OPAALS PROJECT

Contract n° FP6-034824

WP5: Integration with the Digital Ecosystem Platform

D5.2 - OPAALS Platform prototype

Project funded by the European Community under the "Information Society Technology" Programme

Contract Number : FP6-034824

Project Acronym : OPAALS

Deliverable N°: D5.2 OPAALS Platform prototype

Due date : May 2008

Delivery Date : May 2008

Short Description: The paper describes the integration of other partner's components into the OKS platform.

Author : Techideas – David Arcos

Partners contributed : Techideas

Made available to: OPAALS Consortium and European Community

Versioning		
Version	**Date**	**Name, organization**
0.1	November	David Arcos, Techideas
0.2	November	David Arcos & Javier Noguera, Techideas
0.3	November	Jesús E. Gabaldón, Techideas
0.4	December	Nacho Lorente Puchades, Techideas
0.5	May	Juanjo Aparicio, Techideas

Quality check

Internal Reviewers :

Internal Reviewer: Víctor Bayón (University of Nottingham, UK)

Internal Reviewer: Paul Krause (University of Surrey, UK)

Internal Reviewer: Gary Gaughan (University of Limerick, Ireland)

Dependences:

Work Packages	Contributes to WP10 Depends on WP3 and WP4
Partners	All
Domains	Computer science and social science
Targets	OPAALS community

OPAALS Platform prototype

Table of Contents

1 Executive Summary

This report describes the integration procedure followed in WP5 *("Integration with the Digital Ecosystem Platform")* during the last part of Phase 1 of the OPAALS Project. It accounts for the integration of the outcomes of tasks 5.2 *("Integration and test of the automatic code generation deployment")*, 5.3 *("Integration and test of the autopoietic P2P network algorithms")* and 5.4 *("Integration and test of the distributed accountability, identity and trust subsystems")*, and forecasts what should be done in phase 2.

The present deliverable has been categorized as Prototype in the Technical Annex. Therefore, this report essentially describes the integration process of the different components and subsystems that constitute the OKS Desktop at the end of Phase 1. It is also necessary to emphasize that the text of the present deliverable is not aimed to fully and comprehensively describe the integration components; indeed, these are described in the deliverables of the respective WPs.

2 Introduction

The objective of this deliverable is to provide a description of the integration procedure followed during the last part of Phase 1 of the OPAALS Project and an brief overview of the integrated components described in the deliverables of their respective WPs.

Thus, aiming to enrich the OKS, the current integration is centred in adding services and functionalities to the OKSDesktop and the Servent components of the OKS, making them more robust and stable and helping to populate them.

The components to be integrated are:

- OKSDesktop + Servent: base platform where components are to be integrated. [1][18]
- Automatic Code generation from Models: In Task 5.2, the automatic code generation component is integrated. This document defines all procedures for a successful deployment. This code achieved compatibility with OKS, allowing the several advantages that automatically-generated code provides. [2]
- Autopoietic P2P network algorithms: In Task 5.3, the autopoietic P2P network algorithms are integrated. [3]
- Distributed Identity Management Model: In Task 5.4, the distributed identity system implementation is integrated. We deploy this service and explain the changes needed to integrate it in OKS. [4]

2.1 OKSDesktop + Servent

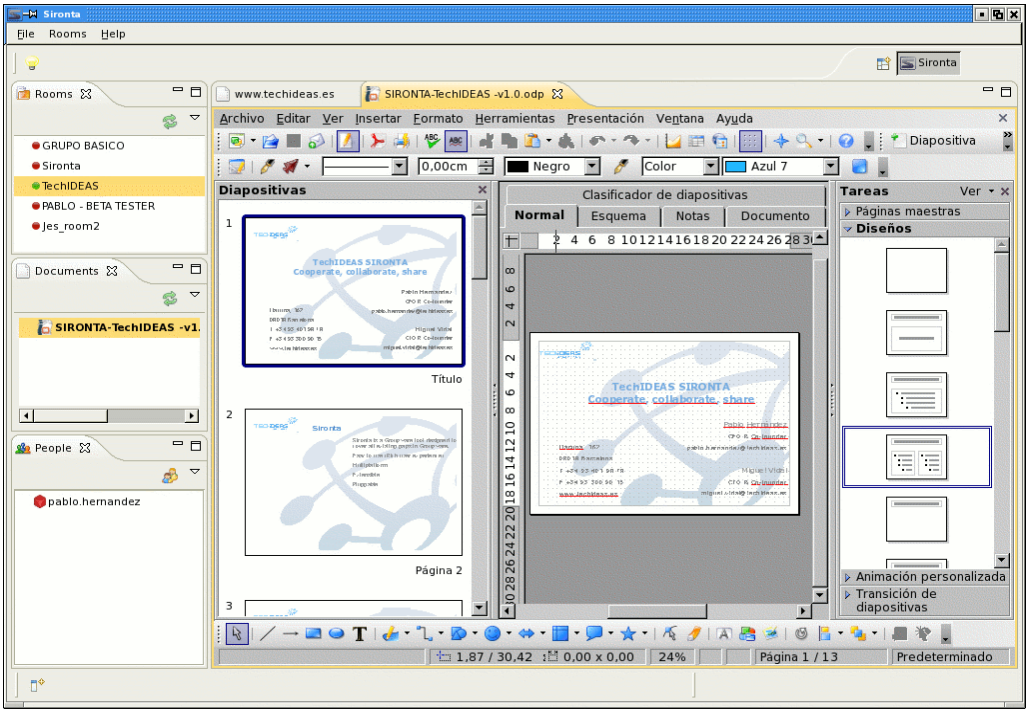
In brief, Servent is a P2P application container for services. A node running the Servent consists in a server component offering services and a client consuming services either from its own server or servers from other nodes.

