NoTube

Networks and ontologies for the transformation and unification of broadcasting and the Internet

FP7 – 231761

D6.1b NoTube System Specifications and Architectural Design

Coordinator: A. Conconi, F. Cattaneo (TXT)

With contributions from:
R. Del Pero, L. Vignaroli, F. Negro (RAI)
D. Brickley (VUA)
L. Miller, V. Buser (BBC)
M. Minno, D. Palmisano (AS)
G. Stoll, R. Zimmermann, M. Riethmayer (IRT)
S. Dietze, Neil Benn (OU)
M. Yankova (OT)

Quality Assessor: Alex Conconi
Quality Controller: Libby Miller

<table>
<thead>
<tr>
<th>Document Identifier:</th>
<th>NoTube/2009/D6.1/b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version:</td>
<td>2.0</td>
</tr>
<tr>
<td>Date:</td>
<td>19/02/2010</td>
</tr>
<tr>
<td>State:</td>
<td>Final</td>
</tr>
<tr>
<td>Distribution:</td>
<td>PU</td>
</tr>
</tbody>
</table>
EXECUTIVE SUMMARY

In the light of the activities performed from M3 to M13, this document presents the updated NoTube specification in terms of functional and non-functional requirements as well as available and planned services, supporting the three NoTube usecases.

The effort produced during the first year of the project by the technological workpackages aimed at translating high-level requirements into a more concrete definition of software services and data structures impacts in turn on the development of the first round of prototypes through an individual integration process that involves a subset of the platform capabilities in the light of the updated use cases requirements. The rationale for the NoTube architecture design process follows the approach introduced in the first issue of this document however the amount of information exchanged and collected by the Consortium partners until now allowed a more specific classification of the NoTube services. The integrated platform then supports vertical customizations as those required to develop the three WP7 use cases.

A section of the document is dedicated to the general description of the envisaged NoTube services grouping them in the recognized categories. A more detailed description of such services is included in D6.2, as a companion to the first integrated prototype.

On the other hand, the definition of the abovementioned reference framework provides the basis for the accommodation and orchestration of a subset of services through the semantic brokerage defined in WP5 in accordance with SESA (Semantically Enabled Service Architecture) principles. See the second issue of D5.1 for more details.

The commonly agreed vision of the NoTube platform, the involved services and components as well as the relationships among them bootstrapped the process that led to the current achievements. Starting from the general design approach defined in the first issue of this deliverable, WP7 partners in partnership with WP6 produced a more detailed vision of the individual architectures envisaged for each application scenario, taking into consideration the SOA paradigm as a reference point.

The first release of the NoTube prototypes served as an important exercise for the RTD activities in the project supporting the integration of the available services.
DOCUMENT INFORMATION

<table>
<thead>
<tr>
<th>IST Project Number</th>
<th>FP7 - 231761</th>
<th>Acronym</th>
<th>NoTube</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Full Title</th>
<th>Networks and ontologies for the transformation and unification of broadcasting and the Internet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project URL</td>
<td><a href="http://www.notube.eu/">http://www.notube.eu/</a></td>
</tr>
<tr>
<td>Document URL</td>
<td>EU Project Officer: Leonhard Maqua</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deliverable Number</th>
<th>6.1</th>
<th>Title</th>
<th>NoTube System Specifications and Architectural Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work Package Number</td>
<td>6</td>
<td>Title</td>
<td>Architecture for personalized TV content</td>
</tr>
</tbody>
</table>

| Date of Delivery | Contractual M13 | Actual 12/02/2010 |
| Status | version 1.3 | final ✓ |
| Nature | prototype □ | report ✓ dissemination □ |
| Dissemination level | public ✓ consortium □ |

| Authors (Partner) | TXT, RAI, VUA, AS, IRT, OT, OU, KT |
| Responsible Author Name | Alex Conconi | E-mail | alex.conconi@txt.it |
| Partner | TXT | Phone | +39.0225771826 |

Abstract (for dissemination) This document illustrates the NoTube specifications in terms of functional and non-functional requirements, leading to the definition of a general architecture as composed by different categories of heterogeneous services that can be composed and verticalised to support the three NoTube usecases. The second release of this deliverable, due M13 describes the first year outcomes in terms of category of services as well as detailed architecture for each of the envisaged application scenarios. This process serves as an important starting point for the development of the three prototypes. Moreover it sets an integrated and commonly agreed vision of the NoTube platform, the involved services and components and the relationships among them.

Keywords Architecture, Requirements, Services
<table>
<thead>
<tr>
<th>Issue Date</th>
<th>Rev. No.</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>29/04/09</td>
<td>1.0</td>
<td>F. Cattaneo, A. Conconi</td>
<td>Final version of D6.1a</td>
</tr>
<tr>
<td>19/02/10</td>
<td>2.0</td>
<td>F. Cattaneo, A. Conconi</td>
<td>Final version of D6.1b</td>
</tr>
</tbody>
</table>
# PROJECT CONSORTIUM INFORMATION

<table>
<thead>
<tr>
<th>Participants</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vrije Universiteit Amsterdam</td>
<td>Guus Schreiber&lt;br&gt;Phone: +31 20 598 7739/7718&lt;br&gt;Email: <a href="mailto:schreiber@cs.vu.nl">schreiber@cs.vu.nl</a></td>
</tr>
<tr>
<td>British Broadcasting Corporation</td>
<td>Libby Miller&lt;br&gt;Phone: +44 787 65 65 561&lt;br&gt;Email: <a href="mailto:libby.miller@bbc.co.uk">libby.miller@bbc.co.uk</a></td>
</tr>
<tr>
<td>Asemantics SRL Uninominale</td>
<td>Alberto Reggiori&lt;br&gt;Phone: +39 0639 7510 78&lt;br&gt;Email: <a href="mailto:alberto@asemantics.com">alberto@asemantics.com</a></td>
</tr>
<tr>
<td>Engin Medya Hizmetleri A.S.</td>
<td>Ron van der Heiden&lt;br&gt;Phone: +31 6 2003 2006&lt;br&gt;Email: <a href="mailto:ron@engin.tv">ron@engin.tv</a></td>
</tr>
<tr>
<td>Institut fuer Rundfunktechnik GmbH</td>
<td>Christoph Dosch&lt;br&gt;Phone: +49 89 32399 349&lt;br&gt;Email: <a href="mailto:dosch@irt.de">dosch@irt.de</a></td>
</tr>
<tr>
<td>Ontotext AD</td>
<td>Atanas Kiryakov&lt;br&gt;Phone: +35 928 091 565&lt;br&gt;Email: <a href="mailto:naso@sirma.bg">naso@sirma.bg</a></td>
</tr>
<tr>
<td>Open University</td>
<td>John Domingue&lt;br&gt;Phone: +44 1908 655 014&lt;br&gt;Email: <a href="mailto:j.b.domingue@open.ac.uk">j.b.domingue@open.ac.uk</a></td>
</tr>
<tr>
<td>RAI Radiotelevisione Italiana SPA</td>
<td>Alberto Morello&lt;br&gt;Phone: +39 011 810 31 07&lt;br&gt;Email: <a href="mailto:a.morello@rai.it">a.morello@rai.it</a></td>
</tr>
<tr>
<td>Semantic Technology Institute International</td>
<td>Lyndon Nixon&lt;br&gt;Phone: +43 1 23 64 002&lt;br&gt;Email: <a href="mailto:lyndon.nixon@sti2.org">lyndon.nixon@sti2.org</a></td>
</tr>
<tr>
<td>Stoneroos B.V.</td>
<td>Annelies Kapteijn&lt;br&gt;Phone: +31 35 628 47 22&lt;br&gt;Email: annelies.kaptein@stoneroos</td>
</tr>
<tr>
<td>Thomson Grass Valley France SA</td>
<td>Raoul Monnier&lt;br&gt;Phone: +33 2 99 27 30 57&lt;br&gt;Email: <a href="mailto:raoul.monnier@thomson.nett">raoul.monnier@thomson.nett</a></td>
</tr>
<tr>
<td>TXT Polymedia SPA</td>
<td>Sergio Gusmeroli&lt;br&gt;Phone: +39 02 2577 1310&lt;br&gt;Email: <a href="mailto:sergio.gusmeroli@txtgroup.com">sergio.gusmeroli@txtgroup.com</a></td>
</tr>
<tr>
<td>KT Corporation</td>
<td>Myoung-Wan Koo&lt;br&gt;Phone: +82 2 526 6347&lt;br&gt;Email: <a href="mailto:mskim@kt.co.kr">mskim@kt.co.kr</a></td>
</tr>
</tbody>
</table>
Table of Contents

TABLE OF CONTENTS ..............................................................................................................................6

LIST OF FIGURES ......................................................................................................................................8

LIST OF TABLES ..........................................................................................................................................9

LIST OF ACRONYMS ....................................................................................................................................10

1. INTRODUCTION .......................................................................................................................................11

1.1. DOCUMENT SCOPE ..........................................................................................................................11

1.2. DOCUMENT OUTLINE .......................................................................................................................11

2. ARCHITECTURAL STYLES OVERVIEW ..........................................................................................12

2.1. SERVICE ORIENTED ARCHITECTURE .........................................................................................13

2.2. DATA CENTRED .............................................................................................................................16

2.3. DATA-FLOW CENTRED ................................................................................................................17

2.4. CALL-AND-RETURN .......................................................................................................................18

2.5. INDEPENDENT COMPONENTS ....................................................................................................20

3. ARCHITECTURAL REQUIREMENTS ...........................................................................................21

3.1. FUNCTIONAL REQUIREMENTS ..................................................................................................21

3.1.1. Personalised Semantic News .....................................................................................................21

3.1.2. Personalised TV Guide with Adaptive Advertising ..............................................................23

3.1.3. Internet TV in Social Web .......................................................................................................24

3.2. NON-FUNCTIONAL REQUIREMENTS (QUALITY ATTRIBUTES) ..............................................28

3.2.1. Interoperability ......................................................................................................................29

3.2.2. Reliability..................................................................................................................................30

3.2.3. Availability..............................................................................................................................31

3.2.4. Usability....................................................................................................................................31

3.2.5. Security.......................................................................................................................................32

3.2.6. Performance............................................................................................................................32

3.2.7. Scalability...................................................................................................................................33

3.2.8. Extensibility............................................................................................................................33

3.2.9. Adaptability.............................................................................................................................34

3.2.10. Testability..................................................................................................................................35

3.2.11. Modifiability................................................................................................................................35

3.3. SUMMARY ..........................................................................................................................................36

4. NOTUBE ARCHITECTURE ..............................................................................................................37

4.1. HIGH-LEVEL ARCHITECTURE ..................................................................................................37

4.2. DETAILED USE-CASES ARCHITECTURE DIAGRAMS ..........................................................40

4.2.1. WP7.a Architecture ................................................................................................................40

4.2.2. WP7.b Architecture ................................................................................................................42

4.2.3. WP7.c Architecture ................................................................................................................43

4.3. NOTUBE PLATFORM SERVICES CATEGORIES .........................................................................43

4.4. SERVICES CATEGORIES Mappings .............................................................................................46

4.4.1. WP7.a........................................................................................................................................46

4.4.2. WP7.b........................................................................................................................................47

4.4.3. WP7.c........................................................................................................................................48

4.4.4. Beancounter ............................................................................................................................49

4.4.5. Design Considerations.............................................................................................................50

Page 6 of 73
5. SEMANTIC WEB SERVICE BASED INTEGRATION MIDDLEWARE ..............................................51
   5.1. SOAP-BASED WEB SERVICES ..............................................................................................51
   5.1.1. XML .................................................................................................................................52
   5.1.2. SOAP ...............................................................................................................................52
   5.1.3. WSDL ..............................................................................................................................52
   5.2. REST SERVICES ...................................................................................................................52
   5.2.1. REST on HTTP ..................................................................................................................53
   5.2.2. REST Web Services .........................................................................................................53
   5.2.3. REST Services Advantages ...........................................................................................54
   5.3. SERVICE INTEGRATION THROUGH THE SEMANTIC BROKER ........................................54
   5.3.1. Semantic Web Services and IRS-III ..................................................................................54
   5.3.2. Supporting applications and developers through Service Semantics .........................56

6. NOTUBE TECHNOLOGIES .........................................................................................................58
   6.1. SEMANTIC MODELS ...........................................................................................................58
       6.1.1. RDF ...............................................................................................................................58
       6.1.2. RDFS .............................................................................................................................58
       6.1.3. OWL ...............................................................................................................................58
       6.1.4. SKOS ..............................................................................................................................58
       6.1.5. SPARQL ..........................................................................................................................59
       6.1.6. Linked Data ....................................................................................................................59
       6.1.7. Envisaged Semantic Services/Models Backbone ..........................................................60
   6.2. METADATA ..........................................................................................................................61
       6.2.1. Annotation and Enrichment ...........................................................................................61
       6.2.2. Interoperability .............................................................................................................62
       6.2.2.1. Introduction ..................................................................................................................62
       6.2.2.2. Metadata Services ....................................................................................................62
       6.2.2.3. Functional Requirements - Metadata Service ..........................................................63
       6.2.2.4. High Level Service Architecture .............................................................................64
   6.3. USER PROFILING AND CONTEXT MODELS ...................................................................64
       6.3.1. User Activities and Profiles ...........................................................................................64
       6.3.2. WP3 Services and their Deployment within the Architecture .......................................65
       6.3.3. Technologies ...............................................................................................................67
   6.4. SECURITY AND PRIVACY PRESERVING POLICIES .........................................................67
       6.4.1. Privacy and Beanounter .................................................................................................67
       6.4.2. Social and Psychological Privacy Challenges .................................................................68
       6.4.3. User Interface and Privacy ............................................................................................68
       6.4.4. Technical Privacy Challenges .......................................................................................69
       6.4.5. Data Access and Privacy in NoTube ..............................................................................69

7. CONCLUSIONS ..........................................................................................................................71

REFERENCES ...................................................................................................................................72
List of Figures

Figure 1 - SOA Roles .........................................................................................................................15
Figure 2 - SOA Layers .......................................................................................................................16
Figure 3 - Data-Centred Style .........................................................................................................17
Figure 4 - Batch Sequential Style ..................................................................................................17
Figure 5 - Pipe-and-Filter Style .....................................................................................................18
Figure 6 - Main-Program-and-Sub-routine Systems ........................................................................18
Figure 7 - Object-Oriented or Abstract Data Type Systems ..........................................................19
Figure 8 - Layered Systems ............................................................................................................19
Figure 9: Personalised Semantic News ..........................................................................................22
Figure 10 – General Architecture ..................................................................................................38
Figure 11 – WP7.a internal architecture – Service Provider ............................................................41
Figure 12 – WP7.a internal architecture – Home Ambient .................................................................41
Figure 13 – WP7.b internal architecture ..........................................................................................42
Figure 14 – WP7.c internal architecture ..........................................................................................43
Figure 15 – WP7.c Beancounter internal architecture ..................................................................45
Figure 16 – WP7.a and NoTube services categories mapping ..........................................................46
Figure 17 – WP7.b and NoTube services categories mapping ..........................................................47
Figure 18 – WP7.c and NoTube services categories mapping ..........................................................48
Figure 19 – WP7.c – Beancounter and NoTube services categories mapping .............................49
Figure 20 – Web Services technologies [6] ......................................................................................51
Figure 21 - SWS-based Broker as part of the NoTube Architecture ..............................................57
Figure 22 - Reference Ontology ......................................................................................................60
Figure 23 – WP3 logical interactions within the NoTube architecture ............................................66
List of Tables
Table 1 - Quality Attributes Impact ................................................................. 36
Table 2 – NoTube Layers Mapping ................................................................. 40
List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMI</td>
<td>Ambient Intelligence</td>
</tr>
<tr>
<td>EPG</td>
<td>Electronic Program Guide</td>
</tr>
<tr>
<td>FOAF</td>
<td>Friend Of A Friend</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>PEPG</td>
<td>Personalised EPG</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDFS</td>
<td>RDF Vocabulary Description Language</td>
</tr>
<tr>
<td>SEE</td>
<td>Semantic Execution Environment</td>
</tr>
<tr>
<td>SESA</td>
<td>Semantically Enabled Service Architecture</td>
</tr>
<tr>
<td>SKOS</td>
<td>Simple Knowledge Organisation System</td>
</tr>
<tr>
<td>SOA</td>
<td>Service-Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SPARQL</td>
<td>Simple Protocol and RDF Query Language</td>
</tr>
<tr>
<td>STB</td>
<td>Set-Top Box</td>
</tr>
<tr>
<td>SWS</td>
<td>Semantic Web Services</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description and Discovery Interface</td>
</tr>
<tr>
<td>VOD</td>
<td>Video On Demand</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Service Modeling Ontology</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
1. Introduction

1.1. Document scope

This document presents the NoTube specification in terms of functional and non-functional requirements, leading to the definition of a general architecture that can be customised and verticalised to support the three NoTube usecases.

