D3.1: Cooperation Incentives and Trust Management Pre-Prototype Software Report

Deliverable Number	D3.1
Lead Beneficiary	UniGe
Nature/Dissemination Level	CO
Working Group/Task	WP3/Task3.1
Editor	UniGe (Carlos Ballester)
List of Authors	***UniGe*** (Carlos Ballester, Jean-Marc Seigneur), ***ULHT*** (Christian Pereira, Paulo Mendes, Rute Sofia, Waldir Moreira), ***UniUrb*** (Alessandro Bogliolo, Lorenz Cuno Klopfenstein, Saverio Delpriori, Gioele Luchetti), ***Level7*** (Paolo Di Francesco, Marzia Mammina).
Date (Project Month and dd.mm.yy)	Month 22, 30.06.2012
QAT Reviewer	Paolo di Francesco (LEVEL7)

Executive Summary

This document provides a description of the software package that is to be delivered on M21 as deliverable D3.1 of the EU FP7 ICT project ULOOP (User-centric Wireless Local Loop, grant Number 257418). The deliverable D3.1 falls into the category of “Others” and is composed of software, as well as this report.

This report provides a guideline to the specification for the ULOOP block Trust Management and Cooperation Incentives, providing details on the design and implementation status of this ULOOP component.

History

Version	Date	Author	Description
0.1	08/05/2012	Carlos Ballester	First draft of the document.
1.0	12/06/2012	Carlos Ballester	Second draft of the document
2.0	26/06/2012	Carlos Ballester	Third draft of the document
final	30/06/2012	Carlos Ballester	Final version of the document

Table of Contents

List of Figures

List of Tables

Acronyms

Acronym	Meaning
ULOOP	User Centric Local Loop
D[XX]	Deliverable numbered XX
OSNs	Online Social Networks
PET	Privacy Enhancing Technologies
DT	Dispositional Trust
EDGE	Enhanced Data Rates for GSM Evolution
3G	3rd Generation Mobile Communications
CAC	Call Admission Control

Acknowledgements

Acknowledgements to all partners for their significant contribution.

1. Introduction

This document provides a description of the software package that is an integrated part of deliverable D3.1 of the ULOOP project.

1.1 Document Scope and Purpose

Deliverable D3.1 presents a pre-prototype of the software being devised in Task 3.1 of the ULOOP. This report aims to be a guideline for the high-level specification architecture and software specification for the pre-prototype software release. The goal is to provide a better insight on how Task 3.1 has been de-composed in main blocks and sub-blocks, how those blocks operate and how they interact between them: to achieve such goal this report details architectural aspects and sub-block operation and procedures, and explains how they fit in the main ULOOP picture.

The document is organized as follows. Section 2 describes the main architectural definition of the trust management and cooperation incentives block, including the global flow-chart for this task. In section 3, we describe the pre-prototype implementation achieved so far, including design choices and limitations. Section 4 provides a set of recommendations to be observed for the first software version.

2. Architecture Definition and Specification

<UniGe: Jean-Marc, Carlos, ULHT : Paulo, Rute>

In ULOOP, trust management and cooperation incentives are related to the understanding of how to define and build circles of trust on-the-fly. Such circles of trust aim of sustaining an environment for allowing devices to share resources to support the dynamic behaviour of user-centric networks. Trust management is based on reputation mechanisms able to identify end-user misbehaviour and to address social aspects, e.g., the different levels of trust users may have in different communities (e.g., family, affiliation). In situations where the created network of trust is not enough to allow resources to be shared, ULOOP devices are able to use a cooperation incentive scheme that allows a node to gain credits in an amount directly proportional to the amount of shared resources: such credits can then be used to gain access to other resources.

Another key aspect relates to the development and validation of a set of methods and techniques that make it possible to optimize network resources in regards to social behaviour, i.e., exploiting shared interests or On-line Social Networks (OSN) information to create/optimize/add trust to ULOOP communities.

Hence, trust management and cooperation incentives aspects are split into three main blocks: i) Trust management; ii) Cooperation Incentives; iii) Identity management. These major blocks are illustrated in Figure 1.

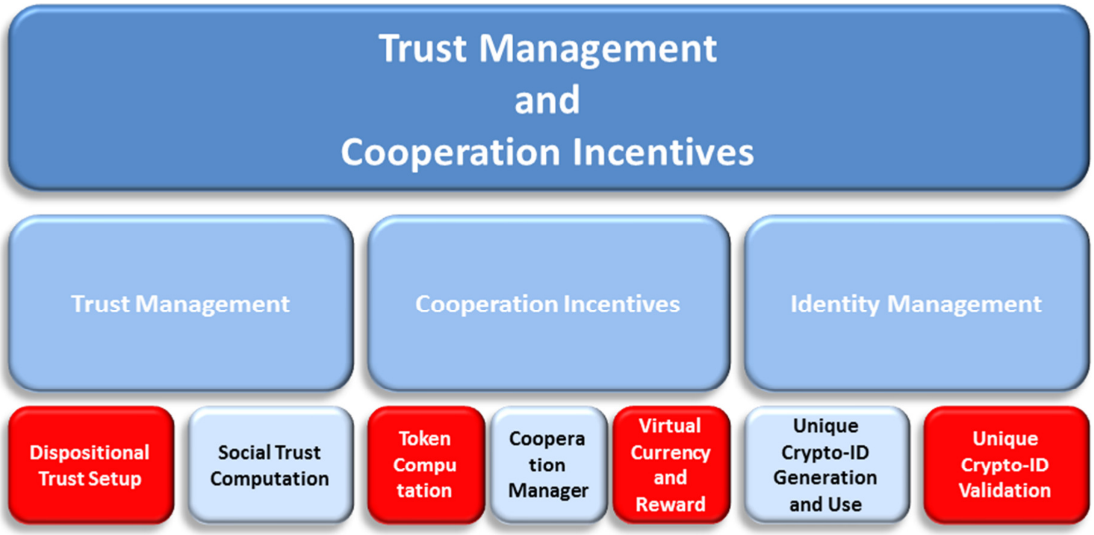


Figure 1: High-level Architecture, Trust Management and Cooperation Incentives Block.

As illustrated, we have further split each of the blocks into sub-blocks, which correspond to different object-oriented modules. From a ULOOP software suite perspective and as has been explained in D2.3 [1] and D3 [2], there are modules which are activated if a ULOOP element plays the role of a regular node, or of a gateway.

To provide a perspective on how the global trust and cooperation framework works on ULOOP, and for the remainder of the document, we shall consider the role of a *requestee* (providing resources) or of a *requester* (requesting access to a set of resources, such as Internet connectivity) in trust and cooperation negotiation. **A requestee in ULOOP can only be a ULOOP gateway**. **While the requester role can be assumed by both a node and a gateway**: nodes perform trust negotiation towards gateways; gateways perform trust negotiation among themselves. The global flow-chart, for trust management and cooperation incentives, is illustrated in Figure 2.

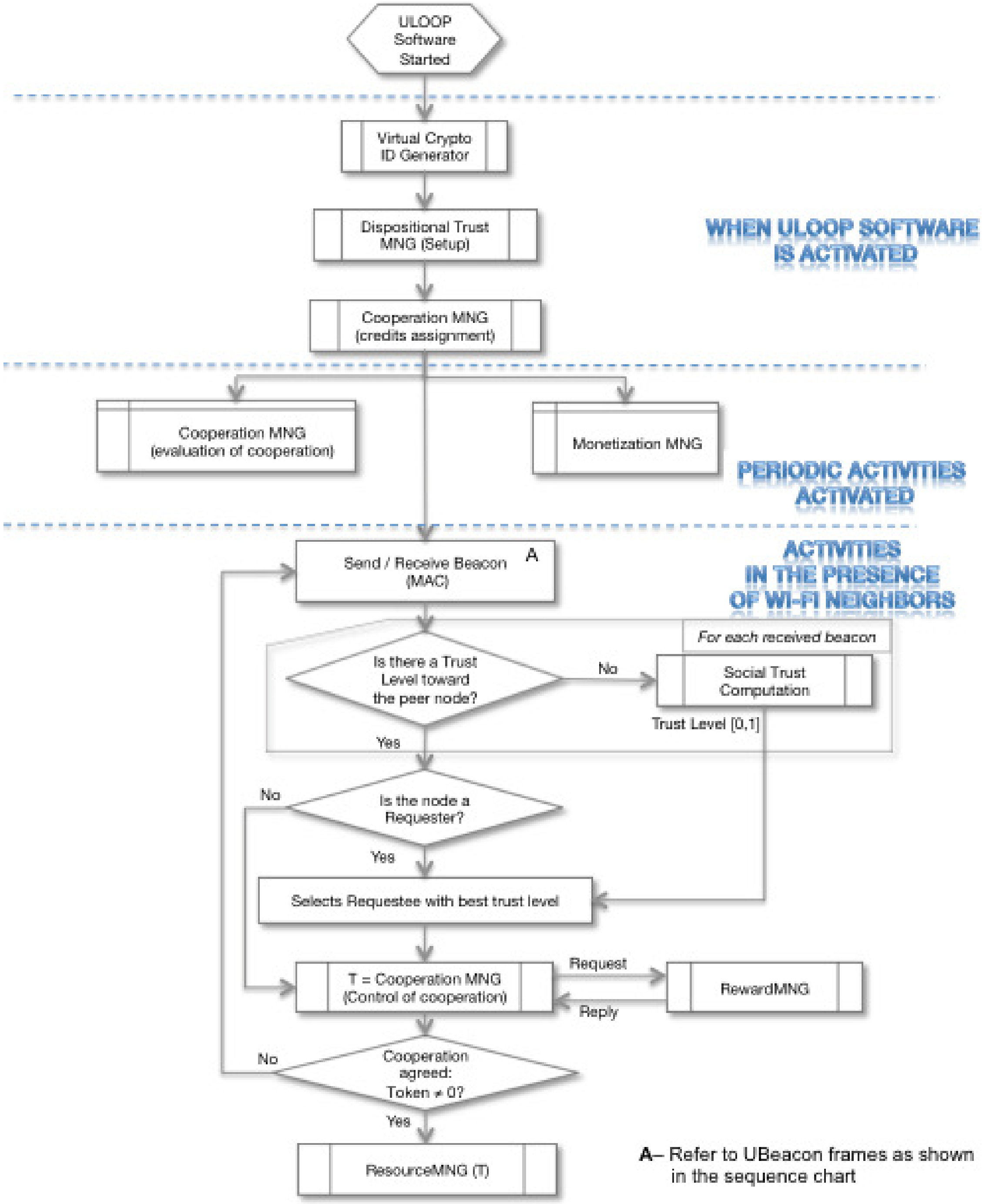

ULOOP Software Started
Virtual Crypto ID Generator
Dispositional Trust MNG (Setup)
Cooperation MNG (credits assignment)
WHEN ULOOP SOFTWARE IS ACTIVATED
Cooperation MNG (evaluation of cooperation)
Monetization MNG
PERIODIC ACTIVITIES ACTIVATED
Send / Receive Beacon (MAC)
A
ACTIVITIES IN THE PRESENCE OF WI-FI NEIGHBORS
For each received beacon
Is there a Trust Level toward the peer node?
No
Social Trust Computation
Trust Level [0,1]
Yes
No
Is the node a Requester?
Yes
Selects Requestee with best trust level
T = Cooperation MNG (Control of cooperation)
Request
Reply
RewardMNG
No
Cooperation agreed: Token ≠ 0?
Yes
ResourceMNG (T)
A– Refer to UBeacon frames as shown in the sequence chart

Figure 2: ULOOP Trust Management and Cooperation Incentives Global Operation flow-chart.

As illustrated in Figure 2, a node has three well defined states:

- boot-up
- activation of periodic activities
- data transfer

The boot-up phase is executed in any ULOOP node, be it a requestee or a requester, since it aims to establish the initial set of conditions for the participation in a ULOOP community. A boot-up steps to be executed, after download of ULOOP software, the initial setup which comprises:

- **Creation of the virtual identity crypto-id.** As has been explained in D3 (refer to section 2.1.4.1 Virtual Identity), in ULOOP the notion of crypto-id has been considered to assist the process of computing and managing the set of trust associations among any pair of ULOOP devices. The goal is to mitigate the impact of impersonation and non-repudiation, while insuring the right level of privacy, e.g. by relying on Privacy Enhanced Technologies (PET). Hence, a crypto-id identifies a unique interconnection of a user and a device. We have opted for the implementation of the unique crypto-id concept that has been presented in D3, section 2.1.4.1.1, and shall further address the current implementation on sections 2.1 and 2.2.
- **Setting up dispositional trus**t. As also defined in D3, dispositional trust (DT) is the general willingness of a given user to trust others, which we have considered to be a value between 0 and 1 that a person will use to configure the ULOOP service. The DT setup is done in the boot-up phase and it may or may not remain the same during the node's lifetime. For the first implementation of ULOOP, we consider a single set-up, without changes during the trust negotiation aspect. **Adaptation is an aspect that we expect to address during year 3 of ULOOP**.
- Initial assignment of credits: At this stage, the ULOOP device has no trust level in the system and its trust on others (as given by the dispositional trust) may prevent such node from interacting with others. Thus, to overcome such issue, a certain amount of credits are assigned to the node. This will motivate the user to interact in the system while its trust level improves (i.e., increases) according to its interaction. The credit assignment process considers the node's dispositional trust level and the amount of credits reflects the nodes interaction in the system.

After a boot-up phase, executed only once, we proceed by activating two periodic activities of the Cooperation Manager and of the Reward Manager: the Cooperation Manager manages credits to motivate volunteer cooperation, and the Reward Manager manages credits when a gateway is only willing to cooperate if rewarded by its actions.

After the phase were periodic activities are started, an ULOOP device enters the data transfer phase, which encompasses all the activities performed when data needs to be exchanged between nodes and gateways. Trust management itself starts each time a node attempts to perform regular MAC authentication, i.e., each time a ULOOP node is in the vicinity of some gateways. Due to regular scanning and overhearing, ULOOP nodes get ULOOP beacons (extension of regular Wi-Fi beacons) that allow them to understand if there are available gateways around. Before MAC authentication, it is necessary for a node to trust some gateways. Hence each node relies on the social trust level that it has toward neighbour gateways: if such value does not exist, the node will compute it based on a social trust computation function (c.f. section 2.4). The computed trust level will be used by a requester (node) to select the most trustful potential requestee (gateway), with which the requester will initiate a cooperation section: the cooperation session may include the establishment of a reward. If the cooperation section is established with success (in case of volunteer cooperation or if credits allocated by requester are above the threshold established by a requestee that wants to be rewarded), the requester will start the resource management procedure. Otherwise the requester will look for another trustful gateway with whom it will try to establish a cooperation session.

The next subsections address each of the sub-blocks in detail.

2.1 Unique Crypto-ID Generation and Use

<UniGe: Jean-Marc, Carlos; ULHT: Paulo>

This section provides a detailed description of a Crypto-ID generation scheme that can be used by ULOOP nodes in order to generate their own crypto-id to further validate it afterwards.

2.1.1 Computational/Algorithmic Aspects

In ULOOP, the crypto-id of a new node that is introduced into the system is calculated using the following formula displayed in Equation 1.

$$Crypto - ID = SHA256\ (\ P_{key}\ )$$

Equation 1

, where:

SHA256(x): cryptographic hash function producing a message digest of 256 bits over x.

P_{key}: is the public key of the user.

2.1.2 Specification

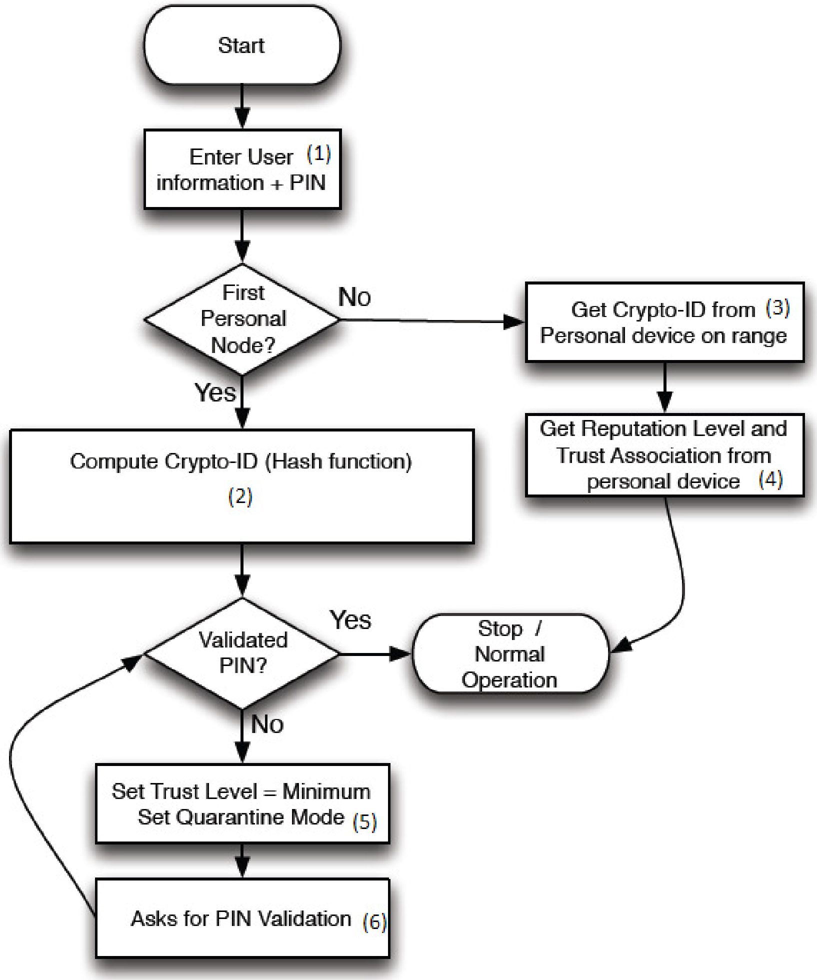

Start
Enter User information + PIN (1)
First Personal Node?
No
Yes
Get Crypto-ID from Personal device on range (3)
Get Reputation Level and Trust Association from personal device (4)
Compute Crypto-ID (Hash function) (2)
Validated PIN?
Yes
No
Stop / Normal Operation
Set Trust Level = Minimum Set Quarantine Mode (5)
Asks for PIN Validation (6)

Figure 3: Crypto-ID generation and use flowchart

Figure 3 shows the flowchart to generate a unique crypto-ID based on a set of information extracted from the user's device, namely a public key. Such information will be used to generate a unique crypto-ID based on a hash function, taken over the previous piece of information, which is implemented in any ULOOP node or gateway. The local generated crypto-ID will need to be verified by an authorized entity in order to allow the ULOOP node/gateway to gain full access to the ULOOP community. While such verification does not happen, the ULOOP device gets a minimum trust level in the community, allowing it to use a predefined set of minimum resources.

In ULOOP, owners (users) are likely to be responsible for more than one active device. One would be a primary device, and the remainder equipment share the same crypto-ID generated by the first personal device, as well as the reputation level and trust associations associated to the unique crypto-ID. This is possible by using secure in range wireless or wired communications. Synchronizing the reputation levels and trust associations among personal devices will allow the user to always make use of the earned reputation level, trust associations and credits that resulted from the usage of the unique crypto-ID in another personal device. Synchronization of trust information can be done by using prior-art on file and data synchronization.

The validation of the unique crypto-ID can be done by making use of any opportunity to access the Internet (limited Internet access should be allowed by the minimum trust level). This may create some problem in extreme cases, in which Internet access is not possible for a long time. However, such scenarios are more related to delay-tolerant networks than to ULOOP. In the latter case it is expected trust management and cooperation incentives to create the conditions to make Internet access more pervasive than today.

Table 1: Mapping of crypto-id generation and use Flow-Chart, major operations.

Flowchart reference	Function method/descriptor	Description	Path to Code

1	setNickname(askUserWantedNickname());
	createLocalKeyPair();
		Establishes a first set of needed parameters for the crypto-id generation	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//simulation/Requester.java

2	public CryptoId(PublicKey publicKey)	Computes the node's crypto-id	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//crypto_id/CryptoId.java
3	getKeyConfigurationBundlefromDeviceInRange(String cryptoId, ChallengeResponse challengeResponse)	Retrieves the keyPair from another node owned by the same user.	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//simulation/Requester.java
4	autoSetupBasedOnConfigurationBundle(ConfigurationBundle configBundle)	Copies the appropriate values from another node owned by the same user.	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//simulation/Requester.java
5		Sets the node in quarantine (low trust level) until validated	
6	validateNickname(String wantedNickname, String phone_number_to_send_sms, CipherParameters publicKey)	Validates and binds the choosen nickname with the crypto-id	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//crypto_id/crypto_id_validat

			ion/IdValidator.java

2.2 Unique Crypto-ID Validation

<Level7: Paolo, Marzia>

This section provides a detailed description of a Crypto-ID validation scheme that can be used by ULOOP nodes that have cellular interfaces with data capacity (e.g. EDGE, 3G), besides the required Wi-Fi interface. The described scheme is an example of a Crypto-ID validation scheme based on SMS messages for the market of Wi-Fi equipped cell-phones that will be used by the demonstrator being prepared by Level7.

For ULOOP nodes without a cellular interface (e.g. some tablets, laptops), the ULOOP proposal for the validation of Crypto-ID could be based on the use of smart id cards, as they are extensively being introduced by many governments when issuing new id cards and they contain certificates which prove the identity of the citizen, or on the use of credit cards, by charging a refundable small amount of money (typically in the range 0.1 - 1 monetary units, be it U.S. dollars, euros or any other currency) in order to verify the identity of the user. One of these options, or perhaps any other that might be deemed suitable, will be chosen and will be described in the D3.4 deliverable.

2.2.1 Specification

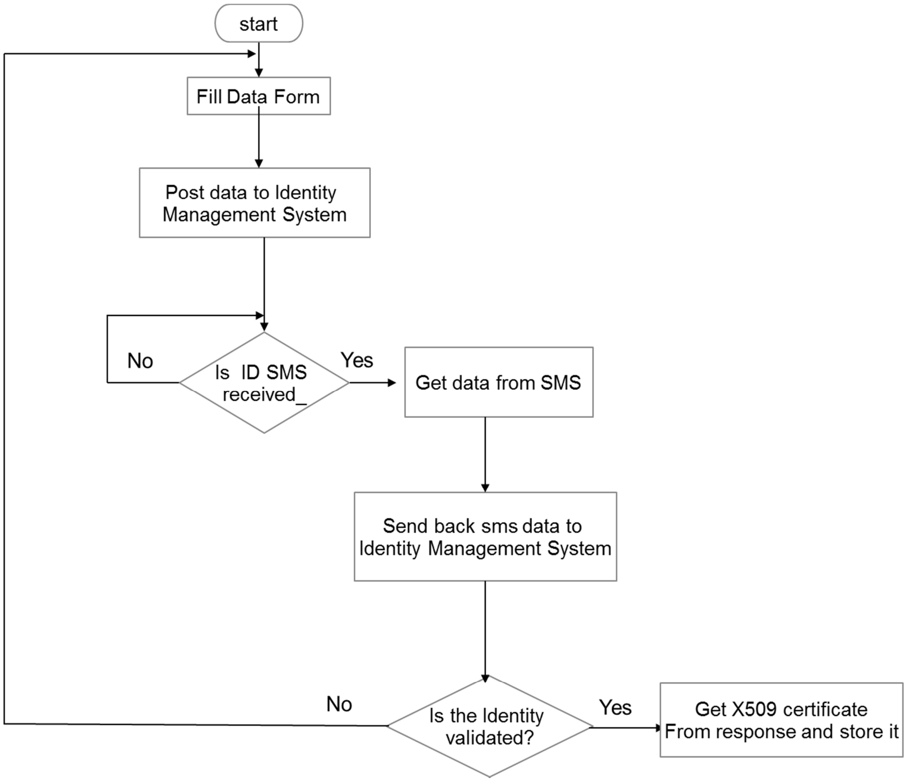

start
Fill Data Form
Post data to Identity Management System
No
Is ID SMS received_
Yes
Get data from SMS
Send back sms data to Identity Management System
No
Is the Identity validated?
Yes
Get X509 certificate From response and store it

Figure 4: Unique Crypto-ID validation, ULOOP node side flowchart.

