

# **Implantation d'un jeu de manipulation pour les systèmes de réputation**

---

## **Rapport du projet annuel de M2 Decim**

**Florian Benavent (numéro étudiant : 21412087)**

**19/02/2015**

## Table des matières

Présentation.....	2
Sujet .....	2
Objectif.....	3
Objectif principaux.....	3
Objectif optionnels.....	4
Madlib .....	4
Sdait .....	5
Étude de l'existant .....	6
Systèmes de réputation .....	6
Représentation graphique automatique de graphe .....	8
Réalisation.....	9
Ajout à madlib.....	9
Modélisation de Graphe .....	9
Réseau et système de réputation .....	12
Modèle de Flot.....	16
Affichage automatique de graphe .....	16
Ajout à Sdait.....	20
Simulation .....	20
Intégration du système de réputation. ....	21
Interface graphique.....	22
Scénario.....	27
Conclusion.....	30
Bibliographie .....	31

## Présentation

Dans le cadre de la seconde année du master DECIM (master informatique spécialité décision et optimisation), un projet annuel est prévu et obligatoire. Ce projet débute au plus tard courant octobre et s'étend jusqu'à fin mars, se fait sous la direction d'un professeur de l'université de Caen Basse-Normandie, sur un sujet spécifique en relation avec le cursus.

Dans le cadre de mon projet, j'ai été sous la direction de M. Gregory Bonnet sur le sujet « Implantation d'un jeu de manipulation pour les systèmes de réputation ». Durant toute la durée, des suivis réguliers, toutes les deux semaines, trois en cas de période d'examen ou de vacance, ont été réalisés avec mon encadrant pour vérifier l'avancée du projet.

## Sujet

Les systèmes de réputation servent à évaluer de manière centralisée ou décentralisée la fiabilité des agents au sein d'une communauté. Dans un tel système, chaque agent évalue les autres agents dont il a connaissance. Ces évaluations locales sont appelées la confiance d'un agent en un autre agent. Ces valeurs sont alors transmises sous forme de témoignages et agrégés pour former des évaluations collectives, que l'on appelle alors réputation. S'ils sont assez efficaces pour détecter les agents non fiables isolés, des collectifs d'agents malhonnêtes (appelés collusions) peuvent tenter de manipuler le système.

Ces systèmes sont très utilisés dans les systèmes d'e-commerce, tel eBay, dans lequel il permet de trouver et de surveiller des agents qui pourraient potentiellement mentir, sur ses produits, ou voler, en n'envoyant pas le produit à l'acheteur, par exemple. Ces systèmes peuvent également être mis en place dans d'autres réseaux multiagents, tels les systèmes d'échanges de fichiers Pair à Pair, pour identifier les agents malveillants qui par exemple transmettent des virus ou se contentent de prendre des fichiers, sans en envoyer.

L'objectif de ce projet est d'implanter sur une plate-forme générique développée au sein de l'équipe MAD un jeu dans lequel un joueur humain devra tenter de manipuler un système de réputation à l'aide de 3 techniques :

- La promotion, où un agent ment, en surévaluant consciemment la confiance dans un autre agent.
- la diffamation, où cette fois-ci l'agent sous-évalue volontairement la confiance dans un autre agent.

- l'attaque Sybil, dans lequel l'utilisateur ajoute de nombreux « faux agents » au système, qu'il contrôle et s'en sert pour se promouvoir tout en diffamant les autres agents.

Dans ce jeu, les agents interagissent en fonction de leur réputation et d'un certain nombre de caractéristiques à définir (type d'interactions, qualité de ces dernières et capacité de montée en charge) et le joueur humain se verra attribué un score en fonction de sa capacité à manipuler le système.

## Objectif

L'objectif du projet se décompose en deux catégories, les objectifs principaux, permettant de répondre aux sujets, et les objectifs optionnels. Ces derniers ont été ajoutés une fois les objectifs principaux atteints, pour compléter l'application au vu du temps restant.

Le langage de programmation utilisé est le Java, et l'accent est mis sur la généricité.

### Objectif principaux

Le principal objectif est d'intégrer à la librairie Madlib un modèle générique capable de représenter les relations entre divers agents, ainsi que deux systèmes de réputation. Les deux systèmes de réputation initialement choisis correspondent à Beta Reputation et EigenTrust. Ces 2 systèmes sont présentés plus en détail dans la section d'étude de l'existant (cf : [étude de l'existant : système de réputation](#)), et, pour l'algorithme en lui-même, dans la partie Réalisation (cf : [BetaReputation](#), [EigenTrust](#)).

Le second objectif est la réalisation d'une interface graphique, permettant de visualiser et de configurer les relations entre une multitude d'agents. Ces agents et leurs relations seront représentés par un graphe orienté, où chaque nœud correspond à un agent du système, et la présence d'un arc correspondra à une relation entre un agent et un second. Cette interface sera créée en utilisant la librairie Sdait.

## Objectif optionnels

Divers objectifs secondaires ont été envisagés, comme des pistes d'amélioration du projet une fois les objectifs principaux remplis. Ces objectifs sont classés par ordre de préférence, selon l'apport concret vis-à-vis du sujet.

Le premier de ces objectifs est un système d'affichage automatique de graphe, pour permettre d'organiser l'interface graphique de manière claire et compréhensible. Cette solution sera à implémenter directement dans la librairie madlib, pour permettre une réutilisation au sein d'autres projets.

En second point, il y a la possibilité de continuer à développer la librairie Madlib par l'introduction de nouveaux systèmes de réputation telle la méthode Flow Trust, présenté dans également à la fois dans la section étude de l'existant (cf : [étude de l'existant : système de réputation](#)) et dans la section réalisation (cf : [FlowTrust](#)).

Enfin, il y a la possibilité d'intégrer une notion de temps et un dynamisme dans la simulation. Les agents pourront alors interagir entre eux, selon un comportement prédéfini, sur un modèle d'échange de service. L'application permettra alors de visualiser l'évolution des confiances et des réputations de chaque agent au fur et à mesure de leurs interactions.

## Madlib

Madlib est une librairie d'intelligence artificielle, créé au sein de l'équipe Modèles, Agents et Décision (abrégé MAD) du laboratoire GREYC de l'université de Caen Basse-Normandie.

Cette librairie a pour objectif de mettre à disposition à un large public, tels des chercheurs, des étudiants ou encore des industriels, des méthodes d'intelligence artificielle facilement adaptable et utilisable pour tout type de problèmes.

Les méthodes actuellement en place dans cette librairie tournent autour des thématiques de l'équipe MAD, et principalement les processus de décision markovien. À long terme, cette librairie n'as pas vocation à être restreinte à certains domaines, mais de proposer aux utilisateurs d'intégrer leurs propres solutions, le tout étant placé sous une licence open source, c'est-à-dire libre d'accès et d'utilisation sous certaines conditions.

Durant ce projet, nous allons intégrer les composants de modélisation et de gestion de système de réputation à madlib.

## Sdait

Sdait est une librairie graphique, créé également au sein de l'équipe Modèles, Agents et Décision (abrégé MAD) du laboratoire GREYC de l'université de Caen Basse-Normandie.

Cette librairie, bien que portée sous licence open source, est destinée à un public plus restreint, qui se constitue actuellement de l'équipe MAD et d'étudiants liés à cette équipe au travers de projet ou de stage.

L'objectif principal est d'offrir la capacité de créer rapidement des applications de démonstration pour des problèmes d'intelligence artificielle. Dans cette optique, la librairie dispose de toute une série de fonctionnalité pour représenter des problèmes ou des processus récurant dans ce genre d'applications, tels l'affichage de graphe, ou le questionnement à l'utilisateur.

Durant ce projet, nous utiliserons Sdait, à la fois pour des raisons d'efficacité, mais également d'unicité vis-à-vis des autres applications de démonstration liée à madlib. Si nécessaire, nous pourrions compléter les fonctionnalités de cette librairie.