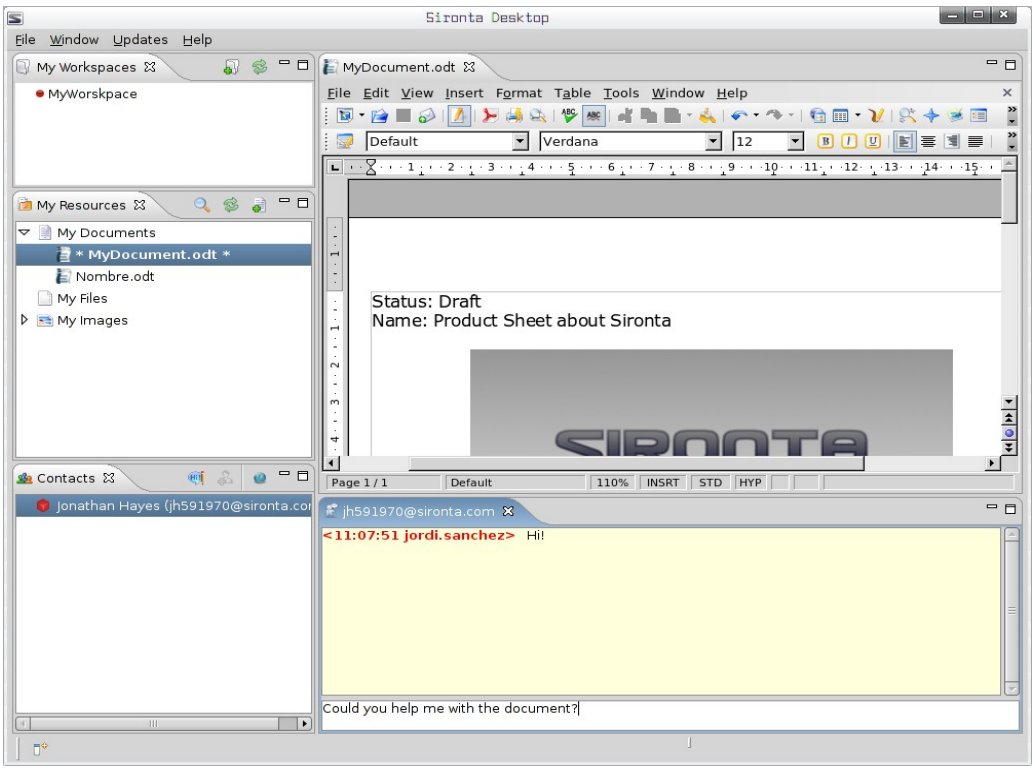
OKSDesktop is the single utility that allow to access in a single view all applications that conform the Open Knowledge Space.

The OKSDesktop is integrated in the Servent as a service consumer and provider, being then considered as a user interface for the Servent in which users can provide and gain access to OKS services.

Some of its main features, are:

- Content generation and edition in a collaborative way
- Content sharing among users
- User groups and categories management
- Instant messaging





In its previous version –before the integration here described– the OKSDesktop was an almost functional tool, featuring the actions listed above. In this new prototype, three more functionalities/components are integrated: *Automatic code generation from models* (WP2), *Autopoietic P2P network algorithms* (WP3) and *Distributed Accountability, Identity and Trust* (WP4).

These three extensions allow OKS to work with computer-generated code and to improve the system scalability by using a P2P behaviour and a distributed identity system.

2.2 Automatic Code Generation from Models

This component is the result of *WP2-Automatic Code Generation from Models*[2]. The goal of this component is to achieve an automatic way to generate machine-understandable code from a controlled natural language model.[6] [7]

It is possible to find several different natural language models. Some of them are:

- Attempto Controlled English (ACE)
- Common Logic Controlled English (CLCE)
- Gellish
- Metalog's Pseudo Natural Language (PNL)
- Ordnance Survey “Rabbit ”
- Processable ENGlish (PENG)
- Semantics of Business Vocabulary and Rules (SBVR)
- Uwe Muegge's Controlled Language Optimized for Uniform Translation (CLOUT)

The selected model input for this component is **SBVR** (*Semantics of Business Vocabulary and Rules*). SBVR is a set of normalized vocabulary and rules, expressed in a standardized format, more human-like and appropriate to the business context than computer languages and data formats. [8] [9]

While natural language is used by humans for general purpose communication, a formal language is used for describing mathematical or machine-understandable formulas. SBVR helps users to give instructions in a easy and comprehensible model. Then a translator process is the one in charge to make the necessary transformations to generate a code model that computers can understand.