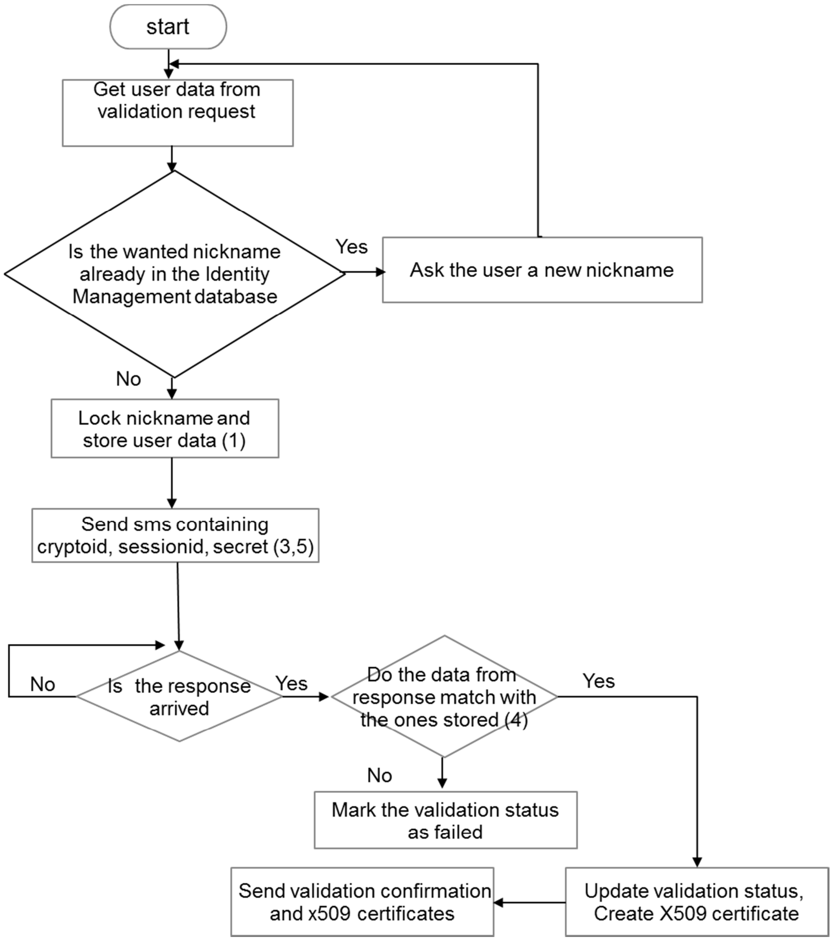

start
Get user data from validation request
Is the wanted nickname already in the Identity Management database
Yes
Ask the user a new nickname
No
Lock nickname and store user data (1)
Send sms containing cryptoid, sessionid, secret (3,5)
No
Is the response arrived
Yes
Do the data from response match with the ones stored (4)
Yes
No
Mark the validation status as failed
Send validation confirmation and x509 certificates
Update validation status, Create X509 certificate

Figure 5: Unique Crypto-ID validation, server side flowchart.

When a node joins ULOOP for the first time, a crypto-id (public-private key pair) is generated. After that the user can request a ULOOP partner (Identity Validator) the validation of such crypto-id, so that the end-user can prove with cryptographic strength that he/she really owns the secret linked to the crypto-id. This would require some steps for the authentication and verification of the user's identity in the real world. For example, the Italian law requires each user to own of a mobile phone number (which is related to the user's real world identity) combined with a one-time verification of the authenticity of the identification.

Figure 4 shows the crypto-id validation from client side (ULOOP node), while Figure 5 shows the validation from server side (Identity Management System). Some user data is required for the validation: first name, last name and mobile phone number in order to be able to perform the SMS validation. Moreover the user is asked to choose a nickname that will be linked to the crypto-id. Verification near the Identity Management System will ensure the uniqueness of the nickname. The

Identity Management System, owned by the Identity Validator, proves the ownership of the provided mobile phone number sending a SMS with a secret to that mobile phone number.

The ULOOP node intercepts incoming SMS messages (with a predefined format) and when recognizes the message sent by the Identity Management System, it sends back the secret received in the SMS via http together with the chosen nickname and other additional information.

If the Identity Management System recognizes that the replied data and secret are the same stored for that Crypto-Id in its database, then the validation can be considered completed and the confirmation is sent to the node together with an X.509 [3] certificate. The purpose of the X.509 certificate is to bind the public key of the node to a particular distinguished name or to an alternative name such as an e-mail address, or in ULOOP case, a ***nickname***. The node marks the Crypto-Id as validated and stores the X.509 certificate.

Table 2: Mapping of crypto-id validation Flow-Chart, major operations.

Flowchart reference	**Function method/descriptor**	**Description**	**Path to Code**
1	*lockingNicknameRequest(wantedNickname)*	If wanted nickname is new in the Validator db, the nickname is locked until the validation procedure is completed	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//crypto_id/crypto_id_validation/IdValidator.java
3	*createValidChallenge()*	Create a challenge	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//crypto_id/crypto_id_validation/IdValidator.java
4	*compareChallengeAndResponse(String challenge, SessionID sessionID,String nickname, ChallengeResponse response)*	Compare on server side the challenge created for the nickname with the one sent back by the devide.	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//crypto_id/crypto_id_validation/IdValidator.java
5	*sendSMS2user(phone_number, challengeToGuess, sessionID)*	Send to the device the SMS containing the challenge	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//crypto_id/crypto_id_validation/level7/SMS_Server.java

2.3 Dispositional Trust Setup

<UniGe: Carlos, Jean-Marc>

Dispositional Trust (DT) is defined in ULOOP as the general willingness of a given user to trust others. As such, in a first implementation of DT this value will be set up manually by the owner of a ULOOP node. The DT setup is done in the boot-up phase as explained in Section 2 and it may or may not remain the same during the node's lifetime. However, as it might provide a better protection of the ULOOP owner, depending on the surrounding environment of the node and other external factors, an adaptation process may be carried out to readjust DT automatically or after asking the user in order to protect the node's integrity. For the first implementation of ULOOP, we consider a single set-up, without changes during the trust negotiation aspect. Adaptation is an aspect that we expect to address during year 3 of ULOOP.

2.3.1 Computational/Algorithmic Aspects

As for this pre-prototype version we are only considering a single set-up of the dispositional trust value, without any means of adaptation over the time, hence the computational and algorithmic aspects of dispositional trust are very simple and straightforward. Dispositional trust is an integer value in the range 0 – 1 as described in Equation 2.

$$D_{Trust} \in [0, 1]$$

Equation 2

2.3.2 Specification

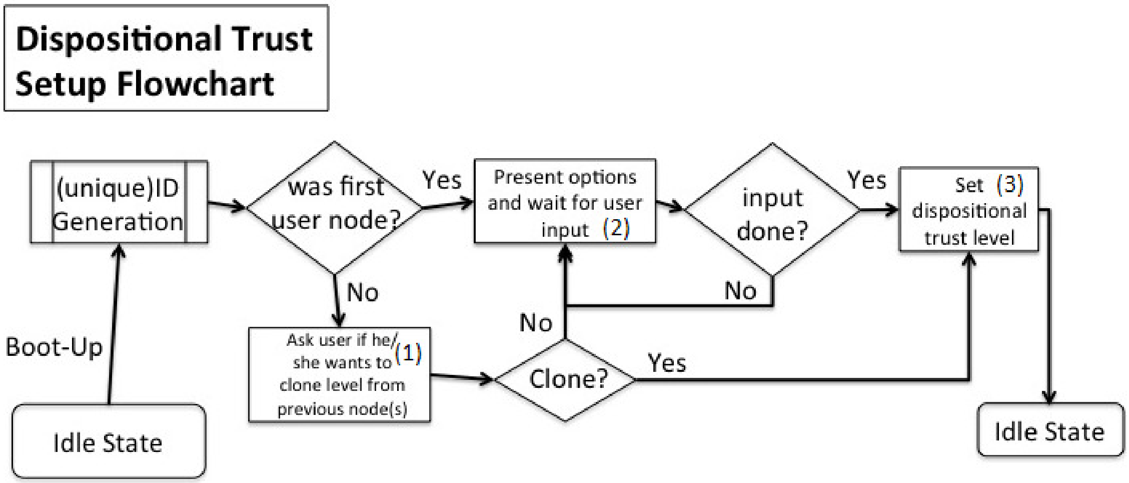

Dispositional Trust Setup Flowchart
(unique)ID Generation
was first user node?
Yes
Present options and wait for user input (2)
input done?
Yes
Set (3) dispositional trust level
No
No
Ask user if he/she wants to clone level from previous node(s) (1)
No
Clone?
Yes
Boot-Up
Idle State
Idle State

Figure 6: Dispositional Trust setup flowchart.

The dispositional trust module allows the user to configure a personal device with his/her disposition to trust other devices. If the user has multiple personal, he/she only has to set his/her dispositional trust for one device: the others will get that information from the first one when in direct contact. For the first device, the owner is prompted to set its DT, e.g. being able to select from a list of predefined values, which range from 0 to 1, being 0 “paranoid”, which means that a priori the node will not trust anyone, and being 1 “blind trust”, which means that the node will trust no matter what.

If the device is not the first one being configured, the user is presented with two options: i) to clone the dispositional trust level assigned to other devices that are already in ULOOP and that she/he owns, as described in D3 section 2.1.4.1.1 for the usage of unique crypto-IDs in different personal devices: ii) to assign a new DT level for the node being introduced, as explained in the previous paragraph. These two cases are depicted in Figure 6.

Table 3: Mapping of dispositional trust setup Flow-Chart, major operations.

Flowchart reference	**Function method/descriptor**	**Description**	**Path to Code**
1	setDispositionalTrust(configBundle.getDispositionalTrust() .getDispositionalTrustValue());	Retrieve the DT value from any other node the user owns and clone it to the new node.	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//simulation/Requester.java
2	Present options	Present the user with the adequate options in order to select which DT value to set	Done in Android, no GUI in the initial Java OModel.
3	setDispositionalTrust(askUserInitialDispositionalTrust() .getDispositionalTrustValue());	Set the DT level selected by the user or cloned from another node owned by the user.	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//simulation/Requester.java

2.4 Social Trust Computation

<ULHT: Rute Sofia >

As described previously in D2.3 and D3, the distributed trust scheme of ULOOP builds trust circles based on social trust modeling. ULOOP considers the use of computational trust management as a complementary approach to security where a level of trust in the requesting entity is automatically computed based on different types of evidence.

Nodes are associated to other nodes by means of *trust associations,* as illustrated in Figure 7. A trust association T_{ij_k} is the k-th directed association between nodes i and j, and is related to the respective owner's interests and social networking perspective. A trust association holds a cost which we name as *trust level.* The trust level provides a measure of previous trust behavior, of *Quality of Experience (QoE)* of nodes, etc. Hence, two nodes may in fact hold more than one trust association among them, as represented in Figure 1, where nodes A and B hold three different trust associations: A has two trust association to node B, T_{AB_1}, which relates to the exchange of data owned by A (where A is the originator), and T_{AB_2}, which relates to the exchange of information which A is relaying (A is not the source). B has one trust association to A, T_{BA_1} with a cost of zero which e.g. could mean that B still does not trust A to relay his data. As shown in Figure 7 the different trust associations have a specific cost, and the computation of such cost is based upon the nodes expectations and beliefs.

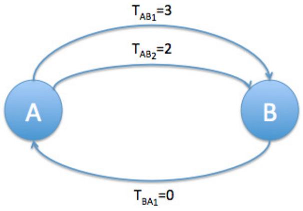


Figure 7: Social trust association examples.

The weight of a specific trust association considers *local* and *external* influences. Examples of local influences are the degree of connectivity and reputation level of node B. External influences are influences that do not relate to the nature of each node but to external networking conditions (e.g. too much overhearing probability around node B).

For example, if Alice has the choice to connect to two nearby ULOOP gateways, Charles' gateway and Bob's gateway but she has never interacted with Bob's gateway before. Bob's ULOOP gateway is the gateway that would give her the quality of service she requires. Fortunately, she has already interacted with Charles' gateway. As she has no direct observation of Bob's gateway, therefore, she asks Charles for a recommendation. Charles has already used Bob's gateway and sends his recommendation to Alice. A third type of evidence used in ULOOP concerns reputation, which is the aggregation of different recommendations from different recommenders that are not exactly known. That reputation value may come from the aggregation of evidence external to ULOOP, for example, from the mining of existing online social networks. In ULOOP, social trust computation relies on the following properties:

1. **Trust is based on a social behavior** of the owners of nodes and is therefore environment-dependent as well as disposition-based.
2. **Trust is not transitive**. A node A may trust a node B with a level of x ($T_{AB} = x$), and this one may trust a node C with a level of $y \geq x$, but it is up to A to compute its trust level towards C.
3. **Trust is asymmetric.** The trust level between a node A and B is not necessarily the same between B and A.
4. **Trust computation** is based upon current and past experience of a node.
5. **Trust is dynamic**. Trust associations are bound to frequent changes depending on the nodes own perception of trust within communities.

2.4.1 Computational/Algorithmic Aspects

The purpose of our scheme is to allow the dynamic propagation of a circle of trust in a way that is robust to malicious usage and attacks. Hence a first step is to provide each node with an initial level $D_i \in [0,1]$ of *dispositional trust*. Dispositional trust corresponds to a node's own disposition to trust others initially, where 0 corresponds to the minimum possible level for no trust (does not trust anybody) to blind trust (trusts any stranger). We expect D_i to evolve throughout a node's lifetime. Since some nodes are carried by Internet end-users, their networking composition, surrounding environment and organization can rapidly change. As such, the dispositional trust level on a given node might not be appropriated in all circumstances and should be able to be adapted and changed over time, in order to protect the node's integrity.

To explain our function we consider three nodes: node i, the node that is about to compute a trust level towards a node z, and node j representing a node in the same community as node i. Node i has a dispositional trust level $D_i \in [0,1]$. Figure 8 provides an example for the different types of recommendations that are the basis to compute the cost of a trust association between two nodes i and a, where the arrows represent an already established trust association.

When i decides to compute its own perspective of a trust association to node a it triggers requests to obtain recommendations about a. For this specific example, this means that its direct nodes j and kl provide their own perspective (*recommendation*) about node a. Such recommendation may be *direct* as occurs for the case of node j who has a direct trust association to node a, t_{ja}, or indirect, as occurs with node l i.e., node l has an indirect trust association to node a through node k, being the indirect recommendation written down as t'_{la} . Direct trust associations are more relevant than indirect recommendations as this is based on direct experience.

Hence, a *direct recommendation* received by node i represents an answer from a node j in the community, and contains the computed cost of one or several trust associations between j and the target node.

An *indirect recommendation* received by node i represents an answer from a node j in the community which contains the computed cost of one or several trust associations between j and the target node, but j is not yet in the trust table of i. For the specific case illustrated in Figure 8,

$$t'_{la} = t_{lk} * t_{ka}\,.$$

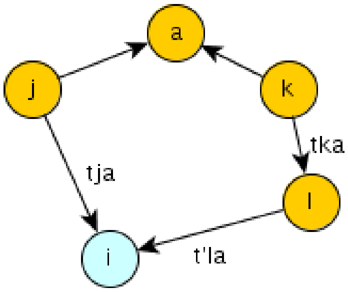


Figure 8: representation of indirect and direct recommendations.

The proposed trust computation function is provided in Equation 3. It considers both direct and indirect recommendation values, as well as the owner's own beliefs - dispositional trust. Moreover, the more stable acquaintances are, the more trusted their recommendations become.

$$t_{iz} = D_i * \left[\alpha * \frac{\sum_{j=0}^{k} t_{jz}}{k} + (\alpha - 1) * \frac{\sum_{j=0}^{p} t'_{jz}}{p} \right]$$

Equation 3

, where

k :number of direct recommendations.

p :number of indirect recommendations.

j : node providing trust recommendation, $j \le n, j \in N \wedge j \ne i \wedge j \ne z$.

z :target node

i : node requesting recommendations

n :total of nodes in the community.

As provided by Equation 3, the value of the dispositional trust parameter is crucial to compute a trust weight towards another node. Assuming such value is zero, then recommendations provided are of no use to the established trust level. We highlight, however, that the dispositional trust parameter is an element that may change with time, even though such adaptation is not part of this specific work. On the other hand, if the dispositional trust value is high (e.g. one), then the trust level becomes dependent on the recommendations provided. Alfa is a parameter that allows us to give more weight to the direct recommendations.

Assuming 2 nodes i and j at the range of each other but without any established trust association, then trust propagation can be based on the multiplication of the trust level perspective of each of nodes i and j, in regards to the path between them, as well as in combination to their own dispositional trust, as shown in Equation 4:

$$t_{ik} = D_i * t_{ij} * D_j * t_{jk} \; t_{ik} \in [0,1]$$

Equation 4

2.4.2 Specification

We provide in Figure 9 the flow chart for trust computation, which is a process that runs periodically since the moment ULOOP nodes boot. Therefore, after boot up (1) the nodes check for their dispositional trust D (2) and activate a trust table (3). The trust table is a structure where each row is a tuple with the following structure: <Node Id, trust level, ageing>. When activated, the node provides each of its neighbors with an equal trust level of D. In other words, in environments where relations were not yet established, ULOOP nodes trust equally all nodes around. Then, periodically, the node emits and hears recommendations (5) – in Figure 10 we provide in an example for the way communication is processed to get/emit such recommendations.

Each time recommendations are obtained the respective entry in the trust table is updated. Hence, the table is kept up to date.

Requests for social trust computation come from the trust manager, cooperation manager and are provided via a look up to the trust table.

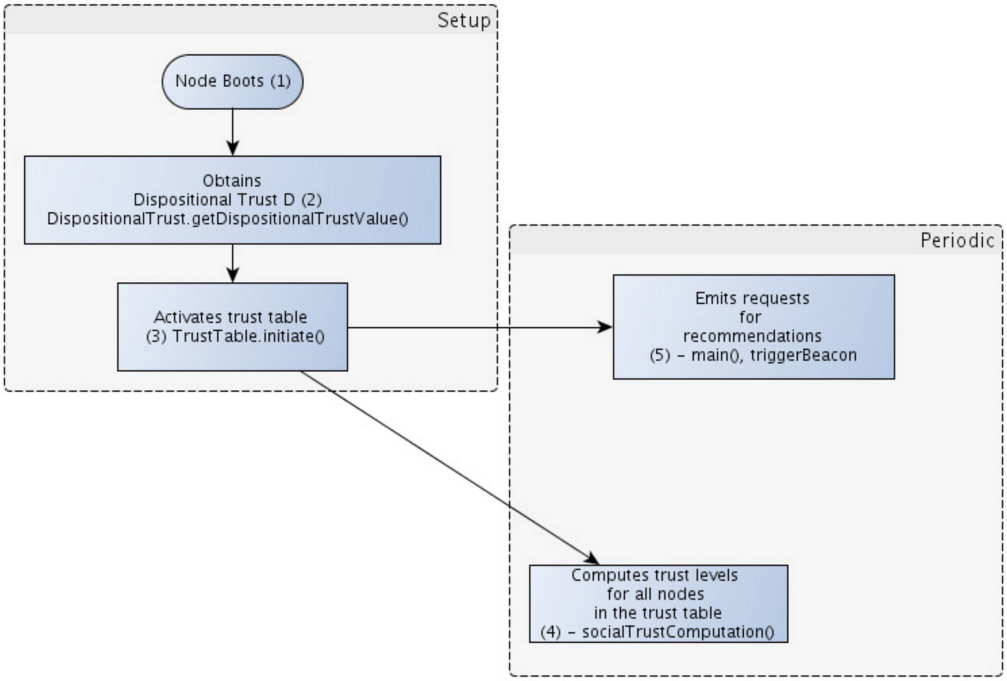

Setup
Node Boots (1)
Obtains
Dispositional Trust D (2)
DispositionalTrust.getDispositionalTrustValue()
Activates trust table
(3) TrustTable.initiate()
Periodic
Emits requests
for
recommendations
(5) – main(), triggerBeacon
Computes trust levels
for all nodes
in the trust table
(4) – socialTrustComputation()

Figure 9: Social Trust Computation flowchart.

Table 4: Mapping of social trust computation Flow-Chart, major operations

Flow-chart reference	**Function/Method Descriptor**	**Description**	**Path**
(1)	public void runUloopClientOn (Requester uloopNode)	Initiates a ULOOP node	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//simulation/User.java
(2)	Public int getDispositionalTrust()	Retrieves the value of the Dispositional Trust of a node. This method relates to class DispositionalTrust,	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//dispositional_trust/DispositionalTrust.java
(3)	Public void Trusttable()	Initiates a new trusttable. This is the constructor of class TrustTable, called from main()	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//social_trust_computation/TrustTable.java
(4)	public float computeSocialTru	Computes trust levels based on input sent by neighboring	TrustManagementAndCooperationIncentives/Common/JavaOOMo

	st(cryptoId NodeID, int tl)	nodes	del/trunk/src//social_trust_computation/SocialTrustComputation.java
(5)	public void sendBeacon(Message message)	Global function to ULOOP which triggers the need to send a beacon, by passing a specific set of data to hostapd	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src//simulation/Node.java

Concerning requests and recommendations for social trust computation, we provide an example of a potential sequence chart in Figure 10, where 2 ULOOP nodes (N1, N2) are in the vicinities of two ULOOP gateways G1 and G2. Node N1 is a new node (no trust relationship established yet), and node N2 has a trust association to G1. Moreover, G1 and G2 are known and trusted by each other.

Based on the ULOOP frame format (a regular MAC where the payload carries additional information) N1 broadcasts a frame requesting recommendations for G1, G2, and N2. G2 replies with its recommendation about G1 (trust level of G2 towards G1). G1 replies with a recommendation for both N2 and G2. While N2 replies with its trust level towards G1.

Each time N1 gets this recommendation, it computes the respective trust level based on the formula provided in Equation 3. The trust table is a structure global to the trust management process.

