the core and filler portions;
Define the grammar graph (see the image below);
Write grammar code;
Test the grammars with “real users”;
Refine the grammar if needed.

Adding a new service

In HOPS, the main grammar corresponding to service selection is named ‘serviceIDSS.gram’: when a new service is added, the corresponding keywords or the references to the top/main service grammars need to be added there.

When adding a new service the specific application grammars need to be created, according to the above described process. When creating a service is important to consider the close relation between prompts and grammars design.

To administrate previous acts or to know more about, user can refer to supporting tools, tutorials and other details found at: http://www.loquendocafe.com/index.asp

2.2 NLPP GRAMMARS

2.2.1 DESCRIPTION

These second grammars are used by the NLPP to analyze syntactically the string we got after submitting previous steps (Frontend's language recognition). In this case system works to get meaningful results from user utterance when applying it to ontological knowledge.

Words and meanings must be introduced for any relevant word that will be considered (considering also operar's name, sing starts, idioms,...), at same time words introduced into dictionary must be linked to ontology if possible in order get sensefull results when analyzing incoming objects.

As to make it easy we can understand that NLPP goes from words to meanings, so in the way to achieve a satisfactory results user utterance is submitted to a syntactic analysis. The syntactic processing is rather complex and cannot be described here in any detail (it's not rellevant at the moment). We only give some hints about the overall
architecture. In order to make possible system to arrive to further internal steps of the process:

Morphological Analysis and access to the dictionary. This is carried out on the basis of tables of suffixes available for the different languages. The English morphology is very simple, but the situation is different for the other languages of the project (Catalan, Italian, Spanish).

POS Tagging. Since the words retrieved in the previous step can be ambiguous (e.g. “plays” can be both a verb and a noun), one of the different alternatives is selected on the basis of the local context where the word occurs (a two-words-before + two-words-after window). The rules that carry out the tagging are largely language independent, though there are a limited number of rules devoted to the different languages.

The sequence of disambiguated words is submitted to the parser. It is a chunk-based dependency parser, based on two types of knowledge that, together, constitute “the grammar” of the language. In a first phase “word chunks” are identified. They are, for instance, adjectival chunks (“very good”), nominal chunks (“the very good dinner of yesterday”), and so on, with chunks of increasing complexity. At the end of the first phase, we have a set of isolated chunks. Then, verb subcategorization rules are applied to attach the chunks to the verbs appearing in the sentence, in order to get a single, integrated, parse tree.

2.2.2 MANIPULATION

To manipulate contents we will take into account reference given in the "introduction". When adding or manipulating contents we will act according to pointed steps (Morpho Analysis, POS tagging, disambiguation).

A) ADDING WORDS AND PORTIONS IN THE DICTIONARY:

In the following examples, each entry refers to a “root”. In Italian “concert” is the root both of the noun concerto and of the verb concertare. In English, it is the root of the noun concert and of the verbs to concert and to concertize. The actual forms are obtained by adding suffixes appearing in suitable tables. For instance, the “classe” 1 is the regular for English verbs, so that from the root “concert”, it is possible to get concert, concerts, concerted. For Italian, “classe” 1, enables one to obtain concerto, concerti, concertata, concertiamo, … and so on for around 40 surface forms for the concertare verbs (according to tense, mood, person, …). Currently, the Italian dictionary includes around 25,800 roots, and the English dictionary around 53,300 roots. To these, the compound words must be added (i.e. entries composed of more than one word, as “ticket counter” in our example), which are represented in a slightly different format.
B) POS TAGGING RULES:

POS Tagging rules are associated with lexical ambiguities. For instance, the English input word concerts produces two results after the morphological analysis:

concerts ((concert noun common n pl))
concerts ((concerts verb main ind pres 3 sing))

In this case the rules associated with the noun-verb ambiguity are applied. An example of such a rule is the following:

(noun-verb5
 :if '(and (prevcat 'verb)
 (prevtype 'mod) (currmood 'infinite))
 :then 'verb)
It can be read: if the previous word is a modal verb and the current word is (as a verb) in the infinite mood, then choose the category verb (He will concert a meeting). Note that this rule applies to all languages involved in HOPS, since for all of them modals govern infinite verbs. An example of a language-specific rule is the following:

\[
\text{if } ('(\text{and } (\text{prevcat} \ ' \ \text{prep})
\quad (\text{nextcat} \ ' \ \text{noun}))
\text{then } \text{noun}
\text{CF } \text{U}
\text{lang } \text{english})
\]

This copes with noun-noun sequences preceded by a preposition. Note that rules include a Certainty Factor (A means “Almost Certain”, while U means “Uncertain”), so that the noun-verbr38a rule is applied just in case all other rules having an higher CF have failed. Other rules apply to inter-categorial ambiguity (as the mood feature, between “I concerted” - indicative - “the meeting concerted by …” - participle).