Next we briefly explain the steps a user follows in defining its code:

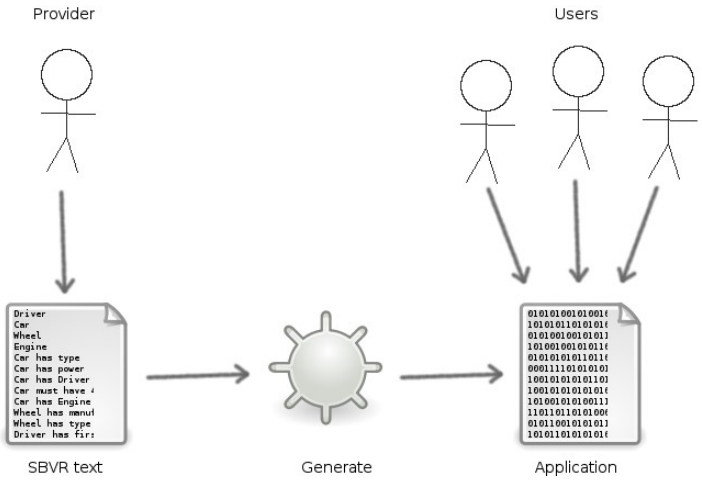


Illustration 1: Provider writes a SBVR text, that is parsed by a process that generates an application for end users

As seen in Illustration 1, service provider writes a document following the SBVR specifications. Then a SBVR translator process (*Model Transformation*) parses the document and generates a metamodel (i.e. UML, *Unified Modeling Language*), which is then used to generate the application code. This generated code is populated to be executed in the final user computer.

Since a natural language expresses a much higher abstraction levelthan a formal programming one, this way is presented as convenient for developing code and applications fast for non-developers.

This tool allows to represent and share knowledge in a way closer to business users, offering them the possibility to easily express their requirements and get code pieces that perform the desired tasks.

Despite all this, we can find some **limitations** to SBVR:

- While it could be expected that writing in SBVR would be as easy as writing in a real natural language, it isn't. It's a controlled natural language, which means that the vocabulary and grammar are restricted to eliminate ambiguity and complexity (this is needed to do a reliable automatic analysis of the language).
- Because of the previous point, users will need some training to acquire knowledge about this language.

Next, we show an example of SBVR code, any English speaker can understand it, and it's easy to learn the syntax with a little training:

Driver
Car
Wheel
Engine
Car has type
Car has power
Car has Driver
Car must have 4 Wheel
Car has Engine
Wheel has manufacturer
Wheel has type
Driver has firstname
Driver can have middlename
Driver has lastname
Driver has age
Driver has gender
Driver is identified by lastname and firstname
Car is identified by type

Table 1: Example of a real BSVR document

For a more detailed explanation, this subject is explained extensively in *WP2-Automatic Code Generation from Models.* [2]

2.3 Autopoietic P2P network algorithms

This component is the result of *WP3-Autopoietic P2P network algorithms*[3] in which several peer2peer behaviours have been studied.

The goal of this component is to find the best auto-organization algorithm that leads to a network without bottlenecks nor clustering with the characteristics of the Scale Free Networks.

Two different approaches have been followed during the research process:

- Allow nodes to decide on their own inputs such as number of neighbours or bandwidth.
- Allow nodes to play the role of "big brother" or network leaders, based on their own inputs and following a rigid algorithms that leads to a mathematical organization of the network.

The integration of this WP in the OKS is necessary for testing in a real environment the results obtained in the simulations. In this spirit, participation of other partners is also fundamental in order to recreate a real environment with peers that communicates and exchange data.

More detailed information, this subject is explained extensively in WP3-*Autopoietic P2P network algorithms.*[3]

2.4 Distributed Identity Management Model

In a typical hierarchical identity model, a single server (the identity provider) is responsible for confirming the identity of all the clients. This server runs identity services and each client must trust its credentials when ask for information about other client identities.

This model has a good performance in early stages, but some scalability problems arise when it grows, being necessary to change the paradigm into a decentralized model. [10] [11]

In a Digital Ecosystem, a **centralized identity model** has some inconveniences, the different ecosystems with diverse institutions and companies inside where collaborators, partners or competitors can forge alliances or work against others in a dynamical way. There is a need to unify criteria and standardize procedures and formats, or the relations between institutions and digital ecosystems will be impeded by the barrier of having different protocols and formats.

Different institutions have different identity technologies (such as X.509, SPKI or Kerberos). There are many types of certificates, many services on independent servers, and many ways for identifying others, causing a high complexity of integration.

To avoid this problem in a Digital Ecosystem it is needed that the Distributed Identity Model component is integrated.

A **distributed identity model** solves many of the problems present on the centralized model: [12]

- Scalability. There are no bottlenecks, the system can grow indefinitely just by adding more peer nodes.
- Fast deployment and ease of administration. While it can take more time to implement the new nodes, adding new ones is faster, because there is no need to integrate different protocols.
- Reliability. There is no "single point of failure". In a centralized model, there is a hierarchical relation within the Certification Authorities (CA), where each one authorizes some identities, but are authorized by a superior CA. If the "root certificate" ceases to be available, then all the identities signed by this root (or signed by the descendants of this root) won't be reliable any

more. In a decentralized model, the failure of one certificate doesn't imply the failure of several other certificates, because each peer is equal to the others.