Individual Use Cases design and specification are out of the scope of WP6 and are performed in WP7, though they clearly inspire and have an impact on the design choices in WP6. In light of this the rationale for the NoTube architecture design process should be twofold:

- on the one hand, to design a generic, integrated platform that can support vertical customizations as those required to develop the three WP7 use cases;
- on the other hand, to define a reference framework that accommodates and orchestrates services developed in WP1-5 in accordance with SESA (Semantically Enabled Service Architecture) principles.

The second issue of this deliverable, due M13, presents a refined architecture with higher details about updated use case scenarios and services (WP1-5). During the first year of the project the technological partners designed and developed the first round of services defining connection interfaces for the planned one, basing on the updated requirements. These services are the building blocks of the NoTube system. In parallel, a detailed vision of the three use case scenarios has been produced as well as the corresponding prototypes.

1.2. Document outline

At a high level the reference architectural paradigm for NoTube is Service Oriented Architecture (SOA); this can be complemented with the introduction of principles derived from different architectural styles presented and analysed. In Section 2 we review the SOA paradigm as a reference starting point and then we survey other commonly adopted architectural styles, analysing their related main principles to evaluate their adoption in NoTube in light of potential benefits and drawbacks.

In Section 3 we present functional requirements derived from an analysis of the use cases scenarios (WP7). We also present non functional requirements (Quality Attributes), identifying their relevance and their impact and priority on the project architecture.

Section 4 describes the NoTube general architecture, distributed between Service Provider and Home Ambient, followed by the detailed internal architecture designed for each of the three application scenarios In addition to that common services categories have been identified and mapped on the individual architectures.

In Section 5 we introduce the Semantic Web Services technology as part of the integration middleware. The role of the Semantic Broker as a smart services orchestrator is described, together with commonly adopted middleware technologies and standards.

Section Error! Reference source not found. is a synopsis of NoTube adopted standards and models with focus on metadata and user-oriented services.
2. Architectural Styles Overview

The term “Architecture” is very generic. For a System Architecture there is no standard, universally-accepted definition of the term, although its roots run deep in software engineering. System architectures are often presented as layered hierarchies that tend to commingle several different structures in one diagram. In the 1970s Parnas pointed out that the term "hierarchy" had become a buzzword, and then precisely defined the term and gave several different examples of structures used for different purposes in the design of different systems.

Describing architectural structures as a set of views, each of which addresses different concerns, is now accepted as a standard architecture practice [37]. We will use the word "architecture" to refer to a set of annotated diagrams, logical schemas and functional descriptions that specify the structures used to design and construct a system. In the software development community there are many different forms used, and proposed, for such diagrams and descriptions [38].

The architecture of a program or computing system is the structure or structures of the system, which comprise software and hardware elements, the externally visible properties of those elements, and the relationships among them [1].

The architecture of a program or, more generally, an IT system is the structure or structures of the system, which comprises software and hardware elements, the externally visible properties of those elements, and the relationships among them. "Externally visible" properties are those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.

So, what is an architecture?

- The set of significant decisions about the organization of a software system
- The selection of the structural elements (components) and their interfaces by which the system is composed
- The behaviour of the structural elements as specified in the collaborations among those elements
- The composition of these structural and behavioural elements into progressively larger subsystems
- The architectural style that guides this organization (i.e. these elements and their interfaces, their collaborations, and their composition).

In the definition above, we assume that components can be made in turn by components (subsystems).

The intent of this definition is that a software architecture must abstract away some information from the system (otherwise there is no point looking at the architecture, we are simply viewing the entire system) and yet provide enough information to be a basis for analysis, decision making, and hence risk reduction. Moreover, properly designed system architecture, will definitely provide support to:

- Represent a solution to the needs of a number of stakeholders;
- Satisfy the requirements of its users;
- Provide added value to its environment.

Components

The architecture embodies information about how the components interact with each other. This means that architecture specifically omits content information about components that does not pertain to their interaction. Moreover it appears clear that systems can comprise
more than one structure, and that no structure holds the irrefutable claim of being the architecture. Thus software components can be services, processes, libraries, databases, commercial products, and more.

**Behaviour**
The behaviour of each component is part of the architecture, insofar as that behaviour can be observed or discerned from the point of view of another component. This behaviour is what allows components to interact with each other, which is clearly part of the architecture.

**Architectural Style**
We have already mentioned the need to organize the system into structures, each defining specific relationships among certain types of components. The architect's chief focus is to organize the system so that each structure helps answer the defining questions for one of the concerns. Key structural decisions divide the product into components and define the relationships among those components. Roughly speaking the approach applied to make such decisions is part of the Architectural Style.

In NoTube the final outcome of this process, namely the system specification and the definition of the architecture, described in Chapter 4, addresses the concerns that are directly derived by the envisaged application requirements (both functional and non-functional) depicted in Chapter 3. Provided these elements, the fundamental starting point on the path to the system architecture appears to be the evaluation of the best architectural style.

In the following paragraphs we intend to present the set of possible architectural styles[2] analysed during this process, in order to highlight possible different approaches and finally focus on the features that best suit the NoTube requirements.

An architectural style consists of a certain number of key features and rules for combining those features so that architectural integrity is preserved. It is thereby determined by:

- A set of component types (i.e., data repository, a process, a procedure) that perform some function at runtime
- A topological layout of these components indicating their runtime interrelationships
- A set of semantic constraints (for example, a data repository is not allowed to change the values stored in it)
- A set of connectors (i.e., subroutine call, remote procedure call, data streams, sockets) that mediate communication, coordination, or cooperation among components.

2.1. **Service Oriented Architecture**
A Service Oriented Architecture[3] (SOA) is an architectural style aimed at supporting the creation of network-based systems around the concept of "software as a service". A system built following the SOA paradigm is composed by applications, called services, well defined and independent from each other, hosted by different machines within a network. Each service provides a certain feature and can leverage on the other exposed services in order to build applications with a greater level of complexity.

SOA is a particular flavour of the concept of Distributed System. The latter is defined as a system built up by different software agents that must work together to complete their tasks. Moreover the software agents in a distributed system typically don't work within the same computational environment, so they must communicate through stacks of hardware/software protocols, over a network. This implies that communications are intrinsically slower in comparison to the one performed via direct invocation and shared memory spaces. From an architectural point of view this forces the developers to consider latencies as an important
factor together with the issues related to the concurrency and the possibility of having partial faults.

The SOA abstraction is not bound to any specific technology; it simply defines some properties aimed at promoting reusability and integration in a heterogeneous environment that all the services building up the system must respect. In particular a service should:

- Be searchable on the basis of its interface and reachable at run-time level. The service definition based on its interface makes the interaction with the other services independent from the implementation.
- Be self-contained and modular. Every service must be well defined, complete and independent from the context and the state of the other services.
- Be defined by an interface independent from the implementation. In other words it should be defined in light of what it actually does, without linkage to technologies used for its implementation. This determines not only the independency from the language used in the development but also from the platform and the operating system underneath.
- Be loosely coupled with the other services. Architecture is loosely coupled if the number of dependencies of its components is limited. This makes the system flexible and easily editable.
- Be available on the network by exposing its interface, and be accessible in a transparent way regardless of its physical location.
- Avoid coarse-grained interfaces. It should make available a limited number of functionalities in order not to have a complex controller for the orchestration activity. It should instead be focused on a high level interaction with other services by means of messages. Because of this and since services could be hosted by platforms running different operating systems, it is required that messages are composed using a standard format.
- Be implemented in a way that allows an easy composition with other services. In a SOA architecture the applications are the outcome of services composition. The process of composing low-level services is called “Service Orchestration”.

However, in real life, a SOA-based architecture usually leverages on widely adopted technologies and standards (i.e. Web Services). Please take a look at Chapter 5, Sections 5.1 and 5.2 for an overview of these technologies. Here it’s just worth mentioning that the adoption of de-facto standards greatly improves the openness of the platform improving extendibility and maintenance, resulting in a smoother development process. In NoTube, since both the media and social fields employ a wide and mostly non-standardised variety of metadata formats, we’ve tried to harmonise the solutions with common approaches. WP2 provides a deep analysis of the related state of the art, exploring possible solutions and services.

In the light of the above requirements, SOA applications usually define the following roles:

- Service Consumer – The entity that requests the service. Could be a module, an application or another service.
- Service Provider – The entity that provides the services exposing the interface.
- Service Contract – It defines the format of the request and the related answer.
- Service Registry – Network directory of the available services
Since services must be dynamically searchable and reachable, the Service Contract must be published on a Service Repository from the entity providing the service. The Service Consumer must ask the Service Registry for the Contract related to the requested service that will be used to execute the service through a transport protocol. Following these concepts it is possible to formulate the definition of an application as composition and orchestration of services.

To this overview we could add that the mentioned services must be application-driven and not only technology-driven. This involves that their definition has to be done together with functional domain experts. For them, and for the reusability attribute, it is very important to follow the described concepts of stateless services and coarse-grained approach.

It is worth noticing that, in comparison to CORBA and J2EE, the SOA paradigm does not specify the architecture in a strict sense, but provides basis for wider objectives, namely:

- define a set of rules for the designing services, related interfaces and interactions among them;
- Improve the sharing/communication/coordination of the roles that traditionally deal with Business and the Architects/Developers.

In other terms the SOA paradigm does not introduce new architectural concepts but an extended and integrated vision of architectural principles adopted in the past (i.e.: CORBA, J2EE).

Summarising, the advantages of using SOA are:

1. Correctly address the service design aimed at reusability and integration;
2. Provide independency in terms of the underlying technologies.

The following picture provides an overview of the SOA paradigm using a layered style, useful to better identify, design and orchestrate services:
Figure 2 - SOA Layers

- Bottom layer – Contains operational systems (existing systems like CMS or legacy applications).
- Component layer – Based on container-based or component-based technologies.
- Service Layer: provides services to upper levels basing on components and actors that can be found in the lower levels.
- Business Process Choreography- It’s the layer that composes the services to implement use cases and/or business processes.
- Presentation – It’s the layer that exposes the processes obtained from the previous layer through presentation components.
- Integration Architecture: it’s the infrastructure that enables the access and composition of services through integration mechanisms (Service Bus).
- QoS, Security, Management, Monitoring – Control tools for the infrastructure management.

As we have seen, services should be able to communicate through a Service Bus (SOA Bus). From an architectural point of view the SOA Bus is the communication layer between services. Its main goal is to develop an infrastructure that provides an individual entry point for services through synchronous/asynchronous communication based on messages, intelligent routing, data transformation support and connectivity support.

We’ve identified the role of the Semantic Broker as an additional component enabling the communications between the topmost layer ("Presentation" in the above figure) and the underlying services, with a strong emphasis on automatic discovery regardless of their physical location and temporal availability. Moreover it’s worth exploiting the semantic capabilities developed in WP5 to support issues common to the different scenarios (i.e.: the adaptation of content presentation) by composition of existing services through semantic reasoning.

Because of the abovementioned facts we feel that it’s not correct to identify the Semantic Broker a simple SOA Bus although they both cover a similar role for what concerning the centralisation of services; the Semantic Broker is more to me considered as a powerful individual entry point providing advanced functionalities related to existing services/contents discovery, classification and composition taking into account heterogeneous metadata formats, physical services location and availability.

2.2. **Data Centred**

The goal of Data-Centred architectures is to achieve high quality for data integrability. The term Data-Centred Architectures refers to systems in which the access and update of a widely accessed data store is an apt description.
The core is nothing more than a centralized data store that communicates with a set of clients. The means of communication distinguishes between repositories and blackboards. A blackboard is basically a notifier that sends notifications to subscribers every time a data of interest is updated. Data-Centred styles are becoming increasingly important because they offer a structural solution to illegibility. Many systems, in particular the one built from pre-existing components, are achieving data integration leveraging on the blackboard mechanisms.

The main advantage is that the clients are independent of each other, and the data store is independent of the clients.

The scalability is not an issue since new clients can be easily added with almost no impact on the system architecture. Moreover, changing functionalities within the clients the overall system is not affected. Coupling among clients will lessen this benefit but may occur to enhance performance.

2.3. Data-Flow Centred
The goal of Data-Flow architectures is to achieve a high level of reusability and modifiability. In this style the system could be viewed as composed by a set of transformers working on successive pieces of input data. Data enters the system and then flows through the components one by one, until the final destination (typically a direct output or a data store).

In the batch sequential style (Figure 4), each component is independent; the only assumption is that each step runs to completion before the next step starts. The “batch” of data is transmitted as a whole between the steps. Data processing is the typical application for this style.
For **Pipe-and-Filter Style** (Figure 5) the emphasis is on the incremental transformation of data. This approach is a typical style in the UNIX-like of operating systems.

Pipes are stateless and are simply used to move data between filters. Filters incrementally transform data, using little contextual information without preserving state between instantiations. The only constraint is the way in which pipes and filters can be combined. A pipe has a source end that can only be connected to a filter’s output port and a sink end that can only be connected to a filter’s input port.

The main advantage of this style is the simplicity connected to the limited ways in which they can interact with the environment. Moreover:

- The system maintenance is simplified by allowing the reuse of filters for the same reason. They can be treated like black-boxes.
- The combination of filters, connected by pipes, can be packaged, appearing to the external world as a filter. This allows a hierarchical organisation of the structure.
- Since filters are independent, an architecture based on this style is easily made parallel or distributed.

On the other hand, there’re some drawbacks as well:

- Filters cannot cooperatively interact to solve a problem.
- Performance is frequently poor:
  - Filters typically force the lowest common denominator of data representation. If the input stream needs to be transformed every filter will pay the parsing/unparsing overhead.
  - If a filter cannot produce its output until it has received all of its input, it will require an input buffer of virtually unlimited size. If bounded buffers are used, the system could deadlock.
  - An overhead is generated each time a filter is invoked since it runs as a separate entity/process.

2.4. **Call-And-Return**

The goal of Call-and-Return architectures is to achieve a high level of modifiability and solvability. They have been the dominant architectural style in large software systems for the past 30 years and include a number of sub styles.

**Main-Program-and-Subroutine Systems**

![Figure 6 - Main-Program-and-Sub-routine Systems](image)
This is the classical programming paradigm. The goal is to decompose a program into smaller pieces to help achieving modifiability. Typically there is a single thread of control and each component in the hierarchy gets this control (optionally along with some data) from its parent and passes it along to its children.

**Remote Procedure Call Systems**
This style is composed by a main-program-and-subroutine system that is decomposed into parts, distributed and connected on a network. The goal is to increase performance by distributing the computations and taking advantage of multiple processors. In such systems, the actual processors workload is deferred until runtime, meaning that the assignment is easily changed to accommodate performance tuning.

**Object-Oriented or Abstract Data Type Systems**

![Object-Oriented Style](image)

These architectures represent the modern version of the call-and-return architectures. They emphasize the bundling of data and methods to manipulate and access that data itself. The concept of class abstracts from the internal components, providing black-box services to the other components requesting those services. The goal is to achieve a high level of modifiability. In other words, this bundle is an encapsulation that hides the internal components from its environment. Access to the object is allowed only through provided operations/services, typically known as methods, which are constrained forms of procedure calls. This encapsulation promotes reuse and modifiability, principally because it promotes separation of concerns.

**Layered Systems**

![Layered Style](image)

This paradigm foresees that the components are assigned to layers to control inter-component interaction. In the pure version of this architecture, each level communicates only with its immediate neighbours. Just like for the previous one, the goal is to achieve high levels of modifiability and portability. The lowest layer provides some core functionality, such as hardware, or an operating system kernel. Each successive layer is built on its predecessor, hiding the lower layer and providing some services that the upper layers make use of.
2.5. Independent Components

Independent component architectures are composed by a set of independent processes/objects that communicate by means of messages. The main goal is to achieve a high-level of component independency, as the name suggest, and the modifiability. This is obtained by decoupling various portions of the computations, sending data to each other but typically without directly controlling each other. The messages may be passed to:

- named participants (Communicating Processes Style);
- unnamed participants using the publish/subscribe paradigm (Event Style).