## Étude de l'existant

L'étude de l'existant fût réalisée en deux temps.

Au début du projet, il a fallu étudier les deux systèmes de réputation au centre du projet, mais également le type d'utilisation de ces systèmes, pour pouvoir proposer une conception la plus générique possible.

Une fois les objectifs principaux atteints, il fallut étudier d'une part les études sur la représentation graphique automatique de graphe, et d'autre part, d'autres systèmes de réputation, pour les intégrer à leurs tours.

### Systèmes de réputation

Les deux premiers articles étudiés correspondent à la présentation d'Eigen Trust [1] et à celle de Beta Reputation [2].

Pour le premier, traitant d'Eigen Trust, l'article propose directement plusieurs algorithmes, en expliquant en détail quelles sont les différences et la raison de ces ajouts. Nous obtenons ainsi une version basique, une version distribuée et une seconde version distribuée plus lourde, mais ayant de meilleures capacités de lutte contre des agents malveillants. La fin de l'article est consacrée à l'évaluation de l'algorithme sur un problème basique de téléchargements dans un réseau Pair à Pair. L'algorithme propre d'EigenTrust est présenté dans la section réalisation (cf : [Système de réputation : EigenTrust](#))

Dans le second article, pour Beta Reputation, seuls le principe général et les diverses fonctions de calculs sont représentés. L'algorithme en lui-même est dépendant du problème, principalement pour l'ordre de traitement des agents. L'article nous explique comment évaluer un agent, combiner le retour de plusieurs agents... La fin de l'article est consacrée à l'étude des comportements de chaque fonction, selon certains critères. L'article se voulant général, il n'y a pas de cas concret présenté. L'algorithme propre de Beta Reputation est présenté dans la section réalisation (cf : [Système de réputation : Beta Reputation](#))

Pour généraliser la vision ainsi formée des modèles de réputation, j'ai étudié 3 articles supplémentaires.

Le premier, rédigé par J. Sabater et C. Sierra [3], propose une catégorisation des divers composants intervenants dans un système de réputation, tels les informations directes/indirectes, le préjudice, les

types d'échanges d'informations... Dans un second temps, l'article présente divers systèmes de réputation, sans toutefois en présenter le fonctionnement.

Le second, de L. Mui, M. Mohtashemi et A. Halberstadt [4], présente d'un point de vue générique certaines notions appartenant aux systèmes de réputation, avant de présenter leur propre système. De cette manière, après un bref rappel des notions de confiance et réputation basée sur des articles existants, les auteurs s'interrogent sur la notion de réciprocité dans un réseau de réputation, expliquant que lorsqu'elle est présente, la réputation la confiance et les actions réciproques forment un cycle, car l'augmentation ou la diminution de l'un se répercutera sur le suivant, qui se répercutera à son tour... LE modèle présenté par la suite prend en compte de cette réciprocité, mais en faisant varier son impact (au travers d'un facteur allant de 0, pas de réciprocité, à 1, une réciprocité totale). Le parcours du réseau ici se fait, pour chaque pair d'agents a et b, par les plus courts chemins disjoints entre a et b, méthode qui sera reprise dans le projet pour Beta Reputation.

Enfin, le troisième article, de T. Vallée, G. Bonnet et F. Bourdon, présente un modèle générique d'échanges de services, considérés comme les problèmes de bases des systèmes de réputation, puis propose une analogie avec le problème du bandit manchot, et conclut sur une étude empirique basée sur les systèmes EigenTrust et Beta Reputation.

Ces trois articles permettent de dégager deux informations primordiales pour la modélisation des problèmes et système de réputation au sein de madlib, et joueront donc un rôle dans la conception, présenté dans la section suivante du rapport.

D'une part, ces divers systèmes de réputations nous permettent d'établir que ces derniers ont uniquement besoin, pour chaque agent, du nombre d'interactions eu avec chaque autre agent, ainsi que l'évaluation de ces derniers. Cette évaluation semble cependant plus problématique pour établir un modèle générique, car là où certains utilisent des valeurs numériques bornées, d'autres utilisent des valeurs qualitatives, pouvant se résumer à la différence bien/mal.

D'autre part, l'ensemble des problèmes de réputation étudiés dans la littérature, e-commerce, réseaux Pair à Pair... se résument bien généralement en problèmes d'échanges de services.

Dans la dernière phase du projet, consacré à la réalisation d'objectifs secondaires, j'ai étudié deux nouveaux systèmes de réputations.

Le premier, intitulé FlowTrust, est présenté par G.Wang et J.Wu [6]. Dans cet article, les 2 auteurs font le rapprochement entre un réseau de confiance et un système de flots. De cette similarité, ils proposent une méthode basée sur le calcul d'un flot maximum, intitulée Basic FlowTrust Algorithm (abrégé B-FlowTrust). Il reproche cependant à cette méthode le fait que les nombreux calculs de flots maximums ont un coût très élevé et propose donc 2 nouvelles méthodes, le Edge Processing-based FlowTrust Algorithm (abrégé E-FlowTrust), offrant une approximation du B-FlowTrust à un coût moindre, et le



Graph Simplification-based FlowTrust Algorithm (abrégé G-FlowTrust), introduisant la notion de la simplification de graphe vis-à-vis de la méthode initiale. L'algorithme B-FlowTrust est présenté dans la section réalisation (cf : [Système de réputation : Flow Trust](#))

Le second, intitulé TrustWalker, est présenté par M. Jamali et M. Ester [7]. Cette méthode se base le principe de la marche aléatoire et est de ce faite très similaire à l'EigenTrust. Du fait de cette ressemblance, il a été décidé de se concentrer sur des méthodes tel que FlowTrust pour apporter plus de diversité au projet.

## Représentation graphique automatique de graphe

La représentation des données sous forme de graphe est extrêmement répandue, la représentation graphique de ces derniers est cependant un problème bien plus complexe.

Une méthode de base a été proposée par P. Eades, en 1984 dans l'article « *A heuristic for graph drawing* ». Dans le cadre de ce projet, j'ai pu étudier 3 articles traitant de ce problème, basé sur le modèle d'Eades.

Les articles de S. Karouach et B. Dousset [8] et T. M. J. Fruchterman et E. M. Reingold [9] commencent par reprendre en détail la méthode de P. Eades, basé elle-même sur la loi de Hooke, chargé de représenté le comportement élastique d'un métal soumis à une force de pression.

Dans cet algorithme, les nœuds trop proches appliquent une force de répulsion les uns aux autres, tandis que les arcs appliquent une force d'attraction aux nœuds qu'ils connectent. Cet ensemble est soumis à une « température », une variable diminuant avec le temps qui, plus elle est élevée, favorise la force de répulsion vis-à-vis de celle d'attraction.

Ces 2 articles, ainsi que le 3e étudié, également S. Karouach et B. Dousset [10], expliquent que l'état initial du système, généré aléatoirement dans la version initiale, a un très fort impact sur les résultats. Au travers des 3 articles, les auteurs présentent alors un ensemble de méthode pour générer un premier classement des nœuds du graphe, accompagné de démonstration, basé sur des clusters, des diagonales,...

Dans notre projet nous nous contenterons de la version simple.

## Réalisation

En se basant sur l'étude de l'existant réalisé, nous avons réalisé une phase de conception dans lequel l'objectif principal était de préparer les différents composants de l'application de manière générique et indépendante, pour pouvoir les ajouter aux librairies madlib et Sdait si cela est pertinent, ou dans l'application de simulation en elle-même.

Le travail c'est ensuite articulé autour de trois pôles, le premier concernant tous les ajouts à la librairie madlib, le second correspondant aux ajouts à la librairie Sdait et le dernier au logiciel de simulation en lui-même, intégrant les fonctionnalités ajoutées aux deux librairies.