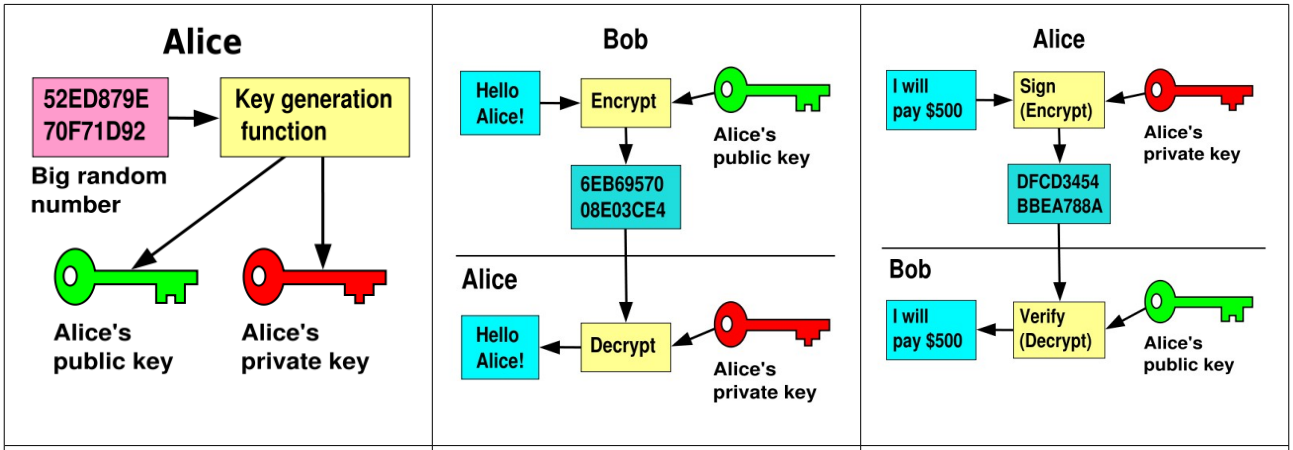
In spite of this, this distributed model has also some limitations:

- There's some added complexity when synchronizing identities, distributing reputation and checks for avoiding ID spoofing.
- It is possible to find false positives: i.e. a user with a valid ID could be not granted because there are not too much peers trusting him.
- It's difficult to revoke ID's, because they are distributed among many peers.
- Earning reputation is slower in comparison with centralized hierarchical systems.

At the time this deliverable is done -considering this topic is still in study-, in the proposed model each user (node, peer, person, entity...) has a pair of keys: [13]

- **Public key**. This key can be (and should be) widely distributed. Public key can be used to digitally sign a message: this ensures *authenticity,* confirming that someone who sends a message is who he claims to be.
- **Private key**. This key is kept secret, the user won't share this key with anyone. Private key cannot be derived from the public key. A message encrypted with the public key can only be decrypted by someone who has the private key: this ensures *confidentiality assuring* that the encrypted message can only be read by someone granted to do so.

All the different certificates of the user are using this pair of keys. [14]



User Alice has both a public and a private key. Private key is secret.

User Bob encrypts a message for Alice, that only Alice can decrypt.

Alice signs a message. This can be verified by using her public key.

The identity information is encrypted with a master password. Only the user knows the password, and it's never stored anywhere. This encrypted information is saved then in several different nodes. These nodes have previously "trusted" by the user, in this way a "web of trust" between nodes trusting each other is created. [15]

Phil Zimmerman (PGP creator) defined this **web of trust** in the PGP manual: [16]

> *As time goes on, you will accumulate keys from other people that you may want to designate as trusted introducers. Everyone else will each choose their own trusted introducers. And everyone will gradually accumulate and distribute with their key a collection of certifying*

> *signatures from other people, with the expectation that anyone receiving it will trust at least one or two of the signatures. This will cause the emergence of a decentralized fault-tolerant web of confidence for all public keys.*

For a more detailed explanation, this topic is explained extensively in *WP4-Distributed Accountability, Identity, and Trust*. [4]Integration and deployment

A software integration consists on merging some components into a existing platform. Some changes are made in both parts to make sure that the integration is clean and seamless, such as refactoring code, or adding layers to achieve compatibility. There is usually a proposal for changes, or in the worst case a bug-report for a broken feature.

Three new components (*Code Generator, P2P layer* and *Distributed Identities*) have to be integrated in a part of DE called **Servent**. This new features will expand Servent's functionalities, and will allow future improvements and more development in the OKSDesktop.

As a part of the integration, Techideas (TI) has developed some extra Servent services needed to adapt the new functionalities to the current Servent in a way that the service is autonomous and consistent with other services. This topic is explained extensively and still in study in *WP4-Distributed Accountability, Identity, and Trust*. [4].

2.5 Servent

The Servent is a P2P service container. *Servent* is a contraction of "*SERV*er" and "cli*ENT*". This means that a Servent node has two differentiated components:

- the **server** part, which offers **services** and communicates transparently with other Servents
- the **client** part, which is used to search, ask and **request** for this services

It communicates with other Servents to form a P2P network, which will be used for offering and accessing more services.