Currently, the tagger include 460 rules; 370 are common; 50 are reserved for English; 40 for Italian (there are also 10 for Spanish). It correctly classifies 98.1% words for Italian and 96.9% for English. Of course, the Italian rules have been more carefully tuned. Some improvements can be obtained for English after further tests. These results apply to general text, while for HOPS sentences the data are not enough to get meaningful results. On the other hand, the rules can be tuned more carefully to HOPS sentences, so that the tagging errors are reduced further on them.

C) PARSING RULES (DISAMBIGUATION):

As stated above, the actual parsing process is split in two parts. The first subprocess takes care of chunk construction. An example of chunk rule is the following:

\[
\text{(NOUN common (precedes (ADJ qualif T (#\#\#\#\#) (ADJ ((type qualif) (agree))) ADJC+QUALIF-RMOD))}
\]

The rule states that if an adjective (ADJ) of type qualif (i.e. standard adjectives, not possessives, anaphoric, etc.) precedes a noun (possibly separated by -, ‘, or “, or by other qualificative adjectives), and if it agrees in gender and number with the noun, then it can be attached to it as a dependent via a ADJ+QUALIF-RMOD link. This rule applies to “una bella casa”, “a nice house”, “my new big house”, …). An example of a rule reserved to English is the following:

\[
\text{(NOUN common (language english}
\quad \text{precedes (PUNCT (char #\#\#) T)
\quad (NOUN ((type common)
\quad (not (word-typ &word-ref))))}
\]

This handles noun-noun modification, as “The art exhibitions”. The condition (not (word-typ &word-ref)) blocks the application to cases as “The word ‘art’”, where “art” modifies “word”, and not vice-versa.

The second subprocess is much more complex to describe, since it involves verbal subcategorization and transformations (as for passives). We can only say that the verbs are associated subcategorization classes (as transitive, intransitive) and that the chunks built in the previous phase are attached to the verbs according to their expectations. This enables the system to assign labels as verb-subj, verb-obj, verb-indobj, and so on.

2.3 ONTOLOGIES

2.3.1 DESCRIPTION
An ontology defines the terms used to describe and represent an area of knowledge. Ontologies are used by people, databases, and applications that need to share domain information (a domain is just a specific subject area or area of knowledge, like medicine, tool manufacturing, real estate, automobile repair, financial management, etc.). Ontologies include computer-useable definitions of basic concepts in the domain and the relationships among them (note that here and throughout this document, definition is not used in the technical sense understood by logicians). They encode knowledge in a domain and also knowledge that spans domains. In this way, they make that knowledge reusable.

The word ontology has been used to describe artifacts with different degrees of structure. These range from simple taxonomies (such as the Yahoo hierarchy), to metadata schemes (such as the Dublin Core), to logical theories. The Semantic Web needs ontologies with a significant degree of structure. These need to specify descriptions for the following kinds of concepts:

- Classes (general things) in the many domains of interest
- The relationships that can exist among things
- The properties (or attributes) those things may have

Ontologies are usually expressed in a logic-based language, so that detailed, accurate, consistent, sound, and meaningful distinctions can be made among the classes, properties, and relations. Some ontology tools can perform automated reasoning using the ontologies, and thus provide advanced services to intelligent applications such as: conceptual/semantic search and retrieval, software agents, decision support, speech and natural language understanding, knowledge management, intelligent databases, and electronic commerce.

Ontologies are used in artificial intelligence, the semantic web, software engineering and information architecture as a form of knowledge representation about the world or some part of it. Ontologies generally describe:
- Individuals: the basic or "ground level" objects
- Classes: sets, collections, or types of objects[1]
- Attributes: properties, features, characteristics, or parameters that objects can have and share
- Relations: ways that objects can be related to one another

2.3.2 FUNCTION
The use of ontologies allows the HOPS platform to properly describe, and differentiate domain information about the services offered by the system. This knowledge is key for the other components of the HOPS platform, enabling NLP and dialogue management facilities to process the user utterance and interact with him/her by means of both text and voice channels. Ontologies provide semantics, i.e. meaning to the information contained in the backend data.

