Deliverable D2.3

Adapted solutions for user identification and authentication

Project Acronym	OASIS
Grant Agreement number	297210
Project Title	Towards a cloud of public services

Project co-funded by the European Commission within the ICT Policy Support Programme

Deliverable reference number and title	OASIS_D2.3
Status	Final

Dissemination level[1]	PU
Nature[2]	R

Due delivery date (project month)	M21
Actual delivery date	5/02/2014

Lead beneficiary	ATOL CONSEILS ET DEVELOPPEMENTS SAS
Contributing beneficiaries	Atol, Open Wide, PN
Author(s)	Christophe Blanchot, Yannick Louvet, Thomas Broyer, Jérôme Poittevin, Andrea Sanna, Marc Dutoo

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise.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

Deliverable abstract

This document is based on the requirements (D1.1) , the privacy requirements (D1.3), governance (the fact of willing to give control to the user, and data governance) and on studies in architecture (D1.2) .
This document also deals with rights management strongly coupled to the authentication process as well as rules for managing personal data.
Based on the identity management needs and the use of personal data requirements this document describes two major functions for the OASIS platform.

Authentication, privacy and data governance in OASIS are based on the social graph, which is also the OASIS directory. This social graph allows user-centric access rights management (each user is at the center of his personal and professional social relations, which he describes in the graph and around which it manages access to his personal data and to data to which he contributes).
The social graph allows one to describe persons and organizations, and different types of relationships between these entities. It can store personal data and manage access to it, as well as create groups and roles to organize rights management. The notion of context is also set up to allow the user to separate his different social roles, if he wishes, while keeping a centralized management under a single identity (and single sign-on).
User authentication and data access authorization (personal data and data shared in datacores and on federated data providers), for all federated services, are centralized and secure by safe, robust and proven protocols OAuth2 and Open Id Connect 1.0. These protocols and associated tools allow services and data sources to request and check authorization tokens from the security module, these tokens enable them to validate permissions for mutual access to resources.

Finally, we define a model for managing data governance rules, organized around the social graph, and based on the security server. This model ensures the restriction of reading and writing access for a data governance scope.

All these mechanisms comply with the security and privacy requirements, as investigated in deliverable D1.32

Project Management Review

	Reviewer 1: WP leader			Reviewer 2: B. Thuillier		
	Answer	**Comments**	**Type***	**Answer**	**Comments**	**Type***
1. Is the deliverable in accordance with						
(i) the Description of Work and the objectives of the project?	☒ Yes ☐ No		☐ M ☐ m ☐ a	☒ Yes ☐ No		☐ M ☐ m ☐ a
(ii) the international State of the Art?	☒ Yes ☐ No		☐ M ☐ m ☐ a	☒ Yes ☐ No		☐ M ☐ m ☐ a
2. Is it the quality of the deliverable in a status						
(i) that allows to send it to the European Commission?	☒ Yes ☐ No		☐ M ☐ m ☐ a	☒ Yes ☐ No		☐ M ☐ m ☐ a
(ii) that needs improvement of the writing by the originator of the deliverable?	☐ Yes ☒ No		☐ M ☐ m ☐ a	☐ Yes ☒ No		☐ M ☐ m ☐ a
(iii) that needs further work by the partners responsible for the deliverable?	☐ Yes ☒ No		☐ M ☐ m ☐ a	☐ Yes ☒ No		☐ M ☐ m ☐ a

* Type of comments: M = Major comment; m = minor comment; a = advice

Table of Contents

1 Introduction

1.1 Authentication issues

In OASIS, personal, proprietary and public data have to be mixed.

It is therefore very important to give the owners or managers data tools to finely manage integrity and confidentiality when necessary.

This requires first the identification of the users of the system, and allows centralized and secure identification and access authorizations management for all services and data sources of the federation (OASIS a federation of services and data). One must then identify the services and the data sources: the authentication mechanism provided by the OASIS kernel should allow everyone (users, services, data sources) to ensure the identification of the entities with which he communicates (user services, data sources): it is necessary for a collaborative ecosystem to be trustworthy, and for access rights to be assigned knowingly.

OASIS manages personal data, for which security issues are very important (see deliverable D1.32), in particular:

- OASIS allows each user to control who uses his personal data (with whom he accepts to share them, and which services can use them), and under what conditions.
- OASIS avoids data breach due to security breaches.
- The platform allows the user to manage his personal data (correct or delete it whenever he wants, for example).

Data management and access rights are user-centric: everyone should describe his data, social relationships (with others, with organizations), which are also personal data, and organize their sharing in the context of their own social interactions. For this purpose, the management of authentication and authorization is based on a social graph.

OASIS also includes datacores, for working collaboratively on data, without necessarily making them public, and not automatically allowing everybody to change them.

To meet these needs, we have introduced in the definition of the OASIS architecture (deliverable D1.2) the concept of governance scope, and various mechanisms to allow datacores to ensure collaboration on data within the defined governance rules: access rights, mediation, quality management.

We must therefore implement some security mechanisms, including a management model of access rights (read and write), forming a strong and consistent system with the social graph and the authentication and authorization solution.

1.2 Document objective and content

This document aims to describe the security mechanisms and access management, to meet requirements from the deliverables D1.1, D1.2 and D1.32 and the above-mentioned issues.

The document first introduces general principles (Chapter 2).

Chapter 3 defines the social graph, which is the basis of identity management and authorizations. The social graph includes management and protection of personal data.
Chapter 4 describes the chosen solution to implement centralized management as well as unique the authentication and authorization module, linked with the social graph.
Chapter 5 describes the mechanisms for managing the rights and rules of governance on the datacores.

Finally, chapter 6 concludes with a focus on how the mechanisms defined in this document allow one to comply with security rules and privacy legislation investigated in deliverable D1.32.

The implementation of the solutions described in this document are explained in the deliverable D2.1, and their use in the deliverable D2.2.

2 General principles

The management of identities is a complex issue with various implementations involving three main entities:

The Identification, or who am I ?

- Technical identity necessary to the system: single identification, technical attributes
- Personal identity I can access in self service: my citizen information, my preferences, the information required for the functioning of the services
- Technical entities which allow me to prove to some extent who I am, credentials: password, certificates, secrets…

The Authentication: How do the services check that I am who I claim to be ?

- Management of passwords and means to identify the identity (card, biometrics…)
- Management of access sessions
- Access securing: two factors authN

The Authorization: What can I do ?

- Division of powers (SoD); access request workflow
- Management of groups: access defined by the groups
- Management of the context: separation between the identity and the user account via the context of use

The management of confidentiality in OASIS is based on the social graph: it allows one to manage personal data under the user's control, to manage the social relations of each user.
It is then the basis of authentication (authentication mechanisms allow one to authenticate a social graph's entity).
It is also the basis of the rights management on data: rights are attributed according to the entities'relationships (working groups, organisations, employees of an organization, family, etc…).
OASIS offers a unique authentication: once the user is authenticated to the OASIS authentication module (from the portal or from an application), this application is valid for the remainder of his working session in his browser.

The authentication module allows applications and data sources to retrieve the identification data of a working session.
The authentication module also allows one to give services the right to access (in reading or writing) to some data, both in the social graph and the datacores, as well as in dataproviders if they wish to benefit from that mechanism.