The Event Systems are derived from this approach. Individual components publish data (announcing the information that will be shared). Other components could register to the class containing interesting data (subscription). In this way, when the data is available, the registered participants are invoked and receive the data.

In these systems, usually there is a message manager that manages communication among the components by invoking the interested one when a message for it is available. On the other side hand the components register types of information that they are willing to provide and that they wish to receive. Information is published by sending it to the message manager that is responsible to forward it to all the interested participants.

The main advantage of this paradigm is that it decouples the component implementations from knowing each others' names and locations. Besides event systems, the other sub style providing independent components is the communicating processes style (multiprocess systems).

Client-Server is a typical example. A server exists to serve data to one or more clients, which are typically located across a network. The client originates a call to the server which answers, synchronously or asynchronously, to the client's request. The goal is to raise the scalability level of the platform.
3. Architectural Requirements

3.1. Functional Requirements

This section presents a sketch of the three NoTube use cases (WP 7a, 7b, 7c), through a descriptive overview that focuses on functional requirements. This review will help in the process of defining the main components for the integrated NoTube architecture and understanding the interactions among the components.

3.1.1. Personalised Semantic News

In this use case a service is defined as a set of functionalities and contents given by the system to the end-user. Service Provider rules are defined as the set of editorial and business rules followed by the Service Provider to create the services to be delivered to the Home Ambient. Home Ambient rules are defined as the set of local rules useful to set requirements for the Home Ambient input (i.e.: pre-filtering). User, Device and presentation adaptation rules are defined as the set of delivery rules used by the Home Ambient server on the basis of the available services to create and deliver the NoTube personalized News experience.

In particular, the Personalized Semantic News use case focuses on the design and development of a system for the creation of a set of local personalized News services, able to acquire News Items from generic broadcast streams, understand the meaning of video News Items, understand the physical context in which News Items are going to be shown and apply criteria for matching the user profile with the available News Items.

We foresee to create three main services: the first one is called the “My News Agency” which will provide the user with an automatically generated local News multimedia channel, customised on his/her preferences and device characteristics; the second one will be called the “Alerting News” (former name: “Breaking News”) service which will allow the final user to have a warning for an incoming News Item of his/her own interest in the Home Ambient and finally there will be a “News Find Engine” service allowing the final user to access News Items performing advanced searches on the available News content inside the Home Ambient.

Considering the whole operations flow, the process foresees the Service Provider to extract from a set of available on-air News-related main streams/repositories a set of related metadata contents. Such contents are then segmented in single pieces called “News Items”, related to individual News. Each extracted News Item could be enriched with additional metadata and internal resources automatically generated by specific tools or external one retrieved from predefined (following Service Provider rules) area of the Web (Service Provider enrichment) through the technological services provided by WP4.

The Remote Service Provider builds one or more services according to its own editorial rules (Service Provider rules), taking into account the behaviour of users groups (privacy preserving) and using semantic filtering (stereotype filtering + service filtering). Obtained services are made available for users following specific business models. Each service is a potential input for the Home Ambient.

Only News Items that match input requirements (Home Ambient rules) are stored in the home ambient (service level semantic filtering), each News Item stored in the Home Ambient has a period of expiration defined at Provider Side or at user side and it’s foreseen to be locally enriched with metadata and resources automatically retrieved from the local repository or from predefined (following Home Ambient rules) area of the Web (Home Ambient enrichment) through the technological services provided by WP4.
Home Ambient services can be created by grouping News Items (My News Agency, News Find Engine) or based on raised events (Alerting News). The automatic creation is based on the interaction of the user with the system in a particular ambient, following personalization aspects performed through smart filtering based on user preferences and habits, besides linguistic preferences and dynamic device adaptation (User, Device and Environment rules). For each News Item best-fitting resources are provided to the fruition device. Envisaged devices involved in the use case will be PC-based (including STB) and mobile compliant.

**Figure 9: Personalised Semantic News**

**Functional Requirements**

*Service Provider side functions*

1. Detect News Items
   a. From an audio-visual stream
   b. From internal News TV archives
2. Extract semantic information about the News Item
   a. On the basis of simple metadata about the segment, e.g. with title/description
   b. On the basis of associated raw text, e.g. speech to text
   c. On the basis of analysis of the audio and video stream
3. Enrich the News Item with associated content from the Web or local repositories
   a. On the basis of Service Provider rules
4. Enrich each News Item with additional contents following Service Provider rules
   a. Equivalent contents, e.g. low quality version
   b. Related contents, e.g. same news in different main streams
5. Store the News Items
   a. Create a persistence layer for enhanced contents and enhanced information related to each News Item (Include expiry date and ensure garbage collection)
6. Build a News Item stream
   a. Using basic Service Provider rules
   b. Taking into account behaviour of user groups (and preserving privacy)
   c. Taking into account semantic filtering (stereotype filtering + service filtering)
Home Ambient server side functions

7. Access to services made available by the Service Provider
   a. Access to service feed
   b. Access to main stream feed (PVR-like)
8. Filter the News Item stream at the Home Ambient side
   a. Using Home Ambient rules
9. Enrich each News Item with additional contents [to be deeply investigated]
   a. Equivalent contents, e.g. low quality version
   b. Related contents, e.g. same news in different main streams
   c. User generated content
10. Manage user and context profiling [to be deeply investigated with WP3]
11. Store the News Items
    a. Create a persistence layer for enhanced contents and enhanced information related to each News Item (Include expiry date and ensure garbage collection)
12. Select News Items for the service
    a. Using basic home ambient rules
    b. Taking into account the user profile
    c. Taking into account the device profile ("Sofa TV", "Hand TV")
    d. Taking into account the viewing context
13. Build final services
    a. Personalized sequence of news items (My news agency)
    b. Alert for a "breaking news" item (Alerting News)
    c. Search over the News Items (News Find Engine)
14. Local Service Delivery

Home Ambient Client side functions

15. STB Device (previously referred as “SOFA-TV”)
    a. UI for the integrated NoTube services
16. Mobile Device (previously referred as "Hand TV" Device)
    a. UI for the NoTube services
17. Capture user and context information

3.1.2. Personalised TV Guide with Adaptive Advertising

The Personalized TV Guide with Adaptive Advertising use case focuses on the design and development of a system for the Personalized TV Guide recommending the viewer TV programs and proposing him/her additional content and services including advertising material.

The 7b Use Case foresees three sub-scenarios built around the concept of the Personalized EPG (PEPG):

- Service 1: WP7b.UC.1: Internet PEPG (iFanzy) ads
- Service 2: WP7b.UC.2: Using iFanzy in social context
- Service 3: WP7b.UC.3: Adaptive ad in video

The first one is will provide the user with a personalized advertisement in iFanzy. The second one will allow the user to send his/her friends a notification of a program to his/her interest. The last one will personalised VOD recommendation and advertisements.

Functional Requirements

WP7b.UC.1: Internet PEPG (iFanzy) ads
Internet ads and functionalities
- Set recordings on Internet client
• Availability of IMDB rating in programmes details
• Serving pre-roll advertisement before video-preview
• Upload video file (and meta-tags) as additional information to a programme page

Relating to: EPG Information of Live broadcast on Internet UI

Context: Dutch nationality, Travelling (alone), morning, day before weekend, Going to Turkey, Speaking: Dutch and Turkish

WP7b.UC.2: Using iFanzy in social context
Using iFanzy in social context
• Receive notifications/reminders on the mobile device
• Overview reminders on STB client
• Multi-user login on STB client
• English STB client interface and ads
• Ad to favourites on Mobile client with multi-modal interface
• Set automatically recording through mobile client with multi-modal interface
• “Send to friend” feature on mobile client with multi-modal interface
• UGC upload mention of 'similar programmes'

WP7b.UC.3: Adaptive ad in video
Adaptive ad in video
• Multi-user login
• VOD portal with personalised ad placement and multi-language
• Function to watch video with or without ads
• Serving one of two available video streams

3.1.3. Internet TV in Social Web
Television watching is known and understood to be an intensely social activity, but as channels and other delivery mechanisms multiply, some of the social aspects of watching are decreasing while other opportunities for social interaction are increasing.

Television watching is social in several different senses. People watch programmes together, commonly but not exclusively in the same room. People talk about television, discussing and recommending programmes, commonly but not exclusively face to face. As video delivery mechanisms and channels get more diverse these social “focal points” of common interest start to evaporate.

Simultaneously however, there are an increasing number of opportunities to watch and discuss television with others on the web, and with on demand video services, an increasing number of opportunities to use the web as a way to create links between people via television programmes.

For example, one person may recommend a programme to a friend; they are able to watch the programme on-demand and then those two have something to talk about. At a more complex level software can be used both to recommend programmes based on the interests of a user and thereby connect people; with the availability of data from the social web, these recommendations could be based on friends and on interests. For example, if Davide’s friend Michele has watched Top Gear then this may make it more probable that Davide will like the programme, even in addition to the fact that Davide is interested in cars and Top Gear is primarily about cars.
As video on demand becomes more prevalent, a new kind of data is being created - data about what the user has been doing online, either user-created ("I'm really enjoying EastEnders tonight") or automatically generated ("User Danbri just started to watch BBC1"). This "attention data" can be about video or other items (viewing webpages, going to football matches or concerts). The data may be available in real time (twitter and Identica are good examples of this), or after the event (for example Radio Pop[32] allows you to add information about what you watched after you watched them).

This attention data can be automatically enhanced and connected in order to create better and more numerous links between people. For example, if a real time attention data broker knows that both Jim and Libby are watching a certain programme at the same time, the system could (with their permission) connect them and allow them to talk about the programme. Attention data could also be automatically enhanced to improve the likelihood of a match between people. For example, if a programme is about lions it may also be about sub-Saharan Africa, which additional piece of information may connect more individuals' interests to the programme.

WP7c is about bridging the social aspects of television with the web, creating new focal points around data aggregation services, and reusing and enhancing social and other data to make better links between people based on real time and traditional television, as well as on-demand services.

In the current social web environment, key concepts are lightweight interoperability, data portability and data reuse, via simple interfaces (RESTful APIs). WP7c focuses on these areas in order to make television a more connected experience. It will be built up by two connected services, one based around aggregating existing attention data and user profiles and the other around providing recommendations based on user profiles. This will include integration code to connect existing Web community sites to these social TV services. Relevant technologies such integration include FOAF, Micro-formats, OpenID, Jabber, RSS/Atom.

This use case is about technologies and strategies that allow users to:
- Watch TV socially as video delivery mechanisms and channels get more diverse and numerous
- Link their existing online presence with their role as users of a social TV platform
- Enrich content metadata provided at producer-side
- Take advantage of new recommendation algorithms which can be built using richer programme, personal and group data and shared attention data

and allow content creators to:
- Create new formats for programmes by mixing existing programmes, web content and user-generated content, and personalization for users and groups of users
- Shape the interaction with audience as a feedback loop
- Integrate social features provided by the social networks with personalized TV content delivery
- Develop applications that embed social graph approaches applied to the TV content delivery

In order to address the abovementioned topics, three sub-scenarios have been considered.

**Scenario 1: Recommendations for me on my TV based on my web behaviour**

Jana wants to see recommendations based on her social activity on her TV when she gets home at night. She talks a lot on twitter and facebook about what she watches in the context of her online social life, and so do her friends, and doesn't see why she should have to explicitly tell any system what her preferences are. She wants to see recommendations clearly featured on the user interface of her media centre.
Scenario 2: Do you want to know more?
While watching on her TV Jana sometimes would like more information about a programme. She'd like to be able to mark a programme to come back to it later and find out more about it. She doesn't necessarily want to have her laptop open all the time during this and neither does she want to interfere with the playing of the programme too much as she often watches TV with other people in the same room.

Scenario 3: Add to playlist
Stephen would like to add programmes to his playlist on the media centre for watching later.

Approach
The approach to the technology is guided by the requirements below, but some aspects should be mentioned now.

- Although TV-watching often takes place alone, nevertheless it is also a social activity, encompassing direct interaction with other individuals while either watching in the same place or separated by distance and communicating online. For the first prototypes we focus on cases where there may be someone else in the same room, and so the user doesn't necessarily want to change channel or otherwise disrupt the viewing experience on the TV screen.

- Conversation is king. Content is just something to talk about (http://www.boingboing.net/2006/10/10/disney-exec-piracy-i.htm) Whether social activity is occurring online or offline it's a crucial part of TV, and once communication is online, there needs to be a way for people to refer to the content in emails, social networking sites, texts and so on. Therefore we require ways to refer accurately, uniquely and dereferencably to items of interest, including programmes, contributors, genres, series, brands.

- Connecting the TV to the Web doesn't have to mean showing the Web on the TV screen. Flowing directly from the face-to-face social user assumption, especially coupled with the difficulties of data input commonly experienced with remote devices on a TV, comes the idea that the Web is a useful companion to TV rather than the Web being on the TV screen. Therefore in this first prototype we are interested in companion devices when watching the TV, used to access the web and get information from the TV.

- The TV is Part of the Web, not a Walled Garden. The third scenario illustrates this nicely: if the user sees something on the Web they can't watch now but are still interested to watch later when they have more time, then why should their TV not understand a link? Similarly, the plethora of APIs and protocols that underlie the modern Web and help users store their data at their convenience and where they get most benefit from it could also be applied to TV.

Functional Requirements

Home Ambient (Client) side functions

Media Centre Requirements
The aims are:
- A box that can perform the usual TV functions, including broadcast TV
- Can be modified, for example to use a separate device as a remote or to add new metadata creation functionalities
The requirements are:

- Play live TV & For example via DVB-T
- Play on demand TV
- Display programme guide (EPG) - Show what's on live TV now and in the future ideally in a grid-based (time-based) format.
- Provide an API to video use and navigation parameters - For example XBMC has an http API; MythTV frontend has a sockets based API.
- Have the ability to modify appearance of displays - In particular the programme guide (EPG) to show recommendations, and the on demand playlist to show recommendations and to add items using an API.
- Have the ability to display text onscreen - In order to subtly alert the user when an important action has taken place using the remote.
- Have the ability to notify devices of events - For example, when a user starts watching a programme, the device is notified what she is watching.
- Have the ability to make changes and additions to the code to add prototype additions such as activities streams, privacy options, XMPP and http protocol additions
- Provide an API to these functionalities
- Have secure LAN and non-LAN network access - For updating and querying the system remotely.

Remote device requirements

The aims are:

- A user experience that will work for both iphone and web-on-iphone or web-on-other-small-device
- A remote that does four groups of things:
  - Basic remote functions for playing and navigating broadcast and ondemand TV
  - View and use configuration and privacy preferences for the media centre
  - View and use information about what is on now and later
  - View and use information about what is being watched now

The requirements are:

- Act as a basic remote: Navigate up / down / left / right and select ('ok')
- Bookmark - This should allow the user to enter some information about the bookmark (such as tags) on the remote itself, and this action may optionally have a small effect on the TV (such as a little 'bug' in the corner saying 'bookmark created').
- Add to Playlist from a programme being viewed (such as a trailer for a programme) or from a remote source such as a web page or application
- Access media centre privacy preferences and configuration options - If the device has a high level of trust, it can access the internal configuration of the media centre for that user, in particular privacy and notification settings, taking advantage of being able to input text more effectively on the device than on the TV.
- Notification settings - enable / disable notifications to device
- Connection settings - detect media centres and create and delete ocnnections with one
- Detect suitable media centres and alert the user in a non-intrusive fashion, detect the capabilities of the media centre and securely 'pair' with the media centre
Service Provider functions

Profiler Requirements
The profiler ('Beancounter') allows the user to generate a machine-processible summary profile of their interests, preferences and dislikes using data aggregated from various social media sources that they already use. It is used as input to a recommender (see below) that can suggest programmes to the user, for example in the first scenario by displaying an EPG with highlighted recommendations. Much more detail about the Beancounter profiler is available in D3.1.