### Ajout à madlib

Les ajouts à madlib se divisent en trois points. Le premier, concerne l'ajout d'un modèle de graphe, et de diverses fonctionnalités autour, pour représenter par la suite efficacement un réseau de réputation. La seconde catégorie correspond à la modélisation du réseau de réputation et des systèmes de réputation. La dernière section, venue plus tard dans le projet, concerne la modélisation des flots, et les fonctionnalités nécessaires pour le système de réputation FlowTrust.

Durant la dernière phase du projet, une dernière fonctionnalité a été intégrée, concernant le positionnement automatique des graphes.

### Modélisation de Graphe

L'ajout d'un modèle de graphe à madlib fut le premier point de mon travail, car il sert de base pour l'intégralité du projet. Pour assurer la généricité de madlib, nous avons implémenté à la fois un modèle de graphe orienté et non orienté. D'autre part, nous avons fait la distinction entre graphe simple et dynamique, et pour finir nous avons intégré la possibilité d'ajouter des données aux nœuds, aux arêtes ou aux deux.

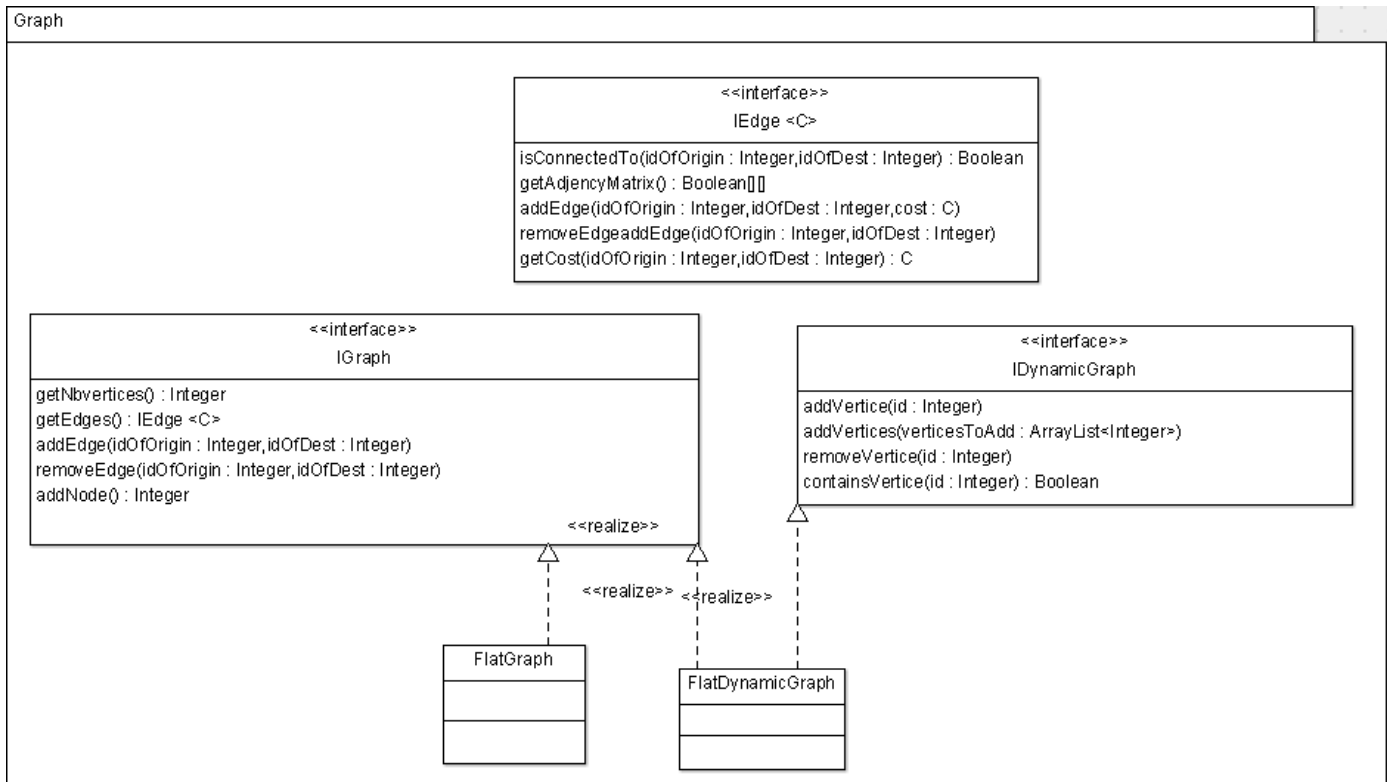
La distinction entre graphe orienté et non orienté se fait en totale transparence du reste, via à une structure dédiée à la gestion des arêtes, utilisé via une interface, dont seule l'implémentation de cette dernière sait si nous sommes dans un cas orienté ou non orienté.

D'autre part nous avons donc distingué les graphes simples et les graphes dynamiques.

Dans le premier type, les sommets sont numérotés de 0 à  $n - 1$ . On peut ajouter un nœud ou ajouté/supprimé une arête. Il est cependant impossible de supprimer un nœud. Ce modèle est donc destiné à des cas où l'on a besoin d'un graphe, mais où les nœuds sont fixes, comme un problème de maximisation de flot...

Dans le type dynamique, chaque sommet possède un identifiant qui lui est propre. Il est alors possible d'ajouter/supprimer aussi bien un nœud qu'une arête. La nécessité de stocker ces identifiants et les vérifications liées à l'existence ou non d'un nœud entraîne un coup supplémentaire, mais qui est alors inévitable. Ce modèle est donc plus orienté pour des problèmes modélisant par exemple un réseau pair-à-pair.

La distinction graphe simple et dynamique sont gérés par le respect, ou non, de l'interface IDynamicGraph, comme on peut le voir dans le diagramme ci-contre.



La dernière division réalisée permet de distinguer 4 types de graphes supplémentaires.

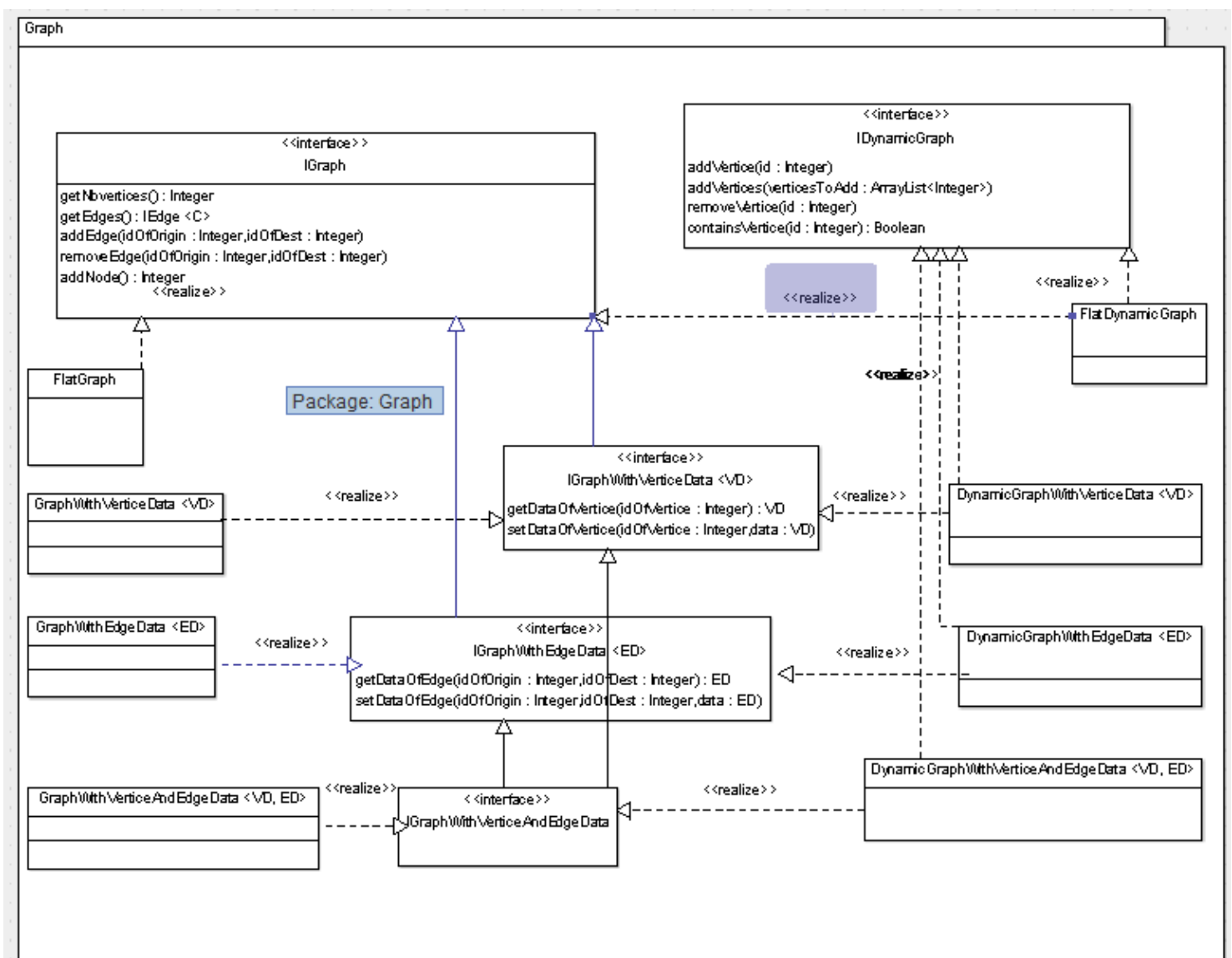
Le premier ne possède aucune information associée aux nœuds, et aucune autre information qu'un potentiel coût, associé aux arêtes. Ce type peut, par exemple, être utilisé pour modéliser des problèmes du plus court chemin