The authentication module then manages authorization tickets required by the applications which attempt to access data and allows applications and datasources to which these tickets are presented to check the authenticity and validity and to retrieve the associated rights.

Of course the management of detailed rights for each type of data is a business process: each data source has its own access rights management rules (based on the governance of data scope for the datacore, on authorized access by each user for personal data and social graph relations, and their own business rules for the other data provider.

The authorization ticket must then allow data sources to retrieve the context information of the user who is connected (or the user for whom the client application makes a request if the latter is made offline) so as to check the advanced rights.

3 The social graph

3.1 Introduction

The social graph allows one to describe the entities involved in OASIS (persons and organisations), relationships between entities, communities of interest, and data in those entities (personal private data when the entity is a physical person).

The social graph is the directory of OASIS, and allows one to manage personal data, especially the access rights to those personal data.

It is also the basis of a management of the rights on datacore data (see chapter 6): indeed, the rights will be given according to the relationships between entities: for example, data managed by a municipality could be accessible by all users who have a work link with this administration, and by all users who work for the Department of this municipality.

We can see that those rights are expressed in term of relationships in a social graph.

The social graph allows the management of personal data, and of the authorizations of access to this data: it's according to the relation established around a person that the rights are managed. It should be noted that the relationships, and the characteristics of relationships, are also personal data and have to be protected by access rights.

The social graph is as its name suggests.

It is constituted by nodes (persons, organisations, etc...) and by relationships between those nodes. Relationships are oriented (relation from A to B); nodes and relations include characteristics.

3.2 Methodology

The social graph being highly structuring for OASIS, and under the strict control of users, it will be hard to organize it in a later version.

We then designed an architecture of the graph which could be enriched, but with no impact on what is already described in the graph.

We then went beyond what is expressed in the project's requirements (D1.1).

We relied on functionalities implemented in the existing social networks (by strengthening the confidentiality by the users), and on OWL ontologies dealing with this subject (in particular: ORG, REL, WAI, GEN et FOAF).

We went further than the study of this first implementation, to validate that all functionalities we could imagine could be implementable and that the model proposed would be coherent, solid and scalable.

3.3 Nodes

We will first implement the following types of nodes (the type of the node will be called “class” to use the RDFS terminology employed):

Class	Description
Person	Describes a physical person
Organization	Describes a legal person (company, association, community, etc...)
Group	Group of people and/or organizations. A group can also contain other groups. This notion allows one to manage organizational units (division of organizations into departments, services, ..., in the sense of the ORG ontology), groups of acquaintances in social networks to manage the rights (similar to Circles in Google+ for example), profiles in the sense of management of access rights in applications, groups of diffusion, etc... This class has a specific characteristic to manage the potential recursivity of the group (are the members of the subgroups contained in this group also considered as members of the group are the employees of the group’s organizations considered as members of the group, etc...) This notion of group also corresponds to organizational units in the “classic” directories, and makes easier the management of the rights and of the profiles in the organizations (and for people too).
Post	It is the position (function) in an organization (in the sense of the class “post” of the ORG ontology). It allows one to define the authorizations for a position, independently of the person who holds this position (and then to make the management of the rights easier when we have staff turnover in the organizations).
Account	The OASIS account. This notion is introduced to allow the definition of codes with restricted access for a same identity (for example, a mobile, a non secured network, etc...)

For all the nodes, a specific attribute [Deleted=True] allows one to indicate that the node or the relationships has been deleted (and is no longer visible and no longer allows connection to OASIS), but that some information is stored for consistency.
We must verify, during the pilot phase, if this attribute has to be stored, or if we can delete the elements directly.

3.4 Relations

The different nodes can be linked according to their class.

It should be noted that relations are oriented: the relation from A to B (which corresponds in fact to a RDF triplet where A is the subject, B the object and the relation is the predicate), is not the same when from B to A, and does no imply an existing relation from B to A.
In the management of the rights on the social graph, the existence of a relation from A to B is not necessarily known from B.

The following table describes the relations generated by the social graph in the first implementation.

Columns of the table are:

- the name of the relation
- the class of the node A
- the class of the node B
- the name of the reverse relation (from B to A), when it exists
- the description of the relation's meaning

It should be noted that the class of nodes and relations combinations mentioned in this table are the only ones authorized.

Relation	Class node A	Class node B	Relation B → A	Description
MemberOf	Person Organization Group Position Account	Group	*Member*	The node A is a part of group B. This relation is transitive: If (A MemberOf B) and (B MemberOf C) then (A MemberOf C) This relation is recursive with no limit in depth.
EmployedBy	Person Account	Organization Person	*Employee*	A is employed by B (in the sense when there is a contractual link of referring subordination: for looser links, for example: members of an association, it will be preferable to create a group and to include members).

Relation	Class node A	Class node B	Relation B → A	Description
Holds	Person	Position	*HeldBy*	A holds the position B (itself linked to an organization) All the rights linked to position B (included the delegations) are automatically sent to A (except explicit blocking: see security model).
MemberOf	Person Organization Group Position Account	Group	*Member*	Node A is part of group B. This relationship can be transitive: If (A MemberOf B) and (B MemberOf C) then (A MemberOf C)
Delegate	Person Organization	Person Position	*HasDelegation*	B can act in the name of A (in the social graph). In the case when B is a "post", physical persons of the position are automatically delegatees. See more information on the subject in the "delegation" paragraph.
HeadOf	Person Position	Organization	*HasHead*	A is the legal representative of B. Does not automatically imply "Delegate". If A is a "position", the holders are automatically "HeadOf".
AccountOf	Account	Person Organization	*HasAccount*	A is an OASIS account linked to B (connected via the A logins, we automatically act in the name of B, with possible restrictions: see security model). An account can be attached to no entity.
MyGroup	Person Organization	Group	*GroupOf*	The B group belongs to A entity (so A manages B).
HasPost	Organization	Position	*PositionIn*	It links a position and an organization (a node as a "position" type is linked to a one and only organization).
Linked	Person Organization Group Position	Person Organization Group Position	*Linked*	Allows one to describe any other link between entities. Some attributes associated to this link (to describe it) could be exploited by the applications.

Remarks:

- The explicit relationships are the business relationships of the social graph; the other relationships ("linked") are relations whose meaning is not exploited by the social graph but can be exploited by the applications (the attributes of "linked" relationships are opened; we will have to determine how to share them, including with the portal).
- The relationships like « father », « mother » and « child » are relations called "Linked". To be used as personal data, they have to be reported into them (parents can be defined by text attributes or by a relation).
- The rights between organizations which have links such as subsidiaries, establishments, etc... are managed by groups, and the relationship is described by a "linked" relation:

indeed, the multiplicity of possible combinations would make too much complex an explicit business management in the social graph.

- The accounts can't be linked to any other entity: this corresponds to the notion of virtual account described in D1.2.
- The accounts linked to an organization allow the access to the directory of the organization by applications in "batch" mode, the connexion of connected objects to OASIS, etc...
- An account can be declared as employed by an organization: this allows IT administrators to create accounts, without the obligation for an agent or an employee to connect them to a physical person.
- But the delegation functions or the definition of the position held, require the creation of a "person" profile.
- The general link « Linked » will allow to manage the extensions according to the services' needs. Those links have no influence on the security model.
- The model is a conceptual model for the applications: the profile management functions in the portal will allow to offer automatisms, so as not to oblige the user to understand the complete model.