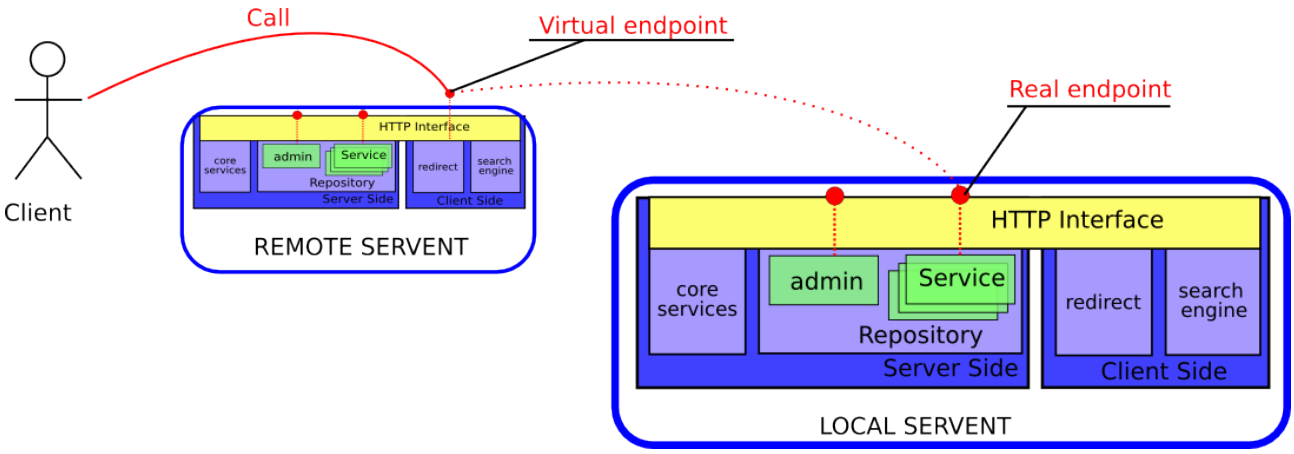


Illustration 2: Client call a remote service using the local Servent

As defined in the Servent manual: [1]

> *The main function of the Servent is to make the remote communication easy as well as the deployment and creation of services, trying to make the development process simple. The Servent itself also must provide different communication protocols and different ways to communicate (UserInterfaces, ProtocolAdapter, DynamicProxy...)*

The OKSDesktop + Servent will be modified following recommendations from the reviewers. In particular, reliance on Java has been minimised. The current implementation is written in the Java language and has to be run in the Java platform, but the interaction between parties of the distributed system is not based in Java. Instead, the protocol (which used to be Java-centric) will be replaced by an implementation based on XMPP [1], an extensible peer-to-peer protocol, which makes it possible for non-Java clients and services to interact with clients and services deployed in the OKSDesktop + Servent. The added benefit from the move to XMPP is that XMPP follows open standards, has created some of its own, published in the form of RFCs, and its extensibility capabilities allows the OKSDesktop + Servent to grow.

One limitation of the XMPP approach is that streams between peers (for example, for the exchange of files) have to be supported by the server, and not all of the publicly available ones do support this XMPP extension. This can be remedied in two ways. First, by using servers that support the standard XMPP stream extensions (XEP-0095, XEP-0096, and/or XEP-0206). Second, by embedding the XMPP server inside the OKS Desktop + Servent and implementing the extensions as plugins. The second approach is more attractive, as it allows also to change the default behaviour of XMPP servers when trying to communicate with a remote server, and to use instead the outcomes of the Autopoietic P2P network algorithms task.

For those users running OKSDesktop in corporate networks, an implementation of streams over UDP by using STUN will be provided, for the performance benefits it offers over plain XMPP.

2.6 Automatic code generation deployment

As part of the task 5.2, SUAS and TI have collaborated in the integration and set up of a Code Generator based on the previous work done by SUAS and other partners of the WP2.

In this collaboration process, the Code Generator service has been deployed. The components of this service are:

- Grails **Code Generator** service. This component has been developed by SUAS and is the responsible for translating SBVR text to computer code. Currently only a Grails plugin is done, aiming to support more computer languages (such as C++ or PHP) via plugins in the future.
- Grails **Deployment** service. Developed by Techideas (TI), this component calls for a transformation from Grails code to a .war file, deploys it in Tomcat, and returns its URL to the user. In the future this service will be able to do the equivalent with other languages.
- Grails **Library** service. This library transforms a Grails code to a .war file (which is executable as an application by a JEE server). Other transformation libraries should be used for other languages.
- Apache Tomcat server. A web server container that runs JEE applications. It will execute the .war files, and provide a web interface for user interaction.