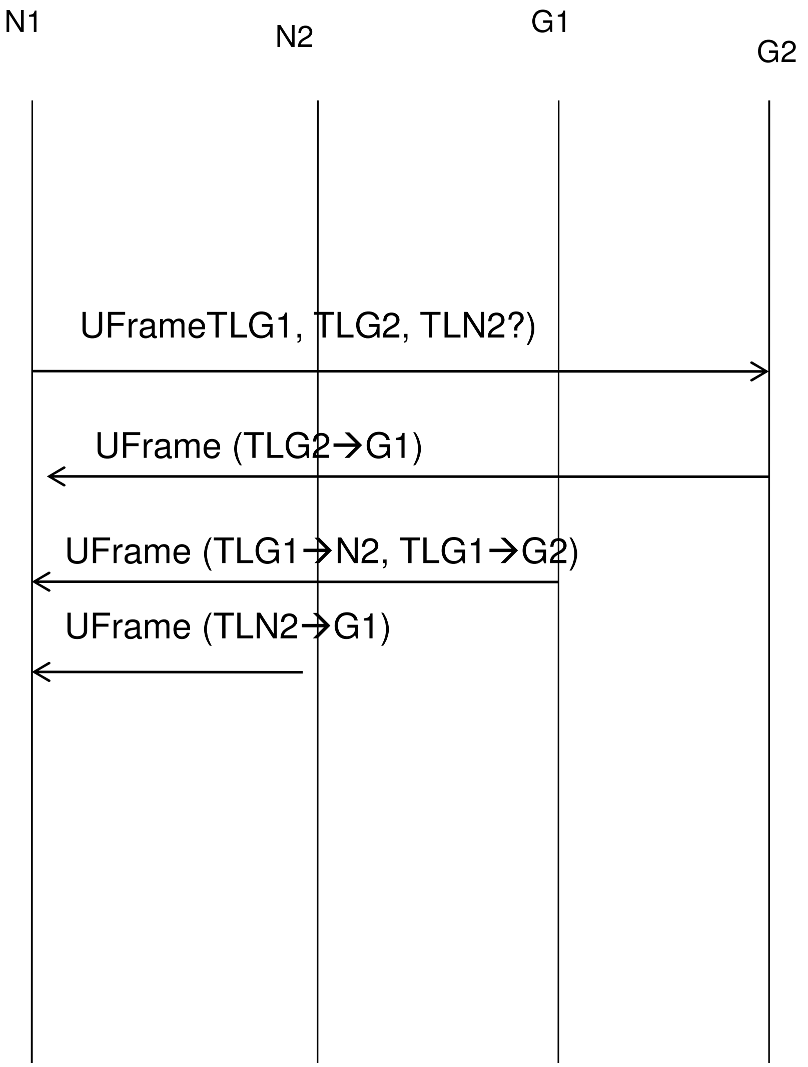


Figure 10: Example of communication for requests/recommendations on trust.

2.5 Cooperation Manager

<ULHT: Paulo, Rute, Waldir, Christian>

This section provides a detailed description of the operation of the Cooperation Manager, which has three components for credit assignment, credit computation, and cooperation evaluation. The cooperation manager aims to control the setup of a cooperation session between a requester and a requestee based on their trust level and a set of credits used by the requester to provide the requestee extra incentives for cooperation. The requestee operates in a volunteer or retailer mode (the device can switch between operating modes at any time). In the latter state, the cooperation is further controlled by the Reward Manager, which will allow the requestee to accept or refuse the cooperation based on the amount of credits offered by the requester.

To manage each cooperation section, the Cooperation Manager controls a cooperation credit set, which will work in coordination with the wallet controlled by the Reward manager (c.f. section 2.6), as follows:

- An ULOOP device has a cooperation credit set and a reward wallet. The former is used to setup cooperation sessions. The latter can be used in a monetization process. The amount of credits in a reward wallet is always lower or equal to the ones in the cooperation credit set (they will be the same is the device is always operating as a retailer).
- When a device boots-up, credits are assigned to the cooperation credit set by the cooperation manager (the reward wallet is still empty)
- When a gateway is operating as a requestee and receives some credits from a requester, the Cooperation Manager places the credits in its cooperation credit set. If the requestee is operating in a retail mode, credits are also placed in the reward wallet by the Reward Manager.

2.5.1 Computational/Algorithmic Aspects

The next two sub-sections provide a description of the components for credit assignment and credit computation, which will be included in the first ULOOP prototype. The component that will execute the cooperation evaluation is not described in this document, since its development is schedule for the third year of the ULOOP project.

2.5.1.1 Credit Assignment

The initial number of credits that will be assigned at the beginning of the cooperation process done by Equation 5 will depend of the node trust level (which most likely will be equal to the dispositional trust).

$$C_{init} = \begin{cases} C_{min} & t_L = 0 \\ (C_{max} - C_{min}) * e^{-\frac{(t_L - \mu)^2}{\sigma^2}} + C_{min} & 0 < t_L < 1 \quad \mathit{where} \quad \sigma^2 \approx 0,03 \\ C_{min} & t_L = 1 \end{cases}$$

Equation 5

Where:

- Cmin and Cmax are the threshold established for the amount of credits an ULOOP device can have
- tL is the trust level Requester-Requestee

- μ is the location of the center of the Gaussian distribution’s peak
- σ is the deviation standard. This variable controls the width of the “bell”

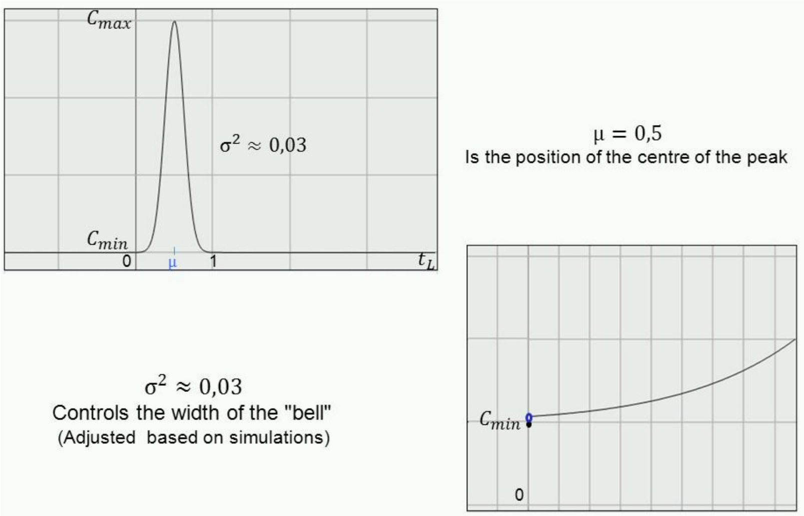


Figure 11: Distribution function for credits assignment

The assignment of an initial set of credits follows a Gaussian distribution behavior with mean equal to the maximum number of credits allowed by the system, as shown in Figure 11. This proposal is based on the following ideas:

- When a node has a high trust level (probably because the output of most of its past interactions has been positive), the impact of having credits will be low, since the node will communicate solely based on its trust level. Hence, in this case the node does not become greedy and will be assigned a minimum number of credits.
- When a node has a low trust level (probably because the output of most of its interactions has been negative or has not had enough interactions yet), the impact of having credits will be low, since the node does not have enough trust to initiate a communication, independently of the number of credits that it has. Moreover, it is appropriate to assign fewer credits because the node behavior “is suspicious”. This is a way to prevent possible attacks.
- When a node has an average trust level, it is a good idea to assign more credits encouraging the node to participate in the community.

When a node has just joined a ULOOP community, it will not have a trust level readily available. Thus, as mentioned before, its trust on other nodes (i.e., dispositional trust) is considered to determine the amount of credits the node will be getting.

2.5.1.2 Credit Computation

The cooperation process refers to encouraging nodes in engaging in this process. In this sense, credits are the most appropriate resource to stimulate the participation and interaction between nodes with an average trust level.

This function will compute the amount of credits that a requester is willing to provide to a requestee as a cooperation incentive. If the provided credit amount is enough to convince the requestee in engaging in the cooperation, the process continues and the association between requester-requestee can be possible.

The amount of cooperation credits sent to the requestee by the requester will be calculated as a function of the number of credits that the requester owns at that specific moment. When the requester is using cooperation credits for the first time, it will start by sending ¼ of the initial credits assigned, hoping that this amount is enough to encourage the requestee in engaging in cooperation.

If the cooperation is accepted, the requester will keep track of the amount sent to the requestee as reference for future cooperations (with the same requestee or even others). This amount of sent cooperation credits refers to a specific service (with same service type and same service level) required by the requester. Thus, the requester computes an average considering the current amount of sent cooperation credits and past ones (i.e., based on cumulative moving average) that it has given for the same service in previous interactions. And this average will serve as reference for sending credits in future cooperation opportunities.

If the cooperation is not accepted, the requester will increase the amount of cooperation credits by 1/8 in an attempt to re-negotiate the cooperation process the same requestee or a newly found one.

2.5.2 Specification

As shown in Figure 12 the operation of the cooperation manager is divided into three phase: an initial assignment of credits when the node enters a ULOOP community; the initiation of a periodic activity for the evaluation of all cooperation sessions; the control of a cooperation process, including the computation of the amount of credits to assign to a new cooperation.

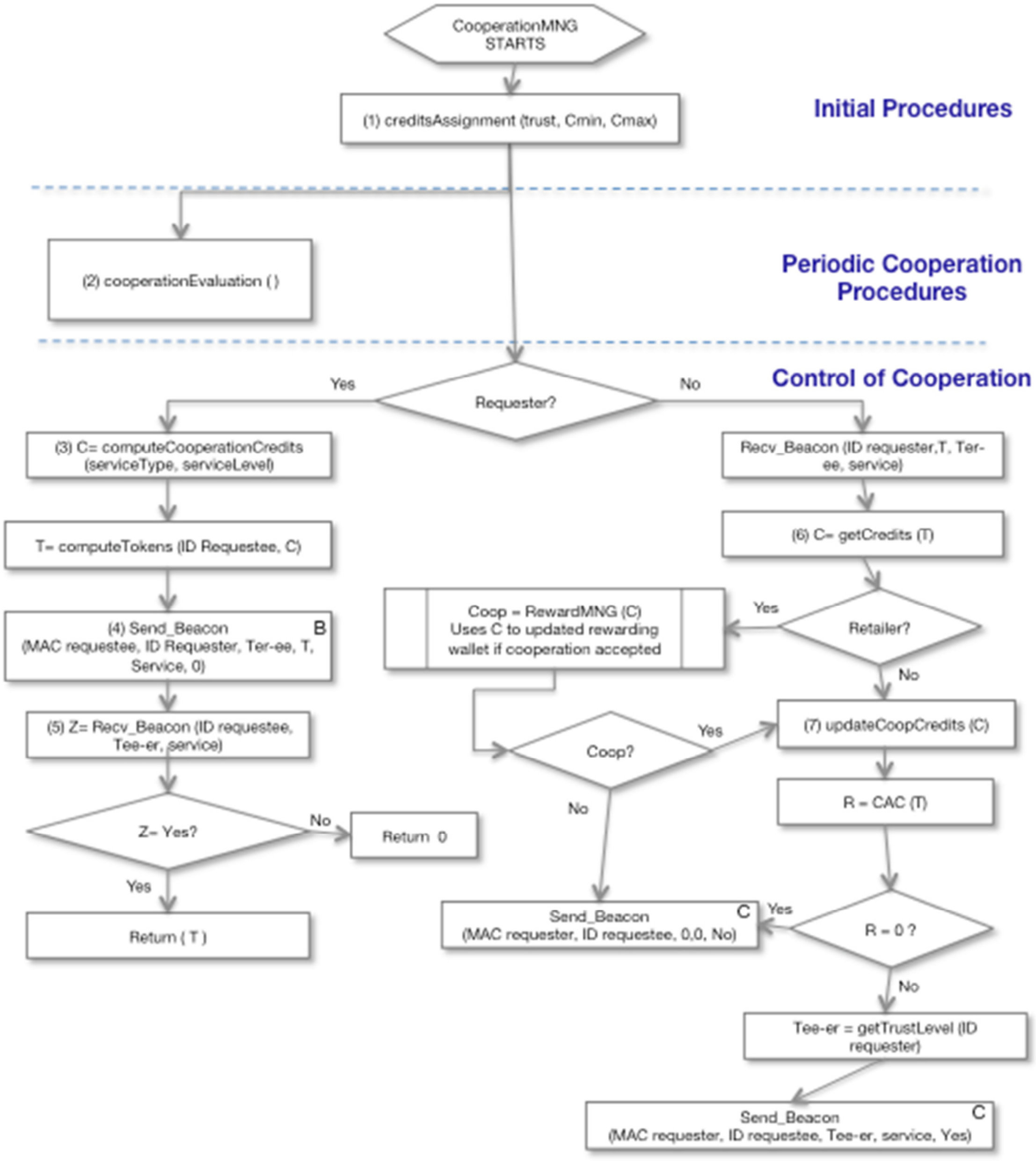
As seen in Figure 12 the control of a cooperation process depend on the role of the ULOOP device as a requester or a requestee.

If the ULOOP device is a requester it will first compute the amount of credits that it wants to assign to the requestee in order to set up a cooperation session. The credits are sent to the requestee in the form of a Token computed by a generic function Comp-Token (c.f section 2.6). The requester sends the token (together with the trust level that it has on the requestee) to the requestee in an ULOOP Beacon (UBeacon). Another UBeacon is used to get from the requestee the result of the cooperation request, which includes information about the trust level that the requestee has on the requester. If the requestee agreed on the cooperation (amount of credits) the Cooperation Manager at the requester will return the value of the cooperation Token. Otherwise, the cooperation manager returns zero.

If the ULOOP device is a requestee it will first extract the information about the credits from the Token, which it got from the requester via an UBeacon. The first condition to be checked is the viability of the cooperation in terms of credits. If the requestee is operating in a retailer mode, the Reward Manager is called to decide about the validity of the requested cooperation. The validation of the cooperation in terms of credits ends up with the requestee adding the received credits to its set of cooperation credits: this operation is always done when the requestee is operating in a volunteer mode, and in a retailer mode, after a positive reply from the reward manager.

If the cooperation is validated in terms of its credits the next validation is in terms of the resources needed to execute the cooperation.

The validation of the cooperation in terms of resources is done by the CAC functionality, which will use the cooperation Token to decide about the amount of resources that should be allocated to this cooperation. If resources are made available, the requestee will compute its trust level towards the requester, and will send that information to the requester via an UBeacon. Otherwise the requester will get a null value, corresponding to a rejected cooperation.


CooperationMNG STARTS
(1) creditsAssignment (trust, Cmin, Cmax)
Initial Procedures
(2) cooperationEvaluation ( )
Periodic Cooperation Procedures
Control of Cooperation
Yes
Requester?
No
(3) C= computeCooperationCredits (serviceType, serviceLevel)
Recv_Beacon (ID requester,T, Ter-ee, service)
T= computeTokens (ID Requestee, C)
(6) C= getCredits (T)
Coop = RewardMNG (C) Uses C to updated rewarding wallet if cooperation accepted
Yes
Retailer?
(4) Send_Beacon (MAC requestee, ID Requester, Ter-ee, T, Service, 0)
B
No
(5) Z= Recv_Beacon (ID requestee, Tee-er, service)
Coop?
Yes
(7) updateCoopCredits (C)
Z= Yes?
No
Return 0
No
R = CAC (T)
Yes
Return (T)
Send_Beacon (MAC requester, ID requestee, 0,0, No)
C
Yes
R = 0 ?
No
Tee-er = getTrustLevel (ID requester)
Send_Beacon (MAC requester, ID requestee, Tee-er, service, Yes)
C

B and C – Refer to UBeacon frames as shown in the sequence chart

Figure 12: Cooperation Manager flowchart.

Table 5 provides a mapping of the flowchart to the major operations executed by the Cooperation Manager.

Table 5: Mapping of cooperation manager Flow-Chart, major operations.

Flowchart reference	Function method/descriptor	Description	Path to Code
1	Double creditsAssignment (TrustTable trusttable, CryptoID NodeID, double Cmin, double Cmax)	This function calculates the initial number of cooperation credits that will be assigned at the boot-up process.	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src/cooperation_manager/CreditsAssignment.java
2	Void cooperationEvaluation ()	This function evaluates the cooperation process that has happened between ULOOP nodes. It results in the increase/decrease of credit value.	TBD
3	Double computeCooperationCredits(String serviceType, int serviceLevel)	This function calculates the amount of cooperation credits that will be sent to the requestee. These credits will be used by the Requester to encourage the Requestee in engaging in cooperation.	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src/cooperation_manager/CooperationManager.java
6	Double getCredits(double token)	This function will be used by the Requestee to obtain the number of credits (from the received token) involved in the cooperation process and to decide if this amount is enough to cooperate.	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src/cooperation_manager/CooperationManager.java
7	updateCoopCredits(Double cooperationCredits)	This function will be called when it is necessary to update the cooperation credits (requestee side) by adding the credits that were received and accepted in the cooperation negotiation. Through this function we can obtain the amount of credits that has been earned by the requestee with a given cooperation.	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src/cooperation_manager/CooperationManager.java

2.6 Virtual Currency and Reward Manager

<UniUrb: Alessandro, >

The virtual currency system provides the necessary primitives needed in order to send monetary cooperation incentives to ULOOP gateways operating in retailer mode and to gather currency needed for untrusted nodes to ensure cooperation. Credits can be converted to fiat money (monetized).

The Reward manager component is responsible for the control of a rewarding system when the requestee operates in retailer mode. The Reward Manager allows the requestee to accept or refuse the cooperation by comparing the amount of credits offered by the requester, with the amount of credit that the requestee assigns as a cost to the requested cooperation.

The control of credits is done based on a reward wallet, which has always an amount of credits equal or lower than the cooperation credit set controlled by the Cooperation Manager (they have the same amount of credits if the ULOOP device is always operating in a retailer mode). The credits in the reward wallet may be used as a virtual currency.

The Reward Manager is also responsible for the transfer (payment) of credits between requester and requestee related to the rewarding process, including ways to monetize the credits used as virtual currency.

2.6.1 Specification

Currently, as of 2.5, the Cooperation Manager receives a cooperation request inside a ULOOP Beacon. The initial message includes the service request, the trust level of the requestee (from the requester's point of view) and an amount of tokens which encode the amount of credits the requester is willing to pay in order to compensate the cooperation effort of the requestee.

Credit transfers are assumed to be immediate, if the cooperation request is accepted. If the request is refused, the credits are assumed not to be transferred. On acceptance, the credits are transferred to the Cooperation Manager (and kept inside the Cooperation Credit Set) and optionally also transferred to the Wallet (if the node is operating in retailer mode).

Those operations are atomic and synchronous.

In order to guarantee the legitimate nature of the credit transfer, a central authority (Bank) can be made available. The bank keeps track of the status of all wallets of registered ULOOP nodes. Credit transfers between nodes can be then confirmed by the bank (this process is needed to ensure there is no double spending of credits and that the requester's wallet contains the transferred amount of credits). Credit transfers are allowed also without an authority, but they cannot be confirmed reliably.

Figure 13 illustrates the full credit transfer system between a requester and a requestee, with optional synchronization via central authority (Bank), if Internet connectivity is available.

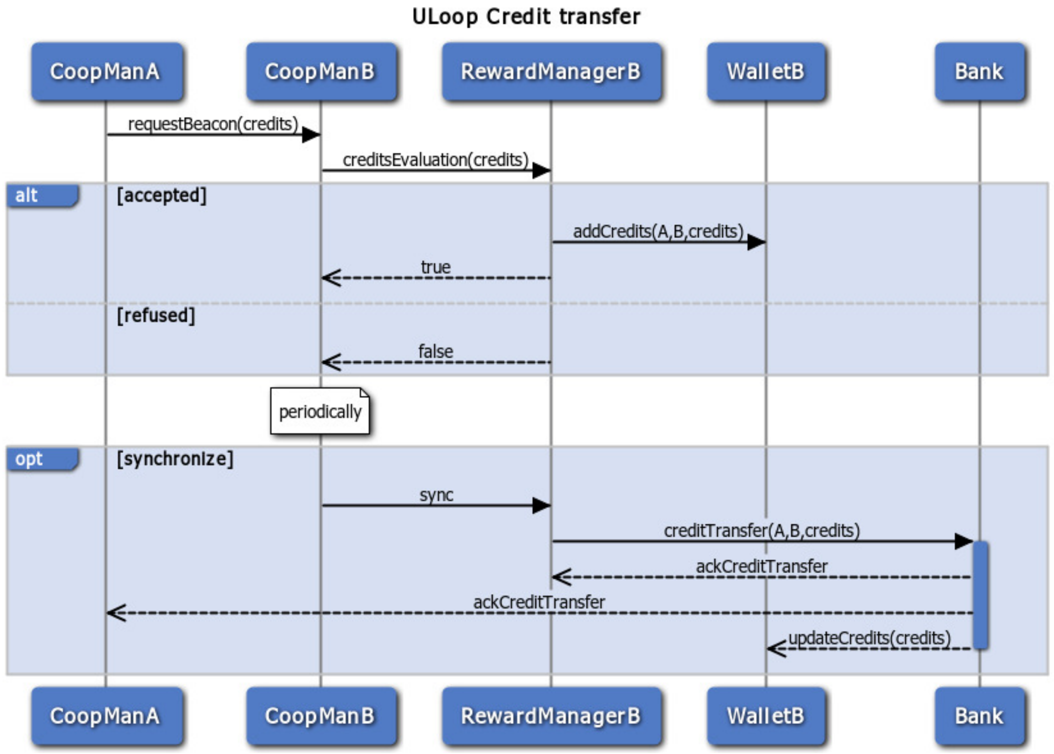


Figure 13: Credit transfer sequence diagram.

The payer assumes that payment succeeds by default, while the payee might wish to confirm payment via authority. Unconfirmed payments, by default, have immediate effect. The Credit Transfer system keeps track of incoming and outgoing credit transfers and waits for acknowledgement by the Bank. If the credit transfer fails while asking for confirmation, the requestee assumes the credits to be lost and appropriate action is taken to notify the illegal behaviour of the requester (influencing its trust on other nodes).

Table 6: Mapping of reward manager Flow-Chart, major operations.

Flowchart reference	Function method/descriptor	Description	Path to Code
NA	RewardManager.creditsEvaluation(credits)	This function evaluates whether to accept a payment or to refuse it (as incentive to a cooperation request). Acceptance is based on trust and transferred credits by the requester.	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src/virtual_currency_and_reward/RewardManager.java
NA	RewardManager.sync()	This functions performs periodic synchronization with the Bank (if available) to confirm pending credit	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src/virtual

		transfers.	_currency_and_reward/RewardManager.java

2.7 Token Computation

<ULHT: Rute Sofia>

In ULOOP, resource assignment and selection of gateways is performed based on two main parameters: trust level (computed by the class SocialTrustComputation) and credits (managed by the class CooperationManager). In order to assist the interfacing towards other classes, ULOOP therefore considers a unique and virtual currency in the form of a *token*: a token is a unit of resources.

For instance, assuming a user at a specific instant in time can benefit of n tokens, then it can be assumed that there is a specific correspondence to bandwidth, or to connectivity time, or to any other form of networking or service resource.

The Trust management and Cooperation Incentives block is responsible for generating tokens. **Then, the responsibility of adequately mapping tokens to resources is delegated to each of the other ULOOP technical blocks**.

2.7.1 Computational Aspects

To generate tokens, ULOOP relies on a utility function that has as input both a trust level and a set of credits that the node is willing to spend to get a specific service. This function must increase with an increase in the trust level and also with an increase in credits. However, we expect it also to vary slowly. Moreover, in ULOOP tokens are dependent also on the trust level. For instance, if a node has a low trust level from the perspective of a source node (requestee), then, even if this node has a high credit level, the resulting token value should progress slowly. When a node has a good trust level, then if it uses a high level of credits, the resulting tokens should also not increase linearly, as this would make the node greedy.

So the basic line of thought for the token function provided in Equation 6 is that when a specific trust level is “low” (below some threshold which ULOOP adjusts), credits are more relevant to generate an adequate number of tokens, than when the trust level is higher than the threshold specified.

Moreover, the trust level varies between [0,1], while credits vary between 0 and infinity. ULOOP is currently assessing (via simulations) different variations of the function proposed as there is not yet consensus from the consortium concerning the function to rely upon. However, for the sake of implementation, we consider in the code the function provided in Equation 6[1].

$$\boldsymbol{tk(i,j) = tl(i,j) * \sqrt{c}}$$

Equation 6

Where:

$$tl(i,j) \in [0,1]: trust\ level\ of\ node\ i\ towards\ j[0,1]; c\ \in [0,\infty]$$

For the sake of representation of the function behavior, we provide in Figure 14 a graph that shows how tokens vary with the trust level and with credits, based on the proposed function.