- Must allow users to create accounts
- Must allow users to add and remove social data accounts to their Beancounter account
- Must output an RDF profile including brands, programmes, series, genres, contributors of interest
- Must provide access to the raw activities data
- Must provide access to the aggregated activities data
- Must provide secure access to any private data
- Must provide appropriate privacy defaults and a suitable UI to the end user for managing privacy

Personalised Filter (Recommender) Requirements
The recommender or personalised filter is some means to turn a list of a user's interests, preferences and dislikes into a list of available on-demand or broadcast programmes to watch.

- Must take as input a profile as output by the Beancounter
- Must be able to connect to a profile generator securely with a username and password or OAuth token, OR be able to connect to a profile generator with a username for public data
- Must allow retrieval of recommendations suggestions via a service
- Must output a list of URIs with channels, dates of broadcast or availability and rationales (explanation of why something was recommended)

Metadata Requirements
- Have globally unique, dereferencable URIs for channels, programmes and any other objects of interest

Enhancement Requirements
- Concept identification: creating new links between items of interest and other vocabularies, using natural language processing or ontology matching
- Matching different identifiers for the same item
- Matching of schedule and TV-Anytime broadcast identifiers and Web programme identifiers
- Matching user channel or programme availability to any given user, for example using schedule data and location data

3.2. Non-Functional Requirements (Quality Attributes)
Non-Functional requirements of an architecture are expressed through Quality Attributes[4]. Starting from the functional requirements that are directly derived from the envisaged application scenarios, quality attribute requirements help the design of the architecture by considering a set of factors that are commonly faced by IT systems.
From NoTube Description of Work (page 21):

**T6.1 - NoTube Architectural design, which will design the system architecture for the NoTube project, with customizations for the single use cases. The reference architecture is a SOA, distributed between the Content Provider site and the Home Environment site, enriched with content metadata and semantic service descriptions.**

The choice of adopting SOA as the reference architecture paradigm has been made in the light of three main factors affecting the NoTube project:

1. **Heterogeneous services** – The technological partners involved in NoTube are going to develop different categories of services specifically focused on their field of knowledge.
2. **Different technologies involved** – At the lower level, the abovementioned services are going to be developed potentially using different languages, frameworks and software environments (including operating systems).
3. **Services distributed on a network** – Since the Consortium partners are displaced across the Europe it’s reasonable to foresee a certain degree of independency in the development of the individual software modules and technologies, including services. Moreover NoTube itself is based on the concept of network for the information retrieval and exchange.

The focus of this section is aimed at finding implications when designing and implementing a SOA. This section describes these implications for a set of quality attributes in the context of a SOA. In particular we will consider the following attributes:

- Interoperability
- Reliability
- Availability
- Usability
- Security
- Performance
- Scalability
- Extensibility
- Adaptability
- Testability
- Modifiability

### 3.2.1. Interoperability

*Interoperability refers to the ability of a collection of communicating entities (components) to share specific information and operate on it according to a commonly agreed operational semantics.*

Increased interoperability is the most prominent benefit of SOA, especially when we consider Web Services technology. Until the advent of Web Services, there was no standard communication protocol or data format that could be used effectively by systems using different technologies to interoperate on a worldwide scale. Today, mainstream development platforms (i.e.: Microsoft .NET, J2EE), as well as open source alternatives (i.e.: Perl, PHP) provide frameworks to implement Web services. Components implemented in disparate platforms using different languages can interact transparently through a call-and-return mechanism. That is possible because Web Services define the interface format and communication protocols but do not restrict the implementation language or platform. However, the promise of cross-vendor and cross-platform interoperability in Web Services begins to fall short when services start to use features beyond the basic Web Service Definition Language (WSDL) and Simple Object Access Protocol (SOAP) standards. Over the last few years, a myriad of Web services standards has emerged, not implementing the same standards and the same versions, so interoperability may not be as seamless in practice as it is in theory. Recognizing that reality, the Web Services-Interoperability
Organization (WS-I) (http://www.ws-i.org) was chartered in 2002 to promote the interoperability of Web Services across platforms, applications, and programming languages. WS-I publishes profiles that prescribe adherence to a group of specific versions of well-defined standards and has grabbed considerable attention in the industry through its 130 (approximately) member organizations.

Though mitigated by the adoption of a SOA, this quality attribute still has an impact on the NoTube architecture since several different technologies are to be integrated in different environments and scenario. As better depicted in Chapters 4 and Error! Reference source not found. the concept of interoperability is enriched with semantics and the role of the Semantic Broker, as described in the NoTube Description of Work, will be exploited for improved integration and service composition towards common aspects like the content presentation adaptation. Since different sources of information adopting different formats are expected to be integrated in NoTube in order for the various services to work together, the interoperability has a high priority with respect to contents.

3.2.2. Reliability

Reliability represents the probability of components and systems to perform their required functions for a desired period of time without failure with a desired confidence. Reliability, in itself, does not account for any repair actions that may take place.

Some aspects of reliability are important within a SOA. First of all the reliability of the messages exchanged between the applications and the services; then, the reliability of the services themselves. It must be noted that applications developed by different companies could be based on different reliability requirements, for the same set of services. In the same way an application that operates in different environments could have different reliability requirements in each one.

One possible issue related to message reliability are connections breaks that causes messages to fail to get delivered, to be delivered more than once or in the wrong sequence. If reliability is addressed by service developers who are incorporating reliability techniques directly into the services and application, there is no guarantee that they will make consistent choices about what approach to adopt. The alternative choice of using middlewares from different vendors, specific for this purpose, might preclude reliable message exchange between applications and services that are using different message-oriented middleware. WS-Reliability (OASIS Consortium) and WS-Reliable Messaging (developed by IBM, BEA, Microsoft, and TIBCO Software) specifications address these issues. They define four basic assurances that can be combined:

1. in-order delivery – The messages are delivered in the same order in which they were sent.
2. at-least-once delivery – Each message that is sent is delivered at least one time.
3. at-most-once delivery – No duplicate messages are delivered.
4. exactly once – Each message is sent only once.

Concerning service reliability the above means that the service operates correctly (does not fail). It also means making sure that the service is obtained from a reliable provider so that a level of trust in the service’s accuracy and reliability can be established. NoTube will clearly distinguish features that access services and content that are provider-based from that dealing with external resources, therefore it is reasonable to expect that a good degree of reliability can be achieved.

The reliability of communication in terms of expected behaviour is a key factor when thinking about an integrated platform composed by heterogeneous services. However the fact that they will perform properly “for a certain period of time” is not crucial as it would be for a mainstream product. We then consider this attribute to be a medium-priority one.
3.2.3. Availability

Availability is the degree to which a service is operational and accessible when required for use.

Availability of services both from the user’s and provider’s perspectives is a concern for the success of a SOA. From the user’s perspective, if the system relies on a set of services in order to meet its functional requirements, and one of those services becomes unavailable, it could prevent the success of the system. On the other side, from service provider’s perspective, in order for the services to be used they must be available when needed. Service providers usually agree to provide to the service users a set of services and to include each service in an SLA that defines the contract for the provision of the service with details about who provides the service and the penalties to the provider if the service level is not met.

From the above overview, it appears reasonable that such attributes has a high impact on commercial systems such as potential NoTube-based application and services, for which the unavailability of one or more services could determine a loss of money for the provider. The NoTube architecture will be designed to minimise critical services so that failure in one service may reduce overall system functionality but still allow the service to run. Availability is related to an application’s reliability, as well as its maintainability, which in turns depends on the complexity of the system and the degree of interdependence among components. In terms of availability NoTube-based applications and services will benefit directly from the effort to support reliability of individual components in the architecture.

Since the prototype development phase continues till the end of the project, the Consortium expects a floating availability of the achieved services due to the fact that they’re subject to be revised/updated and/or extended. As far as the communication interfaces are properly specified they are no strict needs of having a set of 24/7-available services since it won’t impact the depending development and integration activities. The services brokerage activities could even benefit from temporarily unavailable services in order to test the automatic discovering capabilities foreseen in WP5. We then consider this attribute to be a low-priority one.

3.2.4. Usability

Usability measures the quality of the end-users experience in the process of interacting with information or services.

In order to increase the usability of the system, service providers should consider a couple of things that derive from the distributed service nature of SOA: data granularity (services to support usability), and disconnected operation. In SOAs communications over a network usually introduce delays in user interactions. Depending on the application, the service must not only respond to user requests with the data requested but also with other relevant data that may or may not be immediately displayed. This choice should be taken very carefully trying to minimize the network traffic when not strictly requested, especially when the invocation of a service is directly impacted by the front-end.

Concerning “disconnected operations”, let’s think about a scenario where the service user is a mobile device. It may lose connectivity for periods of time. If the service is implemented in a stateless fashion, re-establishing the context is the responsibility of the service user. Service users should probably cache sufficient data to allow for reduced operation for some time until reestablishment occurs. This is a common case in playing back streamed videos. If, however, the service provides some operations that require persistent data (such as adding items to an e-shopping cart), the service should provide a means of re-establishing context or synchronizing the current state of the client with the state maintained by the service.
We foresee a medium impact of this quality factor in NoTube since the core back end of the services is operated by the application logic not strictly in real-time and without directly impacting on the front-end. However, in user-centred environment such as the home ambient, a certain degree of user interaction is foreseen so it is worth considering the abovementioned guidelines in the implementation of the directly involved services.

The presentation layer in NoTube is bound to the different GUIs envisaged for the different devices according to the use case scenarios details. Although it’s not a focal point of the project we consider this attribute as a medium priority one because presentation adaptation aspects are involved to support a seamless content fruition mainly for audiovisuals in particular in WP7.a and WP7.b

### 3.2.5. Security

Security is a complex topic, usually denoting different aspects with respect to software systems. In general it is associated with:

1. **Confidentiality** – Access to information/service is granted only to authorized subjects.
2. **Authenticity** – We can trust that the indicated author/sender is the one responsible for the information.
3. **Integrity** – Information is not corrupted.
4. **Availability** – The information/service is available in a timely manner.

More specifically for SOA and Web Services/REST-based Services, security is a major concern. Developers should pay attention to some characteristics that are inherent to SOAs and directly impact security:

- Messages often contain data in text format (i.e. XML). That means that a malicious user intercepting a message may clearly read the content. Encryption must be adopted to preserve privacy.
- A system built using an SOA approach may leverage on third-party organizations. The identity of the external service provider has to be authenticated.
- Services could have access restrictions based on the identity of the user. An authorization mechanism should be in place to support this scenario.

The downside to be considered is that security mechanisms often have a negative impact on performance and modifiability. In addition to that encryption, authentication, and authorization imply to consider the configuration required for the infrastructure of the chosen technology with respect to security (firewall rules, certificates, etc.).

With reference to the functional requirements described before, NoTube in some cases foresees the exchange of confidential information limited to user profiles and preferences despite the technical partners are already trying to minimise this behaviour in an effort to protect the user privacy (see chapter 6.4 for more information). We thus expect the security factor to have a medium impact on the architecture.

In particular when talking about user profile management in the light of social networks integration, the privacy preservation is an important factor. Since WP3 activities towards these objectives are part of the expected work in NoTube we consider the security as a medium priority attribute.

### 3.2.6. Performance

*In general performance is related to response time, throughput, or timeliness.*

This is an important quality attribute that is usually affected negatively in SOAs because of the overhead generated by the messages exchange. The key factors in SOA that contribute to performance issues are:
• The need to communicate over the network increases the response time. Typical networks do not guarantee deterministic latency (i.e.: Internet). Thus SOA is not suited for near real-time systems.
• The ability to make services on different platforms interoperate seamlessly has a performance cost: intermediaries are needed to perform data transcoding.

Generally speaking the core of NoTube architecture does not present real-time functional requirements although information enrichment (metadata) and the semantic processing as well as multimedia processing could create bottlenecks in the overall response time and affect the user’s experience. This, of course, must be considered and controlled. We classify this quality attribute to have a medium impact.

We don’t expect the 1st NoTube prototype to be the best performing one because the final goal is not to develop a product; optimisation and re-engineering could be applied in a second time so we consider the performance attribute as a low priority one.

3.2.7. Scalability

 Scalability is the ability to maintain performance while system demand increases, or continue to work without degradation of the other quality attributes when the system is changed in size or in volume in order to meet different needs.

Research is still working on how to address the scalability issues related to SOA. The major issue here is the capacity of the system hosting a set of services to accommodate an increasing number of them avoiding degradation of performance.

Options for solving scalability problems include:
• Horizontal scalability: distributing the workload across more computers. Doing so may mean adding an additional tier or more service sites.
• Vertical scalability: upgrading to more powerful hardware for the service site

From an architectural point of view it is a good practice to think about the prototype developed in a research project as a system that could be re-engineered and thus expanded/updated with other components or services. Usually the solution adopted is the vertical one because of the fact that hardware processing power tends to increase year by year and costs reduce as well.

At service provider side applications originating from NoTube use cases may potentially serve many users. The NoTube architecture should therefore be designed to keep scalability into due account. This is enforced also by the need of flexibility to enable response to upcoming technologies particularly in the social Web field (see section 3.1.3 for more details). We consider this quality attribute having a high impact.

However, during the prototype development phase it is quite common to limit the accesses for testing purposes, moreover the prototype development needs however to be re-engineered in terms of software and hardware deployment to became a mainstream product capable of handling a certain target number of users (depending on the stakeholder). Because of the previous reasons we consider this attribute to have a low priority in our research.

3.2.8. Extensibility

 Extensibility represents the ease with which the services capabilities can be extended without affecting other parts of the system.

In modern architectures this parameter is important because the business environment in which a software system lives is continually changing and evolving. Such changes of course imply modifications in the software system as well. Extending a SOA means making changes that include extending:
• The architecture, to add additional services. SOAs allow for the easy addition of new services by its own nature (services can be created and published by the providers and discovered by service users)
• Existing services, without changing the interfaces. Loosely coupled services could be easily extended without changing the service interface.
• Existing services, with changes to interfaces. Here the major impact is on the service user: usually an application learns about a service interface by reading information provided by the directory provider, and the interface may change over time. The service users' application must be able to handle any changes to the interface.

A major obstacle to extensibility is the interface message that must be written in a format, structure, and vocabulary understood by all parties to minimize this issue. Limiting the vocabulary and structure of messages is a necessity for any efficient communication although it comes at the expense of reduced extensibility. Restriction and extensibility are deeply linked. Both are needed, and increasing one comes at the expense of reducing the other. Tradeoffs between them are necessary to achieve the right balance.

The NoTube Description of Work has already split up the main services in different building blocks composing the architecture so a potential extension could be easier to manage by focusing on a subset of the whole platform functionality. Moreover the adoption of a semantic brokerage of heterogeneous services together with metadata conversion tools will mitigate risks. The three NoTube use cases demonstrate the degree of extensibility of the integrated NoTube architecture. Such attribute is regarded to have a medium impact on the project.

It's worthless to mention that from a developer perspective the priority of this attribute is considered high since the NoTube baseline services will be provided during the whole project period at different points in time, basing on the research achievements.

3.2.9. Adaptability

Adaptability is the capability of a system to withstand and easily adapt to changes in its environment, requirements and implementation technologies.

The SOA approach brings various benefits to the ability to adapt by allowing the following:
• Services can be built and deployed using the principles of location and transport independence. If a service needs to adapt, discovery and binding should be automated and not require a change in the application.
• Use-cases that are modelled using services can be adapted, and those services can be combined in new and different ways. What will require changes is the module composing these services.
• Developed services that must operate on different platforms, in different computing environments, using different combinations of sensors, multiple diverse communication protocols, human-computer interaction and applications must be “configurable” to the environment in which they will reside.

To achieve adaptability the services will need to be managed and monitored properly as a single cohesive solution and the interaction between the service and the underlying infrastructure will have to be managed as well.

In NoTube this is exact the purpose of the resource broker described in the DoW, so we are already going to manage this aspect as one of the main building blocks of the architecture. Impact of Adaptability in NoTube can be regarded as low as developed services and overall architecture are expected to be potentially able to accommodate customisations to support novel scenarios as well.
Due to the wide range of thematic involved in the three foreseen use cases we set this attribute as a low priority one because intrinsically part of the project goals.

3.2.10. Testability

Testability is the degree to which a system enables the establishment of test criteria.