2.3.3 FILES

Production
The ontologies used in HOPS are written in the W3C standard language OWL. Compliance with Semantic Web languages increases both the possibilities of uptake by the goal market of the project and support provided by the Semantic Web community in terms of new generation tools and technology.

To produce ontology or to modify existing ones is recommended to use ontology editor (Protegé version 3.2 is recommended). When modifying, it is strongly recommended to reuse existing ones instead of creating new ones from zero in order to not risk system results.

WARNING: it’s important to review NLPP resources.

Location
User can change the location of the ontology and stores it wherever they want. Path is independent to manage the OWL ontology

C:\HOPS\src\hops\isoco\OWLtoLISPParser\ (to check)

2.3.4 RELATIONSHIP
NLPP: the Natural Language Parser Processor is the one who will finally “execute” ontologies to extract meaningful concepts from user incoming sentences. Ontology Manager has been extended with a new functionality which transforms the OWL ontology describing the services supported by the system into a structure necessary for the NLPP component in order to analyze the user utterance. This allows: i) profiting from the expressive capabilities of OWL above Lisp or any other Knowledge Representation languages, which additionally is the W3C standard language for the Semantic Web and ii) taking advantage of the large amount of tools, e.g. Protégé,
OntoStudio, KAON, etc, existing in the market for ontology edition. Both of these points enormously simplifies the Knowledge Representation task.

2.3.5 MANIPULATION

Varying/extending an existing service
Changes on existing ontologies can be introduced at any time, Ontology Manager includes a functionality which updates automatically NLPP resources related with the ontology in a daily basis and stores the changes in the CVS server. This process can also be executed directly by the user.

Adding a new service
The addition of new services requires including the description of such services into the HOPS platform. The information contained in the ontology can be classified as:

Information common to all the services across the domain
Information specific to a particular service..

For maintenance facility and modularity reasons, the most recommended practice is to extend the service-specific part of the ontology with the necessary updates for the new service. This task can be accomplished by means of existing ontology editors like Protégé.

If a new service has been added to the ontology, the NLPP component needs to be reviewed in order to ensure that all the necessary procedures analyzing the user utterance about such service are inline.
The current ontology contains both the generic part described above and specific parts aiming at two particular services, e.g.: a Cultural Agenda (CA) about events and activities available for leisure and a Large Object Collection (LCO) service which informs citizen about possibilities for disposal of furniture and other large items. This can be seen in the figure, which shows the current HOPS ontology and the key concept, i.e. service, which needs to be extended upon addition of a new service.

Adding a new service would require to extend the ontology with its description. In terms of the ontology, this means to include into the taxonomy the necessary concepts, properties, and relations which describe the service. Two policies should be followed in this regard:

Maximal reuse of the existing ontology is recommended. It must be kept in mind that the common part to services is already contained in the ontology. Before adding new terms, it is recommendable to ensure that none of the existing ones already cover our needs.

It is recommended to avoid modification or deletion of pre-existing knowledge contained in the ontology. This would alter the description of other services.

To add a new ontology in the HOPS platform it is required to:

1. Use and ontology editor (Protegé version 3.2 is recommended) to create new project OWL/RDF File
2. To unify criteria, Ontology HOPS URI has to be http://com.isoco.hops instead of something like : http://www.owl-ontologies.com/Ontology1167911531.owl
3. To choose the language profile: OWL DL (by default)
4. To choose the logic view (by default)
5. To fill the property “rdfs:label” with the name of the new class created
6. Optionally the property “rdfs:comment” can be filled with a brief explanation of the class
7. Put the new ontology created into the destination folder “Ontology”

However, it is strongly recommended to reuse the existing ontology.

When adding a new service NLPP resources, e.g. procedures, need to be kept aligned with the peculiarities of the analysis of the new domain, and if necessary, updated.

In order to administrate previous acts an user must know domain in which concepts are going to be modeled.