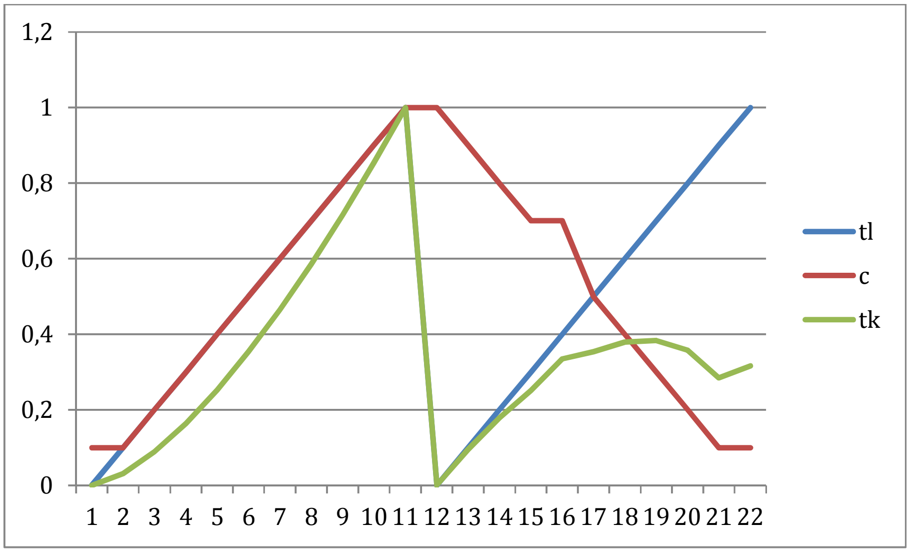


Figure 14: representation of equation 3 when credits and trust level vary.

2.7.2 Specification

The computation of tokens is performed via the java class TokenComputation. The token computation is performed upon request via the TrustManager entity. On the context of the trust management, token computation is performed once the node (Requester) is accepted by a gateway (Requestee), i.e., once the MAC authentication process ends, and the resource management process starts.

[1] Equation 6 is one of the potential embodiments for the token utility function. The consortium is currently evaluating several possibilities. The equation provided is the one that has been implemented in the prototype released. It will be replaced by the function selected by the consortium.

Table 7: Methods of TokenComputation.

Function/Method Descriptor	Description	Path
public float computeTokens(CryptoId requestee, Double credits)	Computes tokens to be provided for a service exchange, based upon the trust level of the node towards the requestee and also based upon the number of credits the requestee is willing to spend on a specific service	TrustManagementAndCooperationIncentives/Common/JavaOOModel/trunk/src/token_computation/TokenComputation.java

3. Software Implementation Aspects

<UniGe: Carlos, Jean-Marc>

This section provides input concerning software design, choices that we have considered for the current release and aspects to be addressed for the next release.

Currently, the pre-prototype implementation in Task 3.1 has been carried out fully in Java, following an object oriented modelling approach. Each of the sub-blocks previously described in Section 2 has been represented by an independent package in the Java project in Eclipse, each of them containing one or more Java classes, in order to implement the main objects and methods needed for ULOOP to work.

Figure 15 shows the main class diagram of ULOOP's architecture implementation, containing the main relevant classes and methods to ULOOP. Some classes have been minimized and their methods omitted for the sake of the figure's clarity.

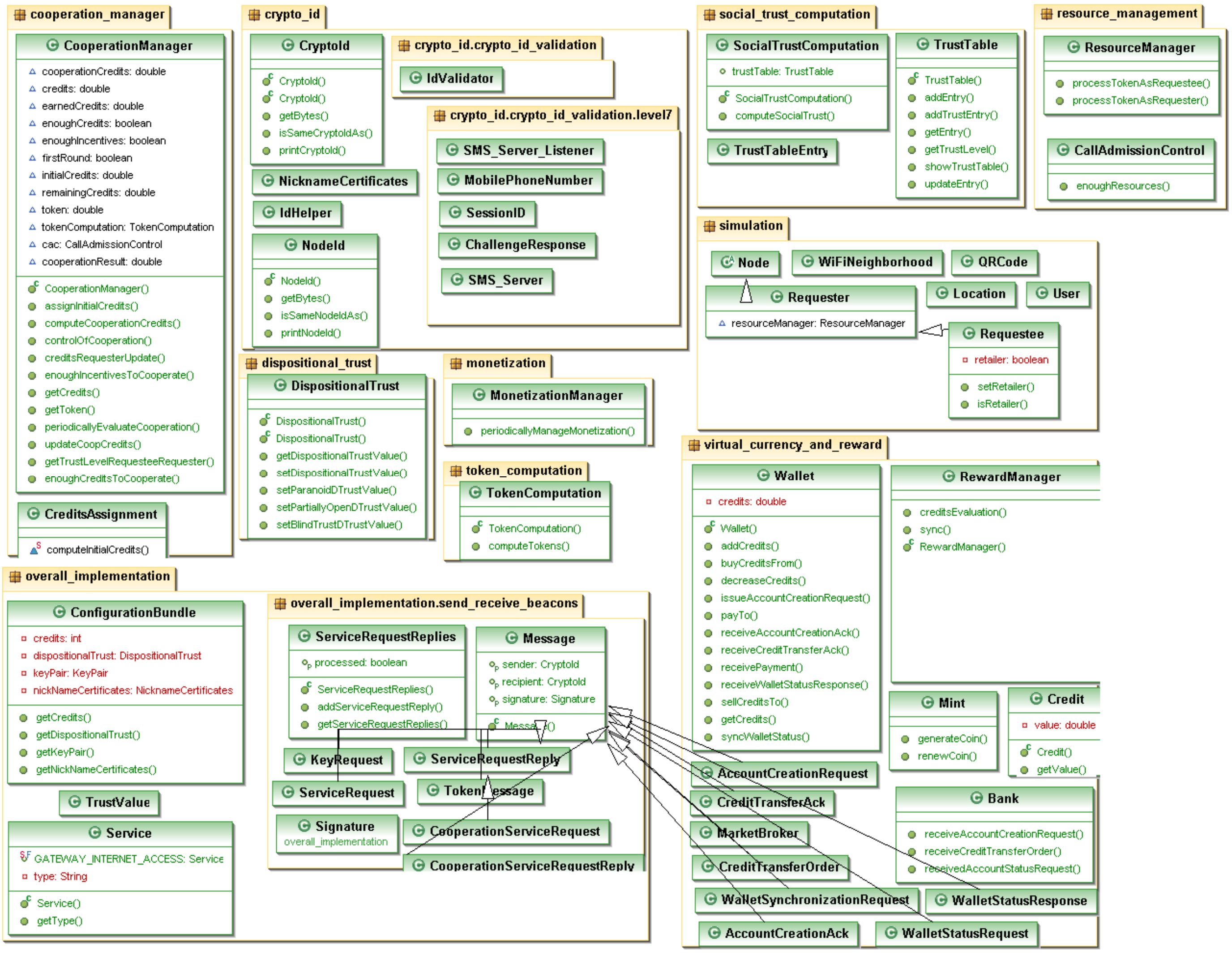


Figure 15: Main ULOOP class diagram

The full description of the classes and methods within the classes shown in Figure 15 can be found in Annex A – Packages, Classes and Methods Description.

4. Guidelines/Next Steps

The main guidelines or next steps to be taken until the release of D3.4 (M24) are:

- To complete the applications in an Android platform, accordingly to the prototype plan of Task 3.4.
- Choosing/adopting some new solutions for the current implementation (i.e. how to validate crypto-ids in a node without cellular interface, etc…).

- Refining the current software in order to ensure a better functioning and integration between sub-blocks.

Also, all the ***TODOs*** present in the java code are presented in Table 8.

Table 8: List of TODOs as listed in the java code in the SVN

Description	Resource	Path	Location
TODO a timeout may occur if no device with this cryptoid is on or other reasons	Requester.java	/uloopjavaoomodelv2/src/simulation	line 415
TODO add real keystore to IdValidator	IdValidator.java	/uloopjavaoomodelv2/src/crypto_id/crypto_id_validation	line 112
TODO add strong secure signature	Requestee.java	/uloopjavaoomodelv2/src/simulation	line 115
TODO add strong secure signature	Requestee.java	/uloopjavaoomodelv2/src/simulation	line 155
TODO add strong secure signature	Requestee.java	/uloopjavaoomodelv2/src/simulation	line 170
TODO add the functions to flash the QRCode	Requester.java	/uloopjavaoomodelv2/src/simulation	line 496
TODO At the moment a requestee replies to any service request whose service is present in the list of services but it should be tailored to the services that it can do and based on a number of other conditions (requester enough trusted, enough resources...)	Requestee.java	/uloopjavaoomodelv2/src/simulation	line 274
TODO At the moment	Requester.java	/uloopjavaoomodelv2/s	line 848

we assuming that all id validators are equal, trust each other and only one id validation is sufficient		rc/simulation	
TODO change back to real implementation from CMS to compute the concatenation of hashes	IdHelper.java	/uloopjavaoomodelv2/src/crypto_id	line 51
TODO create form to ask the user pass nonce to retrieve the secrets	Requester.java	/uloopjavaoomodelv2/src/simulation	line 399
TODO credits should be extracted from the wallet? (This is not clear.)	CooperationManager.java	/uloopjavaoomodelv2/src/cooperation_manager	line 86
TODO currently we can envision tokencomputation to occur during: cooperation negotiation and later, during resource negotiation. right now, we are assuming that after the negotiation, there is a global variable credits, which the node keeps and therefore, we pass that value if that does not occur, then we may need a specific method to get	TokenComputation.java	/uloopjavaoomodelv2/src/token_computation	line 34

creditsc=cooperationcredits.getCredits(); c=computeCooperationCredits(0, 0);			
TODO decreasing the wallet credits or cooperation credits of the requestee in case of no cooperation does not seem on the diagrams	CooperationManager.java	/uloopjavaoomodelv2/src/cooperation_manager	line 186
TODO decreasing the wallet credits or cooperation credits of the requester does not seem on the diagrams	CooperationManager.java	/uloopjavaoomodelv2/src/cooperation_manager	line 199
TODO display form to ask for a validated nickname	Requester.java	/uloopjavaoomodelv2/src/simulation	line 445
TODO display form to ask for a validated nickname	Requester.java	/uloopjavaoomodelv2/src/simulation	line 545
TODO display form to ask for a validated nickname	Requester.java	/uloopjavaoomodelv2/src/simulation	line 556
TODO display form to ask for the CryptoId of another node either typed by hand or flashed or NFCed	Requester.java	/uloopjavaoomodelv2/src/simulation	line 467
TODO display form to ask the user to type the crypto-id by hand	Requester.java	/uloopjavaoomodelv2/src/simulation	line 475
TODO display form to ask user if a QRCdoe is provided	Requester.java	/uloopjavaoomodelv2/src/simulation	line 534

TODO display form to ask user if has already another node	Requester.java	/uloopjavaoomodelv2/src/simulation	line 456
TODO display form to flash the QRCode	Requester.java	/uloopjavaoomodelv2/src/simulation	line 469
TODO display forms to the user to know if she has already a	Requester.java	/uloopjavaoomodelv2/src/simulation	line 817
TODO EMA parameter, we need to tune it better	SocialTrustComputation.java	/uloopjavaoomodelv2/src/social_trust_computation	line 43
TODO for June 30th, this is a simpler version of the function we are not accounting for direct/indirect recommendations we shall do that by keeping state on the truth table about weighted average of both the trustlevel due to direct and to indirect recommendations	SocialTrustComputation.java	/uloopjavaoomodelv2/src/social_trust_computation	line 54
TODO form to ask the user her initial dispositional trust value	Requester.java	/uloopjavaoomodelv2/src/simulation	line 589
TODO form to ask the user her wanted not validated nickname	Requester.java	/uloopjavaoomodelv2/src/simulation	line 578
TODO get real MAC addresses	Node.java	/uloopjavaoomodelv2/src/simulation	line 34
TODO implement credit transfer order	Wallet.java	/uloopjavaoomodelv2/src/virtual_currency_and_reward	line 49
TODO implement	Requester.java	/uloopjavaoomodelv2/s	line 912

isWithinRadius		rc/simulation	
TODO implement more secure signature generation	Requester.java	/uloopjavaoomodelv2/src/simulation	line 730
TODO implement receive account creation ack	Wallet.java	/uloopjavaoomodelv2/src/virtual_currency_and_reward	line 104
TODO implement receive credit transfer ack crypto	Wallet.java	/uloopjavaoomodelv2/src/virtual_currency_and_reward	line 75
TODO implement receive payment	Wallet.java	/uloopjavaoomodelv2/src/virtual_currency_and_reward	line 58
TODO implement secure check of message signature	Requester.java	/uloopjavaoomodelv2/src/simulation	line 900
TODO implement secure check of signature	Requester.java	/uloopjavaoomodelv2/src/simulation	line 888
TODO implement SHA-256 public key + SHA-256 MAC address	Requester.java	/uloopjavaoomodelv2/src/simulation	line 741
TODO implement the form asking the user the already validated nickname	Requester.java	/uloopjavaoomodelv2/src/simulation	line 567
TODO Maybe do other ResourceManagement stuff based on Paulo's end of diagram ResourceMNG(T) T being the token	Requester.java	/uloopjavaoomodelv2/src/simulation	line 649
TODO Maybe do other ResourceManagement stuff based on Paulo's end of diagram	ResourceManager.java	/uloopjavaoomodelv2/src/resource_management	line 24

ResourceMNG(T) T being the token			
TODO Maybe do other ResourceManagement stuff based on Paulo's end of diagram ResourceMNG(T) T being the token	ResourceManager.java	/uloopjavaoomodelv2/src/resource_management	line 33
TODO Maybe has to send the token again or resourceManager.processTokenAsRequester (token);	Requester.java	/uloopjavaoomodelv2/src/simulation	line 650
TODO maybe try to use CMS/PKCS7 messages instead	MainSim.java	/uloopjavaoomodelv2/src/test	line 86
TODO Not sure credits can be transfered	Requester.java	/uloopjavaoomodelv2/src/simulation	line 435
TODO not sure the credits should be transfered now	CooperationManager.java	/uloopjavaoomodelv2/src/cooperation_manager	line 150
TODO not sure why periodically evaluating cooperation should be priority 2 as it is a mandatory feateure to change the trust values	CooperationManager.java	/uloopjavaoomodelv2/src/cooperation_manager	line 209
TODO Not sure why the trust value of the requestee is sent back to the requester because it discloses more information than it seems needed	Requestee.java	/uloopjavaoomodelv2/src/simulation	line 159
TODO Not sure why	Requestee.java	/uloopjavaoomodelv2/s	line 174

the trust value of the requestee is sent back to the requester because it discloses more information than it seems needed		rc/simulation	
TODO Not sure why the trust value of the requester is sent back to the requestee because it discloses more information than it seems needed	Requester.java	/uloopjavaoomodelv2/src/simulation	line 787
TODO refine how it is computed based on the token for the moment it always returns the credits proposed by the requester	CooperationManager.java	/uloopjavaoomodelv2/src/cooperation_manager	line 128
TODO retrieve the value from the node's database	DispositionalTrust.java	/uloopjavaoomodelv2/src/dispositional_trust	line 67
TODO set the dispositional trust value in the node's database	DispositionalTrust.java	/uloopjavaoomodelv2/src/dispositional_trust	line 84
TODO Should check if there are always enough resources as this version always returns true	CallAdmissionControl.java	/uloopjavaoomodelv2/src/resource_management	line 22
TODO should warn the user if the nickname she specified does not exist and stop	Requester.java	/uloopjavaoomodelv2/src/simulation	line 825
TODO substitute node	TokenComputation.jav	/uloopjavaoomodelv2/s	line 32

by the adequate object. Node is the current node	a	rc/token_computation	
TODO to implement getting the key pair from device in range	Requester.java	/uloopjavaoomodelv2/src/simulation	line 278
TODO try not return always true in case of asking if the proposed credits are enough for the wallet	RewardManager.java	/uloopjavaoomodelv2/src/virtual_currency_and_reward	line 37

References

[1] Rute Sofia (Editor, ULHT), ULOOP Consortium, *D2.3: ULOOP Overall Specification*. EU FP7 IST ULOOP project (grant number 257418) deliverable, September 2011.

[2] Rute Sofia (Editor, ULHT), ULOOP Consortium, *D3: ULOOP High Level Architecture Specification*. EU FP7 IST ULOOP project (grant number 257418) deliverable, December 2011.

[3] R. Housley et al., Network Working Group, RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile, http://www.ietf.org/rfc/rfc2459.txt, January 1999.

D3.1 Annex A: Packages, Classes and Methods

Deliverable Number	D3.1
Lead Beneficiary	UniGe
Nature/Dissemination Level	CO
Working Group/Task	WP3/Task3.1
Editor	UniGe (Carlos Ballester)
List of Authors	***UniGe*** (Carlos Ballester, Jean-Marc Seigneur), ***ULHT*** (Christian Pereira, Paulo Mendes, Rute Sofia, Waldir Moreira), ***UniUrb*** (Alessandro Bogliolo, Lorenz Cuno Klopfenstein, Saverio Delpriori, Gioele Luchetti), ***Level7*** (Paolo Di Francesco, Marzia Mammina).
Date (Project Month and dd.mm.yy)	Month 22, 30.06.2012
QAT Reviewer	Paolo di Francesco (LEVEL7)

SEVENTH FRAMEWORK PROGRAMME

FUTURE NETWORKS

Namespace Index

Packages

Here are the packages with brief descriptions (if available):

Class Index

Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Namespace Documentation

Package cooperation_manager

Classes

- class **CooperationManager**
- *Cooperation Manager class.* class **CreditsAssignment**

This function calculates the initial number of credits that will be assigned at the beginning of the cooperation process.

Package crypto_id

Packages

- package **crypto_id_validation**

Classes

- class **CryptoId**
- *The Class **CryptoId***. class **IdHelper**
- *The Class **IdHelper** provides a few helper methods to compute the different id types in ULOOP*. class **NicknameCertificates**
- *The Class **NicknameCertificates***. class **NodeId**

*The Class **NodeId***.

Package crypto_id.crypto_id_validation

Packages

- package **level7**

Classes

- class **IdValidator**

*The Class **IdValidator**.*

Package crypto_id.crypto_id_validation.level7

Classes

- class **ChallengeResponse**
- *The Class **ChallengeResponse**.* class **MobilePhoneNumber**
- *The Class **MobilePhoneNumber**.* class **SessionID**
- *The Class **SessionID**.* class **SMS_Server**
- *The Class **SMS_Server**.* class **SMS_Server_Listener**

The listener interface for receiving SMS_Server_ events.

Package dispositional_trust

Classes

- class **DispositionalTrust**
- *The Class* ***DispositionalTrust****.* class **DispositionalTrustDBHelper**

The Class ***DispositionalTrustDBHelper****.*

Package monetization

Classes

- class **MonetizationManager**

*The Class **MonetizationManager**.*

Package overall_implementation

Packages

- package **send_receive_beacons**

Classes

- class **ConfigurationBundle**
- *The Class **ConfigurationBundle**.* class **CreditsForService**
- *The Class **CreditsForService** keeps track on how many credits were proposed by a requester for a service request.* class **Service**
- *The Class **Service**.* class **ServiceLevel**
- *The Class **ServiceLevel**.* class **Signature**
- *The Class **Signature**.* class **Token**
- *The Class **Token**.* class **TrustValue**

*The Class **TrustValue**.*

Package overall_implementation.send_receive_beacons

Classes

- class **CooperationServiceRequest**
- *A request sent to the selected Requestee who had issued **ServiceRequestReply** if the Requester as selected it with the best trust value.* class **CooperationServiceRequestReply**
- *The Class **CooperationServiceRequestReply**.* class **KeyRequest**
- *The Class **KeyRequest**.* class **Message**
- *The Class **Message**.* class **ServiceRequest**
- *The Class **ServiceRequest**.* class **ServiceRequestReplies**
- *The Class **ServiceRequestReplies**.* class **ServiceRequestReply**
- *The Class **ServiceRequestReply**.* class **TokenMessage**

*The Class **TokenMessage**.*

Package resource_management

Classes

- class **CallAdmissionControl**
- *The Class **CallAdmissionControl**.* class **ResourceManager**

*The Class **ResourceManager**.*

Package simulation

Classes

- class **Location**
- *The Class **Location**.* class **Node**
- *The Class **Node**.* class **QRCode**
- *The Class **QRCode**.* class **Requestee**
- *The Class **Requestee**.* class **Requester**
- *The Class **Requester**.* class **User**
- *The Class **User**.* class **WiFiNeighborhood**

*The Class **WiFiNeighborhood**.*

Package social_trust_computation

Classes

- class **SocialTrustComputation**
- *The Class **SocialTrustComputation** an object of type **TrustTable** it is basically the heart of the way trust levels are computed.* class **TrustTable**
- *The Class **TrustTable** is a linked list of the **TrustTableEntry** class in the node, a trusttable object is initiated (main)* class **TrustTableEntry**

*The Class **TrustTableEntry**.*

Package test

Classes

- class **MainSim**

*The Class **MainSim**.*

Package token_computation

Classes

- class **TokenComputation**

Provides the tokens to be used in exchange of resources.

Package virtual_currency_and_reward

Classes

- class **AccountCreationAck**
- *Message representing the confirmation of a created account.* class **AccountCreationRequest**
- *Message representing the request to create a new account.* class **Bank**
- *The Class **Bank**.* class **Credit**
- *The Class **Credit**.* class **CreditTransferAck**
- *The Class **CreditTransferAck**.* class **CreditTransferOrder**
- *The Class **CreditTransferOrder**.* class **MarketBroker**
- *The Class **MarketBroker**.* class **Mint**
- *The Class **Mint**.* class **RewardManager**
- *The Class **RewardManager**.* class **Wallet**
- *The Class **Wallet**.* class **WalletStatusRequest**
- *The Class **WalletStatusRequest**.* class **WalletStatusResponse**
- *The Class **WalletStatusResponse**.* class **WalletSynchronizationRequest**

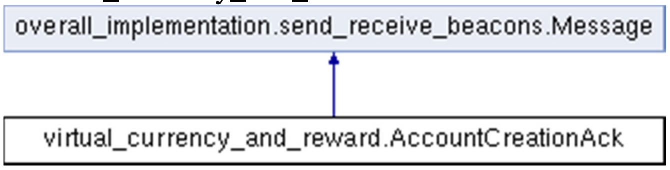
*The Class **WalletSynchronizationRequest**.*

Class Documentation

virtual_currency_and_reward.AccountCreationAck Class Reference

Message representing the confirmation of a created account.

Inheritance diagram for virtual_currency_and_reward.AccountCreationAck:


overall_implementation.send_receive_beacons.Message
virtual_currency_and_reward.AccountCreationAck

Public Member Functions

- **AccountCreationAck** (CryptoId owner, CryptoId bank, Date timestampCreated)
  Instantiates a new account creation ack.
- CryptoId **getAccountOwner** ()
  Gets the account owner.
- CryptoId **getBank** ()
  Gets the bank.
- Date **getTimestampCreated** ()
  Gets the timestamp created.

Detailed Description

Message representing the confirmation of a created account.

Author:

UniUrb

Version:

Jun 27, 2012

Constructor & Destructor Documentation

virtual_currency_and_reward.AccountCreationAck.AccountCreationAck (CryptoId *owner*, CryptoId *bank*, Date *timestampCreated*)

Instantiates a new account creation ack.

Parameters:

owner	the owner
bank	the bank
timestampCreated	the timestamp created

Member Function Documentation

CryptoId virtual_currency_and_reward.AccountCreationAck.getAccountOwner ()

Gets the account owner.

Returns:

the account owner .property name="Account owner"

CryptoId virtual_currency_and_reward.AccountCreationAck.getBank ()

Gets the bank.

Returns:

the bank .property name="Bank"

Date virtual_currency_and_reward.AccountCreationAck.getTimestampCreated ()

Gets the timestamp created.