This relationships model allows one to implement the social graph, the management of the confidentiality and to have a uniform access to the different information.

Of course, the portal (or other services brought to create relationships) will show those concepts in an ergonomic way to users, according to their activities, with possibly a predefined model.

3.5 Creation and approval of relationships

Relationships are created between two nodes.

But, there are some confidentiality constraints to respect:

- A can declare a relationship with B and B doesn't know it
- A can create a relationship with B, while B refutes it (for example, A tells to be a friend of B but B does not confirm)
- A must be able to declare a relation with B even if B doesn't make its profile visible (I want to declare that B is my friend, but I can't see B in the social graph).

We then add some specific properties on the relationships:

Attribute	Description
Reverse	Timekeeper on the opposite relationship, if it exists.
Approval	Statute of the relation's approval by the other extremity: NONE: not requested, without object APPROVED: approved by the other extremity REFUSED: refused by the other extremity (so explicitly not validated (with possible reason and date) PENDING: awaiting, under approval
RequestDate	Date of the approval demand
Date	Date when the relationship was created

To establish a relationship with a non identifiable node in OASIS, or to validate that the node corresponds to the person or entity we know in real life, the following procedure is applied:

- A specific code is created and transmitted to B outside of OASIS (a priori, by an e-mail address known by A and entered into the portal, but can also be transmitted physically).

This code is associated to a direct link (URI).

- This code (or link) allows B, by identifying in OASIS, to verify this relation, to approve or to refuse it and to possibly create an opposite relationship.

3.6 Other attributes of relationships

The model also allows one to manage any other attribute, and all information on relationships.

Interval of validity

The relationships can have an interval of validity: date from when the relationship is active (for example, A employs B since January 2, 2009), and the date of the end of validity (A does no longer employ B since December 31, 2012).

When a relationship is no more valid (or is not valid yet), it is no more used in the rights management (the relationship from A to B doesn't give any rights to B on its data managed by A).

However, this relation becomes an information, that A and B can show if they want to the other users.

3.7 Management of personal data

We will distinguish two types of data:

- Individuals'personal data, defined and protected by the law: everyone can have the power to control precisely what is done with his personal data. This data can't be used beyond the limits defined by its owner.
- Data for which there can be some confidentiality questions (then access restrictions), but which is not personal data according to the law.

In the social graph, we will manage all this data similarly (so, for example, the private address of a person, the address of a municipality...). Of course, the portal will offer a management of access rights by default which is different from case to case.

As far as possible, the federated applications have to avoid the storage of personal data outside the social graph, even temporarily.

Personal data may for specific reasons have to be stored and protected by mechanisms which are no longer controlled by the original owner of data: for example, the address of a customer on an invoice (the invoice must be stored with the original address, the name and the address of this person are then stored with the invoice, under the control of the organisation which sent it and not under the control of the user).

In that case, the access to this data has to be restricted as much as possible, and its use limited to the accounting management of the concerned organization.

It should be noted that this personal information is going beyond the personal data described here. The following information are managed by the same confidentiality mechanisms (see further):

- the relationships from the nodes
- the relationships'attributes
- the nodes' attributes

To manage personal data efficiently, the attributes organized on a tree allow to describe them:

The tree representation allows a subtle management of the rights asked by the user: at the level of the main branches, farthest branches, or at the level of the leaves (the portal will of course offer models by default to make the management of the rights easier: the security model is generic, its use can be specific).

Example (non restrictive)

Personal data:

- General data: *(section for data which could be public)*
- Usual name or pseudo
- Localization = *(city or region for example)*
- Year of birth =
- Description =

- Civil status
  - Name
    - Title =
    - Last name =
    - Usual first name=
    - First names =
    - Middle Name =
  - Status =
  - Sex =
  - Birth
    - Year =
    - Date (month / day) =
    - City =
    - Country =
  - Nationality =
- Phone information
  - Wording = «work»
    - Number=
  - Wording= « mobile phone »
    - Number=
- Addresses
  - Wording = « Perso »
    - Street number=
    - Type of street=
    - Name of the street=
    - Building=
    - City=
    - Postal code=
    - Cedex=
    - Country=
  - Wording = « holidays »
    - Way n°=
    - Type of street=
    - Name of the street=
    - Building=
    - City=
    - Postal code=
    - Cedex=
    - Country =

3.8 Contexts

We defined a particularly important notion to simplify the rights management and the ergonomics in the applications. It is the notion of context which defines a view on « his » social graph.

When we are in a given context, we only see the information validated for this context (« checked » in the context). When we see a node in a relation restricted to a context, we only see what the node « checked » in the context.

The notion of context allows a user to define which relations, which information and which parameters he manages when he is in a specific context.

Contexts are set at the node level.

Everyone can create as many contexts as he wants.

To each context he associates one name and one description.

The context also contains parameters for the portal (for instance element types of the portal displayed when we are in this context).

The context can also be a « master » context: all relations and data are available in this context. At least one context must be a master context.

To define what is authorized in a context, data linked to the node and the outgoing relations of the node are then associated or not to the context: to all contexts, to master contexts only or to contexts explicitely listed.

For personal data, these values can be indicated at any level of the tree.

By default, one branch inherits the contexts of the branch it is linked to.

By default, branches of the first level are all reserved for master contexts (and then by default, data is reserved to master contexts).

Different scenarios will be proposed by the portal to simplify the management of contexts and rights by the user.

When the user places himself in one of these contexts (via the connexion in the portal), he only accesses data (personal data and relationships) for which this context has been activated (see the complete example in the appendix).

When we access data via a relation, data and relations which can be seen are only those for which one activated context at least is common between the relation by which the access is made and data or relation which will be presented.

After this first filter, access rights (see further) are applicable.

It should be noted that for the accounts linked to nodes (individual or organisation), contexts are also defined: the connexion account gives then only access to data and relations of related contexts.

The change of context requires the reentry of the password (for obvious reasons of security).

3.9 Management of access rights

Data (personal data and links) of a node can be seen by the user according to the context and by its relations according to the rights which have been declared.

Data access is also linked to the application which wants to access it (consequently the user must explicitely give his personal data the access rights and his links to the applications).

Please note that the confidentiality management model described in this paragraph is a generic model.Interfaces allowing the management of rights will be able to present modele types according to the cases.

Rights on personal data:

Rights on personal data are defined at the level of each branch or each leaf.

It concerns rights in reading: the writing is possible only for the owner or delegated persons (see the delegation further).

By default, a branch inherits of rights from the parent branch. By default, the rights are reserved to the owner at the root level.