Le second possède des informations sur ses arêtes, en plus du coût. Ces informations n'ont aucune contrainte et peuvent donc représenter ce que l'on souhaite, par exemple des valeurs de flots et de planchers pour les problèmes de flots.

Le troisième type possède des informations uniquement sur le nœud. Il peut servir à représenter un réseau d'utilisateurs ou chaque nœud possède le détail de l'utilisateur, et les liens représentent simplement les personnes connues.

Enfin le dernier type comporte à la fois des informations génériques sur les nœuds et les arête. Concrètement, il s'agit de notre modèle de réseau de réputation. Chaque nœud correspond à un agent, possédant certaines informations telles que sa valeur globale de réputation, et les arêtes correspondent aux relations entre agents, en conservant par exemple les confiances locales ou le nombre d'interactions.

En combinant toutes ces possibilités, nous arrivons à 8 types possibles de graphes, organisés autour de 5 interfaces. Le diagramme ci-dessus représente les relations entre chacun de ces types/classes et les interfaces.



En plus des différents types de graphe, nous avons intégré deux implémentations du gestionnaire d'arêtes directement dans madlib.

Le premier, que l'on peut considérer comme « standard », conserve les 2 agents reliés avec le coût associé. Avec cette représentation nous pouvons soit choisir une version orientée, soit non orientée.

La seconde prend la forme d'une matrice d'adjacente. Bien que plus lourde, car complète, elle peut être utile selon les algorithmes employés (en évitant d'avoir à la générer plusieurs fois).

En termes d'algorithmes, deux fonctionnalités ont été ajoutées, requis pour le projet. Le premier est l'algorithme de recherche de plus court chemin Tarjan, Le deuxième, basé sur Tarjan, permet de récupérer l'ensemble des plus courts chemins disjoints entre deux points, en supprimant au fur et à mesure les nœuds des chemins identifiés.

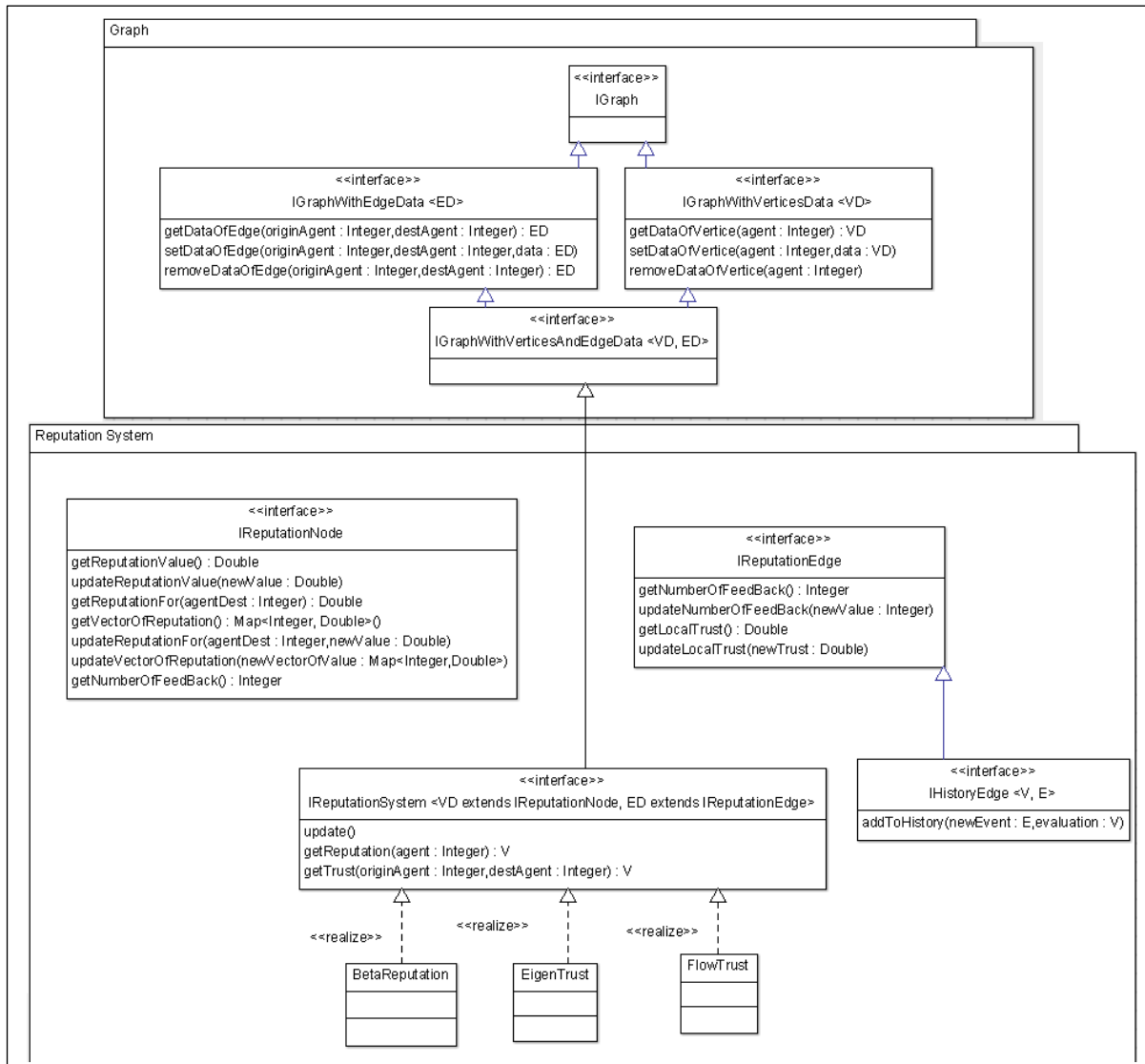
## Réseau et système de réputation

La section système de réputation se forme comme une surcouche à la section graphe.

Un réseau de réputation est alors représenté sous la forme d'un graphe avec comme particularité que les nœuds et les arêtes possèdent des informations relatives au système. Un nœud intègre ainsi sa propre valeur de réputation, et un vecteur de confiance représentant la confiance qu'il a envers chaque autre agent, après agrégation des témoignages qu'il a reçue. D'autre part, les arêtes, représentant les relations entre les agents, possèdent la valeur locale de confiance, le nombre de feedbacks et l'historique des interactions de l'agent d'origine de l'arc, pour l'agent ciblé par l'arc.