Testing a SOA system can be complex for many reasons like:

- Interactions may be required between distributed modules.
- The companies involved may not be able to access the services source code. This problem occurs when services are external to the considered organization.
- Services may be discovered at runtime, so it may not be possible to predict which service is actually used by the system until the system is executing.

If a problem occurs when the system is running, it may be difficult to find the source of the problem. The problem may be:

- Within the application
- Within a service that is being used by the application
- Within the infrastructure that is used by either the application or the service
- Within the “discovery agent” that locates the service

Because of the above reasons, service developers need to build additional tools that support the testing and debugging processes of both the services. Some tools are available to support the individual testing of services, some other to simulate the integration testing when services are used in an application. This should be considered as important guideline for NoTube as well in the light of the prototype integration. This attribute has a medium impact in the project.

We expect the technological partners providing innovative services and, more in general, every developer in the project to provide already tested services in terms of interfaces and behaviour. Some of them may require specific testing tools that could be just internal to the developer, some may require to provide these tools to the external module integrating the considered service; in both cases we face the testability of both the services individually deployed, for which we set the priority as high, and the testability related to the integration within the prototypes, for which we envisage a slightly lower level of priority.

3.2.11. Modifiability

Modifiability represents the possibility to make changes to a system quickly and cost-effectively.

SOA promotes loose coupling between service consumers and providers that contributes to the creation of few, well-known dependencies between services. The cost for modifying the implementation of services is then reduced. However, if modifications need to be applied to interfaces, the change may raise problems because once service interfaces are published and used by applications, it can be difficult to identify who is using a service and what impact changing its interface will have.

NoTube architecture design relies on well defined interfaces therefore we regard this attribute to have low impact.

For the same reason during the prototyping phase it's important to clearly define services, beginning from the exposed interfaces. The modifiability is then reached by transparently changing the services implementation, as a natural process of the research activities. This attribute has a medium priority because implicitly considered by the nature of the SOA architecture.
### 3.3. Summary

The following table provides a summary of the impact analysis performed on architecture quality attributes. For each attribute we identify:

- the level of risk on a generic SOA: this indicates how difficult it can be for a SOA to support the quality attribute;
- impact on NoTube: this indicates the relevance of the quality attribute in NoTube and how critical it is in architecture design.
- priority in the research and development activities aimed at producing integrated prototypes.

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Risk on generic SOA</th>
<th>Impact on NoTube</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interoperability</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Reliability</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Availability</td>
<td>Medium</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>Usability</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Security</td>
<td>High</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Performance</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>Scalability</td>
<td>Medium</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Extensibility</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Adaptability</td>
<td>Medium</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Testability</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Modifiability</td>
<td>Low</td>
<td>Low</td>
<td>Medium</td>
</tr>
</tbody>
</table>

**Table 1 - Quality Attributes Impact**
4. **NoTube Architecture**

Basing on the analysis of the updated functional and non-functional requirements of the platform in this chapter we will describe the envisaged NoTube architecture applied to the individual use cases, designed according to the high level vision of the integrated architecture introduced in the first issue of this document.

The main goal is to highlight the building blocks (components) composing the NoTube platform as well as how they're connected together, providing reference to the involved workpackages.

A more specific description about the envisaged functionality (services) for each component will be given in Chapter 5.

4.1. **High-Level Architecture**

To support the functional requirements of the platform (Section 3.1) directly derived from the application scenarios (WP7) at M3 it appeared reasonable to consider 2 main building blocks for the NoTube system:

- The Service Provider
- The Home Ambient

Following this approach we sketched the following general architecture diagram.
Figure 10 – General Architecture
The idea is to prepare at Provider-side the TV contents to be delivered to the end-user, living in the Home Ambient. With the term preparation we mean:

- the ingestion of TV contents (i.e.: Newscasts, advertisement, etc.)
- the production of content-related metadata (annotation and enrichment)
- implementation of providers’ business policies
- application of generic user-context profile categories.

These activities require a subset of the platform services that can be grouped in:

- User and Context services (generic user/context profile categories)
- Content Annotation and Enrichment services (meta-tagging of TV contents)
- Metadata Conversion services (aimed at adapting metadata to professionals/non-professionals)
- Model and Semantic services (common background for metadata annotation and enrichment)

On the other side, the home ambient represents the physical place where the ambient intelligence (AmI) is implemented, considering the user as the central actor. NoTube receives contents from the provider, applying intelligent filtering based on user’s personal preferences (user and context modelling) and then delivering the final output to the chosen channel (i.e.: STB, PDA, etc.). The Home Ambient is the physical place where the end-user lives and interacts with NoTube, thus from an application point of view, it could simply be a client module/front-end or a complete home server tailored to the specific goals of the considered use case. For instance, with reference to WP7 and the detailed use case architecture diagrams described below, in the 7.a the user relies on an in-house home server (typically a standard PC), responsible to record and store Newscasts, collect news metadata, filter them according to his/her own preferences and habits, etc. In 7.b the Home Ambient just foresees the device(s) used to watch TV programmes, ads, etc. (i.e.: STB, iPhone, ...) capable of running the iFanzy front-end, in 7.c it’s represented by an Open Source media centre and a remote device.

In both the environments we can find the Semantic Broker. This component supports the architecture interconnection on 2 levels:

1. Discovers contents/services and their availability;
2. Smartly composes available services that could be used by the upper level in order to perform more complex tasks through the concept of “goals” (i.e.: presentation adaptation aspects, scenario-specific integration requirements)

The architecture’s components have been grouped in categories and assigned to logical layers. This has been done to help understanding the overall vision and to promote a kind of mapping between the general SOA-based layered architecture previously presented in Figure 2 (Section 2.1) and the NoTube general architecture, in Figure 10.

Of course there’re some differences/extensions that are due to the specific structure of the work plan, the introduction of concepts like automatic service discovery and semantics as well as the need of implementing three different use cases.

The following table maps the NoTube architecture layers with the one presented for a generic SOA-based architecture, provided that WP6 is considered to cover the role of the cross-layer named “Integration Architecture” with security features (WP6 task 6.5):
<table>
<thead>
<tr>
<th>NoTube Layers</th>
<th>Generic SOA Arch. Layers</th>
<th>WP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal/External Contents&amp;Services</td>
<td>Services Layer, Components Layer</td>
<td>WP1, WP2, WP3, WP4, WP5</td>
</tr>
<tr>
<td>Middleware/Broker</td>
<td>Business Process Choreography Layer</td>
<td>WP5 (it adds automatic discovery of services/contents and semantics orchestration)</td>
</tr>
<tr>
<td>Presentation and Business Logic</td>
<td>Presentation Layer, Existing Applications (bottom layer)</td>
<td>WP7a, WP7b, WP7c (includes application logic and internal services tailored on the specific application scenario)</td>
</tr>
<tr>
<td>Delivery Layer</td>
<td>-</td>
<td>This is NoTube-specific. It comes from the common requirement of having a Service Provider ambient and a Home ambient together with presentation adaptation aspects for contents and different devices</td>
</tr>
</tbody>
</table>

Table 2 – NoTube Layers Mapping

4.2. Detailed Use-Cases Architecture Diagrams

This section shows the architecture details for the individual use cases, namely:

- Personalised Semantic News (WP7.a)
- Personalised TV Guide with Adaptive Advertising (WP7.b)
- Internet TV in Social Web (WP7.c).

The presented diagrams reflects the envisaged plans for the final prototype providing an architecture description that shares the common guidelines depicted in the first issue of this document, adding specific requirements proper of each individual application scenario.

4.2.1. WP7.a Architecture

The Personalised Semantic News use case tightly adheres to the general architecture diagram, making a clear distinction between the service provider and the home ambient. The first one prepares the Newscasts and related contents (metadata) following a sequence of steps such as: ingestion, semantic enrichment, content enrichment, item selection, content&service delivery. The output of this block represents the input for the home ambient side, the physical place where the end-user lives and consumes the NoTube experience. Moreover the service provider maintains a persistency layer for the published contents.
The home ambient introduce the concept of home server as the hardware entity capable of ingesting either News items or complete Newscasts coming from the service provider, organise them internally (filtering), managing the personal user profile in terms of static details like username, password, sex, age, etc. as well as dynamic one including his/her habits derived from user activities logging and direct feedback (AmI) provided by the user through his/her interaction with the NoTube front-end. The home ambient server back-end
will enable the front-end to manage local user profiles, to consume personalised Newscasts on the basis of personal interest and behaviour, regardless of the chosen delivery channel (PC, STB, mobile), presenting recommended or linked contents.

Semantic brokering of service is foreseen to help solving presentation adaptation aspects related to the adoption of different delivery channels with different features and hardware capabilities.

More details about the above figures are included in D7a.1, chapter 3.2.

4.2.2. WP7.b Architecture

The personalised TV guide with adaptive advertising use cases leverage – provider side - on the adoption of the iFanzy server providing a set of services for filtering audiovisual contents by synopsis, person, genre, type, service/channel, resource and giving the possibility to customise the response groups (the amount of data returned). iFanzy is augmented with a subset of services made available by the NoTube platform, through the adoption of the semantic broker in order to inject advertisements tailored to the specific profile of the considered user.

The “home ambient” is represented by the different front-ends developed for:
- The Web
- iPhone
- Set-Top Box

Such front-ends foresee to enable the user to access the iFanzy server, in particular to consume audiovisual contents (TV-programmes with adaptive ads) and provide direct feedback to the system.

The following architecture diagram summarises the exposed concepts.

**Figure 13 – WP7.b internal architecture**
4.2.3. WP7.c Architecture

The Internet TV in social web scenario’s is practical and foresees to identify common information models that can be used to inter-related information about TV content regardless of source, beginning from the BBC’s internal knowledge and repositories. The NoTube network, then, will be a virtual and decentralised network of users and their contributions, spanning all modern TV systems touched by the Web including smartphones, linking broadcast content with improved program guides, downloadable and purchasable content (via media centres), and of course user generated content: critiques, parodies, analysis etc. Such an approach brings the unifying and link-oriented approach of the Web as a replacement for television’s increasingly archaic ‘channels’ metaphor and increasingly fragmented online presence.

![Figure 14 – WP7.c internal architecture](image)

The above architecture diagram provides a detailed overview of the considered scenario. The concept of home ambient encloses potential consumer-oriented devices with related interfaces and connection protocols, through which the user can interact with the platform. On the other side, the service provider implements the business logic aimed at monitoring and collecting the user social activities (Beancounter) then adding content enrichment and personalised recommendation functionalities thanks to the integration of a subset of the NoTube platform’s services. Of course this process is not trivial since requires connecting to heterogeneous social services and aggregating the acquired information adopting a unified user model to enable further enrichment and processing for recommendations.

4.3. NoTube Platform Services Categories

From the analysis of the updated use cases scenarios together with the planned activities to be performed by the technological workpackages during the whole project period, we’ve identified common requirements that led to the classification of a set of services that each
use case foresees to integrate, in a subset of them, in order to demonstrate specific capabilities of the platform.

These services can be grouped in 3 main categories:

- User oriented
- Content oriented
- Metadata oriented

As illustrated in the picture above, the Application Logic makes use of these services enabling the user interaction through one additional module, namely the “GUI”, that is expected to develop specific HMI layouts for the considered delivery channels (PC, STB, PDA) taking into account presentation adaptation aspects related not only to the screen size and the hardware limitations of the device but also the consumption of audiovisual contents in terms of format, resolution and quality.

The Semantic Broker is somewhat hidden in this classification because it has a cross-functional role. In particular it could be exploited, depending on the use case focus and priorities, to seamlessly integrate metadata and content oriented services into the application logic giving the possibility to search for the most suitable services by specifying “goals” (improving the extensibility of the platform regardless of the services availability and physical location), on one side, and also to leverage on services composition adding semantic capabilities to help resolving the presentation adaptation issues related to A/V contents and devices.

**User Profile Management**
Since the user profile management is heavily involved in all the planned use cases and since the central module aimed at providing it, namely the BeanCounter, is not so immediate to understand, we also would like to provide an internal architecture diagram for that particular component, trying to describe it more depth.
Each Web source (i.e.: friendfeed, twitter, last.fm, etc.) is wrapped and accessed by a “Tubelet”, built specifically to handle data from the considered source. All the logics that enable the tubelets is embedded in the “Tubelet container”, that manages the life cycle of each Tubelet.

Three external components take into account the scheduling of the activation of each Tubelet (“Scheduler”), the unique identification of the user, mapped to all the different credentials of the user throughout the Web (“User ID manager”) and the computation of the range of data to be pulled out of the source. The Tubelet takes in input the result of the API of the considered source, representing some information about the user within that Web source. Thus the Tubelet parses that result and creates some software model (i.e.: Java Beans) that maps the data pulled from that source. Such models are serialised by the Tubelet and passed as input to a “Pipeline”.

A Pipeline is a pipe with a number of steps which processes the serialised beans to produce an semantic-oriented representation of the data (i.e.: using RDF format). Eventually that content will feed the “RDF storage”. Apart from this mandatory step regarding the conversion, other additional steps could be placed after this conversion in order to perform some ad-hoc actions over this semantic source-specific representation of the user.

The described data flow is oriented to a specific Web source. Data are pulled directly from the source and some automatic lifting is made in order to get a semantic representation of the source content.

Additionally, a model-driven flow is foreseen by the system, instead of just the source-driven one just mentioned. A “Modelet” is the equivalent of a Tubelet in this second flow. A Modelet can accept a structured and very sharp-cut piece of information, regarding the user in a cross-source fashion (i.e.: movies recently watched by the user). Every time a new piece of information is needed, a new Modelet can be hot-plugged into the “Modelets container”. A Modelet feeds then into the proper software structure counterpart which, as for the first flow, is serialised and passed to a pipeline, in order to be further processed and potentially stored.

Finally, a statistic component is responsible for computing all kinds of statistical measurements over the whole subsystem.
Each component could be seen as a black-box providing a set of API that allow the third parts developer to easily integrate the Beancounter, if required.

### 4.4. Services Categories Mappings

The next three sections will provide a mapping between the use-case internal architecture and the identified services categories, giving a more precise idea of the contact points between the prototypes and the NoTube baseline services. For more details about the internal workflow please see WP7.a/b/c reports.

#### 4.4.1. WP7.a

The diagram below provides an overview of the complete envisaged data flow within the Personalised Semantic News scenario.

The main integration activities for the service provider are focused on metadata oriented services, in particular concerning the metadata framework for News modelling, format conversion and metadata injection/extraction.

Content enrichment at metadata level is expected to be integrated in second instance through services developed by WP4 during the project, as part of the research activities.

At the home server side, on the other hand, again it’s foreseen to adopt WP2’s services for metadata conversion and management in order to handle the locally stored News items, but the main focus is on the user oriented services. Such services are not only targeted to manage user authentication, creation, deletion in terms of static information (name, surname, age, sex, etc.) and explicit preferences but are also required to provide contact points with the dynamic profile (the Beancounter concept) coming from user activities and habits logging through his/her social Web interactions as well as through his/her direct feedback derived from the usage of the NoTube platform itself (i.e.: the user may often skips sports News).
WP3 is also expected to provide recommendation services based on the abovementioned information in order to provide an effective way of evaluating how good a piece of News is for a particular user with respect to external sources of potential interest.

In this scenario the interaction with the user is foreseen at home side, through the desired device. The GUI interfaces with the application back-end integrating the just mentioned services thanks to a set of event handlers proving connection for the exposed functionalities (i.e.: playback the personalised Newscast, skip a particular piece of News, search for specific News, etc.). The interesting aspect involving semantic brokerage, at this level, is the resolution of available audiovisual contents in terms of best quality for the chosen device.

4.4.2. WP7.b
Similarly, for the Personalised TV Guide with Adaptive Advertising scenario, the next picture gives an overview of the mapping between the internal architecture and the NoTube services.

WP7b: Architecture

![Figure 17 – WP7.b and NoTube services categories mapping](image)

Here the role of the iFanzy server is a core part for the use case’s envisaged functionalities. The GUIs (top of the figure) have to be considered iFanzy clients, designed to enable a seamless integration with the iFanzy back-end.

The integration with the NoTube baseline services foresee to adopt the semantic broker as the only entry point. The idea behind it is to improve the functionalities already provided by iFanzy with content enrichment and recommendation provided by NoTube (WP3 and WP4). This process is made possible by services composition and orchestration, operated at WP5 level leveraging also on metadata conversion algorithms (WP2).
The WP7.b application logic thus foresees to empower its results by setting specific goals for the broker, without worrying about the availability of specific low-level NoTube services, their physical location as well as potentially heterogeneous data formats used.