In general, access can be defined for each branch or leaf:

- Inherited from the parent branch
- Private access only (alone, the owner and his delegates can access).
- Public access: unlimited access (allowing one to define public information used for instance to be identified in the social graph)
- Limited access to listed groups and entities, except for those on the list of explicit exclusions

The list of groups and entities having an access (or on the contrary excluded from the access) can be:

- All direct relations: information open in reading by any node towards which a relation, whatever it is, is established.
- Relations of my relations (which allows one to show one's name to one's friends'relationships, otherwise they cannot see this information, generally very much used to increase the density of one's network).
- An explicit node (person, organisation)
- A group: all members of the group or sub-groups are concerned
- An account: when connected to this account, one is authorized or blocked.
- Special groups: see below.

Plesae note that in order to give rights to a node, you must have defined a relation with that node: the process of breaking up the relation will automatically prevent the access right (except for the « public » rights).

Special groups:

Special groups allow to define rights according to relations, for instance: all my colleagues, staff of an organisation, persons related to me via a link of such type, etc…

Rights on relations data:

As seen previously, relations are also personal data.

Their access (in reading, writing being reserved for the owner) must also be explicitly checked.

Please note that one relation from A to B is held by A (and consequently the rights of seeing this relation are managed by A): a relation from A to B is a personal data of A.

The reading access to a relation allows one to read the class of the relation and its attributes.

The access to the node's identity being at the end of the relation (then to personal data of that node) is of course submitted to the rights given by the node's owner.

Relations are submitted to the same mechanism of rights than personal data.

They are not visible by default (in the absence of explicit rights) but this right can be modified:

- Private access only (only the owner and his delegates can access)
- Public access: access without restriction (allowing a definition of public information used for instance to be identified in the social graph)
- Access limited to listed groups and entities, except for those in the list of explicit exclusions.

The list of groups and entities having an access (or on the contrary excluded from the access) can be:

- All direct relations: information open in reading by any node towards which a relation whatever it is, is established.
- An explicit node (person,organisation)
- A group: all members of the group or sub-groups who are concerned
- An account: when connected to this account, one is authorized or blocked.

3.10 Accounts

An account is a login and a password, a description and possibly a pseudo and an email account (compulsory if the account is not linked to a node).

An account can be related to a node of « Person » or « Organization » type.

If the account is related to an organisation, it gives the possibility of acting on behalf of this organisation.

If the account is related to a person, it gives the possibilty of acting on behalf of the person.

The link between the person and the account can be restricted to one or more contexts.

When connecting via the portal (or via one portal), once he is connected, the user can choose the context in which he wants to work from the list of the contexts associated with the account.

If the account is related to several nodes, he is first asked on which node he wants to work.

The change of context or node requires at least to reenter the password (or to change the account).

The possibility of linking several accounts to a same node allows one to create accounts with restricted rights for « batch » tasks for instance or for less secured device (on which the password is for example registered).

This will also allow the creation of « anonymous » pseudo-accounts for non authenticated access.

To simplify the management of the accounts, the login is free (but the unity must be verified: it is then recommended to include one's email address).

Logins such as *johndo@gmail.com/pro* or *johndo@gmail.com/tablette/pro* are then authorised.

It is possible to change one's login while keeping the same account.

Please note that the login is not the account internal identifier (see chapter on the identifiers).

3.11 Delegation

The notion of delegation allows one to connect at the level of a node having used an account connexion linked to another node.

The delegation within the meaning of the social graph is not a control of the delegation authorizations in the real world: the business processes are not exempted from making the appropriate checks.

Likewise, the delegation is not used to act on behalf of an organisation in an application: you connect to the applications with an account linked to a physical person and this is the

management of rights in the application which determines who can make a given action (these rights in the application are based on what is defined in the social graph: groups, positions, relations such as « employed by an organisation » or « has a job »).

The delegation (« Delegate » relation, the relation HasDelegation must be approved- then the interrelation « Delegate » must exist- to be valid), allows one to give one person (directly or via a position) the right to act for an entity.

In the case of the delegation of an organisation, this delegation gives all rights: the link « HasDelegation » once validated allows then to connect under the entity's name to manage this organisation (defining rights and relations for example).

The delegation must then not necessarily be total.

It must also be revocable by the node which gave the delegation or must not be revocable (in the case of a guardianship, only the guardian can revoke a delegation).

We then add the following attributes on the delegation's relations:

Attribute	Description
Context	The delegatee is limited to the context mentioned, not a master.
Guardian	**True/False**: (legal guardian) it is a legal guardian, the link cannot be revoked without the guardian's agreement. This link cannot be reduced to a context.

3.12 Connection to OASIS

Considering the concepts introduced previously, the connection procedure to OASIS is:

1) Connection: login/password (allows one to determine an « account »)
   - **If no entity linked to the account**: anonymous account (can have applications, but no personal data nor relationships)
   - **If one entity only is linked to the account**: one is connected to this entity
   - **If several entities are linked to the account:** choice of the entity

   Choice of the context (if the entity connected is a person)

   - **If the relation entity → account is linked to one single context:** we are in this context
   - **If the relation entity → account is linked to several contexts**: choice of the context

Delegations:

- Entities on which we have a delegation are accessible
- We can choose one of these entities:
    - Password request (out of security reasons)
    - If the delegation relation is towards a person and is linked to several contexts:choice of the context
    - The mechanism is recursive
- A return to the previous entity is possible (when entering the password)

Change of context:

- If the context is available with the same account: password request
- If the context is not available with the same account: request of a login/password compatible with the context

At a given moment we are then positioned on a node, we are seen as being that node:

- Account, person or organisation

3.13 Management of the objects' identifiers

The social graph elements (nodes, nodes attributes, relations attributes) must be identified by a sinlgle identifier.

Identifiers must not allow cross-referencing of information: they must then not have meaning and must be totally random (so as not to guess the attribution order).

However, it seems possible to give a prefix to each identifier to determine if this is a node or a relation (to simplify the debugging, as well as the research of this identifier in the grapoh).

The use of an identifier also allows one to rename the nodes and relations without any impact on the applications using these groups.

We then suggest an URI of RDF type:

sg:*type*/*id*

sg:	constant indicating this is the social graph
type	element type
	node node
	rel relation
	natt node attribute
	ratt relation attribute
id	identifier: a set of alphanumeric characters (upper and lower case letters, figures), random and unique.

The creation of identifiers will then have to use a random generator and a table of the identifiers already attributed so as to reusing an identifier.

For special groups (calculated groups and not explicitly described by a node), it is important to give applications and data services a single and permanent identifier but as less significant as possible.

3.14 Additional security

The social graph is particularly sensitive. It is then important that data cannot be useable, even by an OASIS administrator system.

At this stage, we do not implement any encryption: to be really efficient, the access key must be exclusively held by the user and data is then lost if the latter loses the key (physical key or password). This subject will be dealt with for a later version of OASIS.

We particularly consider the following methods:

- Separation of the data and the relations in different bases (being administered by different persons): a base contains the indexed information (data of entities), and the other one contains the relations between indexes
- Use of specific user codes by application, to avoid the risks of inter-applications reconciliations
- Limitation of the data access for every application according to what it really needs (which does not substitute for the explicit, temporary or permanent agreement that the user grants for the access to private data).

4 Authentication and Authorization

4.1 Identity Management in OASIS

4.1.1 Definitions

Identification. OASIS allows one to identify a user (without checking his identity in real life). Association of account (connection parameter) to entity (person or organization), allows each user to manage whole services and data once identified in the OASIS ecosystem. An id is given to each Service Provider and to each Data provider.