Avec l'historique des actions sur les arcs, nous avons ainsi préparé l'ajout d'un côté dynamique pour la simulation, bien que la priorité fût mise sur d'autres fonctionnalités finalement.

Les systèmes de réputations, se basant sur le modèle de réseau présenté, sont totalement configurés à leur instanciation, dans le but d'offrir une utilisation totalement autonome et transparente. Les systèmes de réputation sont définis par la hiérarchie suivante :



Actuellement, nous avons incorporé à la librairie madlib 3 systèmes de réputation : Beta Reputation, EigenTrust et FlowTrust, dont le détail est présenté ci- dessous.

## Beta Reputation

Comme présenté dans la section [étude de l'existant associé](#), le système Beta Reputation intégré est basé sur l'article de R. Ismail et A. Jøsang[2]. Dans cet article, il présente les 3 fonctions principales, la fonction de réduction, d'agrégation et de calcul de la réputation. Le parcours des agents du système n'est cependant pas dicté, car il dépendant de l'application. Nous avons donc formé, à partir des 3 fonctions données, notre version de Beta Reputation correspondant à :

Pour chaque agent X du système  
  Pour chaque autre agent Y du système  
    C <- Calculer tous les plus courts chemins disjoints de X à Y  
    Pour chaque chemin de C  
      Remonter le chemin en appliquant la fonction de réduction  
    Agréger les résultats de tous les chemins de C  
    Calculer la valeur de réputation de Y pour X

Dans ces diverses formules, les variables r et s correspondent aux feedbacks positifs et négatifs. Les variables X, Y et T correspondent à des agents.

- La fonction de réduction :

$$r_T^{X:Y} = \frac{2r_Y^X r_T^Y}{(s_Y^X + 2)(r_T^Y + s_T^Y + 2) + 2r_Y^X},$$
$$s_T^{X:Y} = \frac{2r_Y^X s_T^Y}{(s_Y^X + 2)(r_T^Y + s_T^Y + 2) + 2r_Y^X},$$

- La fonction d'agrégation:

1.  $r_T^{X,Y} = r_T^X + r_T^Y$
2.  $s_T^{X,Y} = s_T^X + s_T^Y$

- La fonction de calcul de réputation :

$$\text{Rep}(r_T^X, s_T^X) = \left( E(\varphi(p | r_T^X, s_T^X)) - 0.5 \right) \cdot 2 = \frac{r_T^X - s_T^X}{r_T^X + s_T^X + 2}.$$

### *Eigen Trust*

Comme présenté dans la section [étude de l'existant associé](#), le système d'EigenTrust est basé sur l'article de S.D. Kamvar, M.T. Schlosser et H. Garcia-Molina. Dans cet article il présente plusieurs versions, dont une seule, la version basique, en centralisé. Il s'agit donc de la version que nous avons choisi d'implémenter et qui est présentée par l'algorithme suivant :

```
 $\vec{t}^{(0)} = \vec{p};$   
repeat  
|  $\vec{t}^{(k+1)} = C^T \vec{t}^{(k)};$   
|  $\vec{t}^{(k+1)} = (1 - \alpha) \vec{t}^{(k+1)} + \alpha \vec{p};$   
|  $\delta = ||\vec{t}^{(k+1)} - \vec{t}^{(k)}||;$   
until  $\delta < \epsilon;$ 
```

Ou chaque variable est définie tel que :

- Le vecteur  $\vec{t}$  correspond au vecteur de confiance globale, mise à jour à chaque itération.
- Le vecteur  $\vec{p}$  correspond au vecteur initial de confiance, utilisé pour représenter les agents dont on a totalement confiance, tels les administrateurs du réseau.
- La matrice  $C$  représente la confiance de chaque agent pour tous les autres agents.
- Le paramètre  $\alpha$  permet de favoriser les agents dont on a confiance initialement, permettant de mettre plus rapidement de côté les agents malveillants et ainsi offrir une convergence plus rapide.
- Le paramètre  $\epsilon$ , le critère d'arrêt, représente la précision minimale que l'on souhaite atteindre.



## FlowTrust

Pour ce système supplémentaire, et optionnel, les 3 versions présentées dans l'article de G.Wang et J.Wu [6] disposent d'une définition complète. Seule la version Basic est actuellement intégrée au projet, les autres seront peut-être intégrés avant la fin du projet. L'algorithme Basic est de la forme suivante :

---

**Algorithm 1** Basic FlowTrust Algorithm from the perspective of *Maximum Flow of Trust Value (MaxT)*

---

- 1: **Input:** A trusted graph  $G = (V, E)$ ,  $u, v \in V$ ,
  - 2:   and the trust value  $t(i, j), \forall (i, j) \in E$
  - 3: **Output:** The normalized trust value  $T_b$
  - 4: Let  $MaxT$  be the maximum flow of trust value of  $G$
  - 5:   from  $u$  to  $v$  with  $t(i, j)$  as the edge capacity
  - 6: Let  $MaxP$  be the maximum number of edge-disjoint
  - 7:   paths of  $G$  from  $u$  to  $v$
  - 8: Return  $T_b = \frac{MaxT}{MaxP}$
- 

## Modèle de Flot

La section de flots ajoutés à madlib est principalement composée de classes permettant la modélisation de flots. Cela permet de former des flots possédants :

- Simplement une capacité maximum à ne pas dépasser.
- Une capacité maximum à ne pas dépasser et une valeur minimale à atteindre.

En prévision de l'implémentation du FlowTrust, un système de réputation étudié dans les objectifs optionnels, l'algorithme d'Edmonds-Karp a été ajouté. Ce dernier permet de calculer un flot de valeur maximum pour un réseau donné.

## Affichage automatique de graphe

Cette fonctionnalité, classée dans les objectifs secondaires du projet, est destinée à permettre une répartition claire et lisible d'un graphe dans notre fenêtre.

Basée sur les articles étudiés et présentés dans la section étude de l'existant (cf : [Représentation graphique automatique de graphe](#)), la méthode utilisée est basée sur le principe d'attraction et de répulsion. Autrement dit, deux nœuds possédant un arc les reliant se rapprocheront, mais deux nœuds trop proches se repousseront. Ce système sera lancé pendant un certain nombre d'itérations, configuré par l'utilisateur, pour permettre d'obtenir une représentation la plus claire possible, dans un temps jugé acceptable par l'utilisateur.

L'algorithme générique est défini dans l'article de S. Karouach et B. Dousset[8] ainsi :

```

Initialisation : Positionnement aléatoire des sommets
Tant que  $k < Max\_iter$  faire
  Pour tout sommet  $u$  faire
    Pour tout sommet  $v \neq u$  faire
      Si  $distance(u ; v) < seuil$  alors
        Calcul de la force de répulsion entre  $u$  et  $v$ 
    Pour toute arête  $(u, v)$  faire
      Calcul de la force d'attraction entre  $u$  et  $v$ 
    Pour tout sommet  $u$  faire
      Cumul des forces
      Déplacement de  $u$  en fonction de la température globale
      Diminuer la température globale

```

Les fonctions d'attraction (notée  $f_a$ ) et de répulsion (notée  $f_r$ ) sont présentées également tel que :

$$f_r(v_i, v_j) = -k^2 / d_{ij}^{\alpha_r}$$

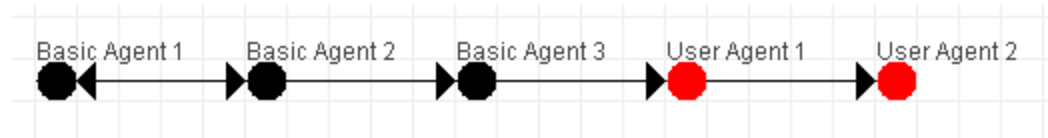
$$f_a(v_i, v_j) = \beta_{ij} d_{ij}^{\alpha_a} / k$$

La variable  $d_{ij}$  correspond à la distance entre les sommets  $i$  et  $j$  du graphe, tandis que  $\beta_{ij}$  correspond au poids de l'arc  $(i, j)$ . Les valeurs  $\alpha_r$  (le facteur de répulsion) et  $\alpha_a$  (le facteur d'attraction) sont 2 variables dépendant à la fois du problème et de ce que souhaite l'utilisateur.  $K$  correspond quant à lui à la racine carrée de l'aire de la fenêtre divisée par le nombre de nœuds dans le graphe.