4.4.3. WP7.c

The WP7.c application scenario is centred on the user and his/her social activities thus the internal structure slightly differs from the previous two use cases. The prototype leverages on the Beancounter (please see WP7 documentation for more details). The latter, as could be easily argued looking at the next diagram, represents a complex software module that interfaces with social networks, monitors the user activities and produces additional services acting as recommender for contents, namely: TV programmes, music, etc.

The picture below shows the mapping between the WP7.c internal architecture and the NoTube services categories.

![Figure 18 – WP7.c and NoTube services categories mapping](image)

The Home Ambient box encloses the front-end potentially running on different devices (i.e.: iPhone, Android Phone, STB, etc.). The user is of course enabled to interact directly with the application front-end (i.e.: through a remote control or the mobile device itself). This in turn reflects on having different GUIs in light of the considered delivery channel, similarly to WP7.a and WP7.b. Here, due to the intrinsic goals of the scenario, presentation adaptation aspects are not so focused on audiovisual contents like in the previously described use-cases. What is really important is the connection with the Beancounter back-end module, representing the integration with user oriented services as well as existing TV-related contents repositories (i.e.: the one provided by BBC) to enable interactive programmes rating for the whole community convenience.

The Beancounter is composed, in turn, by services for aggregating heterogeneous contents (mainly user activities) coming from social networks, producing a local unified user model
that could be further augmented with metadata enrichment services provided by WP4 ("Programme data enrichment service". in the picture).
Of course such dynamic user model is then available for statistical processing. This situation may provide, in a second stage, higher-level services for data enhancement.
EPG data sources, as external contents, are integrated as well in the whole process through the semantic brokerage.

Summarising, The WP7.c application scenario is centred on the user and his/her social activities thus the internal structure slightly differs from the previous two use cases. The idea is that the user's remote "companion" device enables secure identification of the user to the media centre (i.e.: MythTV), which then can pass credentials from the device to other services such as the Beancounter and the recommender, which in turn use the various enhancement services.

Under evaluation is the possibility to run the Beancounter and simple recommenders inside the home ambient for enhanced security and privacy.

4.4.4. Beancounter

The following picture is related to the internal architecture of the Beancounter module, providing again a mapping with NoTube services categories.

![Figure 19 – WP7.c – Beancounter and NoTube services categories mapping](image)

Here we're analysing a low-level diagram. The interesting thing is that the Beancounter API internally works on metadata, thus the exposed services are expected to exchange information that are most likely very light in terms of communication overhead and performance impact (in opposition to audiovisual contents). Moreover, at the current stage of development, the API is provided as a set of RESTful services that allow remote integration through the Web with minimal technical effort.
4.4.5. Design Considerations

The SOA paradigm has been widely adopted, focusing on service providers on one side (WP1, WP2, WP3, WP4) and service users on the other (WP7). The Semantic Broker implements semantic composition of underlying services to achieve specific goals.

From an architectural style point of view, just like for “event systems” (Independent Component Architecture) the SOA paradigm appears to include message-management functionalities, handling the communication among the service components. This feature increase the Interoperability which is one of the most impacting attributes for NoTube.

On the other hand the concept of “layered system” (Call-and-Return Architectures) is loosely applied since we can distinguish 4 logical layers for which each level communicates only with its immediate neighbour:

- Services
- Semantic Bus
- Applications
- Delivery

We cannot use the same style for the individual components because the need of combining services requires to independently invoking them at different levels.

A hidden style of this architecture is derived from the fact that the content preparation foresees several sequential steps that could be summarised in:

1. Ingestion and annotation
2. Enrichment
3. Recommendation
4. Delivery with presentation adaptation

This is comparable to one of the principles of the Data Flow Architectures characterized by viewing the system as a series of transformations/enrichment on successive pieces of input data (or metadata). Here we’re not applying this approach to the whole system, but to a subset of it.
5. Semantic Web Service based Integration Middleware

When talking about integration we can distinguish between two levels:
- Technical integration
- Functional integration

The pure technical integration deals with the hardware/software techniques exploited to make the communication process easier and robust. In SOAs the widest adopted technology for the communications among the various components of the architecture are Web Services that could be SOAP-based or REST-based. The reason why we've introduced also RESTful Services is that Internet TV in Social Web described in Chapter 3.1.3 foresees to make a wide use of them. In fact social Web portals often expose RESTful Services because of their lightness in comparison with traditional SOAP-based Web Services. An overview of this paradigm is presented in this chapter as well (Section 5.2).

On the other hand, the functional integration required in NoTube is focused on the automatic discovery, collection and composition of services, thus on the Semantic Broker functionalities envisaged at both the Provider and Home sides. Semantics requires the introduction of other specific technologies and standards. More details about such component will be part of the work performed in WP5. D5.1 sets basic requirements for it while section 5.3 of this Chapter provides an overview of the SWS Broker and the involved technologies.

5.1. SOAP-Based Web Services

Web Services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processible format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards[5].

Web Services involve many layered and interrelated technologies. There are many ways to visualize these technologies, just as there are many ways to build and use Web Services.
The above illustration provides a layered-style overview of these technology families with respect to the widely adopted standards used. The following sections (5.1.1, 5.1.2, 5.1.3) briefly describe the purpose of these standards

### 5.1.1. XML
XML solves a key technology requirement that appears in many places. By offering a standard, flexible and inherently extensible data format, XML significantly reduces the burden of deploying the many technologies needed to ensure the success of Web Services.

### 5.1.2. SOAP
SOAP provides a standard, extensible, composable framework for packaging and exchanging XML messages.

### 5.1.3. WSDL
WSDL is a language for describing Web services. WSDL describes Web services starting with the messages that are exchanged between the requester and provider agents. The messages themselves are described abstractly and then bound to a concrete network protocol and message format.

Web service definitions can be mapped to any implementation language, platform, object model, or messaging system. Simple extensions to existing Internet infrastructure can implement Web Services for interaction via browsers or directly within an application. The application could be implemented using COM, JMS, CORBA, COBOL, or any number of proprietary integration solutions. As long as both the sender and receiver agree on the service description, (i.e. WSDL file), the implementations behind the Web Services can be anything.

### 5.2. REST Services
The acronym REST stands for Representational State Transfer. Both terms were introduced in 2000 in the doctoral dissertation of Roy Fielding, one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification [33]. REST is a style of software architecture for distributed hypermedia systems such as the World Wide Web. As such, it is not strictly a method for building Web Services, even if, in practice, it's emerging as an alternative of traditional SOAP-WSDL based web services, and later we will see how and why.

It should be clear from the outset that REST is not a standard (while it uses standards). You will not see the W3C putting out a REST specification. You will not see IBM or Microsoft or Sun selling a REST developer's toolkit. Why? Because REST is just an architectural style. You can only understand it, and design your Web services in that style. REST refers in the strictest sense to a collection of network architecture principles which outline how resources are defined and addressed. Resources are a fundamental concept in REST. As Fielding writes: "REST-based architectures communicate primarily through the transfer of representations of resources". A resource is a source of information or, more practically speaking, any item of interest (a car, a user, an airplane, etc...). For example, the Boeing Aircraft Corp may define a 747 resource. Clients may access that resource with this URL: http://www.boeing.com/aircraft/747.

A representation of the resource is returned (e.g., Boeing747.html or Boeing747.xml, etc…). The representation places the client application in a state. The result of the client traversing a hyperlink in Boeing747.html is another resource is accessed. The new representation places the client application into yet another state. Thus, the client application changes (transfers) state with each resource representation (Representational State Transfer). URIs are another key concept that will be discussed later [34].
In summary, the key principles of REST architectural style are:
1. Application state and functionality are abstracted into resources.
2. Every resource is uniquely addressable using a universal syntax for use in hypermedia links.
3. All resources share a uniform interface for the transfer of state between client and resource, consisting of a constrained set of well-defined operations and a constrained set of content types, optionally supporting code on demand.
4. A protocol which is: Client-server, Stateless, Cacheable, Layered

5.2.1. REST on HTTP

While REST is not a standard, it does use standards:
- HTTP
- URL (Resource Naming)
- XML/HTML/GIF/JPEG/JSON/etc (Resource Representations)
- text/xml, text/html, image/gif, image/jpeg, etc (MIME Types)

Better still, REST is a set of principles that define how Web standards, such as HTTP and URIs, are supposed to be. The promise is that if you adhere to REST principles while designing your application, you will end up with a system that exploits the Web’s architecture to your benefit.

Why HTTP? Because HTTP has a uniform interface for accessing resources, which consists of URIs, methods, status codes, headers, and content distinguished by MIME type. The most important HTTP methods are Post, Get, Put and Delete. These are often respectively associated with the Create, Read, Update, Delete (CRUD) operations. HTTP separates the notions of a web server and a web browser. This allows the implementation of each to vary from the other based on the client-server principle. When used RESTfully, HTTP is stateless. Each message contains all the information necessary to understand the request when combined with state at the resource. HTTP provides mechanisms to control caching and layered components (intermediaries) such as proxy servers, cache servers, gateways, etc, can be inserted between clients and resources to support performance, security, etc. Thus it fulfills all the key principles of REST stated before.

5.2.2. REST Web Services

A RESTful Web Service is a Web Service implemented using HTTP and the principles of REST. The definition of such a Web Service can be thought of as comprising three aspects:
1. The URI for the web service such as http://example.com/resources/cars. This URI represents a resources collection “cars”. Members of this collection (i.e. a specific car) are addressed by ID using URIs in the form of “<resource-collection-uri>/<ID>”.
2. The MIME type of the data supported by the web service. This is often JSON, XML or YAML but can be anything.
3. The set of operations supported by the web service using HTTP methods (e.g. POST, GET, PUT or DELETE).

REST is fundamentally different from the classic Remote Procedure Call (RPC) approach that encapsulates the notion of invoking a procedure on the remote server. Hence, RPC messages typically contain information about the procedure to be invoked or action to be taken. This information is referred to as a verb in a Web Service request. In the REST model, the only verbs allowed are GET, POST, PUT, and DELETE. In the RPC approach, typically many operations are invoked at the same URI. This is to be contrasted with the REST approach of having a unique URI for each resource [35].
5.2.3. **REST Services Advantages**

Here’s a brief list of possible advantages of using REST Services [36]:

- Provides improved response time and reduced server load due to its support for the caching of representations.
- **Improves server scalability** by reducing the need to maintain session state. This means that different servers can be used to handle different requests in a session (high performance).
- Requires **less client-side software** to be written than other approaches (think about SOAP stubs generator etc..), because a single browser can access any application and any resource.
- Depends less on vendor software and mechanisms which layer additional messaging frameworks on top of HTTP
- Provides equivalent functionality when compared to alternative approaches to communication
- Does not require a separate resource discovery mechanism, due to the use of hyperlinks in representations
- Provides better long-term compatibility and evolvability characteristics than RPC.
- RESTful implementation allows a user to bookmark specific "queries" (or requests) and allows those to be conveyed to others across e-mail, instant messages, or to be injected into wikis, etc (increase portability).
- You can turn your application’s Web UI into its Web API — after all, API design is often driven by the idea that everything that can be done via the UI should also be doable via the API. Conflating the two tasks into one is an amazingly useful way to get a better Web interface for both humans and other applications.
- Ease of use, flexibility, simplicity.

5.3. **Service Integration through the Semantic Broker**

In order to abstract from different service implementations and to enable a rather automated way to discover distributed services and resolve heterogeneities broker based on Semantic Web Services (SWS) technology will be utilised. Semantics-based abstraction from services is particularly required to automate the discovery of distributed and to allow transparent access to Web Services independent from their different communication standards and underlying data models.

The Semantic Broker component of the architecture is described in detail in D5.1. In this section we highlight some background information (SWS and IRS-III) and present an overview of the envisaged functionalities provided by the Semantic Broker. According to Chapter 4 above, the Semantic Broker will play different roles in the provider and in the home architecture; however, we envisage the same SWS-based platform to be used in both cases.

5.3.1. **Semantic Web Services and IRS-III**

Semantic Web Services (SWS) are ontological descriptions of Web services in terms of their capabilities, interfaces and non-functional properties. Semantic Web Services technologies enable the automatic discovery, selection and composition of distributed services for a particularly expressed user request. Note that the term service here refers to software functionalities which are exposed to and accessible through the Web (i.e. based on HTTP).

In that, a Web service might utilise standard Web service technology such as SOAP, UDDI and WSDL but also more light-weight approaches such as REST or XML-RPC. Semantic Web Services are being deployed to facilitate interoperability and to increase the degree of automation in a wide range of applications from different domains, such as eLearning or business process management.
Current results of SWS research are available in terms of reference ontologies, such as **OWLS**, **WSMO**\(^1\), and **SAWSDL**\(^2\) as well as comprehensive frameworks (i.e. DIP project\(^3\)). Whereas WSMO is intended to enable fully automated service matchmaking based on comprehensive semantic specifications of service capabilities, recent derivations of WSMO, enable representation of rather light-weight service descriptions based on RDF and the hREST microformat:

- **WSMO-Lite**\(^4\): Lightweight Descriptions of Services on the Web – a lightweight set of semantic service descriptions in RDFS that can be used for annotations of various WSDL elements using the SAWSDL annotation mechanism. Exploits the standard languages of W3C including RDF and RDFS as well as various extensions of those languages such as OWL, WSML and RIF for semantic service descriptions.

- **MicroWSMO**\(^5\): Semantic Annotations for RESTful Services – a semantic annotation mechanism for RESTful Web services based on the hRESTs microformat.

- **hRESTs**\(^6\) (HTML for RESTful Services) - microformat for machine-understandable descriptions of Web APIs, backed by a simple service model. The hRESTS microformat describes main aspects of services, such as operations, inputs and outputs. Also available are two extensions of hRESTS: SA-REST, which captures the facets of public APIs important for mash-up developers, and MicroWSMO (see above), which provides support for semantic automation.

While the above mentioned lightweight approaches are less costly to apply, they envisage a much lower degree of automation and do not facilitate comprehensive matchmaking scenarios as foreseen by more complex frameworks such as WSMO.

IRS-III\(^7\) is a **Semantic Execution Environment** (SEE) that also provides a development and broker environment for SWS. A client sends a request which captures a desired outcome (Goal) and, using the set of Semantic Web Service capability descriptions, IRS-III proceeds through the following steps:

1. Discover potentially relevant Web Services.
2. Select set of Web Services which best fit the incoming request.
3. Invoke the selected Web Services whilst adhering to any data, control flow and Web Service invocation constraints.
4. Mediate any mismatches at the data or process level.

IRS-III adopts the WSMO\(^8\) conceptual model, thus implementing the following top-level knowledge-based models:

- **Ontologies**. Provide the foundation for semantically describing data in order to achieve semantic interoperability and are used by the three other elements;

- **Goal**. Describes the task that a service requester expects a Web Service to fulfil. In this sense they express the client request;

---

\(^1\) http://cms-wg.sti2.org/TR/d1/v1.0/
\(^2\) http://www.w3.org/2002/ws/sawstdl/
\(^3\) DIP Project: http://dip.semanticweb.org
\(^4\) http://cmu-wg.sti2.org/TR/d11/v0.2/20090310/d11v02_20090310.pdf
\(^5\) http://cmu-wg.sti2.org/TR/d12/v0.1/20090310/d12v01_20090310.pdf
\(^6\) http://knoesis.wright.edu/research/srl/projects/hRESTs
\(^7\) http://technologies.kmi.open.ac.uk/irs/
\(^8\) http://www.wsmo.org
- **Web Service.** Describes the capability of an existing deployed Web Service. The description also outlines how Web Services communicate (choreography) and how they are composed (orchestration);
- **Mediator.** Describes connections between the components above and represent the type of conceptual mismatches that can occur. In particular, four types of mediators are provided: oo-mediators link and map between heterogeneous ontologies; wg-mediators connect Web Services to goals; gg-mediators link different goals and ww-mediators connect different Web Services.