Returns:

the timestamp created .property name="Creation timestamp"

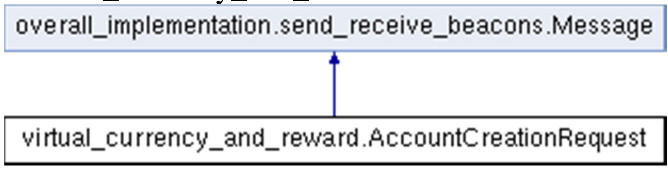
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/virtual_currency_and_reward/**AccountCreationAck.java**

virtual_currency_and_reward.AccountCreationRequest Class Reference

Message representing the request to create a new account.

Inheritance diagram for virtual_currency_and_reward.AccountCreationRequest:


overall_implementation.send_receive_beacons.Message
virtual_currency_and_reward.AccountCreationRequest

Public Member Functions

- **AccountCreationRequest** (CryptoId owner, CryptoId bank, Date timestampIssued)
  Instantiates a new account creation request.
- CryptoId **getAccountOwner** ()
  Gets the account owner.
- CryptoId **getBank** ()
  Gets the bank.
- Date **getTimestampIssued** ()
  Gets the timestamp issued.

Detailed Description

Message representing the request to create a new account.

Author:

UniUrb

Version:

Jun 27, 2012

Constructor & Destructor Documentation

virtual_currency_and_reward.AccountCreationRequest.AccountCreationRequest (CryptoId *owner*, CryptoId *bank*, Date *timestampIssued*)

Instantiates a new account creation request.

Parameters:

owner	the owner
bank	the bank
timestampIssued	the timestamp issued

Member Function Documentation

CryptoId virtual_currency_and_reward.AccountCreationRequest.getAccountOwner ()

Gets the account owner.

Returns:

the account owner .property name="Account owner"

CryptoId virtual_currency_and_reward.AccountCreationRequest.getBank ()

Gets the bank.

Returns:

the bank .property name="Bank"

Date virtual_currency_and_reward.AccountCreationRequest.getTimestampIssued ()

Gets the timestamp issued.

Returns:

the timestamp issued .property name="Issue timestamp"

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/virtual_currency_and_reward/**AccountCreationRequest.java**

virtual_currency_and_reward.Bank Class Reference

The Class **Bank**.

Public Member Functions

- void **receiveAccountCreationRequest** (**AccountCreationRequest** request)
  Receive account creation request.
- void **receiveCreditTransferOrder** (**CreditTransferOrder** transferOrder)
  Receive credit transfer order.
- void **receivedAccountStatusRequest** (String accountOwner)
  Received account status request.

Detailed Description

The Class **Bank**.

Author:

UniUrb

Version:

Jun 27, 2012

Member Function Documentation

void virtual_currency_and_reward.Bank.receiveAccountCreationRequest (AccountCreationRequest *request*)

Receive account creation request.

Parameters:

request	the request

void virtual_currency_and_reward.Bank.receiveCreditTransferOrder (CreditTransferOrder *transferOrder*)

Receive credit transfer order.

Parameters:

transferOrder	the transfer order

void virtual_currency_and_reward.Bank.receivedAccountStatusRequest (String *accountOwner*)

Received account status request.

Parameters:

accountOwner	the account owner

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/virtual_currency_and_reward/**Bank.java**

resource_management.CallAdmissionControl Class Reference

The Class **CallAdmissionControl**.

Public Member Functions

- boolean **enoughResources** (double token)
  Check if there are enough resources.

Detailed Description

The Class **CallAdmissionControl**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 29, 2012

Member Function Documentation

boolean resource_management.CallAdmissionControl.enoughResources (double *token*)

Check if there are enough resources.

Parameters:

token	

Returns:

true if there are enough resources

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/resource_management/**CallAdmissionControl.java**

crypto_id.crypto_id_validation.level7.ChallengeResponse Class Reference

The Class **ChallengeResponse**.

Public Member Functions

- **ChallengeResponse** ()
  Instantiates a new challenge response.
- String **getChallenge** ()
  Gets the challenge.
- String **getResponse** ()
  Gets the response.

Detailed Description

The Class **ChallengeResponse**.

Author:

Marzia (Level7)
Paolo (Level7)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

crypto_id.crypto_id_validation.level7.ChallengeResponse.ChallengeResponse ()

Instantiates a new challenge response.

Member Function Documentation

String crypto_id.crypto_id_validation.level7.ChallengeResponse.getChallenge ()

Gets the challenge.

Returns:

the challenge

String crypto_id.crypto_id_validation.level7.ChallengeResponse.getResponse ()

Gets the response.

Returns:

the response

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/crypto_id/crypto_id_validation/level7/**ChallengeResponse.java**

overall_implementation.ConfigurationBundle Class Reference

The Class **ConfigurationBundle**.

Public Member Functions

- KeyPair **getKeyPair** ()
  Gets the key pair.
- **NicknameCertificates getNickNameCertificates** ()
  Gets the nick name certificates.
- **DispositionalTrust getDispositionalTrust** ()
  Gets the dispositional trust.
- int **getCredits** ()
  Gets the credits.

Detailed Description

The Class **ConfigurationBundle**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Member Function Documentation

int overall_implementation.ConfigurationBundle.getCredits ()

Gets the credits.

Returns:

the credits

DispositionalTrust overall_implementation.ConfigurationBundle.getDispositionalTrust ()

Gets the dispositional trust.

Returns:

the dispositional trust

KeyPair overall_implementation.ConfigurationBundle.getKeyPair ()

Gets the key pair.

Returns:

the key pair

NicknameCertificates overall_implementation.ConfigurationBundle.getNickNameCertificates ()

Gets the nick name certificates.

Returns:

the nick name certificates

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/**ConfigurationBundle.java**

cooperation_manager.CooperationManager Class Reference

Cooperation Manager class.

Public Member Functions

- **CooperationManager** (TokenComputation tokenComputation, **RewardManager** rewardManager)
  Constructor method of the Cooperation Manager class.
- double **assignInitialCredits** (float trustLevel, double cMin, double cMax)
  This function calculates the initial number of credits that will be assigned at the boot-up process.
- double **computeCooperationCredits** (String serviceType, int serviceLevel, String serviceRequestId)
  This function compute the credits that will be sent to the Requester to encourage the Requestee in engaging in cooperation.
- double **getToken** (CryptoId cryptoId, double credits)
  Token is a function of the credits the Requester is willing to provide to Requestee for the perspective cooperation and its trust level towards the Requestee.
- double **getCredits** (double token, String serviceRequestId)
  This function will be used by the Requestee to obtain the credits involved in the cooperation process and to decide if this amount is enough to cooperate.
- boolean **enoughCreditsToCooperate** (boolean retailer, double credits)
  This function will be used in the retailer cooperation case.
- double **updateCoopCredits** (double newCredits)
  This function updates the cooperation credits in the Requestee side.
- boolean **enoughIncentivesToCooperate** (double token, String serviceRequestId)
  This function will evaluate if the incentive sent (Token value) is or is not enough to establish the association.
- void **creditsRequesterUpdate** (String serviceRequestId)
  This function will be called only if the result of the cooperation process (be it voluntary or retailer) is positive.
- void **periodicallyEvaluateCooperation** ()
  Periodically evaluate cooperation.

Detailed Description

Cooperation Manager class.

Author:

Christian Silva (ULHT)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

cooperation_manager.CooperationManager.CooperationManager (TokenComputation *tokenComputation*, RewardManager *rewardManager*)

Constructor method of the Cooperation Manager class.

Member Function Documentation

double cooperation_manager.CooperationManager.assignInitialCredits (float *trustLevel*, double *cMin*, double *cMax*)

This function calculates the initial number of credits that will be assigned at the boot-up process.

Parameters:

the	trust level
cMin	a double argument. Minimum credits to assign
cMax	a double argument. Maximum credits to assign

Returns:

The initial amount of credits assigned

double cooperation_manager.CooperationManager.computeCooperationCredits (String *serviceType*, int *serviceLevel*, String *serviceRequestId*)

This function compute the credits that will be sent to the Requester to encourage the Requestee in engaging in cooperation.

Parameters:

serviceType	a String value. It is the service type requested
serviceLevel	a integer value. It is the service level requested

Returns:

The credits exchanged in the cooperation process

void cooperation_manager.CooperationManager.creditsRequesterUpdate (String *serviceRequestId*)

This function will be called only if the result of the cooperation process (be it voluntary or retailer) is positive.

The cooperation credits (requester side) will be updated by, decreasing the credits that were sent during the cooperation.

boolean cooperation_manager.CooperationManager.enoughCreditsToCooperate (boolean *retailer*, double *credits*)

This function will be used in the retailer cooperation case.

With this function the Requestee will evaluate if the credits received are or are not enough to continue with the cooperation process

Parameters:

rewardManager	is the reference associated to the RewardManager class
retailer	a boolean argument. It is refers to the node status: retailer or voluntary
credits	a double argument. Amount of credits received by the Requestee

Returns:

the boolean value. This function tell us if the credits sent are or are not enough to continue with the cooperation

boolean cooperation_manager.CooperationManager.enoughIncentivesToCooperate (double *token*, String *serviceRequestId*)

This function will evaluate if the incentive sent (Token value) is or is not enough to establish the association.

Parameters:

token	a double argument.

Returns:

the boolean value. This function tells us if the Token value is or is not enough to cooperate

double cooperation_manager.CooperationManager.getCredits (double *token*, String *serviceRequestId*)

This function will be used by the Requestee to obtain the credits involved in the cooperation process and to decide if this amount is enough to cooperate.

Parameters:

token	a double argument. It is the "weight" that the request has.

Returns:

The amount of credits exchanged in the cooperation process

double cooperation_manager.CooperationManager.getToken (CryptoId *cryptoId*, double *credits*)

Token is a function of the credits the Requester is willing to provide to Requestee for the perspective cooperation and its trust level towards the Requestee.

Parameters:

the	CryptoId of the trusted
the	credits to be provided

Returns:

Token value

void cooperation_manager.CooperationManager.periodicallyEvaluateCooperation ()

Periodically evaluate cooperation.

double cooperation_manager.CooperationManager.updateCoopCredits (double *newCredits*)

This function updates the cooperation credits in the Requestee side.

Parameters:

credits	a double parameter.

Returns:

The amount of credits earned by the Requestee with a given cooperation

---

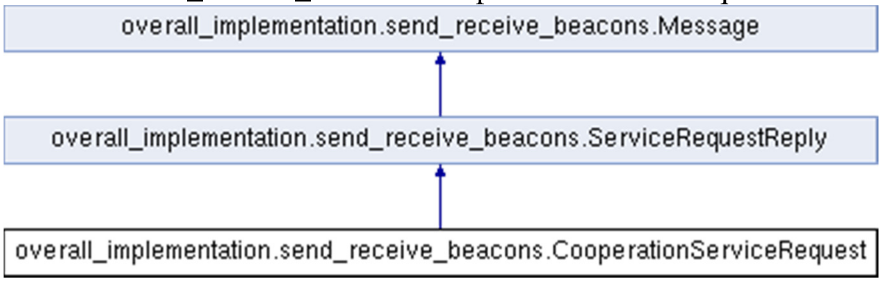
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/cooperation_manager/**CooperationManager.java**

overall_implementation.send_receive_beacons.CooperationServiceRequest Class Reference

A request sent to the selected Requestee who had issued **ServiceRequestReply** if the Requester as selected it with the best trust value.

Inheritance diagram for overall_implementation.send_receive_beacons.CooperationServiceRequest:


overall_implementation.send_receive_beacons.Message
overall_implementation.send_receive_beacons.ServiceRequestReply
overall_implementation.send_receive_beacons.CooperationServiceRequest

Public Member Functions

- **CooperationServiceRequest** (**ServiceRequestReply** selectedServiceRequestReply, **Signature** signature, **TrustValue** trustValueOfRequesterInRequestee, **Token** tokenWillingToBeGivenByTheRequesterToTheRequestee)
  *Instances a **CooperationServiceRequest**.*
- **TrustValue getTrustValueOfRequesterInRequestee** ()
  Gets the trust value of requester in requestee.
- **Token getTokenWillingToBeGivenByTheRequesterToTheRequestee** ()
  Gets the token willing to be given by the requester to the requestee.

Detailed Description

A request sent to the selected Requestee who had issued **ServiceRequestReply** if the Requester as selected it with the best trust value.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 30, 2012

Constructor & Destructor Documentation

overall_implementation.send_receive_beacons.CooperationServiceRequest.CooperationServiceRequest (ServiceRequestReply *selectedServiceRequestReply*, Signature *signature*, TrustValue *trustValueOfRequesterInRequestee*, Token *tokenWillingToBeGivenByTheRequesterToTheRequestee*)

Instances a **CooperationServiceRequest**.

Parameters:

selectedServiceRequestReply	the selected service request reply
signature	the signature
trustValueOfRequesterInRequestee	the trust value of requester in requestee
tokenWillingToBeGivenByTheRequesterToTheRequestee	the token willing to be given by the requester to the requestee

Member Function Documentation

Token overall_implementation.send_receive_beacons.CooperationServiceRequest.getTokenWillingToBeGivenByTheRequesterToTheRequestee ()

Gets the token willing to be given by the requester to the requestee.

Returns:

the token willing to be given by the requester to the requestee

TrustValue overall_implementation.send_receive_beacons.CooperationServiceRequest.getTrustValueOfRequesterInRequestee ()

Gets the trust value of requester in requestee.

Returns:

the trust value of requester in requestee

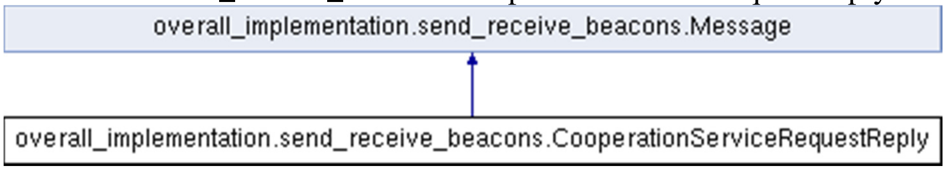
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/send_receive_beacons/**CooperationServiceRequest.java**

overall_implementation.send_receive_beacons.CooperationServiceRequestReply Class Reference

The Class **CooperationServiceRequestReply**.

Inheritance diagram for overall_implementation.send_receive_beacons.CooperationServiceRequestReply:


overall_implementation.send_receive_beacons.Message
overall_implementation.send_receive_beacons.CooperationServiceRequestReply

Public Member Functions

- **CooperationServiceRequestReply** (**CryptoId** sender, **CryptoId** recipient, **Signature** signature, **TrustValue** trustValueOfRequesteeInRequester, boolean cooperationSuccessReply, String serviceRequestId)
  Instantiates a cooperation service request reply.
- boolean **isCooperationSuccessReply** ()
  Checks if is cooperation success reply.
- **TrustValue getTrustValueOfRequesteeInRequester** ()
  Gets the trust value of requestee in requester.
- String **getServiceRequestId** ()
  Gets the service request id.

Detailed Description

The Class **CooperationServiceRequestReply**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

overall_implementation.send_receive_beacons.CooperationServiceRequestReply.CooperationServiceRequestReply (CryptoId *sender*, CryptoId *recipient*, Signature *signature*, TrustValue *trustValueOfRequesteeInRequester*, boolean *cooperationSuccessReply*, String *serviceRequestId*)

Instantiates a cooperation service request reply.

Parameters:

sender	the sender
recipient	the recipient
signature	the signature

trustValueOfRequesteeInRequester	the trust value of requestee in requester
cooperationSuccessReply	the cooperation success reply
serviceRequestId	the service request id
token	the token

Member Function Documentation

String overall_implementation.send_receive_beacons.CooperationServiceRequestReply.getServiceRequestId ()

Gets the service request id.

Returns:

the service request id

TrustValue overall_implementation.send_receive_beacons.CooperationServiceRequestReply.getTrustValueOfRequesteeInRequester ()

Gets the trust value of requestee in requester.

Returns:

the trust value of requestee in requester

boolean overall_implementation.send_receive_beacons.CooperationServiceRequestReply.isCooperationSuccessReply ()

Checks if is cooperation success reply.

Returns:

true, if is cooperation success reply

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/send_receive_beacons/**CooperationServiceRequestReply.java**

virtual_currency_and_reward.Credit Class Reference

The Class **Credit**.

Public Member Functions

- **Credit** (double value)
  Instantiates a new credit.
- double **getValue** ()
  Gets the value.

Detailed Description

The Class **Credit**.

Author:

UniUrb

Version:

Jun 27, 2012

Constructor & Destructor Documentation

virtual_currency_and_reward.Credit.Credit (double *value*)

Instantiates a new credit.

Parameters:

value	the value

Member Function Documentation

double virtual_currency_and_reward.Credit.getValue ()

Gets the value.

Returns:

the value

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/virtual_currency_and_reward/**Credit.java**

cooperation_manager.CreditsAssignment Class Reference

This function calculates the initial number of credits that will be assigned at the beginning of the cooperation process.

Detailed Description

This function calculates the initial number of credits that will be assigned at the beginning of the cooperation process.

The credits to consider are dependent on the trust level following a Gaussian distribution behavior. To more information for this function: ULOOP/WorkPackages/WP3/Task3.1/CooperationManager/Coop-Manager-Functions.pdf

Author:

Christian Silva (ULHT)

Version:

Jun 27, 2012

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/cooperation_manager/**CreditsAssignment.java**

overall_implementation.CreditsForService Class Reference

The Class **CreditsForService** keeps track on how many credits were proposed by a requester for a service request.

Public Member Functions

- **CreditsForService** (String **serviceRequestId**, double credits)
  Instantiates a new credits for service.
- String **getServiceRequestId** ()
  Gets the service request id.
- **Credit getCredit** ()
  Gets the credit.

Public Attributes

- String **serviceRequestId**
  The service request id.
- **Credit credit**
  The credit.

Detailed Description

The Class **CreditsForService** keeps track on how many credits were proposed by a requester for a service request.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 30, 2012

Constructor & Destructor Documentation

overall_implementation.CreditsForService.CreditsForService (String *serviceRequestId*, double *credits*)

Instantiates a new credits for service.

Parameters:

serviceRequestId	the service request id
credits	the credits

Member Function Documentation

Credit overall_implementation.CreditsForService.getCredit ()

Gets the credit.

Returns:

the credit

String overall_implementation.CreditsForService.getServiceRequestId ()

Gets the service request id.

Returns:

the service request id

Member Data Documentation

Credit overall_implementation.CreditsForService.credit

The credit.

String overall_implementation.CreditsForService.serviceRequestId

The service request id.

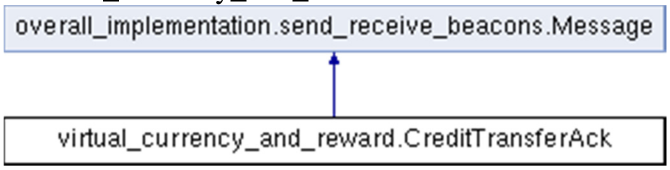
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/**CreditsForService.java**

virtual_currency_and_reward.CreditTransferAck Class Reference

The Class **CreditTransferAck**.

Inheritance diagram for virtual_currency_and_reward.CreditTransferAck:


overall_implementation.send_receive_beacons.Message
virtual_currency_and_reward.CreditTransferAck

Public Member Functions

- **CreditTransferAck** (long orderId, double amount, CryptoId payeeCryptoId, CryptoId payerCryptoId, Date done)
  Instantiates a new credit transfer ack.
- **CreditTransferAck** (long orderId, long negotiationId, double amount, CryptoId payee, CryptoId payer, Date done)
  Instantiates a new credit transfer ack.

Detailed Description

The Class **CreditTransferAck**.

Author:

UniUrb

Version:

Jun 27, 2012

Constructor & Destructor Documentation

virtual_currency_and_reward.CreditTransferAck.CreditTransferAck (long *orderId*, double *amount*, CryptoId *payeeCryptoId*, CryptoId *payerCryptoId*, Date *done*)

Instantiates a new credit transfer ack.

Parameters:

orderId	the order id
amount	the amount
payeeCryptoId	the payee crypto id
payerCryptoId	the payer crypto id
done	the done

virtual_currency_and_reward.CreditTransferAck.CreditTransferAck (long *orderId*, long *negotiationId*, double *amount*, CryptoId *payee*, CryptoId *payer*, Date *done*)

Instantiates a new credit transfer ack.

Parameters:

orderId	the order id
negotiationId	the negotiation id
amount	the amount
payee	the payee
payer	the payer
done	the done

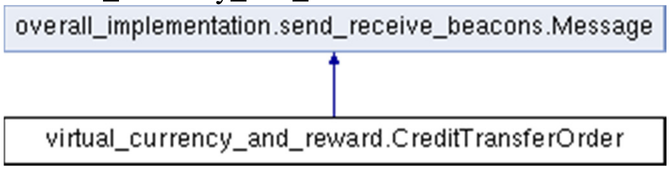
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/virtual_currency_and_reward/**CreditTransferAck.java**

virtual_currency_and_reward.CreditTransferOrder Class Reference

The Class **CreditTransferOrder**.

Inheritance diagram for virtual_currency_and_reward.CreditTransferOrder:


overall_implementation.send_receive_beacons.Message
virtual_currency_and_reward.CreditTransferOrder

Public Member Functions

- **CreditTransferOrder** (long orderId, double amount, CryptoId payee, CryptoId payer, Date issue)
  Instantiates a new credit transfer order.
- **CreditTransferOrder** (long orderId, long negotiationId, double amount, CryptoId payee, CryptoId payer, Date issue)
  Instantiates a new credit transfer order.

Detailed Description

The Class **CreditTransferOrder**.

Author:

UniUrb

Version:

Jun 27, 2012

Constructor & Destructor Documentation

virtual_currency_and_reward.CreditTransferOrder.CreditTransferOrder (long *orderId*, double *amount*, CryptoId *payee*, CryptoId *payer*, Date *issue*)

Instantiates a new credit transfer order.

Parameters:

orderId	the order id
amount	the amount
payee	the payee
payer	the payer
issue	the issue

virtual_currency_and_reward.CreditTransferOrder.CreditTransferOrder (long *orderId*, long *negotiationId*, double *amount*, CryptoId *payee*, CryptoId *payer*, Date *issue*)

Instantiates a new credit transfer order.

Parameters:

orderId	the order id
negotiationId	the negotiation id
amount	the amount
payee	the payee
payer	the payer
issue	the issue

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/virtual_currency_and_reward/**CreditTransferOrder.java**

crypto_id.CryptoId Class Reference

The Class **CryptoId**.

Public Member Functions

- **CryptoId** (PublicKey publicKey)
  Instantiates a new crypto id.
- **CryptoId** (String userProvidedCryptoId)
  Instantiates a new crypto id.
- byte[] **getBytes** ()
  Gets the bytes of the crypto id, which is the hash of the public key.
- void **printCryptoId** ()
  Prints the crypto id.
- boolean **isSameCryptoIdAs** (**CryptoId** cryptoId)
  Checks this crypto id to another one.

Detailed Description

The Class **CryptoId**.

Author:

Daniel Romao (CMS)
Nuno Martins (CMS)
Alfredo matos (CMS)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

crypto_id.CryptoId.CryptoId (PublicKey *publicKey*)

Instantiates a new crypto id.

Parameters:

publicKey	the public key

crypto_id.CryptoId.CryptoId (String *userProvidedCryptoId*)

Instantiates a new crypto id.

Parameters:

the	String provided by the user either by hand or QRCode

Member Function Documentation

byte [] crypto_id.CryptoId.getBytes ()

Gets the bytes of the crypto id, which is the hash of the public key.

Returns:

the crypto id

boolean crypto_id.CryptoId.isSameCryptoIdAs (CryptoId *cryptoId*)

Checks this crypto id to another one.

Parameters:

cryptoId	

Returns:

true if same crypto id

void crypto_id.CryptoId.printCryptoId ()

Prints the crypto id.