1 http://www.xmpp.org/

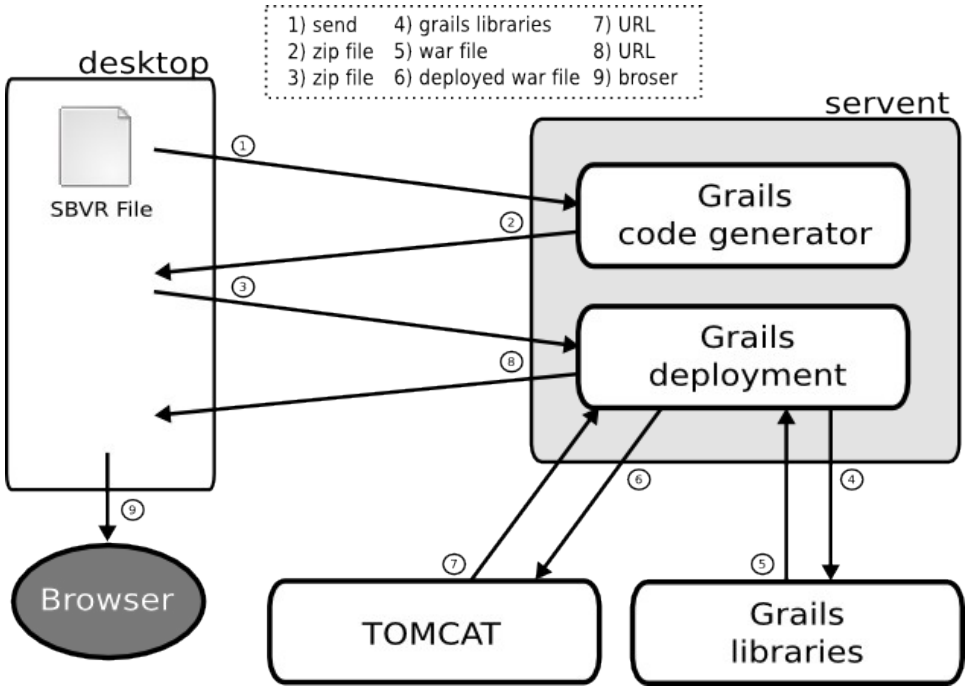


Illustration 3: SBVR execution process

This process has 9 steps:

1. SBVR text is sent to Servent. The Grails Code Generator translates SBVR language to Grails (Groovy on Grails) language.
2. The code is sent back to the user in a compressed file. But this code won't be in any use unless the user loads it into a server. This is why Techideas (TI) made the Grails Deployment plugin for Servent, as a step for the integration.
3. The users submits the compressed groovy code to the Grails Deployment plugin. This plugin is the easiest way to set up the code.
4. The plugin accesses a Grails library to transform this code to a .war file. This file will be usable directly by the server.
5. Grails library parses the code and sends back a .war file to the Grails Deployment plugin.
6. The plugin deploys this file into a JEE environment. It uploads the .war file to a Apache Tomcat server, which will be able to execute the application and provide a usable web interface.
7. The URL of this web is sent back to the plugin.
8. This URL is sent back to the user.
9. The user opens the URL in the browser, where he gets the application executed on the Tomcat server.

Note that this is a fully automated and transparent process, so the user won't notice any of the steps. The user will just write the SBVR text, submit it, and get it automatically opened in his browser.

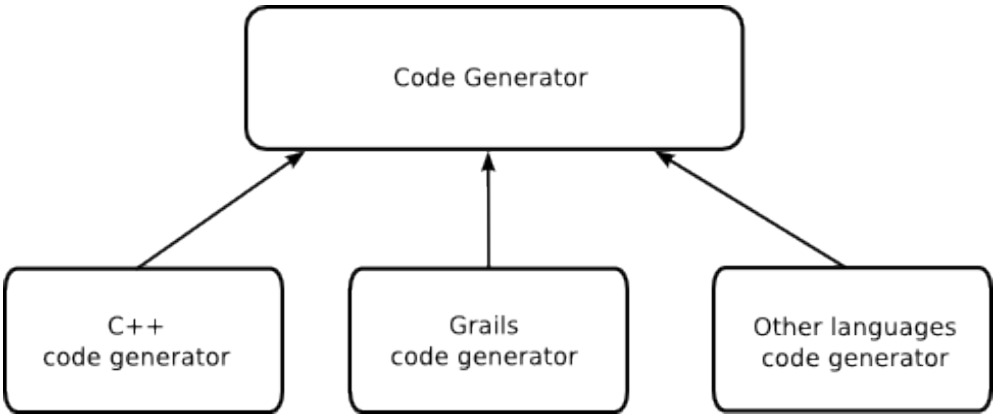


Illustration 4: Using different services that use the same interface, it is possible to generate code for different platforms

Note also that the current version of the Code Generator transforms SBVR text to Grails code. In the future we will create a generic CodeGeneration interface to be shared by all implementations. It will transform SBVR to other languages, and steps 3 to 9 will change. For example:

- For C or C++ code, the application would be executed on the local machine
- For PHP, it would be executed on a PHP server, and the URL would be returned to the user.
- For Python, Perl or Ruby scripts, it would be run against its respective interpreter, and the result will be returned to the user.
- For SQL, it would be run against a database

2.7 Autopoietic P2P network algorithms

As a part of task 5.3, the integration of the autopoietic P2P network component is based on the work done during the project. This component aims to allow the existing Servent to perform efficient searches and communication.

The Servent itself was previously released with a peer2peer framework named FADA [17]. This component is isolated from the Servent code using a interface that can be implemented in any other way. That is, the peer-to-peer component of the Servent is already pluggable.

Following the previous approach, the new P2P algorithms can implement that interface. In that case, the OKS infrastructure will use, in a transparent way, the new approach during the P2P communication.

In the case of the transactional model, also included as a task of the WP3, the integration with the existing OKS infrastructure differs. In that case the behaviour of the transactional model does not follow the typical P2P actions and it is impossible to implement that rigid interface. In these cases, we can integrate such components as a *core services* deployed and running in the Servent. Core services are running within the Servent but can access to all its functionality: deploy, p2p network, search, execute, undeploy, etc. A core service can make its own decisions and then call the appropriate methods when it is necessary to execute any action.