Authentication. It's the identity validating process, which allows one to give a proof of this identity (virtual identity). Authentication process could have several confidence levels according to the process used.

Authorizations. A user will have to allow a Service Provider to use his personal data given by a Data Provider, and his access authorizations to the scope of shared data are given to federated services in accordance with the scope governance.
The first role of the security server is to manage authorization. Prerequisite is identification and authentication.

So security server is at the same time identity provider (Identity Data Provider, IdP), in charge of the authenticity of the identity and manager of authorizations). Services Provider delegate authentication and authorization request to this security server. Data provider validates authorization with it.
Security server provides at least an authentication by password with the possibility to activate authentication with two factors (single use password, TOTP, RFC 6238).
Security server could delegate authentication to third servers: social networks to facilitate daily user connection; or STORK 2.0.
OASIS is also provider of different attributes of identity. Notice that STORK doesn't define minimal data that a member state has to provide, and leaves Service Providers the possibility to ask the user complementary necessary informations. When STORK 2.0 is used, OASIS will play this role and centralize personal information in the social graph.

4.1.2 Authentication Delegation

Authentication is centralized with the security server. Services providers delegate users' authentication according to OpenID Connect 1.0 protocol, developed by Google, Facebook,

Yahoo!, etc. within the OpenID foundation. OpenID Connect 1.0 is an extension of authorization protocol OAuth 2.0 standardized for identification and authentication at the Internet Engineering Task Force (IETF).

4.1.2.1 Overview of OpenID Connect 1.0

The service provider redirects users to the security server (for an installed application –ie. Mobile application) it will open the internet browser or integrate it into the application) giving client ID, list of the information which he wants to know about the user (name, first name, email, address…), and list of data he wants to update with data providers. Service Provider could indicate some essential data (required or optional).

```
HTTP/1.1 303 See Other
Location: https://security-server/authorize?
      response_type=code
      &client_id=s6BhdRkqt3
      &redirect_uri=https%3A%2F%2Fservice-provider%2Fcb
      &scope=openid%20profile%20http://data-access-protocol/1.0
      &state=af0ifjsldkj
```

In this simple example, service provider redirects user to security server at `https://security-server` indicating its id (`client_id`) and a private state, asking access to information of user profile (name, firstname, sex, date of birth, language, etc.) and a data Provider answering to identified specifications with `http://data-access-protocol/1.0`. It indicates URL which security server could contact another time.

Security server authenticates the user. If the user is recently authenticated to the security server, this step could be transparent for the user, the Security Server recognizes his browser. Then the Security Server checks authorizations given in the past by the user according to Service Provider asking. If needed the user will have to give his consent. If he does not agree, Security Server redirects him to the Service Provider with an error code. If it's OK, authorizations given are saved for the next utilization. Users can refuse some optional requests, step by step, without hindering the proper functioning of the authentication process. Security Server creates an authorization code and gives it to the service provider with redirecting user.

```
HTTP/1.1 303 See Other
Location: https://service-provider/cb?
      code=SplxlOBeZQQYbYS6WxSbIA
      &state=af0ifjsldkj
```

Notice: URL used here is, apart query-string, the one given during the initial request.

Service Provider has to exchange this authorization code by contacting on its own (without the user), the Security Server. Service Provider authenticates to the Security server during this call.

In return of the authorization code, Security Server creates, on request of Service Provider, an identity token (id token) and/or an access token eventually giv with a refresh token. It transmits also authorizations to which user has agreed.

All these tokens, and authorizations code, are specifically linked to user, Service provider and authorizations agreed.
Authorization code is single use and limited in time.
Identity token contains personal information about user (name, first name, etc.). It's e-signed with asymmetric keys. Service Provider has to validate the token before using it: validating the signature and checking that it was exclusively made for him.
Acces token and refresh token are used during communications between Service Providers and Data Providers (see below).

4.1.3 Communication between Service Providers and Data Providers

When a Service Provider needs access to user data from a Data Provider, it asks for authorization via the same process as the authentication delegation and asks the Security Server an access a token and possibly a refresh token.
An access token is transmitted by the Service Provider to the Data Provider during requests to transmit granted authorizations, using authorization protocol OAuth2.0.

4.1.4 Overview of OAuth 2.0

When data provider receives an access token, he contacts the security server to check if the token is valid and gives access to data asked. It is Data Provider's responsibility to transmit data asked only if the token is valid and contains necessary authorizations.
The access token is valid only for a limited time, after this time Service Provider will have to ask for a new one to Security Server. It could be done with the help of the user, with the same process than authentication delegation (user already authenticated and who already gave authorizations, the request of Access Token will be transparent), or with the help of a refresh token.
Refresh token is valid indefinitely and allows Service provider to ask new access token, even if the user is not connected during the request (for example for a recurrent task).
On the authorization request, user will have to agree that Service Provider accesses to his data even when he has not access to Service Provider.
The use of a refresh token to get an access token is similar to the exchange an authorization code.

4.1.5 Access to non-personal data

A service provider can also need to access data of Data provider which are not linked directly to a user (for example, some collaborative data in the data core). These data can be accessible without their owner being connected (and the owner could be a group, for example a public bodies group which collaborate on a scope of data).

Authorization server knows the scope of data (scopes stored in an OASIS datacore or in a federated data provider): so service provider will authenticate himself to the security server, to request an access token with a specific login if the request is not in an interactive process (with connected user), or with connected user login if it's the case. Accounts described in the social graph allow to manage used login by service in non-interactive mode.

4.1.6 Security

- All communications occur with a secured connexion (TLS).
- Service providers and data providers authenticate to the security server with a login/password, security brought by TLS being enough. Password is automatically generated by security server. Next evolutions of the platform could append support of signed JWT (authentication with asymmetric cryptographic keys rather than a password).
- Service providers use in a first time some Bearer tokens to authenticate users to Data Providers, security brought by TLS is enough and all parts belong to confident network (tokens could be revoked if needed). Next evolutions could bring MAC support (calculation of nonce and signature, in addition to token).
- In case of service provider's or data provider's compromise, his password could be changed (prevents new authorization requests or tokens validations), and all linked tokens to this provider could be revoked (preventing all token validation by other Data Providers). All is centralized at the security server level, only guarantor of credentials validity. The service provider could automatically revoke a token if he estimates that it is compromised (ex: high number of requests in a short laps of time and potentially on several Data providers); well a provider can be temporarily disabled automatically if the profiling of his applications seems abnormal (eg abnormally high number of requests in a short period of time and potentially several Data Providers, but using different tokens). Data providers who wish it could set up similar mechanism.
- Security Server does not store any sensitive information other than users 'passwords (if any) (encrypted) and the necessary various security protocol codes and tokens.
- Service Provider can theoretically store personal data of users, instead of asking the Data Providers. This should only be used temporarily to ensure a good level of performance (temporary caching of data). OASIS is a trusted network and a Service Provider who does not play the game can be excluded. However only an audit will detect such faults.

4.2 Implementation of the Identity Management

4.2.1 Identity

Each person is associated with a global unique identifier in OASIS, which identifies personal data. Data identity (name, address, email address, phone number, etc..) Data is stored in the graph of users associated with the unique identifier.