Shows on the console the hexadecimal values of the CryptoID

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/crypto_id/**CryptoId.java**

dispositional_trust.DispositionalTrust Class Reference

The Class **DispositionalTrust**.

Public Member Functions

- **DispositionalTrust** ()
  Instantiates a new dispositional trust.
- **DispositionalTrust** (int dispositionalTrustValue)
  Instantiates a new dispositional trust.
- int **getDispositionalTrustValue** ()
  Gets the dispositional trust value.
- void **setDispositionalTrustValue** (int dispositionalTrustValue)
  Sets the dispositional trust value.
- void **setParanoidDTrustValue** ()
  Sets the paranoid d trust value.
- void **setPartiallyOpenDTrustValue** ()
  Sets the partially open d trust value.
- void **setBlindTrustDTrustValue** ()
  Sets the blind trust d trust value.

Static Public Attributes

- static final **DispositionalTrust PARANOID** = new **DispositionalTrust**(0)
  The Constant PARANOID.
- static final **DispositionalTrust PARTIALLY_OPEN** = new **DispositionalTrust**(50)
  The Constant PARTIALLY_OPEN.
- static final **DispositionalTrust BLIND_TRUST** = new **DispositionalTrust**(100)
  The Constant BLIND_TRUST.

Detailed Description

The Class **DispositionalTrust**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

dispositional_trust.DispositionalTrust.DispositionalTrust ()

Instantiates a new dispositional trust.

dispositional_trust.DispositionalTrust.DispositionalTrust (int *dispositionalTrustValue*)

Instantiates a new dispositional trust.

Parameters:

dispositionalTrust Value	the dispositional trust value

Member Function Documentation

int dispositional_trust.DispositionalTrust.getDispositionalTrustValue ()

Gets the dispositional trust value.

Returns:

the dispositional trust value

void dispositional_trust.DispositionalTrust.setBlindTrustDTrustValue ()

Sets the blind trust d trust value.

void dispositional_trust.DispositionalTrust.setDispositionalTrustValue (int *dispositionalTrustValue*)

Sets the dispositional trust value.

Parameters:

dispositionalTrust Value	the new dispositional trust value

void dispositional_trust.DispositionalTrust.setParanoidDTrustValue ()

Sets the paranoid d trust value.

void dispositional_trust.DispositionalTrust.setPartiallyOpenDTrustValue ()

Sets the partially open d trust value.

Member Data Documentation

final DispositionalTrust dispositional_trust.DispositionalTrust.BLIND_TRUST = new DispositionalTrust(100) `[static]`

The Constant BLIND_TRUST.

final DispositionalTrust dispositional_trust.DispositionalTrust.PARANOID = new DispositionalTrust(0) `[static]`

The Constant PARANOID.

final DispositionalTrust dispositional_trust.DispositionalTrust.PARTIALLY_OPEN = new DispositionalTrust(50) `[static]`

The Constant PARTIALLY_OPEN.

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/dispositional_trust/**DispositionalTrust.java**

dispositional_trust.DispositionalTrustDBHelper Class Reference

The Class **DispositionalTrustDBHelper**.

Public Member Functions

- **DispositionalTrustDBHelper** ()
  Instantiates a new dispositional trust db helper.
- int **getDispositionalTrust** ()
  Gets the dispositional trust.
- Boolean **setDispositionalTrust** (int dTrust)
  Sets the dispositional trust.

Detailed Description

The Class **DispositionalTrustDBHelper**.

Warning! Database name, username, password and table and column names are generic, this is not working until we don't define the "real" db

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

dispositional_trust.DispositionalTrustDBHelper.DispositionalTrustDBHelper ()

Instantiates a new dispositional trust db helper.

Member Function Documentation

int dispositional_trust.DispositionalTrustDBHelper.getDispositionalTrust ()

Gets the dispositional trust.

Returns:

the dispositional trust

Boolean dispositional_trust.DispositionalTrustDBHelper.setDispositionalTrust (int *dTrust*)

Sets the dispositional trust.

Parameters:

dTrust	the d trust

Returns:

the boolean

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/dispositional_trust/**DispositionalTrustDBHelper.java**

crypto_id.IdHelper Class Reference

The Class **IdHelper** provides a few helper methods to compute the different id types in ULOOP.

Static Public Member Functions

- static final byte[] **SHA256Bytes** (byte[]key)
  SHA 256 bytes.
- static final byte[] **generateNodeId** (**CryptoId** cryptoId, byte macAddress[])
  Generate node id.
- static final void **printBytes** (byte[]data)
  Prints the bytes.
- static final boolean **isSameCryptoId** (**CryptoId** cryptoId1, **CryptoId** cryptoId2)
  Checks if the CryptoIds are the same.
- static final boolean **isSameNodeId** (**NodeId** nodeId1, **NodeId** nodeId2)
  Checks if the NodeIds are the same.

Detailed Description

The Class **IdHelper** provides a few helper methods to compute the different id types in ULOOP.

Author:

Daniel Romao (CMS)
Nuno Martins (CMS)
Alfredo matos (CMS)

Version:

Jun 27, 2012

Member Function Documentation

static final byte [] crypto_id.IdHelper.generateNodeId (CryptoId *cryptoId*, byte *macAddress*[]) `[static]`

Generate node id.

Parameters:

macAddress	the network identifier

Returns:

the byte[]

static final boolean crypto_id.IdHelper.isSameCryptoId (CryptoId *cryptoId1*, CryptoId *cryptoId2*) `[static]`

Checks if the CryptoIds are the same.

Returns:

true if same crypto-ids

static final boolean crypto_id.IdHelper.isSameNodeId (NodeId *nodeId1*, NodeId *nodeId2*) [static]

Checks if the NodeIds are the same.

Returns:

true if same node ids

static final void crypto_id.IdHelper.printBytes (byte[] *data*) [static]

Prints the bytes.

Support function for hex byte output

@param data the data

static final byte [] crypto_id.IdHelper.SHA256Bytes (byte[] *key*) [static]

SHA 256 bytes.

Parameters:

key	the key

Returns:

the byte[]

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/crypto_id/**IdHelper.java**

crypto_id.crypto_id_validation.IdValidator Class Reference

The Class **IdValidator**.

Public Member Functions

- **IdValidator** (String name, char[] keyStorePass)
  Instantiates a new id validator.
- boolean **validateNickname** (String wantedNickname, String phoneNumberToSendSms, PublicKey publicKey)
  Validate nickname.
- byte[] **sign** (byte[] data) throws GeneralSecurityException, CMSException, IOException
  Sign.
- KeyStore **getKeystore** (char[] password)
  Gets the keystore.
- X509CertificateObject[] **getNicknameCertificates** (String nickName)
  Gets the nickname certificates.
- **CryptoId getCryptoId** (String nickname)
  Gets the associated cryptoId to the nickname.

Static Public Attributes

- static final Hashtable< String,
- **NicknameCertificates** > **GLOBALLY_USED_NICKNAMES_WITH_PUBLIC_KEYS_AND_CERTIFICATES** = new Hashtable<String, **NicknameCertificates**>()
  The Constant GLOBALLY_USED_NICKNAMES_WITH_PUBLIC_KEYS_AND_CERTIFICATES.
- static final HashMap< String,
- String > **WANTED_NICKNAMES** = new HashMap<String, String>()
  The Constant WANTED_NICKNAMES.

Detailed Description

The Class **IdValidator**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

crypto_id.crypto_id_validation.IdValidator.IdValidator (String *name*, char[] *keyStorePass*)

Instantiates a new id validator.

Parameters:

name	the name
keyStorePass	the key store pass

Member Function Documentation

CryptoId crypto_id.crypto_id_validation.IdValidator.getCryptoId (String *nickname*)

Gets the associated cryptoId to the nickname.

Parameters:

nickname	

Returns:

the **CryptoId**, null if nickname does not exist

KeyStore crypto_id.crypto_id_validation.IdValidator.getKeystore (char[] *password*)

Gets the keystore.

Parameters:

password	the password

Returns:

the keystore

X509CertificateObject [] crypto_id.crypto_id_validation.IdValidator.getNicknameCertificates (String *nickName*)

Gets the nickname certificates.

Parameters:

nickName	the nick name

Returns:

the nickname certificates

byte [] crypto_id.crypto_id_validation.IdValidator.sign (byte[] *data*) throws GeneralSecurityException, CMSException, IOException

Sign.

Parameters:

data	the data

Returns:

the byte[]

Exceptions:

GeneralSecurityEx	the general security exception

ception	
CMSException	the cMS exception
IOException	Signals that an I/O exception has occurred.

boolean crypto_id.crypto_id_validation.IdValidator.validateNickname (String *wantedNickname*, String *phoneNumberToSendSms*, PublicKey *publicKey*)

Validate nickname.

Parameters:

wantedNickname	the wanted nickname
phoneNumberToSendSms	the phone number to send sms
publicKey	the public key

Returns:

true, if successful

Member Data Documentation

final Hashtable<String, NicknameCertificates> crypto_id.crypto_id_validation.IdValidator.GLOBALLY_USED_NICKNAMES_WITH_PUBLIC_KEYS_AND_CERTIFICATES = new Hashtable<String, NicknameCertificates>() `[static]`

The Constant GLOBALLY_USED_NICKNAMES_WITH_PUBLIC_KEYS_AND_CERTIFICATES.

final HashMap<String, String> crypto_id.crypto_id_validation.IdValidator.WANTED_NICKNAMES = new HashMap<String, String>() `[static]`

The Constant WANTED_NICKNAMES.

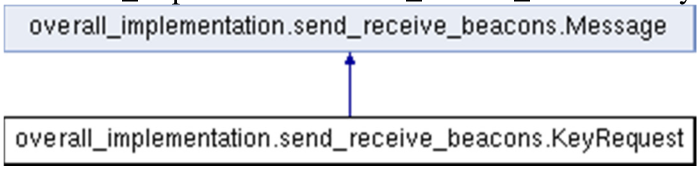
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/crypto_id/crypto_id_validation/**IdValidator.java**

overall_implementation.send_receive_beacons.KeyRequest Class Reference

The Class **KeyRequest**.

Inheritance diagram for overall_implementation.send_receive_beacons.KeyRequest:


overall_implementation.send_receive_beacons.Message
overall_implementation.send_receive_beacons.KeyRequest

Public Member Functions

- **KeyRequest** (**CryptoId** sender, **CryptoId** recipient, **Signature** signature)
  Instantiates a new key request.

Detailed Description

The Class **KeyRequest**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 30, 2012

Constructor & Destructor Documentation

overall_implementation.send_receive_beacons.KeyRequest.KeyRequest (CryptoId *sender*, CryptoId *recipient*, Signature *signature*)

Instantiates a new key request.

Parameters:

sender	the sender
recipient	the recipient
signature	the signature

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/send_receive_beacons/**KeyRequest.java**

simulation.Location Class Reference

The Class **Location**.

Public Member Functions

- **Location** (double lat, double lon)
  Instantiates a new location.
- double **getLat** ()
  Gets the lat.
- double **getLon** ()
  Gets the lon.

Detailed Description

The Class **Location**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

simulation.Location.Location (double *lat*, double *lon*)

Instantiates a new location.

Parameters:

lat	the lat
lon	the lon

Member Function Documentation

double simulation.Location.getLat ()

Gets the lat.

Returns:

the lat

double simulation.Location.getLon ()

Gets the lon.

Returns:

the lon

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/simulation/**Location.java**

test.MainSim Class Reference

The Class **MainSim**.

Static Public Member Functions

- static void **main** (String[] args)
  The main method.

Detailed Description

The Class **MainSim**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Member Function Documentation

static void test.MainSim.main (String[] *args*) `[static]`

The main method.

Parameters:

args	the arguments

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/test/**MainSim.java**

virtual_currency_and_reward.MarketBroker Class Reference

The Class **MarketBroker**.

Public Member Functions

- void **placeOffer** (String n, double money)
  Place offer.
- void **placeOrder** (String n, double money)
  Place order.

Detailed Description

The Class **MarketBroker**.

Author:
UniUrb

Version:
Jun 27, 2012

Member Function Documentation

void virtual_currency_and_reward.MarketBroker.placeOffer (String *n*, double *money*)

Place offer.

Parameters:

n	the n
money	the money

void virtual_currency_and_reward.MarketBroker.placeOrder (String *n*, double *money*)

Place order.

Parameters:

n	the n
money	the money

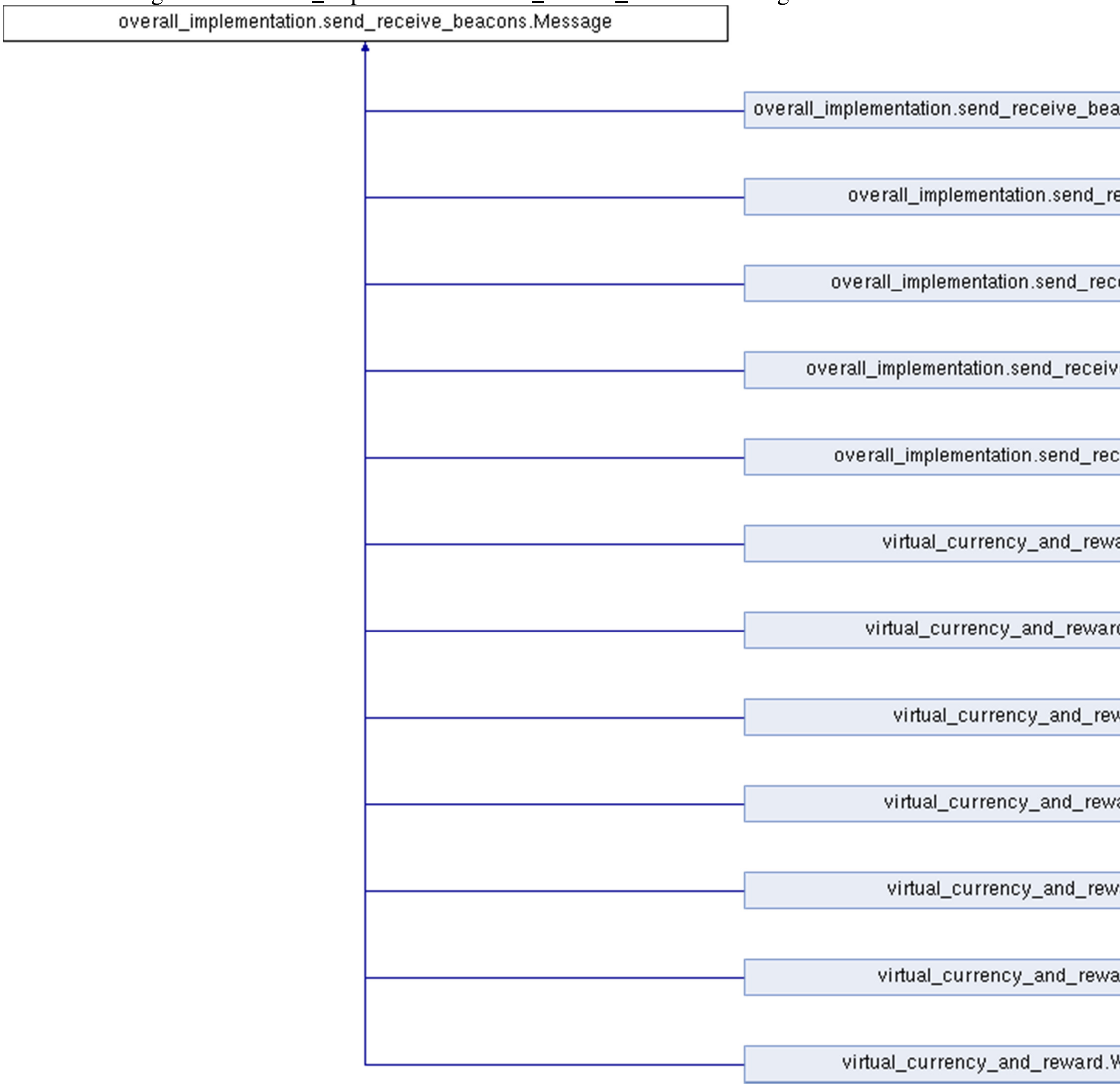
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/virtual_currency_and_reward/**MarketBroker.java**

overall_implementation.send_receive_beacons.Message Class Reference

The Class **Message**.

Inheritance diagram for overall_implementation.send_receive_beacons.Message:


overall_implementation.send_receive_beacons.Message
overall_implementation.send_receive_bea
overall_implementation.send_re
overall_implementation.send_rec
overall_implementation.send_receiv
overall_implementation.send_rec
virtual_currency_and_rewa
virtual_currency_and_reward
virtual_currency_and_rew
virtual_currency_and_rewa
virtual_currency_and_rew
virtual_currency_and_rewa
virtual_currency_and_reward.W

Public Member Functions

- **Message** (**CryptoId** sender, **CryptoId** recipient, **Signature** signature)

Instantiates a new message.

- **Signature getSignature** ()
  Gets the signature.
- **CryptoId getRecipient** ()
  Gets the recipient.
- **CryptoId getSender** ()
  Gets the sender.

Detailed Description

The Class **Message**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 30, 2012

Constructor & Destructor Documentation

overall_implementation.send_receive_beacons.Message.Message (CryptoId *sender*, CryptoId *recipient*, Signature *signature*)

Instantiates a new message.

Parameters:

sender	the sender
recipient	the recipient
signature	the signature

Member Function Documentation

CryptoId overall_implementation.send_receive_beacons.Message.getRecipient ()

Gets the recipient.

Returns:

the recipient

CryptoId overall_implementation.send_receive_beacons.Message.getSender ()

Gets the sender.

Returns:

the sender

Signature overall_implementation.send_receive_beacons.Message.getSignature ()

Gets the signature.

Returns:

the signature

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/send_receive_beacons/**Message.java**

virtual_currency_and_reward.Mint Class Reference

The Class **Mint**.

Public Member Functions

- void **generateCoin** (String nickname)
  Generate coin.
- void **renewCoin** (**Credit** coin)
  Renew coin.

Detailed Description

The Class **Mint**.

Author:

UniUrb

Version:

Jun 27, 2012

Member Function Documentation

void virtual_currency_and_reward.Mint.generateCoin (String *nickname*)

Generate coin.

Parameters:

nickname	the nickname

void virtual_currency_and_reward.Mint.renewCoin (Credit *coin*)

Renew coin.

Parameters:

coin	the coin

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/virtual_currency_and_reward/**Mint.java**

crypto_id.crypto_id_validation.level7.MobilePhoneNumber Class Reference

The Class **MobilePhoneNumber**.

Public Member Functions

- **MobilePhoneNumber** (String phone_number)
  Instantiates a new mobile phone number.
- boolean **isValidPhoneNumber** ()
  Checks if is valid phone number.
- String **getE164PhoneNumber** ()
  Gets the e164 phone number.
- String **getOnlyNumbers** ()
  Gets the only numbers.

Detailed Description

The Class **MobilePhoneNumber**.

Author:

Marzia (Level7)
Paolo (Level7)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

crypto_id.crypto_id_validation.level7.MobilePhoneNumber.MobilePhoneNumber (String *phone_number*)

Instantiates a new mobile phone number.

Parameters:

phone_number	the phone_number

Member Function Documentation

String crypto_id.crypto_id_validation.level7.MobilePhoneNumber.getE164PhoneNumber ()

Gets the e164 phone number.

Returns:

the e164 phone number

String crypto_id.crypto_id_validation.level7.MobilePhoneNumber.getOnlyNumbers ()

Gets the only numbers.

Returns:

the only numbers

boolean crypto_id.crypto_id_validation.level7.MobilePhoneNumber.isValidPhoneNumber ()

Checks if is valid phone number.

Returns:

true, if is valid phone number

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/crypto_id/crypto_id_validation/level7/**MobilePhoneNumber.java**

monetization.MonetizationManager Class Reference

The Class **MonetizationManager**.

Public Member Functions

- void **periodicallyManageMonetization** ()
  Periodically manage monetization.

Detailed Description

The Class **MonetizationManager**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Member Function Documentation

void monetization.MonetizationManager.periodicallyManageMonetization ()

Periodically manage monetization.

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/monetization/**MonetizationManager.java**

crypto_id.NicknameCertificates Class Reference

The Class **NicknameCertificates**.

Public Member Functions

- **NicknameCertificates** (String nickname, **CryptoId** cryptoId, X509CertificateObject firstCertificate)
  Instantiates a new nickname certificates.
- X509CertificateObject[] **getCertificates** ()
  Gets the certificates.
- String **getNickname** ()
  Gets the nickname.
- **CryptoId getCryptoId** ()
  Gets the crypto id.

Detailed Description

The Class **NicknameCertificates**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

crypto_id.NicknameCertificates.NicknameCertificates (String *nickname*, CryptoId *cryptoId*, X509CertificateObject *firstCertificate*)

Instantiates a new nickname certificates.

Parameters:

nickname	the nickname
cryptoId	TODO
firstCertificate	the first certificate

Member Function Documentation

X509CertificateObject [] crypto_id.NicknameCertificates.getCertificates ()

Gets the certificates.

Returns:

the certificates

CryptoId crypto_id.NicknameCertificates.getCryptoId ()

Gets the crypto id.

Returns:

the crypto id

String crypto_id.NicknameCertificates.getNickname ()

Gets the nickname.

Returns:

the nickname

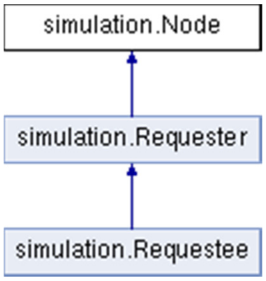
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/crypto_id/**NicknameCertificates.java**

simulation.Node Class Reference

The Class **Node**.

Inheritance diagram for simulation.Node:



Public Member Functions

- void **moveIntoWiFiNeighborhood** (**WiFiNeighborhood** wifiNeighborhood)
  Move into wi fi neighborhood.
- void **leaveWiFiNeighborhood** (**WiFiNeighborhood** wifiNeighborhood)
  Leave wi fi neighborhood.
- void **sendBeacon** (**Message** message)
  Send beacon.

Protected Member Functions

- String **getMac** ()
  Gets the mac.

Detailed Description

The Class **Node**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Member Function Documentation

String simulation.Node.getMac () `[protected]`

Gets the mac.

Returns:

the mac

void simulation.Node.leaveWiFiNeighborhood (WiFiNeighborhood *wifiNeighborhood*)

Leave wi fi neighborhood.

Parameters:

wifiNeighborhood	the wifi neighborhood

void simulation.Node.moveIntoWiFiNeighborhood (WiFiNeighborhood *wifiNeighborhood*)

Move into wi fi neighborhood.

Parameters:

wifiNeighborhood	the wifi neighborhood

void simulation.Node.sendBeacon (Message *message*)

Send beacon.

Parameters:

message	the message

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/simulation/**Node.java**

crypto_id.NodeId Class Reference

The Class **NodeId**.

Public Member Functions

- **NodeId** (**CryptoId** cryptoId, byte[] macAddress)
  Instantiates a new node id.
- byte[] **getBytes** ()
  Gets the bytes of the node id, which is the hash of the public key concatenated with the hase of the MAC address.
- void **printNodeId** ()
  Prints the crypto id.
- boolean **isSameNodeIdAs** (**NodeId** nodeId)
  Checks this node id to another one.

Detailed Description

The Class **NodeId**.

Author:

Daniel Romao (CMS)
Nuno Martins (CMS)
Alfredo matos (CMS)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

crypto_id.NodeId.NodeId (CryptoId *cryptoId*, byte[] *macAddress*)

Instantiates a new node id.

Parameters:

cryptoId	the crypto id
macAddress	the MAC address

Member Function Documentation

byte [] crypto_id.NodeId.getBytes ()

Gets the bytes of the node id, which is the hash of the public key concatenated with the hase of the MAC address.

Returns:

the node id

boolean crypto_id.NodeId.isSameNodeIdAs (NodeId *nodeId*)

Checks this node id to another one.