The approach discussed between the University of Surrey and Techideas is based in the creation of

a *core service* that acts as an Agent. The final user (client2service) or other services (service2service) call these local agent. The agent control all active transactions and gives feedback to the caller (human or service).

The behaviour of the agent can be changed just modifying the deployed service, and without the need of changes in the core application of the OKS or waiting for releases. New services can be deployed in runtime, changing the behaviour and the performance but without affecting the OKS platform.

2.8 Distributed accountability, identity and trust subsystems

As a part of task 5.4, the integration of the WIT's component is based on the previous work seen on [4]. We have developed two services for Servent, S1 and S2:

- (S1) **LookupUser** service. It receives a request for an id, asks to S2 services of the trusted Servents, and returns this id.
- (S2) **ExistUser** service. It receives a request for an id, and returns the id if it's registrated in its server

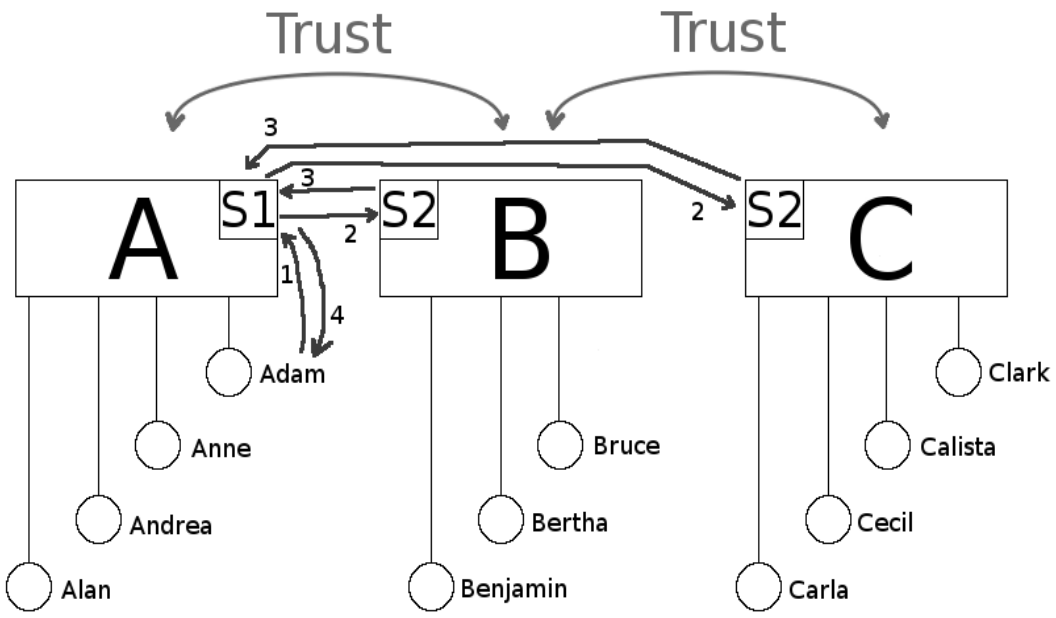


Illustration 5: Example process (Adam looks for Carla)

In Illustration 5 we can see three companies/entities (A, B and C). Each one has a Servent and several employees/users. There is a trust relationship between A and B, and another trust relationship between B and C. There is no direct trust between A and C, but A can confirm C's identity via B.

This example process has 4 steps:

1. Adam wants to know Carla's identity. He asks his company's Servent (A). A request is made to S1 service: "*Can I trust Carla?"*
2. A-S1 service looks for S2 services in other Servents, asking: "*Is Carla one of your users?*"
3. Each Servent performs a local search against its registers, to check if they have Carla. Each service will have their own rules according with the distributed model delivered in [4]. B-S2 service answers that it doesn't has Carla. C-S2 service answers with Carla's id.
4. Adam gets back Carla's ID from Servent A.

Adam can confirm Carla's id: Adam trusts A, who trusts B, who trusts C, who trusts Carla. This is a very simple "web of trust". In a real life situations, more peers (maybe Alan, Bruce, Cecil and Clark) would confirm Carla's identity, building a more complete (and reliable) "web of trust".

3 Future work

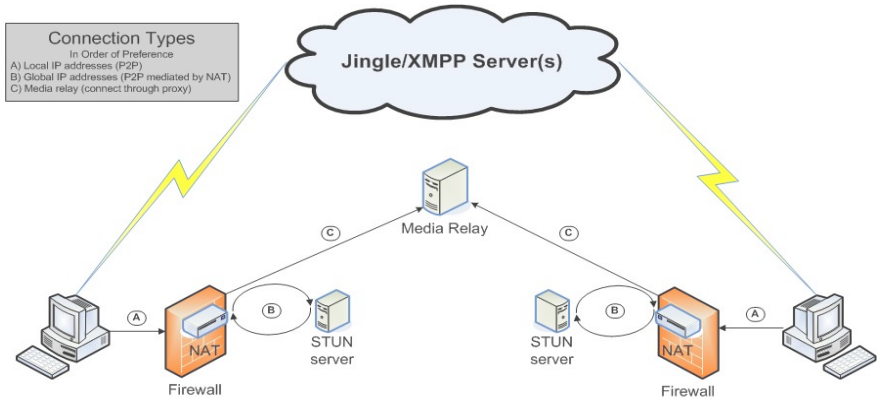
To implement the features described in this document, it may be needed to change the implementation of some already existing components. But not as many as could be initially thought. The DE is extensible, so it's open for more services, functionalities and plugins. More components are to be integrated in a near future as part of the DE evolution and growth. The addition of the OKS Desktop to the Servent offers us great possibilities.