Whereas WSMO is intended to enable fully automated service matchmaking based on comprehensive semantic specifications of service capabilities, recent derivations of WSMO (WSMO-Lite\(^9\), Micro-WSMO\(^10\)), enable representation of rather light-weight service descriptions based on RDF and the REST\(^11\) microformat. While such lightweight approaches are less costly to apply, they envisage a much lower degree of automation and do not facilitate comprehensive matchmaking scenarios.

### 5.3.2. Supporting applications and developers through Service Semantics

Basically, service semantics will be exploited within WP5 to support two major goals:

G.1. Support of distributed NoTube developers with light-weight service annotations
G.2. Support of application automation with Semantic Web Service brokerage via the WP5 Broker

#### G.1 Support of distributed NoTube developers with light-weight service annotations

While the NoTube development takes place in a highly distributed setting and follows service-oriented principles which involve the reuse of public and WP-specific services, NoTube developers need to provided with smart means to find and identify appropriate services and data sources for their specific applications. Hence, as an initial step, light-weight service semantics will be produced to support the NoTube SOA developers in finding and re-using appropriate services. Therefore, light-weight service schemas such as WSMO-Light will be utilised and refined for the needs of NoTube to allow service providers to document services in RDF what forms the basis for a more structured search for relevant services by potential service consumers.

In addition, such service annotations can be gradually enriched in order to produce more comprehensive and formal service specifications which are the basis for automated service brokerage, i.e., discovery and orchestration of services via a service broker (G.2).

#### G.2 Support of application automation with Semantic Web Service brokerage via the WP5 Broker

Following the service-oriented approach described in the previous Section 2.1, the overall aim of the Broker is to abstract from available software entities (Web Services) by means of semantic annotations. In that, it ensures a high level of autonomy and flexibility by providing a means to expose certain functionalities in an implementation-independent manner. While requests – i.e. for a certain piece of content in a specific format – are achieved by sending so-called goal requests to the Broker platform, such requests are resolved by automatic discovery and execution of available services based on their semantic annotations. This ensures that service implementations can be modified, replaced and added to the lower architectural levels without having to adopt the upper levels, e.g. the application logic of the architecture. Figure 21 shows the SWS-based broker as part of the NoTube architecture. It maps to the “Business Process Choreography” logical layer presented early on in Figure 2 (Section 2.1).

---

\(^9\) http://www.wsmo.org/TR/d11/v0.2/
\(^10\) http://www.wsmo.org/TR/d38/v0.1/
\(^11\) http://knoesis.wright.edu/research/srl/projects/hRESTs/
This highly generic version is particularly supported by the SWS technology introduced in Section Error! Reference source not found., in that it allows to discover and orchestrate a set of services which suit the needs of a given goal G. Note, while goals, i.e. WSMO Goals, are notions represented semantically, i.e. within the SWS-based TV Resource Broker, Web services themselves represent external software entities, provided by either other NoTube components/WPs or external providers. In that, the SWS-based TV Resource Broker will make use of the services provides by WP1, WP2, WP3 and WP4 and will integrate with the architectural components and interfaces provided by WP6 and WP7.

The SWS-based Resource Broker serves two main purposes:
   a) it abstracts from available software functionalities by means of semantics and
   b) allows to automatically discover and execute services for a given goal request.

In that, the upper level application logic can be developed independent from the implementation specifics of the underlying software functionalities by simply requesting particular computations by means of WSMO goal achievement requests. The SWS-based broker resolves such a request by discovering the most appropriate services for a given request, handling their execution and if desired, mediating mismatches occurring during execution-time. Note, service orchestrations might be created either within the SWS-based Broker but also the upper layers of the NoTube architecture, such as the application logic, dependent on the actual need and abstraction level.
6. NoTube Technologies

The goal of this chapter is to provide a synopsis of the NoTube technologies in the light of the general architecture.

Envisaged functionalities will be detailed in the related work packages as part of the research effort and subject to be revised during the project period.

6.1. Semantic Models

This section introduces several background technologies (RDF/OWL, SKOS, etc.), which provide the envisaged semantic services/models backbone for the interoperation of the various components (helping WP5 and WP7). NoTube's architecture draws upon the W3C family of Semantic Web technology standards for describing things.

6.1.1. RDF

The core underlying technology is RDF. RDF defines a simple model for representing collections of simple claims about the properties of things. It uses URIs (increasingly, IRIs, an internationalised successor to URI) to identify things. RDF claims are often called "triples", since they consist of three parts: a thing being talked about (the "subject"), a relationship or property type (the "predicate") and the value, or "object" of the 3-part statement. Hence RDF stores are often called "triple-stores", although (see SPARQL "GRAPH" section below), they typically store and expose RDF information in at least quadruples ("quads"). Within NoTube, RDF stores will typically be SPARQL quad stores.

6.1.2. RDFS

The RDF Schema language (RDFS) provides a basic framework for defining different kinds of properties and classes, and making statements about some basic aspects of their meaning. For example, in RDFS we can say that some things are Books, some things are Agents, and that "author" is a property that applies to a thing that is a Book, and will refer to things that are Agents.

6.1.3. OWL

The OWL family of ontology languages provides a much richer modelling. In OWL, we can make more complex claims about properties and classes, and making statements about some basic aspects of their meaning. For example, in RDFS we can say that some things are Books, some things are Agents, and that "author" is a property that applies to a thing that is a Book, and will refer to things that are Agents.

6.1.4. SKOS

SKOS provides an alternative set of representational conventions on top of RDF. Instead of emphasising formal logical rules, SKOS is based around a single RDF class called skos:Concept. This is, roughly, equivalent to the notion of a subject or topic, and is used for describing systems such as thesauri, blog categories, and other knowledge organization systems that do not readily (or cheaply) map onto the object-property-class formalism of RDFS/OWL.

At the time of writing, SKOS is approaching finalisation at W3C. The core RDF model has been a W3C standard since 1999. RDFS and a cleaned up version of RDF were ratified as the "RDF Core" in 2004, alongside the first set of OWL specifications. Recently, a new version of OWL, OWL 2 has been in preparation. OWL 2 updates the OWL technology in the light of implementer experience. It contains several measures to reduce the difficulty of
staying within the "OWL DL" (i.e. OWL Description Logic) profiles of OWL 1. However it is currently something of a moving target, with recent changes to the specification being tracked by active implementers. W3C is also creating a full Rule Language ("RIF"). This has something of a hybrid nature, due to the diversity of interests and traditions represented in the Working Group. It is not clear at this stage how well RIF integrates with current RDF and OWL tools, but there are activities and developments underway exploring this.

Many of the datasets needed for WP1 and WP3 are expressed in terms of SKOS RDF.

6.1.5. SPARQL

Aside from these essentially representational technologies, the W3C Semantic Web standards also include a Query Language, SPARQL. SPARQL is a core technology for NoTube. SPARQL defines not only a rich query language, but is also the first W3C SW technology to include a semi-formal notion of "data source", "provenance" or "context" within an RDF-based system. SPARQL is a language for querying RDF-based databases and similar services. It looks superficially like SQL, and has a syntax that shares many elements with Turtle, an alternative text-based syntax for RDF graphs. SPARQL queries can include references to the "GRAPH" within queries, where the GRAPH is one of possibly several independent layers of data that can be mixed and un-mixed in ad hoc fashion.

In addition to the SPARQL query language, SPARQL also defines a Web-based query protocol and supporting data formats. The protocol is defined in abstract terms, but the most common and generally preferable concrete binding of the protocol is to simple REST-style HTTP. A SOAP binding is also available, as is an experimental XMPP/Jabber binding. The result set formats defined by W3C include simple tabular XML, as well as JSON.

In (Web 2.0-like) contexts where query client code is running in a Web browser, the REST/JSON profile has several advantages, since JSON can be more easily loaded across internet domains. SPARQL services typically emphasise access to raw RDF data, but there are implementations (eg. Pellet, Cliopatria) that allow to execute SPARQL queries on a more sophisticated knowledge system, where the query response is the result of some reasoning activities mode on the ontology underneath.

6.1.6. Linked Data

In the project, we use Semantic Web3 vocabularies and technologies where useful, particularly Linked Data. Linked Data is rather like putting multiple databases on the web and giving each item of interest a globally unique key that allows you to make links between different databases. In the Linked Data world, these keys are also dereferencable URLs, which can give more machine-readable information about themselves when fetched, such as what they are connected to, what sort of a thing they are and what properties or attributes they have. The items of interest can be anything - people, documents, places, pictures, videos, anything that can be identified.

Using Linked Data enables interesting links to be made between items in a partially automated way. We believe that a profile for a user stating that she is interested in particular things (such as a particular kind of music) will allow useful personalised content filters to be created, by narrowing the types of linked data connections that should be followed for this user. Further, if the user profile can state in a machine-processible fashion the preferences of the user over multiple interests in context then a smart linked data filter should be able to order the suggestions in priority of interestingness.

As part of its website the BBC aims to produce a URL for each programme created, available in HTML and RDF formats, and linked to other items of interest, such as series, genre and contributors, using the BBC Programmes Ontology and in some cases MusicBrainz and DBpedia identifiers. This makes for a useful testbed for these ideas:
without adding to the BBC data, we can easily create bookmarking systems, because there has a URL for most recent programmes - for the BBC, TV already links to the Web.

6.1.7. Envisaged Semantic Services/Models Backbone

![Reference Ontology](image)

Figure 22 - Reference Ontology

In order to enable interaction between several various components we need to address the following issues (cf. OWL-S):

- How does one interact with the service? The answer to this question is given in the "Service grounding".
- What does the Service provide for other services/agents? This can be described in a "Service Profile".
- How is the service used? The answer to this question is given in the "Process model".

**Service grounding**

For a service ground we look at the following levels:

- Communication. Symbol level transport or data. Common technology is TCP/IP
- Protocol: request/response standard between two services. Common technology is HTTP
- Syntax: format of representing content of requests and responses. Common syntax is XML/RDF
- Semantic: model of exchanged content. Common Models are:
  - Visual Content Annotation: i.e.: VRA
  - User modelling: i.e.: [http://www.foaf-project.org](http://www.foaf-project.org)
  - Context modelling: inputs from WP1.

**Service Profile**

A service Profile describes the functions a service offers. State-of-the-art is:

- WSDL: Web service description language
- OWL-S: Standard vocabulary for describing semantic web services.

Typical services functions of semantic services are:

- Content classification
- Content enrichment by annotation
- Ontology alignment
• Concept search

Process model
The process model anticipates on the role of the service in a process. Although one should not hard-code the role of a service in a service, one can define the quality of service (QoS), such as:

• Response time
• Security
• Value adding

6.2. Metadata

6.2.1. Annotation and Enrichment
Semantic annotation of textual and multimedia content requires hybrid techniques of processing language and processing visual and audio material. Approaches for extracting knowledge from multimedia content investigate the convergence of multimedia and knowledge technologies.

Main requirement in processing audio/video material is bridging the semantic gap between low-level information of processing content (i.e.: pixel, color, motion vectors) that can be easily generated by computers and high-level semantic information that is understandable by humans. Solutions for automatic semantic annotation of both images & video use a robust methodology for discovering complex relationships between a numerical image data and perceptually higher-level concepts. Such solutions exploit domain specific spatial knowledge with spatial context of objects within an image in order to discriminate between objects with similar visual characteristics.

The annotation of video content will exploit and develop techniques for automatic identification and annotation of the region of interest. Different techniques are adapted to these tasks, e.g. computing a topographic saliency map for identification of the Region of Interest and probabilistic model techniques like Hidden Markov Model and Dynamic Bayesian Networks for extraction of the sequence of interest.

The following services will be provided for the processing of audio/video materials:

• Identification of the Region of Interest in A/V materials
• Extraction of Sequences of Interest from A/V materials

Loudness metadata which characterize the loudness level of each incoming audio stream or are required in order to normalize the loudness level before reproduction of the various audio clips. These Metadata have to be generated by loudness measurements at e.g. the same stage in the NoTube chain as the metadata generation for the individual cropping of the video for different displays. These metadata have to be provided with the various audio clips to the rendering engine, i.e. after decoding of the audio in the NoTube terminal.

On a textual level, semantic annotation is the process of identification of knowledge elements in text and mapping them to instances and entities in a given knowledge base. This is a model of semantic content enrichment. It is a process of automatic generation of named-entity annotations with class and instance references to a semantic repository. The named-entity type is specified by reference to an ontology, and the semantic annotation requires identification of the entity. Semantic annotation is assigning to entities in the text links to their semantic description, giving both class and instance information about the entities referred to in the documents. This approach comprises two processes:

a) Information extraction
b) Identity resolution.

**Information extraction** refers to a shallow process of detecting pieces of relevant informational units in texts, and representing them in the form of attribute value templates. The core problem in the process of information extraction is the correct identification of the textual units, segments that describe particular information, knowledge element. This is done with natural language processing techniques.

**Identity resolution** is a process that makes possible to determine that two or more data representations can be resolved into one representation of a unique object. It analyzes all of the information relating to individuals and/or entities from multiple sources of data, and then determines which identities are a match and what non-obvious relationships exist between those identities. Identity resolution solutions provide efficient ways to search huge amounts of information looking for comparisons that match. Identity resolution process assigns every individual or entity a unique identifier. The process of identity resolution is closely related to the ontology development and use.

The named entity references in the text are linked to an entity individual in the knowledge base keeping the semantic description and extending it with the information extraction process. Thus, the service **Semantic annotation** of texts will be provided as well.

### 6.2.2. Interoperability

#### 6.2.2.1. Introduction

To ensure an effort-less exchange of metadata between various components of the NoTube system, a unified ruleset is necessary. All IT-Systems taking part in the exchange and processing of metadata therefore have to speak a common language. Currently different metadata formats are used in professional TV productions to exchange and save information. Those differing models impede the interoperability between systems of distinct broadcasters and production facilities. Moreover, external information sources (e.g. Internet portals or News agencies) are more and more used in the TV production. These external sources again have their own metadata format – if any.

To simplify the production, a unified metadata model is required within one domain (i.e. TV production). This will ensure system integration of all sources and increase interoperability between miscellaneous components. A detailed overview of currently used professional and non-professional metadata formats can be found in Deliverable 2.1.

#### 6.2.2.2. Metadata Services

If not all systems in a specific domain use a uniform metadata exchange format, converters services are required, that will adapt or transform the used format. These services will also be essential if information needs to be exchanged between different domains, which might require different formats depending on the field of application.

Within NoTube, two different domains must be distinguished: The first domain is the Service Provider ambient, which is responsible for the creation and delivery of the TV content. It will also receive metadata from external sources. The second domain is the Home Ambient, which will mainly be receiving the content offered by the Service Provider, but might also contribute to the system with user generated content. The end-user will be found in this second domain (see also Chapter 4).

Both environments require a metadata model. These will need to be appointed during the further development of the project with regard to the use cases. For future reference, the metadata models that will eventually be used in the NoTube architecture will be referenced as follows:
• NoTube metadata model for Service Provider (NMSP): This metadata model will be used in the Service Provider ambient and should hold all necessary data to be used in the NoTube system.

• NoTube metadata model for Home Ambient (NMHA): This metadata model will be used in the Home Ambient. This model can also be a subset of the model used in the Service Provider domain (NMSP).

If external metadata (e.g. PrestoSpace from RAI, see Use Case 7a, or the Broadcast Metadata Exchange Format (BMF)) is to be processed and enriched in the Service Provider Ambient, a metadata transformation service is required to adapt these external metadata formats to the NMSP. A similar approach must be taken to enable the exchange of information between the both “worlds” (Service Provider and Home Ambient). Again, a service for the metadata transformation or extraction is required. This service will therefore be responsible for the adaption between the formats of the Service Provider (NMSP) and the Home Ambient (NMHA) as well as the adaption from external metadata formats into the NMSP.

On both sides, the Service Provider as well as the Home Ambient, external information sources can be used to further detail the content and refer to additional information. This external information is likely to be taken from the Internet (e.g. Wikipedia, photo or video communities, user review portals etc.). Within both ambiances, the operation of the service will be to insert the links to the external sources to the respective NoTube internal format (NMSP and NMHA).

6.2.2.3. Functional Requirements - Metadata Service

Within the NoTube system, the transformation of the metadata models (as mentioned above) will have to take place on the Service Provider side. On the Home Ambient side, further enrichment of the metadata has to be done.

In the following section, the base functionality of these services will be examined.