Knowing that classes are definitions associate the names of entities in the universe of discourse (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. Considering that Properties can be distinguished between two main categories:

- Object properties link individuals to individuals.
- Datatype properties link individuals to data values

In summary, a user knowledgeable familiar with basic ontology engineering tasks would be required. Related to this but as part of the maintenance of the NLPP component, NLPP resources must be kept consistent with the changes in the ontology.

2.4 LANGUAGE PATTERNS

2.4.1 DESCRIPTION

Language patterns are the contents which fill expected gaps in VXML/TXML templates, furthermore this prompts will become the final words/sentences or sounds produced as the system-to-user feedback.

The prompts production follows several linguistic patterns and rules which are language dependent. Linguistic patterns are general to both modes of interaction, text or voice.

2.4.2 FUNCTION

Each interaction with the user is defined in an interaction file. There are two types of interaction files: voice (in VoiceXML format) and text (in an XML we called TextXML). All interaction files include the system prompts. Voice interaction files which are going to be interpreted also include the grammar pointer to be used to process
next user intervention (this is, the grammar containing suitable lexicons rules for expected response).

The system prompts could be generated manually for each service. Alternatively, they can be generated automatically when a new service is incorporated by using the general prompts and the linguistic patterns we have defined. However, the generated prompts have to be supervised during the development phase (some of them could look unnatural for a specific service).

2.4.3 Files

Production

As can be seen in the figure extracted from main excel file, for generating the system’s prompts we use sentences general to more than one service and linguistic patterns. General sentences can include variables that are instantiated during communication using context dialogue information.

In this prompts we should supply specific words for each new service and its type. In current version different types of words are classified considering their semantic (their type of content), e.g.: place, date, type, event,... Once a new domain word is included, its type has to be inserted.

In order to get satisfactory prompts would be desirable to have at disposal people skilled in Natural Language Processing, on the other hand non-skilled people could also create it by studying existing linguistic patterns.

In this point is important to state that generally language patterns are not inherent or specific for a service or domain. However, new patterns could be needed when adapting system to a new service.

Location

In the current implementation this folder are:

C:\HOPS\src\hops\talp\LG\misc\pg
For initializing the LG, the full path to find the lexicon and sentences files is required to be declared.

2.4.4 Relationship

The specific vocabulary for each new service is related to the service parameters. As said regularly along the guide, parameters’ name must be common in all components.

By the moment, there aren’t interrelation to any other customizable components, but they could.

2.4.5 Manipulation

Varying/extending an existing service

When extending an existing service, we should differentiate two cases:

- If attribute already exist we should be interested in adding speechacts related to.
- If attribute doesn’t exist we should add the new attribute and afterwards attend the previous situation.

All attributes and speechacts are listed/indexed in a file looking like /phrases.properties, in current version the path would be:

/hops_folder/talp/LG/etc/phrases.properties

This file is pointing to other files containing specific language resources:

/hops_folder/talp/LG/etc/<LANG>_lexicon.properties
/hops_folder/talp/LG/etc/<LANG>_phrases.properties

User can add or modify this contents but when done system must be reinitialized (but not recompiled) to assume new data, see necessary steps at the end of this point.

Another option to have into account is to generate an enumeration of attribute’s accepted values in order to help user to choose an answer. To afford this capability we should edit proper values on enumerable attributes list:

/hops_folder/talp/LG/etc/menuattr.properties

WARNING: In case Dialogue Manager attribute names are not the same than used by the LG, this file contains the translation from DM attribute names and LG attribute names:

/hops_folder/talp/LG/etc/HopsAttributes.properties

As stated below is a recommendable practice to keep attributes’ name all along the system resources.
Once we know how to vary contents properly, we can summarize required steps to add/modify prompts templates:

- Edit the excel file containing the patterns
- Run the Macro that generates a xml file with the patterns
- Run the “GeneraHopsResources” program in /hops_folder/talp/LG/dist/TALP-LG.jar to generate the lexicon and phrases language dependent files.

When extending an existing service can happen that specific service words belongs to a non-considered type on existing patterns. In this situation new patterns could be included.

**Adding a new service**

To assume a new service, Natural Language Generator will need all the specific words related to the service parameters belong to one of the types considered. If a new type is needed for the new service it has to be incorporated and linked to the appropriate pattern. When introducing a new services, you should add to this file the new attribute and all the speechacts related to each attribute:

```
/hops_folder/talp/LG/etc/phrases.properties
```

This index of speechacts and attributes links to the specific language resources, that are in the files:

```
/hops_folder/talp/LG/etc/<LANG>_lexicon.properties
/hops_folder/talp/LG/etc/<LANG>_phrases.properties
```

You should actualize these files with the new specific sentences and lexicon, either manually or automatically (GeneraHopsResources program)

To set a desired new service, the specific words related to the service parameters have to be given. The type of those words have also to be given.