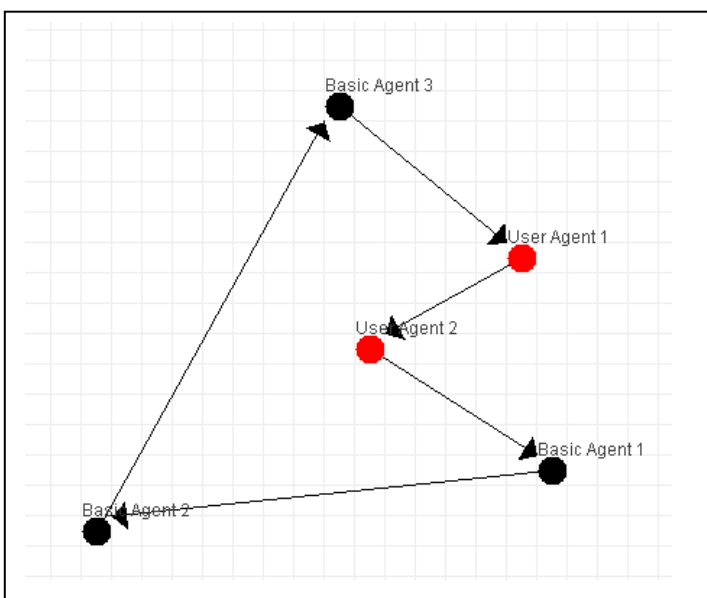
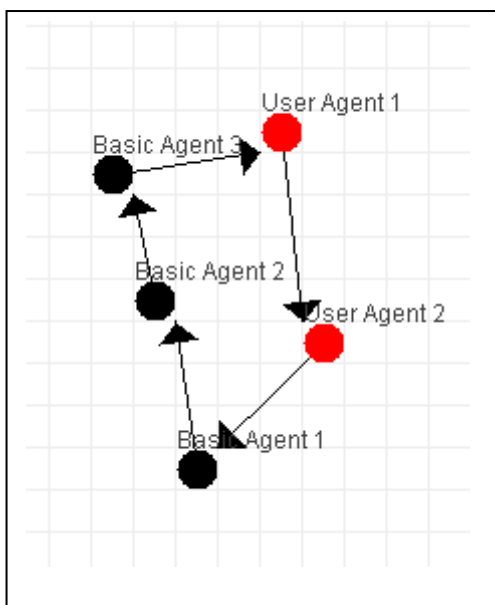
Les résultats sont satisfaisants, bien qu'ajoutés une phase de pré traitement, comme indiqué dans les articles pourraient potentiellement améliorer les choses, il a été jugé préférable de s'arrêter ici pour cette fonctionnalité, et se concentrer sur de nouveaux systèmes de réputation, tel FlowTrust.

Voici des exemples de résultats obtenus via les scénarios 1 et 2. Les résultats obtenus sont cependant très variables, à cause de l'initialisation totalement aléatoire :

Le scénario 1 sans l'algorithme offrait la représentation par défaut :



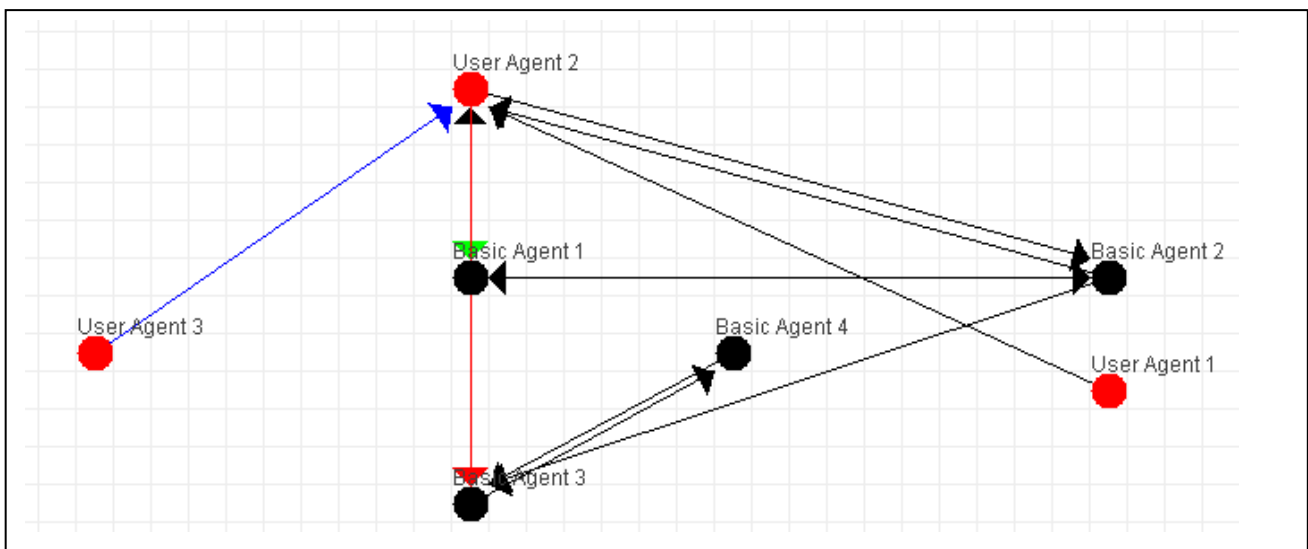
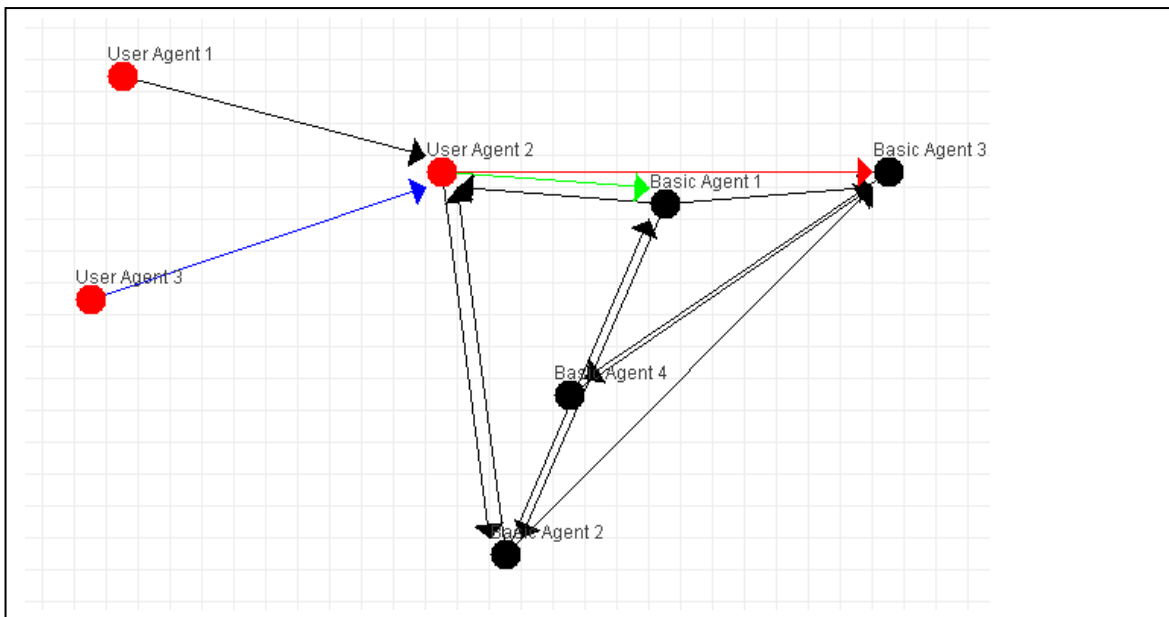
Avec application de l'algorithme, nous obtenons :



Le scénario 2 sans l'algorithme offrirait la représentation par défaut :



Avec application de l'algorithme, nous obtenons :



## Ajout à Sdait

Seules quelques modifications ont été apportées à Sdait, tels que :

- La coloration des arcs, pouvant être activés ou non.
- L’affichage d’arc orienté, au moyen de flèches.
- Fonction de nettoyage et de suppression d’arcs/nœud en dynamique dans Sdait.