Parameters:

nodeId	

Returns:

true if same node id

void crypto_id.NodeId.printNodeId ()

Prints the crypto id.

Shows on the console the hexadecimal values of the node id

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/crypto_id/**NodeId.java**

simulation.QRCode Class Reference

The Class **QRCode**.

Public Member Functions

- **QRCode** (Image image)
  Instantiates a new qR code.
- String **getCryptoId** ()
  Gets the crypto id.

Detailed Description

The Class **QRCode**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

simulation.QRCode.QRCode (Image *image*)

Instantiates a new qR code.

Parameters:

image	the image

Member Function Documentation

String simulation.QRCode.getCryptoId ()

Gets the crypto id.

Returns:

the crypto id

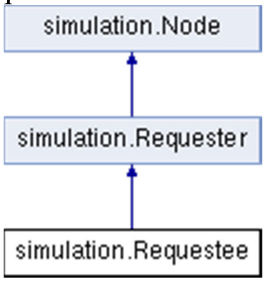
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/simulation/**QRCode.java**

simulation.Requestee Class Reference

The Class **Requestee**.

Inheritance diagram for simulation.Requestee:



Public Member Functions

- **Requestee** (**Bank** defaultBank, **IdValidator** defaultIdValidator, Vector< **Service** > serviceTypes, boolean retailer, **TrustValue** trustValueThresholdForServiceDelivery)
  Instantiates a new requestee.
- boolean **isRetailer** ()
  Checks if is in retailer mode.
- void **setRetailer** (boolean retailing)
  Sets the gateway.
- void **receiveBeacon** (**Message** message)
  Receives a beacon message and process it.
- String **requesteeId** ()
  ***Requestee** id.*
- **TrustValue getTrustValueThresholdForServiceDelivery** ()
  Gets the trust value threshold for service delivery.
- void **setTrustValueThresholdForServiceDelivery** (**TrustValue** trustValueThresholdForServiceDelivery)
  Sets the trust value threshold for service delivery.
- void **addAbleToServeServiceType** (**Service** service)
  Adds the able to serve service type.
- void **removeAbleToServeServiceType** (**Service** service)
  Removes the able to serve service type.
- boolean **canServeServiceType** (**Service** service)
  Can serve service type.

Additional Inherited Members

Detailed Description

The Class **Requestee**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

simulation.Requestee.Requestee (Bank *defaultBank*, IdValidator *defaultIdValidator*, Vector< Service > *serviceTypes*, boolean *retailer*, TrustValue *trustValueThresholdForServiceDelivery*)

Instantiates a new requestee.

Parameters:

defaultBank	the default bank
defaultIdValidator	the default id validator
serviceTypes	the service types
retailer	the retailer
trustValueThresholdForServiceDelivery	the trust value threshold for service delivery

Member Function Documentation

void simulation.Requestee.addAbleToServeServiceType (Service *service*)

Adds the able to serve service type.

Parameters:

service	the service

boolean simulation.Requestee.canServeServiceType (Service *service*)

Can serve service type.

Parameters:

service	the service

Returns:

true, if successful

TrustValue simulation.Requestee.getTrustValueThresholdForServiceDelivery ()

Gets the trust value threshold for service delivery.

Returns:

the trust value threshold for service delivery

boolean simulation.Requestee.isRetailer ()

Checks if is in retailer mode.

Returns:

true, if is in retailer mode

void simulation.Requestee.receiveBeacon (Message *message*)`[virtual]`

Receives a beacon message and process it.

Parameters:

the	message

Reimplemented from **simulation.Requester** (*p.89*).

void simulation.Requestee.removeAbleToServeServiceType (Service *service*)

Removes the able to serve service type.

Parameters:

service	the service

String simulation.Requestee.requesteeId ()

Requestee id.

Returns:

the string

void simulation.Requestee.setRetailer (boolean *retailing*)

Sets the gateway.

Parameters:

retailing	the new retailer

void simulation.Requestee.setTrustValueThresholdForServiceDelivery (TrustValue *trustValueThresholdForServiceDelivery*)

Sets the trust value threshold for service delivery.

Parameters:

trustValueThresholdForServiceDelivery	the new trust value threshold for service delivery

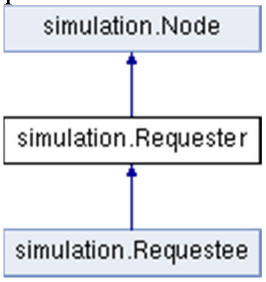
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/simulation/**Requestee.java**

simulation.Requester Class Reference

The Class **Requester**.

Inheritance diagram for simulation.Requester:



Public Member Functions

- **Requester** (**Bank** defaultBank, **IdValidator** defaultIdValidator)
  Instantiates a new requester.
- **NodeId getNodeId** ()
- void **createLocalKeyPair** ()
  Creates the local key pair.
- void **setValidationCertificates** (X509CertificateObject[] newCertificates)
  Sets the validation certificates.
- int **getDispositionalTrust** ()
  Gets the dispositional trust.
- void **setDispositionalTrust** (int dispositionalTrust)
  Sets the dispositional trust and the trust manager.
- **ConfigurationBundle getKeyConfigurationBundlefromDeviceInRange** (**CryptoId cryptoId**, String oneTimePassword)
  Gets the key configuration bundle from device in range.
- void **setLocalKeyPair** (KeyPair localKeyPair)
  Sets the local key pair.
- String **getNickname** ()
  Gets the nickname.
- void **setNickname** (String nickName)
  Sets the nickname.
- String **getPhoneNumber** ()
  Gets the phone number.
- void **setPhoneNumber** (String phoneNumber)
  Sets the phone number.
- boolean **isRunForTheFirstTime** ()
  Checks if is run for the first time.
- boolean **hasValidatedNickname** ()
  Checks for validated nickname.
- boolean **hasDirectInternetAccess** ()
  Checks for direct internet access.
- boolean **hasIndirectInternetAccess** ()
  Checks for indirect internet access.

- void **setHasValidatedNickname** ()
  Sets the has validated nickname.
- void **setHasDirectInternetAccess** (boolean hasDirectInternetAccess)
  Sets the checks for direct internet access.
- void **setHasIndirectInternetAccess** (boolean hasIndirectInternetAccess)
  Sets the checks for indirect internet access.
- boolean **isItClearToSend** ()
  Checks if is it clear to send.
- void **setCredits** (int credits)
  Sets the credits.
- void **receiveBeacon** (**Message** message)
  Receive beacon.
- boolean **isTheNodeARequester** ()
  Checks if is the node a requester.
- **Token computeToken** (**Credit** credit, **TrustValue** trustRequesterInRequestee)
  Compute token.
- void **userAsksToGetInternetAccess** ()
  ***User** asks to get internet access.*
- String **requesterId** ()
  ***Requester** id.*
- void **run** ()
  Called when the ULOOP app is started on a device: the first time, the device is configured ULOOP with the new key pair or a private key moved from a previous ULOOP node owned by the user along with the right crypto-id, the validation of a crypto-id is also done if needed and wanted by the user; then and next times a few periodic processes are run: process of incoming service requests, cooperation evaluation...

Public Attributes

- **CryptoId cryptoId**
  The crypto id.
- **NodeId nodeId**
  The node id.

Static Public Attributes

- static KeyPairGenerator **RSA_KEYPAIR_GENERATOR**
  The rsa keypair generator.

Additional Inherited Members

Detailed Description

The Class **Requester**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

simulation.Requester.Requester (Bank *defaultBank*, IdValidator *defaultIdValidator*)

Instantiates a new requester.

Parameters:

defaultBank	the default bank
defaultIdValidator	the default id validator

Member Function Documentation

Token simulation.Requester.computeToken (Credit *credit*, TrustValue *trustRequesterInRequestee*)

Compute token.

Parameters:

credit	the credit
trustRequesterInRequestee	the trust requester in requestee

Returns:

the token

void simulation.Requester.createLocalKeyPair ()

Creates the local key pair.

int simulation.Requester.getDispositionalTrust ()

Gets the dispositional trust.

Returns:

the dispositional trust

ConfigurationBundle simulation.Requester.getKeyConfigurationBundlefromDeviceInRange (CryptoId *cryptoId*, String *oneTimePassword*)

Gets the key configuration bundle from device in range.

Parameters:

cryptoId	the crypto id
oneTimePassword	the one time password that the user should see on the other device

Returns:
the key configuration bundlefrom device in range

String simulation.Requester.getNickname ()

Gets the nickname.

Returns:
the nickname

NodeId simulation.Requester.getNodeId ()

String simulation.Requester.getPhoneNumber ()

Gets the phone number.

Returns:
the phone number

boolean simulation.Requester.hasDirectInternetAccess ()

Checks for direct internet access.

Returns:
true, if successful

boolean simulation.Requester.hasIndirectInternetAccess ()

Checks for indirect internet access.

Returns:
true, if successful

boolean simulation.Requester.hasValidatedNickname ()

Checks for validated nickname.

Returns:
true, if successful

boolean simulation.Requester.isItClearToSend ()

Checks if is it clear to send.

Returns:
true, if is it clear to send

boolean simulation.Requester.isRunForTheFirstTime ()

Checks if is run for the first time.

Returns:

true, if is run for the first time

boolean simulation.Requester.isTheNodeARequester ()

Checks if is the node a requester.

Returns:

true, if is the node a requester

void simulation.Requester.receiveBeacon (Message *message*) `[virtual]`

Receive beacon.

Parameters:

message	the message

Implements **simulation.Node** (*p.76*).

Reimplemented in **simulation.Requestee** (*p.83*).

String simulation.Requester.requesterId ()

Requester id.

Returns:

the string

void simulation.Requester.run ()

Called when the ULOOP app is started on a device: the first time, the device is configured ULOOP with the new key pair or a private key moved from a previous ULOOP node owned by the user along with the right crypto-id, the validation of a crypto-id is also done if needed and wanted by the user; then and next times a few periodic processes are run: process of incoming service requests, cooperation evaluation...

void simulation.Requester.setCredits (int *credits*)

Sets the credits.

Parameters:

credits	the new credits

void simulation.Requester.setDispositionalTrust (int *dispositionalTrust*)

Sets the dispositional trust and the trust manager.

Parameters:

dispositionalTrust	the new dispositional trust

void simulation.Requester.setHasDirectInternetAccess (boolean *hasDirectInternetAccess*)

Sets the checks for direct internet access.

Parameters:

hasDirectInternetAccess	the new checks for direct internet access

void simulation.Requester.setHasIndirectInternetAccess (boolean *hasIndirectInternetAccess*)

Sets the checks for indirect internet access.

Parameters:

hasIndirectInternetAccess	the new checks for indirect internet access

void simulation.Requester.setHasValidatedNickname ()

Sets the has validated nickname.

void simulation.Requester.setLocalKeyPair (KeyPair *localKeyPair*)

Sets the local key pair.

Parameters:

localKeyPair	the new local key pair

void simulation.Requester.setNickname (String *nickName*)

Sets the nickname.

Parameters:

nickName	the new nickname

void simulation.Requester.setPhoneNumber (String *phoneNumber*)

Sets the phone number.

Parameters:

phoneNumber	the new phone number

void simulation.Requester.setValidationCertificates (X509CertificateObject[] *newCertificates*)

Sets the validation certificates.

Parameters:

newCertificates	the new validation certificates

void simulation.Requester.userAsksToGetInternetAccess ()

User asks to get internet access.

Member Data Documentation

CryptoId simulation.Requester.cryptoId

The crypto id.

NodeId simulation.Requester.nodeId

The node id.

KeyPairGenerator simulation.Requester.RSA_KEYPAIR_GENERATOR`[static]`

The rsa keypair generator.

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/simulation/**Requester.java**

resource_management.ResourceManager Class Reference

The Class **ResourceManager**.

Public Member Functions

- void **processTokenAsRequester** (**Token** token)
  Process token as requester.
- void **processTokenAsRequestee** (**Token** token)
  Process token as requestee.

Detailed Description

The Class **ResourceManager**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 30, 2012

Member Function Documentation

void resource_management.ResourceManager.processTokenAsRequestee (Token *token*)

Process token as requestee.

Parameters:

token	the token

void resource_management.ResourceManager.processTokenAsRequester (Token *token*)

Process token as requester.

Parameters:

token	the token

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/resource_management/**ResourceManager.java**

virtual_currency_and_reward.RewardManager Class Reference

The Class **RewardManager**.

Public Member Functions

- **RewardManager** (**Bank** centralBank)
  Instantiates a new reward manager.
- boolean **creditsEvaluation** (double credits)
  Evaluates whether an amount of credits is deemed acceptable to offer service.
- void **sync** ()
  Perform periodical synchronization with central authority.
- void **transferCreditToWallet** (double credits)
  Transfer credit to wallet.

Detailed Description

The Class **RewardManager**.

Author:

UniUrb

Version:

Jun 30, 2012

Constructor & Destructor Documentation

virtual_currency_and_reward.RewardManager.RewardManager (Bank *centralBank*)

Instantiates a new reward manager.

Parameters:

centralBank	the central bank

Member Function Documentation

boolean virtual_currency_and_reward.RewardManager.creditsEvaluation (double *credits*)

Evaluates whether an amount of credits is deemed acceptable to offer service.

Parameters:

credits	the credits

Returns:

true, if successful

void virtual_currency_and_reward.RewardManager.sync ()

Perform periodical synchronization with central authority.

void virtual_currency_and_reward.RewardManager.transferCreditToWallet (double *credits*)

Transfer credit to wallet.

Parameters:

credits	the credits

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/virtual_currency_and_reward/**RewardManager.java**

overall_implementation.Service Class Reference

The Class **Service**.

Public Member Functions

- **Service** (String type)
  Instantiates a new service.
- String **getType** ()
  Gets the type.

Static Public Attributes

- static final **Service GATEWAY_INTERNET_ACCESS** = new **Service**("Gateway Internet Access")
  The Constant GATEWAY_INTERNET_ACCESS.

Detailed Description

The Class **Service**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

overall_implementation.Service.Service (String *type*)

Instantiates a new service.

Parameters:

type	the type

Member Function Documentation

String overall_implementation.Service.getType ()

Gets the type.

Returns:

the type

Member Data Documentation

final Service overall_implementation.Service.GATEWAY_INTERNET_ACCESS = new Service("Gateway Internet Access") `[static]`

The Constant GATEWAY_INTERNET_ACCESS.

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/**Service.java**

overall_implementation.ServiceLevel Class Reference

The Class **ServiceLevel**.

Public Member Functions

- **ServiceLevel** (int level)
- int **getIntegerValue** ()
  Get the integer value of the service level.

Static Public Attributes

- static final **ServiceLevel DEFAULT_SERVICE_LEVEL** = new **ServiceLevel**(-1)

Detailed Description

The Class **ServiceLevel**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 30, 2012

Constructor & Destructor Documentation

overall_implementation.ServiceLevel.ServiceLevel (int *level*)

Member Function Documentation

int overall_implementation.ServiceLevel.getIntegerValue ()

Get the integer value of the service level.

Returns:

int the integer value of the service level

Member Data Documentation

final ServiceLevel overall_implementation.ServiceLevel.DEFAULT_SERVICE_LEVEL = new ServiceLevel(-1) `[static]`

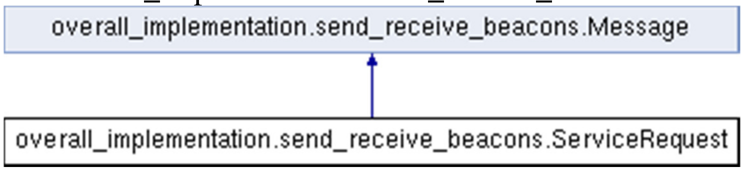
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/**ServiceLevel.java**

overall_implementation.send_receive_beacons.ServiceRequest Class Reference

The Class **ServiceRequest**.

Inheritance diagram for overall_implementation.send_receive_beacons.ServiceRequest:


overall_implementation.send_receive_beacons.Message
overall_implementation.send_receive_beacons.ServiceRequest

Public Member Functions

- **ServiceRequest** (**CryptoId** sender, **CryptoId** recipient, **Signature** signature, **Service** service)
  Instantiates a new service request.
- String **getId** ()
  Gets the id.
- **Service getService** ()
  Gets the service.
- Date **getCreationDate** ()
  Gets the creation date.

Static Public Attributes

- static final **CryptoId BROADCAST_TO_ALL_POTENTIAL_REQUESTEES_IN_RANGE** = new **CryptoId**("Broadcast to all potential requestees in range")
  The Constant BROADCAST_TO_ALL_POTENTIAL_REQUESTEES_IN_RANGE.

Detailed Description

The Class **ServiceRequest**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 30, 2012

Constructor & Destructor Documentation

overall_implementation.send_receive_beacons.ServiceRequest.ServiceRequest (CryptoId *sender*, CryptoId *recipient*, Signature *signature*, Service *service*)

Instantiates a new service request.

Parameters:

sender	the sender

recipient	the recipient
signature	the signature
service	the service

Member Function Documentation

Date overall_implementation.send_receive_beacons.ServiceRequest.getCreationDate ()

Gets the creation date.

Returns:

the creation date

String overall_implementation.send_receive_beacons.ServiceRequest.getId ()

Gets the id.

Returns:

the id

Service overall_implementation.send_receive_beacons.ServiceRequest.getService ()

Gets the service.

Returns:

the service

Member Data Documentation

final CryptoId overall_implementation.send_receive_beacons.ServiceRequest.BROADCAST_TO_ALL_POTENTIAL_REQUESTEES_IN_RANGE = new CryptoId("Broadcast to all potential requestees in range")`[static]`

The Constant BROADCAST_TO_ALL_POTENTIAL_REQUESTEES_IN_RANGE.

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/send_receive_beacons/**ServiceRequest.java**

overall_implementation.send_receive_beacons.ServiceRequestReplies Class Reference

The Class **ServiceRequestReplies**.

Public Member Functions

- **ServiceRequestReplies** (**ServiceRequest** serviceRequest)
  Instantiates a new service request replies.
- void **addServiceRequestReply** (**ServiceRequestReply** serviceRequestReply)
  Adds the service request reply.
- boolean **isProcessed** ()
  Checks if is processed.
- void **setProcessed** (boolean processed)
  Sets the processed.
- **ServiceRequestReply**[] **getServiceRequestReplies** ()
  Gets the service request replies.

Detailed Description

The Class **ServiceRequestReplies**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 30, 2012

Constructor & Destructor Documentation

overall_implementation.send_receive_beacons.ServiceRequestReplies.ServiceRequestReplies (ServiceRequest *serviceRequest*)

Instantiates a new service request replies.

Parameters:

serviceRequest	the service request

Member Function Documentation

void overall_implementation.send_receive_beacons.ServiceRequestReplies.addServiceRequestReply (ServiceRequestReply *serviceRequestReply*)

Adds the service request reply.

Parameters:

serviceRequestReply	the service request reply

ServiceRequestReply [] overall_implementation.send_receive_beacons.ServiceRequestReplies.getServiceRequestReplies ()

Gets the service request replies.

Returns:

the service request replies

boolean overall_implementation.send_receive_beacons.ServiceRequestReplies.isProcessed ()

Checks if is processed.

Returns:

true, if is processed

void overall_implementation.send_receive_beacons.ServiceRequestReplies.setProcessed (boolean *processed*)

Sets the processed.

Parameters:

processed	the new processed

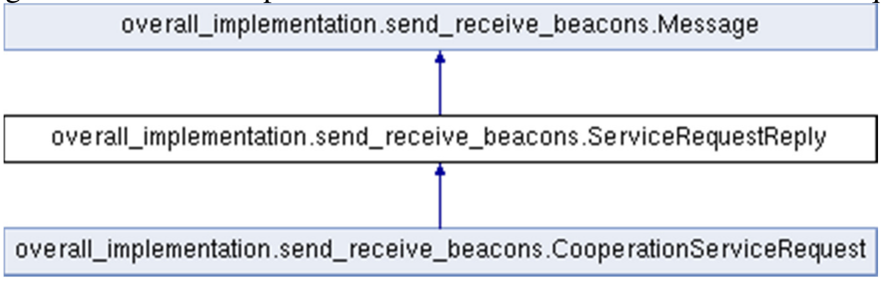
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/send_receive_beacons/**ServiceRequestReplies.java**

overall_implementation.send_receive_beacons.ServiceRequestReply Class Reference

The Class **ServiceRequestReply**.

Inheritance diagram for overall_implementation.send_receive_beacons.ServiceRequestReply:


overall_implementation.send_receive_beacons.Message
overall_implementation.send_receive_beacons.ServiceRequestReply
overall_implementation.send_receive_beacons.CooperationServiceRequest

Public Member Functions

- **ServiceRequestReply** (**CryptoId** sender, **CryptoId** recipient, **Signature** signature, String serviceRequestId)
  Instantiates a new service request reply.
- String **getServiceRequestId** ()
  Gets the service request id.

Detailed Description

The Class **ServiceRequestReply**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 30, 2012

Constructor & Destructor Documentation

overall_implementation.send_receive_beacons.ServiceRequestReply.ServiceRequestReply (CryptoId *sender*, CryptoId *recipient*, Signature *signature*, String *serviceRequestId*)

Instantiates a new service request reply.

Parameters:

sender	the sender
recipient	the recipient
signature	the signature
serviceRequestId	the service request id

Member Function Documentation

String overall_implementation.send_receive_beacons.ServiceRequestReply.getServiceRequestId ()

Gets the service request id.

Returns:

the service request id

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/send_receive_beacons/**ServiceRequestReply.java**

crypto_id.crypto_id_validation.level7.SessionID Class Reference

The Class **SessionID**.

Public Member Functions

- **SessionID** ()
  Instantiates a new session id.
- int **getSessionID** ()
  Gets the session id.

Detailed Description

The Class **SessionID**.

Author:
Marzia (Level7)
Paolo (Level7)

Version:
Jun 27, 2012

Constructor & Destructor Documentation

crypto_id.crypto_id_validation.level7.SessionID.SessionID ()

Instantiates a new session id.

Member Function Documentation

int crypto_id.crypto_id_validation.level7.SessionID.getSessionID ()

Gets the session id.

Returns:
the session id

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/crypto_id/crypto_id_validation/level7/**SessionID.java**

overall_implementation.Signature Class Reference

The Class **Signature**.

Public Member Functions

- **Signature** (String content)
  Instantiates a new signature.
- String **getContent** ()
  Gets the content.

Static Public Attributes

- static final **Signature DUMMY_SIGNATURE** = new **Signature**("Test dummy signature")
  The Constant DUMMY_SIGNATURE.

Detailed Description

The Class **Signature**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

overall_implementation.Signature.Signature (String *content*)

Instantiates a new signature.

Parameters:

content	the content

Member Function Documentation

String overall_implementation.Signature.getContent ()

Gets the content.

Returns:

the content

Member Data Documentation

final Signature overall_implementation.Signature.DUMMY_SIGNATURE = new Signature("Test dummy signature") `[static]`

The Constant DUMMY_SIGNATURE.

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/**Signature.java**

crypto_id.crypto_id_validation.level7.SMS_Server Class Reference

The Class **SMS_Server**.

Public Member Functions

- **SMS_Server** ()
  Instantiates a new sM s_ server.
- **SMS_Server_Listener sendSMS2user** (**MobilePhoneNumber** phoneNumber, String challenge, **SessionID** sessionID)
  Send sm s2user.

Detailed Description

The Class **SMS_Server**.

Author:

Marzia (Level7)
Paolo (Level7)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

crypto_id.crypto_id_validation.level7.SMS_Server.SMS_Server ()

Instantiates a new sM s_ server.

Member Function Documentation

SMS_Server_Listener crypto_id.crypto_id_validation.level7.SMS_Server.sendSMS2user (MobilePhoneNumber *phoneNumber*, String *challenge*, SessionID *sessionID*)

Send sm s2user.