See the previous point for further information.

2.5 **VOICE AND TEXT TEMPLATES**

2.5.1 **DESCRIPTION**

The Voice and Text Templates are XML files used by the OG. This XML files contain GAPS for inserting dynamic information of the dialogue, which will be provided by the Natural Language Generator attending resources details in the previous point. For example, there is a GAP for inserting the grammar path to use in the current interaction, another GAP for inserting the sentence to be shown to the user, and so on.
2.6 Files

Production
These templates are general, so you don’t need to generate new files for new services. New templates are needed when you change the Output Generator Module. So these files are created and designed by the OG developer in collaborative with the Front Ends.

Location
The OutputGenerator reads a file when begins to run, expecting to find it in:

C:\HOPS\src\hops\talp\OG\etc\templates\vxml

This file contains, apart from other information, the absolute path to the file that contains templates information. This second file contains the list of available templates and their absolute path.

2.6.1 Relationship
The templates (either vxml and txml) has some gaps for different user interventions acts as: no input reaction, no match reaction or help reaction. The NLG should provide the OG the needed sentences (prompts) to fill these specific GAPS.

There is an interrelation with the FrontEnd modules, since the final vxml and txml documents built will be read from these modules. For example, the vxml files contains the complete path of the voice grammars in the VoxNauta Server. These customizable variables are set in the file:

/hops_folder/talp/OG/etc/OG.properties

Furthermore, one of main gaps to fill is the one related to grammars which are going to be used in the frontend in order to evaluate the expected utterance. As a consequence all existing grammars in the frontend are listed in the following file:

/hops_folder/talp/OG/etc/OG/LangGrammars.properties

2.6.2 Manipulation

When varying/extending an existing service
Despite modifications on voice/text templates are not considered as usual activity, administrator can add/remove templates by updating the index in the file:

/hops_folder/talp/OG/etc/OG/NLGTemplates.properties

To add a new template we will have to develop it in VXML/TXML language considering GAPS to fill. As an example we can see following schemas:
The OG builds VXML pages on run time. After completing the built of the VXML or TXML page, there is a process that checks all the remaining GAPS. For each GAP found,

If it corresponds to an attribute name:
    Change the GAP with the attribute value
Else
    Remove the Gap

In order to find the attribute values, we need to translate from GAP name to DM attribute names. These translation is listed in the file:

/hops_folder/talp/OG/etc/OG/HopsAttributes.properties

Once we know how to vary contents properly, we can summarize required steps to add/modify voice and text XML templates:

1. write/rewrite the complete xml page
2. Check which are the dynamic parts, and substitute them by a GAP
3. Check which are the recursive parts, and substitute them by a GAP and create a new template for the recursive part.
4. modify the OG in order to fill up the new template correctly.

Adding a new service

Despite previous points include all related knowledge, here we summarize needed contents to assume a new service:

New service Attributes names DTMF grammars for new service attribute (if needed)
Voice grammar path for new service attributes

Then, to set a new service we should:

Add the attributes translation to these files:
   /hops_folder/talp/OG/etc/OG/HopsAttributes.properties

Add the voice grammar path for each attribute to this file:
   /hops_folder/talp/OG/etc/OG/LangGrammars.properties

Add the DTMF grammars to this file:
   /hops_folder/talp/OG/etc/OG/AttributesDTMF.properties

WARNING: it’s strongly recommended to consider attribute-grammar relationship and also prompts templates as altogether, stating again the importance of keeping names all along.