La seconde partie du travail vis-à-vis de Sdait, autre que la simple utilisation, concernait l’affichage automatique de graphe. Originellement prévu pour une intégration directement dans Sdait, il a été plus intéressant de l’intégrer à madlib. L’application se sert alors de l’algorithme pour déterminer directement la position des nœuds et la récupérer via le réseau de réputation. Chaque nœud de ce dernier ayant un équivalent dans Sdait, il suffit alors d’utiliser la fonction de mapping déjà présente pour y appliquer les nouvelles positions.

## Simulation

La simulation, cœur du projet est divisé en 3 parties :

- D’une part, nous avons l’intégration du système de réputation depuis madlib.
- Ensuite nous avons l’interface graphique, liée à la librairie Sdait.
- Enfin nous retrouvons l’intégration d’un système de scénario, permettant une utilisation rapide et simple.

L’ensemble du système est articulé autour de 2 classes « manager ». La première, dédiée au graphique, est en charge des actions déclenchées par les différents boutons, tels l’ajout d’agent ou la suppression, et la mise à jour automatique des différents composants. La seconde est en charge de la simulation en elle-même. Gérant à la fois le réseau et le système de réputation, il a été créé de façon générique pour pouvoir être facilement remplacé par des systèmes similaires, tels qu’un système évoluant avec le temps. Cette notion de temps étant déjà apportée par Sdait.

Au sein de notre simulation, nous distinguons 2 agents, les « Basic agent » et les « User agent ». Les premiers sont considérés comme des agents appartenant au système. Ils ne peuvent être créés en dynamique et sont donc présents uniquement avec les scénarios. La seconde catégorie correspond aux agents de l’utilisateur. Il peut alors en ajouter autant qu’il souhaite, et surtout modifier leurs comportements, ce qui n’est pas possible pour les « Basic agent »

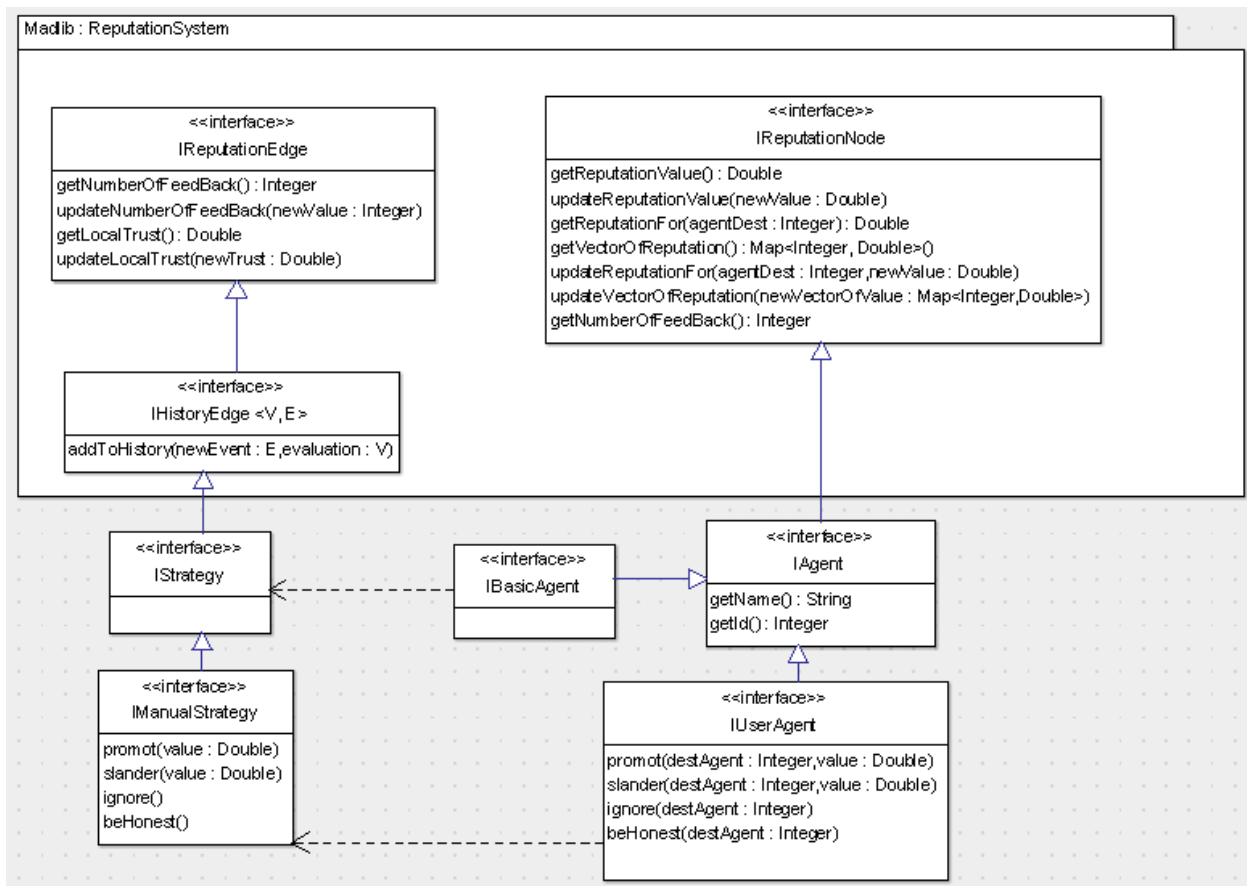
## Intégration du système de réputation.

L'intégration du système de réputation de madlib dans la simulation se fait au moyen de 2 groupe de classes, les Agents d'une part, représentant la structure de donnée des nœuds du graphe, et les Stratégies d'autre part, représentant les données sur les arrêtes.

Les différentes classes de Stratégies, représentant donc la relation entre l'agent x et l'agent y, détermine également le comportement de l'agent x envers l'agent y. Ce comportement s'exprime en modifiant la valeur renvoyée par la fonction `getTrustLocal()` en respectant le schéma suivant :

- Si x ignore y, alors on renvoie null
- Si x est honnête envers y, alors on renvoie la vraie confiance locale.
- Si x cherche à promouvoir y d'une valeur v, on renvoie alors la confiance locale + v.
- Si x cherche à calomnier y d'une valeur v, on renvoie alors la confiance locale - v.

Cette intégration est représentée par le schéma suivant :



On y voit ici la distinction faite entre Basic Agent et User Agent. Les User agents disposent d'une surcouche aux arrêtes leur permettant de mentir au système, alors que les Basic agents eux non. Dans la

suite du programme, on utilise uniquement les interfaces de madlib, permettant donc de limiter au maximum les dépendances vis-à-vis de la librairie.

Dans un second temps, la configuration du réseau et du système se fait au moment de l'instanciation, selon des valeurs fournies par l'utilisateur, ou avec les valeurs par défaut si l'utilisateur ne change rien. L'application s'occupe ensuite simplement de relayer les appels de l'interface graphique aux fonctionnalités de madlib tel que :

- Ajout/suppression d'un agent dans la simulation, correspondant à un nœud dans madlib.
- Ajout/suppression d'une relation entre 2 agents, correspondant à un arc dans madlib.
- Mise à jour des valeurs de réputation, correspondant au fait de relancer le calcul du système de réputation.
- Récupération des valeurs de réputation (valeur du système pour un agent) et de confiance (valeur d'un agent pour un autre agent).