4.2.2 Authentication

A user account is linked to an identity (global unique identifier).
In the case where the user creates an account to connect to OASIS, he may opt (at any time and reversibly) for strong authentication, 2 factors (use of a password to use more unique passwords associated with the account). The password is either disposable generated upstream (scratch list or code) and printed by the user or generated by a third-party application as a function of time and with a time-based expiration window (One Time Password, TOTP, standardized by RFC 6238).

In case of loss of the password, the user can request a reset. A message is then sent to his e-mail with one-time login link to reset the password and enter a new one (if strong authentication was enabled for the account, the second factor must be used)

Any change in the terms of authentication automatically means sending a message to the e-mail address of the user. Similarly, in case of change of e-mail, a message is sent to the previous address. Any change requires the prior second factor (if strong authentication has been enabled for the account), or another authentication on a third department (if possible with this service) or uses another third department (if the user has associated several of them with his OASIS account); modality will depend on how the user is logged in to the session.

The specific authentication of the application should be replaced by OpenID Connect 1.0.

The application maintains its own mechanism to determine if the user is connected (usually a cookie managed by the CDM session development framework). If it is not, it redirects (HTTP Redirect) to the security server by specifying its client_id, the redirect_uri, scopes corresponding to the information that he wants to access (at least openid which provides the user ID) and, for greater security, the first single-use token (state) he will also store in a session to avoid cross-site requests forged (XSRF) and another single-use token (nonce) to prevent replay attacks.

For example, single-use tokens can be generated in Java using the SecureRandom class in PHP with the rand () function or with the Python random module. The URL to which the user will be redirected for example, asking only access to the data user's profile (the scope profile) (returns the line added for readability):

```
https://accounts.oasis.eu/authorize?
response_type=code&
client_id=s6BhdRkqt3&
scope=openid%20profile&
redirect_uri=https://application.example.com/cb&
```

```
state=security_token%3D138r5719ru3e1%26url%3Dhttps://application.ex
ample.com/myHome&
nonce=af0ifjsldkj
```

After authentication and authorization of the application to access the requested resource, the security server redirects the user to the redirect_uri indicating an error (if the application has not been approved), or a temporary code, accompanied in all cases of the single-use token provided previously. For example:

```
https://application.example.com/cb?
state=security_token%3D138r5719ru3e1%26url%3Dhttps://application.ex
ample.com/myHome&
code=SplxlOBeZQQYbYS6WxSbIA
```

The application must then, after checking that the single-use token state that is expected to make an HTTPS request to the server security by providing its client_id and client_secret as user name and password respectively HTTP authentication Basic and client_id, redirect_uri and his temporary code received in the query body as application / x-www-form-urlencoded format. For example:

```
POST /token HTTP/1.1
Host: accounts.oasis.eu
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpjbGllbnRfc2VjcmV0Cg==

grant_type=authorization_code&
client_id=s6BhdRkqt3&
redirect_uri=https://application.example.com/cb&
code=SplxlOBeZQQYbYS6WxSbIA&
```

It receives in response to JSON, an access token, which it will use to access Data Providers, a period of validity of the token (in minutes), and the profile data of the user electronically signed (JSON Web token format) and the second single-use token (nonce). For example (the id_token is edited for readability):

```
{
   "access_token": "SlAV32hkKG",
   "token_type": "Bearer",
   "expires_in": 3600,
   "id_token": "eyJ0 … NiJ9.eyJ1c … I6IjIifX0.DeWt4Qu … ZXso"
}
```

JSON Web Token (JWT) is composed of 3 parts Base64url coded and separated by a point. The first part shows the algorithm used for signature, the second is the message, and finally the third is the signature of the message with the private key of the security server and the algorithm described in the first part. All communications with the security server in HTTPS being performed, the signature does not need to be checked, but may be for increased safety. Public key server security arrangements are made by the security server to a known address. The application can then retrieve to validate the signature.

The message contains the JWT global identifier for the user and the second single-use token, for example:

```
{
                    "iss":                    "accounts.oasis.eu",
                "sub":                    "113945685385052458154",
                    "aud":                    "s6BhdRkqt3",
                    "iat":                    1370271794,
                    "exp":                    1370275694,
                    "nonce":                  "af0ifjsldkj"
}
```

The token disposable nonce must be validated.
The application must be able to make the link between global identifier for the user (sub) and the internal account for the user.
Profile information of the user are accessed via the user info end point whose access is authenticated as a data provider.

4.2.3 Personal data access via data providers

This case includes data cores, data-providers and federated social graph.
The access token provided by the security server for authentication can be used to access the Data Providers authorized by the user (including scopes were requested during authentication).

The token is passed in HTTP header to authenticate requests to the Data Providers (RFC 6750), for example:

```
GET /data?user=113945685385052458154 HTTP/1.1
Host: dataprovider.example.net
Authorization: Bearer SlAV32hkKG
```

In case of authentication error, the server will respond with a HTTP 401 error, for example:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer
    scope="http://dataprovider.example.net/scopes/data",
    error="invalid_token",
    error_description="The access token expired"
```

At any time, an application may request new authorizations by repeating the authentication process with different scopes.

4.2.4 Revocation of tokens

At any time, the application may revoke an access token by making a request to the security server. The request is authenticated in the same way to obtain the token from the temporary code. For example:

```
POST /revoke HTTP/1.1
Host: accounts.oasis.eu
Authorization: Basic czZCaGRSa3F0MzpjbGllbnRfc2VjcmV0Cg==

client_id=s6BhdRkqt3&
token=SlAV32hkKG
```

4.2.5 Data Providers

As for applications, Data Providers must replace their authentication mechanism by a mechanism based on OAuth 2.0 (RFC 6749) and Bearer tokens (RFC 6750). As applications, Data Providers must also be registered in advance with the security server to obtain a client_id and client_secret.

Applications to authenticate using an access token issued by the security server (see above). And the Data Providers must validate the token from the security server to check if it is valid (not expired, for example) and if it gives access to the Data Provider and possibly, if a user ID is known, if it provides access to data that user.

The Data Provider is therefore an HTTP request to the server indicating the safety (s) scope (s) corresponding (s) requested data, the access token and an optional user ID (sub). The request is authenticated with the client_id and client_secret the Data Provider. For example:

```
POST /validate HTTP/1.1
Host: accounts.oasis.eu
Authorization: Basic Y2JuYjFvczk6JTdCY2xpZW50X3NlY3JldCU3RAo=
Content-Type: application/x-www-form-urlencoded

client_id=cbnb1os9
scope=http://dataprovider.example.net/scopes/data&
token=SlAV32hkKG&
sub=113945685385052458154
```

If the token is valid in all points, the server responds with an HTTP response 204:

```
HTTP/1.1 204 No Content
```

Data Provider and continues processing the initial request.

In case of error, the answer will be a 400 and a JSON content indicating the reason for the error. For example:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "error": "invalid_token",
   "error_description": "The access token expired"
}
```

The JSON object contains an error property whose value can be:

invalid_request
The request is invalid (missing parameter, double, etc.).

invalid_token
The token is invalid (expired, revoked, non-existent)

insufficient_scope
The token does not provide access to data (or user scope)

The JSON object can also contain properties error_description to give precision error for developers and / or error_uri specifying a URL with more information about the error (to application developers).