2.7 DIALOGUE PLANS

2.7.1 DESCRIPTION

The DM uses communication plans to recognize user’s intentions and to determine the next action the system should perform to satisfy the user’s needs. These plans are defined when a new service is incorporated and stored in a library of plans. We consider each service task is a possible user’s goal. For this reason, for each task service we define a communication plan that has to be followed in order to perform the task. The communication plans basically define when to ask the user the information the service task needs, when to access the service back-end and when and how to present the results obtained. Mainly, this information is related to the parameters of the service task. A communication plan can be decomposed into simple and complex actions. In current implementation of the DM, the simple actions can be: Asking, the system asks the user the information needed, Answering, the system gives information to the user, Ask to confirm (feedback) and Back-end access (dotransaction and exequery). Complex actions correspond to subtasks and consist of a set of simple actions. Preconditions governing in which context a particular action must be done are also included. Additional information about the parameters that has to be asked to the user is also included in the plans. This information consists of the set of possible values (when is a closed set), whether the data is optional or mandatory, if the value is simple (only one value) or compound (a set of values). Other information, such as the type of the value
could also be included. Examples of plans used by our system are given next. The first of these plans corresponds to the action to guide the user to choose a plan. The second one corresponds to the CA service. Notice there is a subplan for each service task. Plans are represented in ordered lists (the first elements in the bottom). The elements in the list are simple action (actions in the figure) and complex actions (subactions in the figure). Information on the attributes (service parameters) asked to the user include: if the value is simple (uniary) or compound (multiary), the value type (only when it is Boolean) and preconditions (commitment parameter value, meaning, the parameter and its value(s) are represented in the commitment stack).

```
#init plan
plan serviceIdentification init
action asking serviceId uniary
endactions
```

The plan to guide the user to choose a plan
The DM uses these plans to address issues introduced by the user which had not been previously raised. For instance, if the user initially says “I want information about classical music concerts” the system can search for a plan that contains an Ask action which its answer could be classical music concerts. Then, the DM would continue executing the other actions in the plan.

2.7.2 FILES
2.7.3 **RELATIONSHIP**

The plans are generated from the service tasks, considering the type of service, the input parameters and the output parameters.

Plans are static. During communication they are not changed.

2.7.4 **MANIPULATION**

Plans are generated when a new service is incorporated to the system. When varying/extending an existing service the plans also has to be revised.

The generation of plans requires some knowledge about the system plans (main concepts and the syntax used) as well as the service parameters. In order to facilitate the generation of plans for a new service, we have defined general structure (the general steps) of the plans for the two types of web services the system currently support: informational and transactional services.

2.8 **BACKEND CONTENTS**

2.8.1 **DESCRIPTION**

When talking about backend contents we're refering to information contained in an external database. This database will be normally the storage system of any city or institution willing to deploy HOPS, since there's no standard for stored data exchange or data can defer from one city to other will be necessary integrate it through JAVA objects conceived to wrap data in.

2.8.2 **RELATIONSHIP**

Will be necessary to define clearly which parameters will be involved in the data exchange, both uplink and downlink.

It's strongly recommended to know about local data storage system capabilities in order to know which services take sense to develop. Somehow, HOPS systems acts like an interface so it doesn't assume certain logical tasks when providing information desired by user if backend doesn't supply it properly.

2.8.3 **MANIPULATION**

We recommend local integrators experts to assume this task.
3 VARYING OR UPDATING AN EXISTING SERVICE

As varying or updating a service is understood to add proper information or data, or to update it. If we have a service already running in the platform, a normal modification would be to extend the vocabulary (e.g. to introduce new cultural event names when they're available at the backend), in that case modification can be afford at any time by introducing right knowledge in the grammars.

When updating a service we basically will modify vocabulary and sometimes grammar rules. These actions are easy and clear defined in previous points and could be done for anyone without special skills on language recognition or programming.

Somehow, our interest will require us to extend the service by introducing concepts which haven't been considered yet. This case can be considered as a simple situation for next section, ‘Creating a new service’, please refer there to know how to deal with this action.
4 CREATING A NEW SERVICE

Gantt chart

When creating or deploying a new service on the HOPS platform few tasks must be assumed considering some prior considerations.

As can be seen in the Gantt chart timeline, it is expected to spend up to one week to establish preliminary concepts in order to drive next steps. To succeed in different tasks it’s important to have a clear idea about what to do but mostly how to.

First task to be done will be to decide which attributes are going to play any role, how will be named, which information will system work with and how will it do. If we’re consistent with prior definitions further development will become much easier!!!!!!

Important considerations to take into account when planning a new service:

- Resources reusability is desired when possible, please consider similarities to other services in order to profit already working resources.
- HOPS platform acts like an user-backend interface, so consider backend capability properly to provide required information without assuming HOPS to manipulate it (or to take critical decisions on provided data).