More in detail, the automatic code generator exports code only to Grails language. In the future it's expected to export to more languages, with some changes on the Code Generator component. These changes will need to be done, probably as plugins to the Code Generator component. A service similar to “Grails Deployment” will be needed for each new language.

Concerning to the distributed identity system, next steps are related to the implementation a new service that stores statistics about the requests. It would compile information about the most wanted peers, and this would be very useful to avoid possible attacks (spam attacks, abuse from a peer, and other theoretical problems). Regarding the Servent, as it was mentioned before, a shifting to a message passing protocol is needed in order to open the systems to any other programming language. Following the XMPP standards makes the boot strapping mechanism easier, letting us to use the already existing XMPP networks, such as Jabber and Google Talk.

Other issue raised during the first stages by the early users has been the NAT problem. Network address translators (NATs) are something every software engineer has heard of, not to mention networking professionals. NAT has become as ubiquitous as the Cisco router in networking terms. Fundamentally, a NAT device allows multiple machines to communicate with the Internet using a single globally unique IP address, effectively solving the scarce IPv4 address space problem. Though not a long-term solution, as originally envisaged in 1994, for better or worse, NAT technology is here to stay, even when IPv6 addresses become common. This is partly because IPv6 has to coexist with IPv4, and one of the ways to achieve that is by using NAT technology. This can be solved by joining the problem, instead of fighting it. In order to achieve peer-to-peer traffic across NATs, we can modify the P2P model slightly to make it a hybrid of a traditional star model and modern mesh model.

Hole punching is a possible solution to solve the NAT problem for P2P protocols. Hole punching is a well known technique for establishing communications between two parties in separate organizations who are both behind restrictive firewalls. Using this strategy, both clients establish a connection with an unrestricted third-party server that uncovers external and internal address information for them. Since each client initiated the request to the server, the server knows their IP addresses and port numbers assigned for that session, which it shares one to the other.



The concept of a rendezvous server, or mediator server, which listens on a globally routable IP address, will be introduced. Almost all peer-to-peer protocols have traditionally relied on certain supernodes. Some nodes always have acted as key players in any P2P protocol (BitTorrent tracker is an example). The rendezvous concept is nothing new in the P2P world, nor is the star model totally done away with in P2P. We will use the existing XMPP infrastructure and protocols to act as rendezvous for the establishment of data streams between peers, and perforate holes in the NAT where possible, so we are able to route XMPP traffic directly between two peers. At the worst case, that is, if a hole can not be punched in the NAT, or the networking environment doesn't use NAT but stronger limitations due to network administrator policies, the existing XMPP infrastructure will allow, as long as XMPP traffic is allowed at all, any two peers to communicate by using the XMPP servers as relays.

4 References

All Workpackages and deliverables can be found in http://files.opaals.org/OPAALS/

[1] Swallow User Guide - http://swallow.sourceforge.net/user_guide.html

[2] Eder R., Kurz T., Heistracher T., Bayon V., Russo M, Filieri A, 2007. "D2.1 - Design of Software Generation Prototype"

[3] Razavi A., Moschoyiannis S., Krause P., 2007. "D3.2 - Report on formal analysis of autopoietic P2P network, together with predictions of performance"

[4] Noguera J., 2007. "D4.1 - Distributed Identity Model for the Digital Ecosystem"

[5] Noguera J., 2007. "D5.1 - Open Source Software Engineering Process Methods and Tools"

[6] Controlled natural language - http://en.wikipedia.org/wiki/Controlled_natural_language

[7] Natural language processing – http://en.wikipedia.org/wiki/Natural_language_processing

[8] Semantics of Business Vocabulary and Rules – http://www.businessrulesgroup.org/sbvr.shtml

[9] Interchange of Human-oriented Business Rules – http://www.w3.org/2005/rules/wg/wiki/Interchange_of_Human-oriented_Business_Rules

[10]Koshutanski H., Ion M., Telesca L., 2007. "Distributed Identity Management Model for Digital Ecosystems"

[11]HK, MI, LT. 2007. "Identity Management and Certificates in Digital Ecosystems"

[12]Scalable Definition – http://www.linfo.org/scalable.html

[13]Public-key cryptography - http://en.wikipedia.org/wiki/Public-key_cryptography

[14]Sources for diagrams:

- http://en.wikipedia.org/wiki/Image:Public_key_making.svg
- http://en.wikipedia.org/wiki/Image:Public_key_encryption.svg
- http://en.wikipedia.org/wiki/Image:Public_key_signing.svg

[15]Web of trust – http://www.rubin.ch/pgp/weboftrust.en.html

[16]PGP User's Guide – http://www.cl.cam.ac.uk/PGP/pgpdoc1/how-does-pgp-keep-track.html

[17]FADA – http://fada.sf.net

[18]OKSDesktop – http://wiki.opaals.org/OKSDesktop