This information may be passed as shown in the Data Provider response to the application, for example:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer
    scope="http://dataprovider.example.net/scopes/data",
    error="invalid_token",
    error_description="The access token expired"
```

4.2.6 Revocation

At any time, the client_secret an application or a data provider may be revoked, and another generated by OASIS administrator or an administrator of the application. When client_secret is revoked for an application, all the tokens issued for this application are automatically revoked.

A user can also revoke permissions at any time he gave an application from its OASIS account (on the security server).

4.3 Implementation example: Archiland

Alfresco is composed of two applications: Alfresco Share and Alfresco Explorer.
Each of these applications also exposes its data via Web APIs that are also data providers.

The data is stored only in Alfresco Explorer and Alfresco Share uses the Alfresco Explorer Web APIs to manipulate (and expose them to turn).

Archiland based solely on Alfresco Explorer, but could make use of repositories stored in the heart of data.

CIFS, IMAP, WebDAV (see below) and SharePoint are not taken into account here.
In our example case we are interested in Alfresco Explorer only, this is the simplest case. Yet, Alfresco has multiple modes of access, and each has a replaceable authentication mechanism.

4.3.1 Registration

As an application , Alfresco Explorer is registered with the security server and is assigned a client_id couple and client_secret .
As a data provider, a couple and client_id different client_secret be awarded.

The application and data provider are stored in the same "project" and appear as a single entity for the user, so that if an application requests access to data stored in Alfresco , the user correctly identifies the data that it manages itself by the application.

It may be differentiated based on the level of access required authentication (user vs . Administrator), transaction type (read only or read / write) or family WebScripts and manage multiple scopes. OASIS will however only be able to manage authorization service access according to these access levels , Alfresco continues to guarantee access to authorized according to the user, according to its own rules of authorization data.

4.3.2 Authentication

To change the authentication mechanism Alfresco Explorer, create a new "Authentication Subsystem" and configure Alfresco to use.

4.3.2.1 Authentication to application

For authentication purposes, we define a class `eu.oasis.alfresco.web.app.servlet.OasisAuthenticationFilter` which implements `org.alfresco.web.filter.beans.DependencyInjectedFilter` repo. and in that the subsystem configures as `authenticationFilter` name. Its role will be to check if the user is already authenticated and otherwise redirect to the security server OASIS. Whether the user is already authenticated, you can use the servlet session (used by Alfresco in the normal case). When redirecting the originally requested URL is included in the state parameter. Differential access to the admin interface can be implemented based on the requested URL, thus requiring a different scope.

This class, however, manages access to the Alfresco Explorer application itself. Alfresco also exposes Web Scripts, that develop both Web APIs and user interfaces, with several possible authentication modes, including a user based on the authentication Alfresco Explorer application.
For authentication to Web Scripts, we will create a similar to the previous class, configured in the subsystem under the name `cookieBasedAuthenticationFilter`, and adding support for different scopes for differentiated access described above.

To implement the redirect_uri, given the two possible access points above, use the `globalAuthenticationFilter` intercepting most requests to the server. This must be a class that implements `DependencyInjectedFilter.` Its role is to:

1. receive temporary code from the security server
2. exchange it against an access token and profile information of the user
3. create or update the user in the database Alfresco users from profile information (the user ID is the identifier Alfresco OASIS)
4. create a session to see the authenticated user in its subsequent requests redirect to the originally requested URL (and recorded in the state parameter)

Finally, replace the authenticationComponent by an instance of `org.alfresco.repo.security.authentication.SimpleAcceptOrRejectAllAuthenticationComponentImpl` who reject any attempt authentication password.

4.3.2.2 Authentication to Web APIs

Web Scripts Web APIs used as our Data Provider in the case of OASIS, use a different authentication. In continuation of the above changes, a class that implements the interface will be created `org.springframework.extensions.webscripts.servlet.ServletAuthenticationFactory` and configured in the subsystem under the name `webscripts.authenticator.basic`.

Its role will be to authenticate the user on the basis of an access token: The access token will be required on each application, and will send to the security server for verification. A caching mechanism can be added to limit such requests to the server security and thus improve performance, but at the expense of safety cover will therefore have a timeout as short as possible to maximize safety, a compromise will be found to achieve a balance between performance and security, within the order of a few seconds should suffice. The cache can also check the validity of the ticket in the background so as not to impact the client.

Likewise, we remove the access authentication via Facebook portlets and mechanisms, by replacing their respective `ServletAuthenticationFactory` by denying all access classes.

4.3.2.3 Authentication WebDAV

WebDAV can be considered as a Web API, in which case we will add a class that implements the subsystem `org.alfresco.repo.web.filter.beans.DependencyInjectedFilter` and registered under the name `webDavAuthenticationFilter.`

However, most WebDAV clients (to use Alfresco as a network drive on the user's computer, for example) are not compatible with OAuth 2.0. In this case, a specific password could be used, generated specifically for the Service Provider and scope (s). This is however outside the initial scope of OASIS.

4.3.3 Access to external data

Archiland uses repositories that could be stored in the heart of data. We will replace the code that loads the Alfresco repository from a version that queries the heart and puts the data repositories cache (you can also keep the current code, but add a recurring task that will update local repositories from heart data).

4.4 Authentication and social graph

4.4.1 Link between two services

The end point of user info OpenID connect accessed social graph to retrieve the profile information of the user filtered according to scopes associated with the authentication token and the context of use of the user associated with the token itself.

4.4.2 Access to personal data and implementation of the context notion

The notion of context can meet three needs:

- Use a single account, a single identity in order to reduce the cognitive load (only one username, one password, no need to remember what information I have entered (or not) on each account, remember to update all my accounts when information exchange, etc.).
- Have full control over their personal data:
  - what is shown in the apps when they are used
  - what shows when they use other apps that read the relations (corollary: to keep control, our information is shared, not everyone re-enters the information he knows from his relations, as opposed to a "book Address "for example)
- Identify applications in which context we are (personal, professional, association, etc..)

4.4.2.1 Indicate to applications in which context we are (perso, pro, association, etc.)

As we have only one identity used in several contexts, it is necessary to indicate to applications in which context they are used. There is therefore a notion of context of use, which must be communicated to applications.

4.4.2.2 Monitor what we show to used applications

One of the most common cases provides different results in different contexts (eg. private, political, religious, pregnancy, etc. - hidden in a professional context, confidential business information hidden in a non-pro).

For applications used in different contexts, the user does not necessarily show the same information in different contexts.

The right of access to personal data applications is attached to the context and not defined per application globally. There is no need to define a per-application -level rights context, the context selection is sufficient. If the user wants to show something different applications for the same "context" of use (personal, pro, etc.) Has the ability to create contexts demand all he wants.

In terms of ergonomics, for each personal data, grouped by "blocks" of information, it simply defines which context shares information with applications. A context is duplicated by copying access rights. A context is created ex nihilo by definition rights for the list of applications used.

4.4.2.3 Monitor what we show to others (when they use applications which read relationship)

Access rights are positioned on their personal data for relationships that want to access (via applications, most of the time). In terms of ergonomics, simplify management, relations are grouped (groups of relationships: my colleagues, my family, my home, my friends of football, etc...) And rights are assigned to these groups.