Functional Requirements – Service Provider Metadata Service:

• Transforming instances of external metadata models (e.g. PrestoSpace) into instances of the metadata model of the Service Provider. In order to achieve future compatibility, the transformation will be done using a special exchange format for metadata, the Broadcast Metadata Exchange format.

• Transformation of instances on non-TV metadata (e.g. EGTA, a metadata model for advertisements) into the instances of the metadata model of the Service Provider.

• Provide an interface to the stored data in the instance to enable semantic enrichment.

• Enriching the metadata instance with links to additional information (e.g. YouTube videos, Wikipedia articles etc.)

• Transforming instances of the Service Provider metadata model (NMSP) into instances of the Home Ambient metadata model (NMHA). Note: This transformation may possibly be conceived as a filtering as the same base model will be implemented on both sides and the NMHA is a subset of the NMSP.

Functional Requirements – Home Ambient Metadata Service:

• Provide information for transforming or filtering within instances of the Home Ambient metadata model base on user preferences that are not stored on the Service Provider side (or where a filtering according to those preferences is not possible on the Service Provider side). Note that source and destination model are identical, only the information stored within the model is modified.

• Provide an interface to the stored data in the instance to enable semantic enrichment.
- Enriching the metadata instance with links to additional information (e.g. YouTube videos, Wikipedia articles etc.)

6.2.2.4. High Level Service Architecture

In the following chapter, a rough draft of the architecture of the metadata service will be given. This description has to be verified and described in more detail as the project continues to specify the metadata models to be used internally and the external formats to be transformed.

The service for the metadata transformation will provide an interface for external access. The source and destination format for the transformation can be specified via this interface. The destination format will either be the Service Provider or the Home Ambient metadata model, whereas the source can be either one of these or external models. The specific transformations will be implemented as separate modules that will be provided to the service as “plug-ins”. This approach ensures a flexible adoption of the service for different transformations. Furthermore, this modular approach provides for future extensibility. This way, new transformations, implemented in additional modules added to the service will not require the interface to change for external access.

The service could also be implemented in an intelligent way that would detect the source metadata format automatically and transform the data by the means of the correct corresponding module. In this case the service only has to be called with information about the destination format, whereas the source (as optional parameter) could be left empty. The transformation can only be done if a module exists that supports the conversion of the source model in the specified destination model. If, and if yes: in which way, such intelligence can be added to the service has to be ascertained within the subsequent work in this project.

The other important functionality of the metadata service is to provide access to the information itself. For that reason the service will offer different interfaces to read and to set several information within the metadata instances. These interfaces will be designed and implemented in a generic way, thus the access for other services is independent from the metadata format used.

6.3. User Profiling and Context Models

The User Context and Profile specification is described in details in deliverable D3.1. In the following short passage, some background about user web activity and profiling is outlined (Section 6.3.1), the main functionalities and their logical separation between client and server sides are sketched (Section 6.3.2) and in the end a little overview about used and/or envisioned technologies is provided (Section 6.3.3).

6.3.1. User Activities and Profiles

An informal and introductory overview on the main ideas and principles behind the services responsible for the user profiles and recommendation management is needed in order to motivate the following architectural choices. Actually the user and context models, jointly with the algorithms for their update, heavily impact on the deployment of these services on the overall NoTube architecture.

The main principles behind the adopted user model rely on the assumption that the everyday interaction of a web user with several different social web applications can be exploited to build meaningful user profiles [17, 18]. Although several works already showed the potential of using the social data in content-filtering algorithms, from web search engine personalization [19, 20] to TV content recommendation [21], such collaborative-filtering techniques could be further improved with a mechanism to lift such data to a semantic level. Bridging the gap between the Web 2.0 and the Semantic Web could be the way to produce rich and semantically described user profiles, as a recent effort [22] in such direction tries to demonstrate.
With this aim in mind, we can leverage the quick proliferation of the Linked Open Data clouds [23] that will provide us with the possibility to enrich the social data representing user interactions, the content metadata and the contextual aspects of the interaction.

For example, a user could be described with a FOAF profile, while the content of a video he/she bookmarked on YouTube can be described as an OWL ontology linked to DBPedia for some concepts, to GeoNames for enrich some geographical locations, to MusicBrainz for some descriptions about the author of the soundtrack and so forth, according to what hereby is called the "Layer Cake".

With the term "Layer Cake" (refer to the NoTube deliverable D3.1 for deeper insights) we aim to a layered dataspace [24], made up of a collection of ontologies (schemas and instances) as Dbpedia [25], GeoNames [26], the BBC Programmes Ontology [27], MusicBrainz [28], etc., jointly with a formal representation of the user's activity streams [29] and profile, represented themselves with specific ontologies (and thus in specific layers of the cake).

Using these interconnected and continually-updated knowledge bases, it will be possible to build user profiles where the inferred implicit knowledge is encoded and usable to make meaningful semantic-based content recommendations. With regard to this aspect, it is important to underline the key role of the semantic-lifting process, applied to the so-called activity stream of a user.

Moreover, within the project we have designed an experimental vocabulary for describing user interests supporting filtering and recommendation services with a summary of a user's preferences when they are in different environments (contexts). It can be combined or used instead of straightforward FOAF interest profiles, and be combined with and used instead of information traditionally used to make recommendations to users, in particular age, gender, and location. The idea is to allow people to describe their preferences as orderings within a certain context (i.e.: "I prefer radio 4 over radio 5 when I am working at home). Please see D3.1 Section 6 for more details.

The last key aspect of the user model is represented by the feedback-based updating mechanism of the user profile. It adheres to the assumption that a user profile, being a machine-readable snapshot of a user interests, evolves as if it were a "biological entity". This last statement could be motivated by the consideration that the interests of a person can change during his/her whole life, following a biological evolution pattern. For this reason, the real-time activity of the user as well as his/her resulting profile will be fed into the user modelling engine in order to take in account profile changes as deltas between different chronological versions of the same profile.

### 6.3.2. WP3 Services and their Deployment within the Architecture

The WP3 Profile and Recommendation Engine modules will be split into Home side functionalities and Content Provider side functionalities, in order to realise a logical loop starting from the user (what he/she is doing and in which context, what he/she is) and ending to the user itself (who gets the recommended content items).
With regards to the pictures above, where the logical interactions between the components are showed, we could identify three main components and their deployment within the NoTube overall architecture.
Informally we have the following three components:

- **The User Profiler component**, that is responsible for lifting a set of items to a semantic level (including user activities, subset of his/her profile spread over several different data silos, generic attention data). It will then link them to the layer cake, inferring a user profile that will encode also some implicit knowledge about user interests, preferences and so forth. Updating and storing profiles operations, as well as CRUD operations in general, will be addressed within this module.

- **The Recommendation Engine**, that is responsible for producing content recommendation. It takes as input a snapshot of the user activity streams, the actual context the user is into (the device used, the country in which is used, etc.) and the user profile. To execute the recommendation action, the recommender will leverage client information about the current user activity as well as profiler statements about the user.

- **The Home interface**, that is responsible for capturing users’ interaction and activities to feed the remote profiler. More specifically, users’ activity streams will be gathered and submitted to the User Profiler according to a set of user-defined privacy rules. Privacy rules play a key role for the overall system behaviour: it will
offer functionalities to the user for the definition of what he/she wants to be profiled by the remote User Profiler. More details about privacy issue are described below.

6.3.3. Technologies
Each one of the three outlined services (the client module and the two server modules) will require specific languages to describe and reason about activities, profiles and recommendations. RDF, SPARQL, SKOS, OWL will be all used in order to be able to manage semantic descriptions and statements about user, context and content.

Moreover, since the static part of a user profile (built of such attributes and properties characterized by a low degree of variability, i.e. religion, sex, relationship status ...) results from the merging of such kind of different information contained in the various social networks the user belongs to, it’s quite natural to adopt technologies aimed to solve such complex data interoperability issues.

Typically, different data models (OpenSocial, Facebook, Portable Contacts, domain specific ones as microblogging “follower” relationship) will be semantically mediated, making use of the FOAF ontology. Moreover, technologies belonging to the so-called OpenStack [30] will be employed in this scenario. In particular, the OAuth protocol [31] will be used to enable a secure API-based access to such partial information contained in several different user profiles repositories (the so-called "social networks").

6.4. Security and Privacy Preserving Policies
The following sections provide an overview of the privacy protection issues and the way the technical partners plan to address them within NoTube project. During the project this topic will be further deepened.

Since the user profile management is strictly related to WP3 activities, in the text there will be a number of references to the “Beancounter”, which is the codename of the module aimed at managing user identities, profiles, etc.

6.4.1. Privacy and Beancounter
NoTube's mission is to 'put the user back in the driving seat', through making their needs and interests central to a modernised experience of television. This is attempted by building systems based around rich and expressive representation of user interests, but also by adopting architecture that embraces the entire Web as an environment in which users can explore and share content. In particular, NoTube expects 'television' to melt into the wider Web, rather than remain a separate medium. Many of the characteristics of the Web experience such as linking, annotation, bookmarking and universal access will eventually merge with television. Given these background assumptions, it is critical that NoTube specify mechanisms that keep users in control of their data, especially when it is being linked, shared and integrated between diverse sites and services. NoTube's approach to user profiles is to ground them in users' online behaviour, and with their permission to derive higher level summaries of user interests from analysis of activity streams and other data collected from around the Web.

Several aspects of the Beancounter are therefore of particular significance in respect to privacy. These are:

- The aggregation and analysis of users' previously disparate and unconnected data
- The 'enhancement' of users’ data by linking it to other data, thereby creating relationships where none may have existed before
- The collective aggregation of users' data for statistical analysis
- The access and use of private data and combining it with public data
While we believe there will be significant benefits to the user of collecting their data in this way (by delivering more personalised recommendations and previously unavailable functionality such as "show me which TV series I watched the most last year"), the Beancounter faces several challenges in finding ways of:

- effectively communicating the associated privacy risks to users without scaring them and
- enabling users to have full control over their data while still being willing to share it appropriately

### 6.4.2. Social and Psychological Privacy Challenges

There are numerous and well-known problems with online privacy from a user point of view.

- People find it very difficult to think about privacy in an abstract way: it’s hard to define what we mean by ‘privacy’ (ID theft? credit card fraud?). In this context we are talking about intimate data: information users might prefer not to share, but they need to understand the concrete implications of the Beancounter.
- Perceptions of privacy vary across nations and cultures. For example, people in India are much more comfortable about giving out personal details on social networks than people in America. (Source: SYNOVATE 2008: Social Network Users)
- Managing privacy in everyday life is an intuitive process: the front you present is not only tailored to the pertinent audience but also to the context (e.g. at work/at home) and it determines the amount of information you are willing to disclose to the audience (http://en.wikipedia.org/wiki/The_Presentation_of_Self_in_Everyday_Life_Erving_Goffman). It is subtle and complex, but it comes automatically and you don’t have to think about it very much.
- People systematically underestimate privacy risks online because the urge to sociality is such a highly motivating psychological force.
- The reaction of users to hypothetical or artificial situations is not a good predictor of actual behaviour and feelings experienced in real life.
- Reassuring people about online privacy tends to make them more, not less, concerned. The results of a series of experiments conducted by Carnegie Mellon University showed that people who were reminded about privacy were less likely to reveal personal information than those who were not. As The Guardian (http://www.guardian.co.uk/technology/2009/jul/15/privacy-internet-facebook) commented "Users care about privacy, but don't really think about it day to day. The social networking sites don't want to remind users about privacy, even if they talk about it positively, because any reminder will result in users remembering their privacy fears and becoming more cautious about sharing personal data."
- It becomes even more complicated if anyone can aggregate anyone else’s data.

### 6.4.3. User Interface and Privacy

We are taking several steps within the user interface to mitigate these issues, specifically:

- By default all profile data should be private until the user actively chooses to share it.
- The user can only 'share' their Beancounter profile from the screen where they can see their profile. This is to protect the user from sharing things inadvertently which they may prefer to keep private.
- Any updates or changes to the profile are private by default. The user can be alerted to changes by email if they choose to subscribe. If not, there is an 'updated' icon on
the Profile screen and facility to see most recent changes. The user must approve any updates before they are shared.

- Users must be able to edit their profiles at any time (e.g. to delete / hide / strengthen / weaken specific topics)

We are also looking at technical solutions to the problem of accessing private data.

### 6.4.4. Technical Privacy Challenges

The Beancounter accesses various data sources with the user's permission and currently using the username and password of the user directly. This works because this pattern is common across all secure sites, but is not a good approach to keeping user data private, for the following reasons:

- It encourages users to give away their password to sites that could be 'phishing', i.e. misusing the user's account for social engineering or other fraudulent purposes. Twitter users, for example, are frequently on the receiving end of such attacks.
- It lays open the opportunity for passwords to be stored in a way (even if hashed) that developers may accidentally or deliberately make them available to attackers via attacks on their software, social hacking, or security breaches

### 6.4.5. Data Access and Privacy in NoTube

In the future we expect to resolve some of the difficulties with use and storage of passwords in Beancounter with OAuth, a technology that makes it possible for a Web site to request data held at a second site, through a Web-based interface in which the user gets to grant (or revoke) data access tokens.

For example, a Notube-based site might want to negotiate access to information (book purchases, micro-blog messages, bookmarks, favourite videos) which are not themselves publically available. OAuth provides a framework for this. We expect to use OAuth 1.0a and monitor the ongoing IETF standards work around OAuth and the newly proposed (SSL-centric) OAuth-Wrap work.

A second scenario is the use of OAuth for controlling access to data held by NoTube systems; for example, a recommender component might want to negotiate access to a summary of some users' YouTube viewing history. Although that summary might have been produced by NoTube-inspired software and methods, this does not mean that the data can be handed over freely to any other piece of software that wants it. So, even within the NoTube ecosystem, we need a permissioning framework that puts users firmly in control of data flow.

The OAuth setup process works best in a full (desktop) Web browser. It can be achieved on smart-phones too (eg. iPhone’s Safari) but the user experience is suboptimal. However once a software component (eg. a recommender tool embedded in media-centre software such as MythTV, Boxee, MediaPortal or Freevo) has - probably via WWW - acquired an access token to restricted data, that access token can be used directly and invisibly for inter-component communication. This is typically but not necessarily achieved with HTTP; however an OAuth binding to XMPP has been defined (XEP-0235: OAuth Over XMPP, http://xmpp.org/extensions/xep-0235.html); we anticipate exploring this for remote controls. For example, a smartphone-based remote in possession of such an access token could refresh its list of user profile fields by consulting a database of profiles which was otherwise inaccessible.

OAuth is also a key component of the wider OpenSocial framework for extensible, interoperable Social Network systems. High level user summaries in NoTube will be
accessible also via OpenSocial APIs, so there is likelihood we can exploit OAuth for mediated access to these too.
7. Conclusions

The designed SOA-based architecture appears to present a hybrid style. Indeed it picks up different principles from different styles. The main reason is that the Consortium has to deal with a heterogeneous IT background that should be merged in light of the functional requirements, to increase interoperability, extensibility, modifiability, scalability and testability (thus module independence) that represent the main quality attributes impacting on NoTube.

The second revision of this document presents a more detailed vision of the internal architectures of the three envisaged use cases, showing a mapping with the general architecture diagram that reflects the original idea and, more specifically, provides a link between the architectures detailed for each application scenario and the categorisation of the baseline services planned for NoTube. Moreover, refined description of the involved technologies producing such set of services, in light of the requirements consolidated during the first year of the project, have been included.

We have decided to preserve the document’s original purpose which is to provide every technical partner with the broad lines of the NoTube architecture and integration approach, leaving the low-level description of the platform’s technical achievements overview within the yearly-planned documentation (d6.2, d6.3, d6.4) reflecting the evolving status of the platform itself. On the other hand, individual use cases and related prototypes will be detailed in WP7.a,b,c following the same timeline.
References

1. D. Spinellis, G. Gousios - Beautiful Architecture - O'Reilly Media inc. – 2009


27. http://bbc.co.uk/programmes


29. Atom Activity Base Schema (Draft),
   http://martin.atkins.me.uk/specs/activitystreams/activityschema


32. Radio Pop, http://www.radiopop.co.uk/

33. Wikipedia article on REST,
   http://en.wikipedia.org/wiki/Representational_State_Transfer

34. S. Tilkov, A Brief Introduction to REST, http://www.infoq.com/articles/rest-introduction