Parameters:

phoneNumber	the phone number
challenge	the challenge
sessionID	the session id

Returns:

the sM s_ server_ listener

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/crypto_id/crypto_id_validation/level7/**SMS_Server.java**

crypto_id.crypto_id_validation.level7.SMS_Server_Listener Class Reference

The listener interface for receiving SMS_Server_ events.

Public Member Functions

- **SMS_Server_Listener** (**SessionID** listenID)
  Instantiates a new sM s_ server_ listener.
- String **getChallengeResponse** ()
  Gets the challenge response.

Detailed Description

The listener interface for receiving SMS_Server_ events.

The class that is interested in processing a SMS_Server_ event implements this interface, and the object created with that class is registered with a component using the component's `addSMS_Server_Listener method. When the SMS_Server_ event occurs, that object's appropriate method is invoked.`

Author:

Marzia (Level7)
Paolo (Level7)

Version:

Jun 27, 2012

See also:

SMS_Server_Event

Constructor & Destructor Documentation

crypto_id.crypto_id_validation.level7.SMS_Server_Listener.SMS_Server_Listener (SessionID *listenID*)

Instantiates a new sM s_ server_ listener.

Parameters:

listenID	the listen id

Member Function Documentation

String crypto_id.crypto_id_validation.level7.SMS_Server_Listener.getChallengeResponse ()

Gets the challenge response.

Returns:

the challenge response

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/crypto_id/crypto_id_validation/level7/**SMS_Server_Listener.java**

social_trust_computation.SocialTrustComputation Class Reference

The Class **SocialTrustComputation** an object of type **TrustTable** it is basically the heart of the way trust levels are computed.

Public Member Functions

- **SocialTrustComputation** (**CryptoId** localNodeCryptoId, **DispositionalTrust** localNodeDispositionalTrustLevel)
  Instantiates a new social trust computation.
- float **computeSocialTrust** (**CryptoId** nodeID, float trustLevel)
  Computes social trust for a specific node and updates its entry in the trusttable It is the core of the trust computation.

Public Attributes

- **TrustTable trustTable**
  The trust table.

Detailed Description

The Class **SocialTrustComputation** an object of type **TrustTable** it is basically the heart of the way trust levels are computed.

Author:

Rute Sofia (ULHT)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

social_trust_computation.SocialTrustComputation.SocialTrustComputation (CryptoId *localNodeCryptoId*, DispositionalTrust *localNodeDispositionalTrustLevel*)

Instantiates a new social trust computation.

Parameters:

localNodeTrustTable	the local node trust table

Member Function Documentation

float social_trust_computation.SocialTrustComputation.computeSocialTrust (CryptoId *nodeID*, float *trustLevel*)

Computes social trust for a specific node and updates its entry in the trusttable It is the core of the trust computation.

Parameters:

nodeID	the node id
trustLevel	the tl

Returns:

the float

Member Data Documentation

TrustTable social_trust_computation.SocialTrustComputation.trustTable

The trust table.

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/social_trust_computation/ **SocialTrustComputation.java**

overall_implementation.Token Class Reference

The Class **Token**.

Public Member Functions

- **Token** (double value)
- double **getValue** ()

Detailed Description

The Class **Token**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

overall_implementation.Token.Token (double *value*)

Member Function Documentation

double overall_implementation.Token.getValue ()

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/**Token.java**

token_computation.TokenComputation Class Reference

Provides the tokens to be used in exchange of resources.

Public Member Functions

- **TokenComputation** (**TrustTable** trustTable)
- float **computeTokens** (**CryptoId** dst, double credits)

Detailed Description

Provides the tokens to be used in exchange of resources.

Author:

Rute Sofia (ULHT)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

token_computation.TokenComputation.TokenComputation (TrustTable *trustTable*)

Member Function Documentation

float token_computation.TokenComputation.computeTokens (CryptoId *dst*, double *credits*)

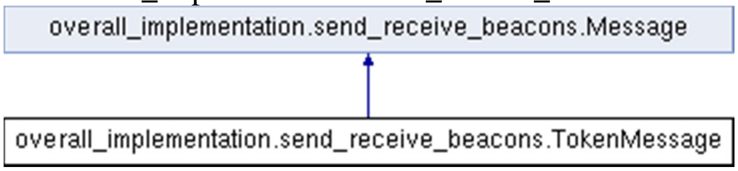
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/token_computation/**TokenComputation.java**

overall_implementation.send_receive_beacons.TokenMessage Class Reference

The Class **TokenMessage**.

Inheritance diagram for overall_implementation.send_receive_beacons.TokenMessage:


overall_implementation.send_receive_beacons.Message
overall_implementation.send_receive_beacons.TokenMessage

Public Member Functions

- **TokenMessage** (**CryptoId** sender, **CryptoId** recipient, **Signature** signature, **Token** token)
  Instantiates a new token message.
- **Token getToken** ()
  Gets the token.

Detailed Description

The Class **TokenMessage**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

overall_implementation.send_receive_beacons.TokenMessage.TokenMessage (CryptoId *sender*, CryptoId *recipient*, Signature *signature*, Token *token*)

Instantiates a new token message.

Parameters:

sender	the sender
recipient	the recipient
signature	the signature
token	the token

Member Function Documentation

Token overall_implementation.send_receive_beacons.TokenMessage.getToken ()

Gets the token.

Returns:

the token

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/send_receive_beacons/**TokenMessage.java**

social_trust_computation.TrustTable Class Reference

The Class **TrustTable** is a linked list of the **TrustTableEntry** class in the node, a trusttable object is initiated (main)

Public Member Functions

- **TrustTable** (**CryptoId** localNodeCryptoId, float dispositionalTrustLevel)
- void **addEntry** (**TrustTableEntry** t)
  Adds the trust table entry.
- **TrustTableEntry getEntry** (**CryptoId** nodeId)
  Gets the entry based on a node's CryptoId.
- float **getTrustLevel** (**CryptoId** nodeId)
  Gets the trust level based on the node CryptoId.
- boolean **updateEntry** (**TrustTableEntry** trustTableEntry)
  Update the trust table entry values.
- void **showTrustTable** ()
  Show all entries in the trust table.
- void **addTrustEntry** (**CryptoId** nodeCryptoId, float trustLevel)
  Adds the trust entry to a trust table.

Detailed Description

The Class **TrustTable** is a linked list of the **TrustTableEntry** class in the node, a trusttable object is initiated (main)

Author:

Rute Sofia (ULHT)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

social_trust_computation.TrustTable.TrustTable (CryptoId *localNodeCryptoId*, float *dispositionalTrustLevel*)

Member Function Documentation

void social_trust_computation.TrustTable.addEntry (TrustTableEntry *t*)

Adds the trust table entry.

void social_trust_computation.TrustTable.addTrustEntry (CryptoId *nodeCryptoId*, float *trustLevel*)

Adds the trust entry to a trust table.

Parameters:

nodeCryptoId	the id
trustLevel	the t

TrustTableEntry social_trust_computation.TrustTable.getEntry (CryptoId *nodeId*)

Gets the entry based on a node's CryptoId.

Parameters:

nodeId	the node id

Returns:

the trust table entry, null if no trust table entry found

float social_trust_computation.TrustTable.getTrustLevel (CryptoId *nodeId*)

Gets the trust level based on the node CryptoId.

Parameters:

nodeId	the node id

Returns:

the trust level, -1 if no CryptoId not found

void social_trust_computation.TrustTable.showTrustTable ()

Show all entries in the trust table.

boolean social_trust_computation.TrustTable.updateEntry (TrustTableEntry *trustTableEntry*)

Update the trust table entry values.

Parameters:

t	the trust table entry

Returns:

true if successful

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/social_trust_computation/**TrustTable.java**

social_trust_computation.TrustTableEntry Class Reference

The Class **TrustTableEntry**.

Public Member Functions

- **TrustTableEntry** (**CryptoId** nodeId, float trustLevel, int ageing)
  Instantiates a new trust table entry.
- float **getTrustLevel** ()
  Gets the trust level.
- **CryptoId getNodeId** ()
  Gets the node id.
- int **getAgeing** ()
  Gets the ageing.
- int **getTimeout** ()
  Gets the timeout.
- void **setTrustLevel** (float trustLevel)
  Sets the trust level.
- void **setAgeing** (int ageing)
  Sets the ageing.

Detailed Description

The Class **TrustTableEntry**.

Author:

Rute Sofia (ULHT)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

social_trust_computation.TrustTableEntry.TrustTableEntry (CryptoId *nodeId*, float *trustLevel*, int *ageing*)

Instantiates a new trust table entry.

Parameters:

nodeId	the node id
trustLevel	the trust level
ageing	the ageing

Member Function Documentation

int social_trust_computation.TrustTableEntry.getAgeing ()

Gets the ageing.

Returns:

the ageing

CryptoId social_trust_computation.TrustTableEntry.getNodeId ()

Gets the node id.

Returns:

the node id

int social_trust_computation.TrustTableEntry.getTimeout ()

Gets the timeout.

Returns:

the timeout

float social_trust_computation.TrustTableEntry.getTrustLevel ()

Gets the trust level.

Returns:

the trust level

void social_trust_computation.TrustTableEntry.setAgeing (int *ageing*)

Sets the ageing.

Parameters:

ageing	the new ageing

void social_trust_computation.TrustTableEntry.setTrustLevel (float *trustLevel*)

Sets the trust level.

Parameters:

trustLevel	the new trust level

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/social_trust_computation/**TrustTableEntry.java**

overall_implementation.TrustValue Class Reference

The Class **TrustValue**.

Public Member Functions

- **TrustValue** (float value)
- float **getValue** ()
- boolean **higherTrustThan** (**TrustValue** otherTrustValue)

Detailed Description

The Class **TrustValue**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

overall_implementation.TrustValue.TrustValue (float *value*)

Member Function Documentation

float overall_implementation.TrustValue.getValue ()

boolean overall_implementation.TrustValue.higherTrustThan (TrustValue *otherTrustValue*)

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/overall_implementation/**TrustValue.java**

simulation.User Class Reference

The Class **User**.

Public Member Functions

- **User** (String nickname, **DispositionalTrust** dispositionalTrust, boolean hasAnotherNode)
  Instantiates a new user.
- void **runUloopClientOn** (**Requester** uloopNode)
  Run uloop client on.
- boolean **hasAlreadyANickName** (boolean answer)
  Checks for already a nick name.
- boolean **hasAlreadyAnotherUloopNode** ()
  Checks for already another uloop node.
- boolean **wantsToValidateNicknameWithProvider** (**IdValidator** idValidator, boolean validate)
  Wants to validate nickname with provider.
- int **getDispositionalTrust** ()
  Gets the dispositional trust.
- String **getNickname** ()
  Gets the nickname.
- void **pushesButtonToBeConnectedToTheInternet** (**Requester** requester)
  Pushes button to be connected to the internet.
- void **movesIntoWiFiNeighborhoodWithDevice** (**WiFiNeighborhood** wifiNetwork, **Requester** node)
  Moves into wi fi neighborhood with device.

Detailed Description

The Class **User**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Constructor & Destructor Documentation

simulation.User.User (String *nickname*, DispositionalTrust *dispositionalTrust*, boolean *hasAnotherNode*)

Instantiates a new user.

Parameters:

nickname	the nickname
dispositionalTrust	the dispositional trust

hasAnotherNode	the has another node

Member Function Documentation

int simulation.User.getDispositionalTrust ()

Gets the dispositional trust.

Returns:

the dispositional trust

String simulation.User.getNickname ()

Gets the nickname.

Returns:

the nickname

boolean simulation.User.hasAlreadyANickName (boolean *answer*)

Checks for already a nick name.

Parameters:

answer	the answer

Returns:

true, if successful

boolean simulation.User.hasAlreadyAnotherUloopNode ()

Checks for already another uloop node.

Returns:

true, if successful

void simulation.User.movesIntoWiFiNeighborhoodWithDevice (WiFiNeighborhood *wifiNetwork*, Requester *node*)

Moves into wi fi neighborhood with device.

Parameters:

wifiNetwork	the wifi network
node	the node

void simulation.User.pushesButtonToBeConnectedToTheInternet (Requester *requester*)

Pushes button to be connected to the internet.

Parameters:

requester	the requester

void simulation.User.runUloopClientOn (Requester *uloopNode*)

Run uloop client on.

Parameters:

uloopNode	the uloop node

boolean simulation.User.wantsToValidateNicknameWithProvider (IdValidator *idValidator*, boolean *validate*)

Wants to validate nickname with provider.

Parameters:

idValidator	the id validator
validate	the validate

Returns:

true, if successful

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/simulation/**User.java**

virtual_currency_and_reward.Wallet Class Reference

The Class **Wallet**.

Public Member Functions

- **Wallet** ()
  Instantiates a new wallet.
- void **payTo** (CryptoId payee, double amount)
  Pay to.
- void **receivePayment** (**CreditTransferOrder** transferOrder)
  Receive payment.
- void **receiveCreditTransferAck** (**CreditTransferAck** transferAck)
  Receive credit transfer ack.
- void **issueAccountCreationRequest** (CryptoId bankId)
  Issue account creation request.
- void **receiveAccountCreationAck** (**AccountCreationAck** creationAck)
  Receive account creation ack.
- void **syncWalletStatus** (**Bank** bank)
  Sync wallet status.
- void **receiveWalletStatusResponse** (**WalletStatusResponse** response)
  Receive wallet status response.
- double **getCredits** ()
  Gets the amount of credits stored inside the wallet.
- void **sellCreditsTo** (CryptoId broker, double amount)
  Sell credits to.
- void **buyCreditsFrom** (CryptoId broker, double amount)
  Buy credits from.
- void **addCredits** (double amount)
- void **decreaseCredits** (double amount)

Detailed Description

The Class **Wallet**.

Author:

UniUrb

Version:

Jun 27, 2012

Constructor & Destructor Documentation

virtual_currency_and_reward.Wallet.Wallet ()

Instantiates a new wallet.

Member Function Documentation

void virtual_currency_and_reward.Wallet.addCredits (double *amount*)

void virtual_currency_and_reward.Wallet.buyCreditsFrom (CryptoId *broker*, double *amount*)

Buy credits from.

Parameters:

broker	the broker
amount	the amount

void virtual_currency_and_reward.Wallet.decreaseCredits (double *amount*)

double virtual_currency_and_reward.Wallet.getCredits ()

Gets the amount of credits stored inside the wallet.

Returns:

the amount of credits

void virtual_currency_and_reward.Wallet.issueAccountCreationRequest (CryptoId *bankId*)

Issue account creation request.

Parameters:

bankId	the bank id

void virtual_currency_and_reward.Wallet.payTo (CryptoId *payee*, double *amount*)

Pay to.

Parameters:

payee	the payee
amount	the amount

void virtual_currency_and_reward.Wallet.receiveAccountCreationAck (AccountCreationAck *creationAck*)

Receive account creation ack.

Parameters:

creationAck	the creation ack

void virtual_currency_and_reward.Wallet.receiveCreditTransferAck (CreditTransferAck *transferAck*)

Receive credit transfer ack.

Parameters:

transferAck	the transfer ack

void virtual_currency_and_reward.Wallet.receivePayment (CreditTransferOrder *transferOrder*)

Receive payment.

Parameters:

transferOrder	the transfer order

void virtual_currency_and_reward.Wallet.receiveWalletStatusResponse (WalletStatusResponse *response*)

Receive wallet status response.

Parameters:

response	the response

void virtual_currency_and_reward.Wallet.sellCreditsTo (CryptoId *broker*, double *amount*)

Sell credits to.

Parameters:

broker	the broker
amount	the amount

void virtual_currency_and_reward.Wallet.syncWalletStatus (Bank *bank*)

Sync wallet status.

Parameters:

bank	the bank reference

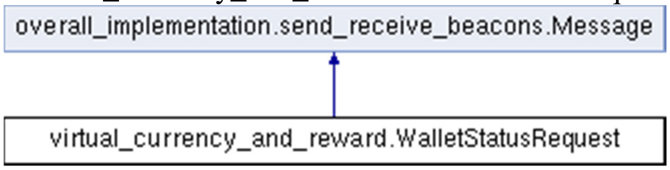
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/virtual_currency_and_reward/**Wallet.java**

virtual_currency_and_reward.WalletStatusRequest Class Reference

The Class **WalletStatusRequest**.

Inheritance diagram for virtual_currency_and_reward.WalletStatusRequest:


overall_implementation.send_receive_beacons.Message
virtual_currency_and_reward.WalletStatusRequest

Public Member Functions

- **WalletStatusRequest** (CryptoId owner, CryptoId bank, Date timestampIssued)
  Instantiates a new wallet status request.
- CryptoId **getAccountOwner** ()
  Gets the account owner.
- CryptoId **getBank** ()
  Gets the bank.
- Date **getTimestampIssued** ()
  Gets the timestamp issued.

Detailed Description

The Class **WalletStatusRequest**.

Author:

UniUrb

Version:

Jun 27, 2012

Constructor & Destructor Documentation

virtual_currency_and_reward.WalletStatusRequest.WalletStatusRequest (CryptoId *owner*, CryptoId *bank*, Date *timestampIssued*)

Instantiates a new wallet status request.

Parameters:

owner	the owner
bank	the bank
timestampIssued	the timestamp issued

Member Function Documentation

CryptoId virtual_currency_and_reward.WalletStatusRequest.getAccountOwner ()

Gets the account owner.

Returns:

the account owner

CryptoId virtual_currency_and_reward.WalletStatusRequest.getBank ()

Gets the bank.

Returns:

the bank

Date virtual_currency_and_reward.WalletStatusRequest.getTimestampIssued ()

Gets the timestamp issued.

Returns:

the timestamp issued

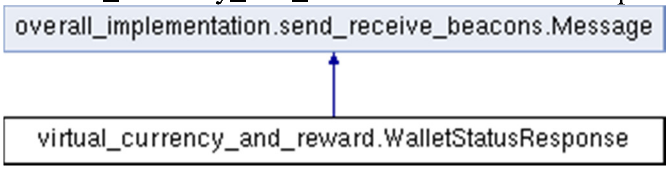
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/virtual_currency_and_reward/**WalletStatusRequest.java**

virtual_currency_and_reward.WalletStatusResponse Class Reference

The Class **WalletStatusResponse**.

Inheritance diagram for virtual_currency_and_reward.WalletStatusResponse:


overall_implementation.send_receive_beacons.Message
virtual_currency_and_reward.WalletStatusResponse

Public Member Functions

- **WalletStatusResponse** (CryptoId owner, CryptoId bank, Date timestampResponse, double convertibleCredits, double unconvertibleCredits, double convertedCredits)
  Instantiates a new wallet status response.
- CryptoId **getAccountOwner** ()
  Gets the account owner.
- CryptoId **getBank** ()
  Gets the bank.
- Date **getTimestampResponded** ()
  Gets the timestamp responded.
- double **getConvertibleCredits** ()
  Gets the convertible credits.
- double **getUnconvertibleCredits** ()
  Gets the unconvertible credits.

Detailed Description

The Class **WalletStatusResponse**.

Author:

UniUrb

Version:

Jun 27, 2012

Constructor & Destructor Documentation

virtual_currency_and_reward.WalletStatusResponse.WalletStatusResponse (CryptoId *owner*, CryptoId *bank*, Date *timestampResponse*, double *convertibleCredits*, double *unconvertibleCredits*, double *convertedCredits*)

Instantiates a new wallet status response.

Parameters:

owner	the owner
bank	the bank

timestampResponse	the timestamp response
convertibleCredits	the convertible credits
unconvertibleCredits	the unconvertible credits
convertedCredits	the converted credits

Member Function Documentation

CryptoId virtual_currency_and_reward.WalletStatusResponse.getAccountOwner ()

Gets the account owner.

Returns:
the account owner

CryptoId virtual_currency_and_reward.WalletStatusResponse.getBank ()

Gets the bank.

Returns:
the bank

double virtual_currency_and_reward.WalletStatusResponse.getConvertibleCredits ()

Gets the convertible credits.

Returns:
the convertible credits

Date virtual_currency_and_reward.WalletStatusResponse.getTimestampResponded ()

Gets the timestamp responded.

Returns:
the timestamp responded

double virtual_currency_and_reward.WalletStatusResponse.getUnconvertibleCredits ()

Gets the unconvertible credits.

Returns:
the unconvertible credits

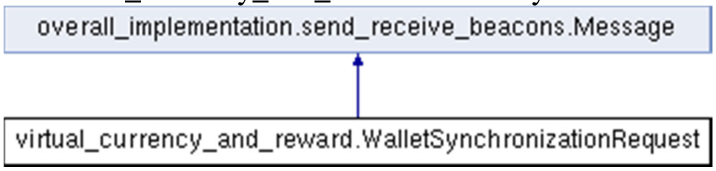
The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/virtual_currency_and_reward/**WalletStatusResponse.java**

virtual_currency_and_reward.WalletSynchronizationRequest Class Reference

The Class **WalletSynchronizationRequest**.

Inheritance diagram for virtual_currency_and_reward.WalletSynchronizationRequest:


overall_implementation.send_receive_beacons.Message
virtual_currency_and_reward.WalletSynchronizationRequest

Public Member Functions

- **WalletSynchronizationRequest** (CryptoId owner, CryptoId bank, Date timestampIssued)
  Instantiates a new wallet synchronization request.
- CryptoId **getAccountOwner** ()
  Gets the account owner.
- CryptoId **getBank** ()
  Gets the bank.
- Date **getTimestampIssued** ()
  Gets the timestamp issued.

Detailed Description

The Class **WalletSynchronizationRequest**.

Author:

UniUrb

Version:

Jun 27, 2012

Constructor & Destructor Documentation

virtual_currency_and_reward.WalletSynchronizationRequest.WalletSynchronizationRequest (CryptoId *owner*, CryptoId *bank*, Date *timestampIssued*)

Instantiates a new wallet synchronization request.

Parameters:

owner	the owner
bank	the bank
timestampIssued	the timestamp issued

Member Function Documentation

CryptoId virtual_currency_and_reward.WalletSynchronizationRequest.getAccountOwner ()

Gets the account owner.

Returns:

the account owner

CryptoId virtual_currency_and_reward.WalletSynchronizationRequest.getBank ()

Gets the bank.

Returns:

the bank

Date virtual_currency_and_reward.WalletSynchronizationRequest.getTimestampIssued ()

Gets the timestamp issued.

Returns:

the timestamp issued

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/virtual_currency_and_reward/**WalletSynchronizationRequest.java**

simulation.WiFiNeighborhood Class Reference

The Class **WiFiNeighborhood**.

Public Member Functions

- void **addNewNodeToWiFiNeighborhood** (**Node** newNode)
  Adds the new node to wi fi neighborhood.
- void **removeNodeFromWiFiNeighborhood** (**Node** leavingNode)
  Removes the node from wi fi neighborhood.
- void **broadcastNewMessage** (**Message** newMessage)
  Broadcast new message.

Detailed Description

The Class **WiFiNeighborhood**.

Author:

Jean-Marc Seigneur (UNIGE)
Carlos Ballester Lafuente (UNIGE)

Version:

Jun 27, 2012

Member Function Documentation

void simulation.WiFiNeighborhood.addNewNodeToWiFiNeighborhood (Node *newNode*)

Adds the new node to wi fi neighborhood.

Parameters:

newNode	the new node

void simulation.WiFiNeighborhood.broadcastNewMessage (Message *newMessage*)

Broadcast new message.

Parameters:

newMessage	the new message

void simulation.WiFiNeighborhood.removeNodeFromWiFiNeighborhood (Node *leavingNode*)

Removes the node from wi fi neighborhood.

Parameters:

leavingNode	the leaving node

The documentation for this class was generated from the following file:

- /Users/jean-marcseigneur/Documents/workspaceindigo/uloopjavaoomodelv2/src/simulation/WiFiNeighborhood.java