Depending on the application used by his relations, the user may not want to expose the same information: he shares many things with his wife, which is very useful for administrative procedures, but does not wants "fun" applications to have access to all this information; as his wife can choose what she presents to each application through the choice-of-context, he must also be able to choose what he presents to the applications when the latter are used by others. Thus, for each personal data, grouped in blocks of information, we can know who has the right to access (rights of social graph) and with which application (security module).

As with any function, defining access to personal data must be simple ergonomics and features shown unambiguously.

- The default access are created and used by all applications.
- In a menu is presented the list of applications used and / or have accessed the personal information of the user via other users
- When the user edits the default path, he must be able to know if defined custom application to access this information block (eg date of birth available in A, B and C, you have defined custom duties for X and Y applications)
- When an application accesses personal data for the first time (or, for the first time in a given relation), the user is notified and can easily switch to the customization screen rights for this application.

5 Management of rights in datacore

5.1 Reminder of the need

In the study of architecture (document D1.2), we introduced the need to manage data access rights in the data cores, with two objectives:

- Ensuring governance rules on data.
- Allowing applications to manage access rights, especially for business data for which this concept is important.

As indicated earlier in this document, rights management is based on the social graph, and on its different nodes (including groups), but also on computed groups, that is to say from the definition of a rule linking.

For example:

- All employees of this organization
- All employees of data owner
- The relationship of my relationships

5.2 Scope of governance

We introduced the concept of "scope", which corresponds to a perimeter data governance.

Note that this concept also used for the authorization system allowed to access an application with a "scope" of data.

For example, a scope can represent the street furniture in an *Agglomeration*.

The scope is then defined by data types, and by a perimeter which can be geographical or of any other type.

In order not to include complex business rule in the datacore, the scope of data will be reported by the application that creates the data (and its modification requires "rights management" rights)

In reading, it should be possible to read any data type without knowing all relevant scopes: scopes are used exclusively for writing permissions at the level of the authentication server.

In reading, the authorization may be given for a type (schema) data for a full container, or type / container couple, or indeed for a scope.

The scope is a queryable attribute (using one or more scopes in a read request to restrict the search scope).

On direct access to an element, the scope specification not being that of the element generates a specific error message (this is a specific access restriction)

The scope defines governance rules and includes the following information:

- Default owner and authorized owners
- Mediation Rules: types of mediation authorized, default mediator, authorized mediators
- Conservation rules of Histories
- For each type of access: default access rights, maximum access rights (i.e. limited access), and minimum access rights (to prevent an application from limiting access beyond a certain limit)
- The data license (this is a searchable information via the broker, which does not trigger automatic processing)

5.3 Elements for rights management

5.3.1 Access lists

Access is given for the social graph nodes, or computed groups.
The social graph therefore provides groups and pseudo user groups.

Access lists (defined by type of access: reading, writing, etc ...) are either:

- "Public" (the access being submitted to the fact that the application can access this type of data in the authorization server)
- Limited to a given list (consisting of users, organizations, groups, and advanced groups)
- Calculated by the social graph (the rule of definition of rights is the responsibility of the social graph, which thus provides a resource for checking the rule according to authentication settings)

Default access lists are defined at the scope level.

5.3.2 Owner

The owner of a given node is a social graph:

- A group
- An organization
- One (in an organization)
- A physical user

- An account

This is the application by creating a data set owner.

A default owner is defined at the scope.

Authorized owners are also defined at the scope.

If proposed by the application owner is not allowed, the default owner is assigned.

5.3.3 Mediator

The mediator of a given node is a social graph:

- A group
- An organization
- One (in an organization)
- A physical user
- An account

The application defines the mediator upon creating data. The mediator is defined by default at the level of the scope:

- Either the owner
- Or the explicit node.

The authorised mediators are also defined at the level of scope. If the mediator proposed by the application isn't authorized, the default mediator is applied.

Note that for an organization the mediation request is sent to the organization as well as to delegatees of the organization

For a group, mediation is sent to all members of the group (using recursion groups)

5.3.4 Owner and mediator defined in the scope.

For each scope (for each type of data included in the scope) are defined:

- The default owner
- The default mediator
- The list of owners who may be affected
- The list of mediators who may be affected

These values are not necessarily static. The values may be:

- An explicit node (one group)
- A field of authentication (the datacores then replaced by the corresponding value retrieved from the authorization ticket)
  - The authenticated node
  - The delegatee
  - "of" value

6 Conclusion

The principles and security solutions which have been set-up allow the OASIS platform to respect the constraints of confidentiality and the protection of privacy studied in délivrable D1.32.

Personal data is stored in the social graph, which allows each user to finely control its use, management of access rights and control of the history of effective accesses.

OASIS doesn't manage sensitive private data (as defined the relevant legal codes) during the pilot phase. Since data are fully controlled by their owners, who explicitly choose to whom to provide access and when to delete data, such data could be included in the next stage. However, we consider that this requires complementary security mechanisms (including certification of services as to their security levels, an issue which will be studied during the pilot phase, as well a cryptography with no possible access to the key without explicit approval by the data owner.)

In the data core, we introduced the notion of "governance scope", allowing the guarantee the application of governance rules chosen for data.
The possibility of implementing multiple data cores (and thereby located a data core within a specific country), allows one to meet the requirements of national legislation about public administrations' data. A data core will be hosted in Italy with this aim, and the main data core will be hosted in France.

The implemented technical architecture (see D2.1) guarantees that security and availability constraints are respected.
The authentication and autorisation mécanisms presented above (OAuth2) as well as the systematic use of encrypted exchanges on the network (protocole HTTPS) reduce to a minimum the risk of usurpation.

Critical access (management of personal data, access authorization for data, authentication and management of passwords) are approved on the provided web interfaces directly by the core's security module and not by the portal or a federated service.

D1.32 addresses the specific requirements for the social graph and authentication described in the current document. They are summarized in the table below.

Requirement	How we adress it
Robust authentication system that includes: • password management and means of confirming the identity (card, biometrics, etc.) • management of access sessions • access security: two factor authentication	These topics are adressed by OAuth 2.0 and Open ID Connect 1.0. This is described in paragraphs 5.2.2, 5.2.4, 5.3
Protection against malicious insiders	We propose mainly two solutions to reduce this threat: • the social graph (where personnal data are stored) will be divided into two database (see paragraph 4.14) • Service and data providers are strictly identified, and can be revoked
Protection of underage users	The principle of delegation set up allows parents or legal guardians to manage data confidentiality for child (see paragraph 4.11) By default, the data is confidential, sharing must be explicit, which contributes to users sensibiliation about privacy (see paragraphs 4.8 and 4.9)
Adequate safeguards to secure authentication and authorization	We use Open ID Connect 1.0 and OAuth 2.0 solutions (see paragraphs 5.2.2, 5.2.4, 5.3): The OASIS kernel doesn't verify real identity: for sensitive process, federated services operating this process have their own mechanisms to verify real identity, exactly as when used outside OASIS.