## Interface graphique

L'interface doit répondre à plusieurs objectifs :

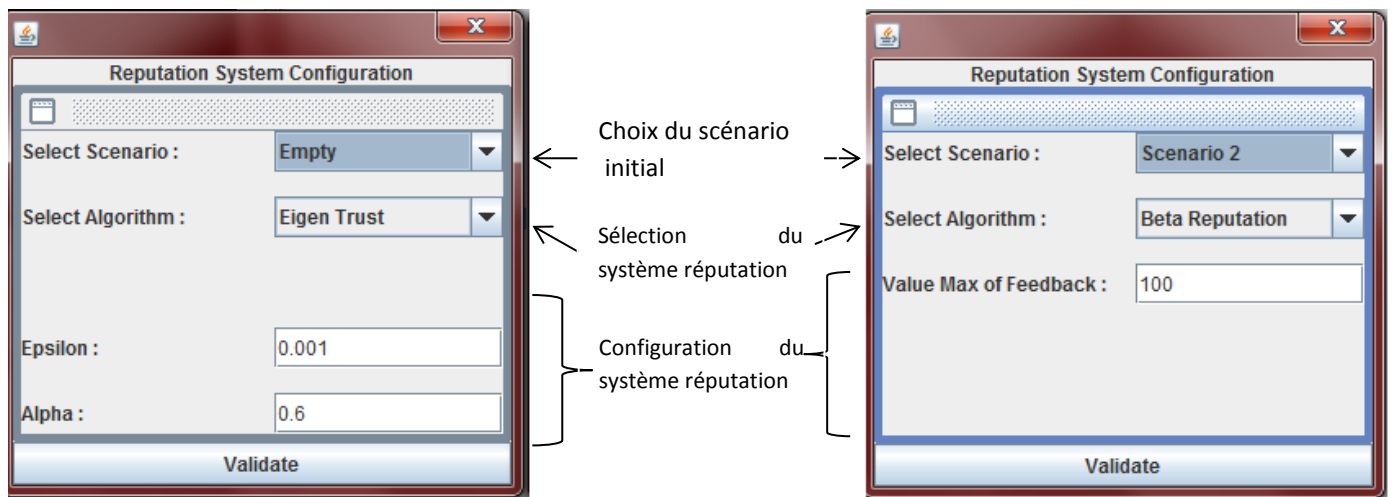
- Tout d'abord, il doit permettre de choisir et de configurer le système de réputation.
- Ensuite il doit permettre d'afficher à la fois le réseau de réputation et les résultats du système de réputation.
- Enfin il doit offrir la possibilité de modifier le réseau de réputation et de mettre à jour le système de réputation, pour voir l'évolution de ce dernier.

La fenêtre étant totalement gérée par Sdait, nous avons décidé d'utiliser un template de fenêtre à onglet, proposé par Sdait.

### *Fenêtre de Configuration.*

Le choix et la configuration du système de réputation se font intégralement au lancement de l'application. La modification dynamique d'un tel système n'ayant pas lieu d'être.

Cette configuration se fera donc par l'ouverture au préalable d'une fenêtre permettant la sélection du système de réputation, et, selon le choix en cours, de remplir les variables de l'algorithme. Pour permettre une utilisation plus rapide, un système de scénario a été ajouté permettant de charger des réseaux de réputations déjà construits. On peut alors soit sélectionner un de ces scénarios déjà remplis, soit en créer un nouveau totalement vide.



### *Onglet 1 : Représentation du réseau*

Cet onglet est composé de 2 parties.

La première, prenant 75 % de la fenêtre, est destinée à l’affichage du graphe, tel que présenté dans la figure 1.

La seconde, située à droite de l’écran, est un petit panel, présenté en figure 2, permettant d’ajouter un de supprimer un agent du réseau, mais également de gérer les couleurs associées aux différents comportements. Un bouton permet également de réorganiser automatiquement le graphe. Il peut être utilisé plusieurs fois d’affilé, la section aléatoire de l’algorithme de placement formera des représentations différentes.



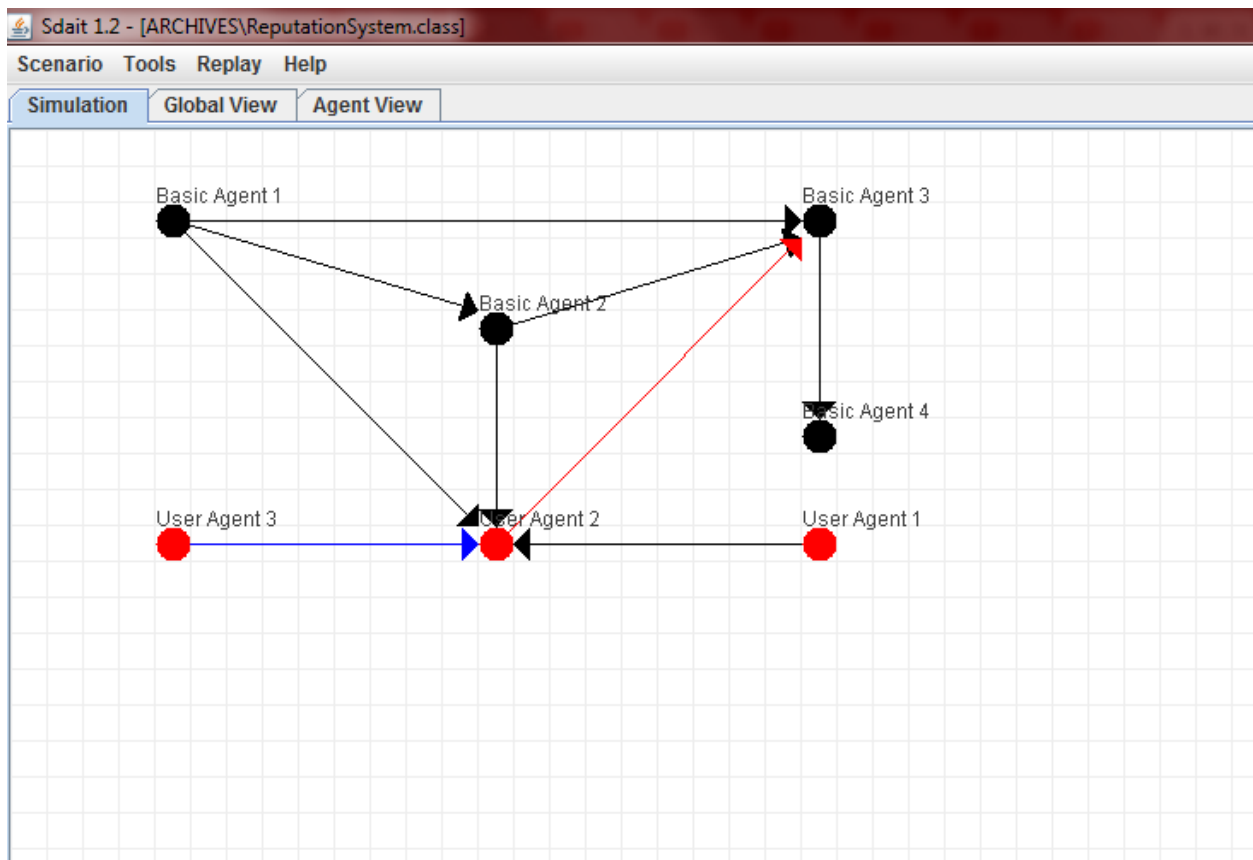


Figure 1

Add Agent

User Agent 1 ▼

Remove Agent

☒ Use color by type

Honest Color : Black ▼

Promot Color : Blue ▼

Slander Color : Red ▼

Ignore Color : Green ▼

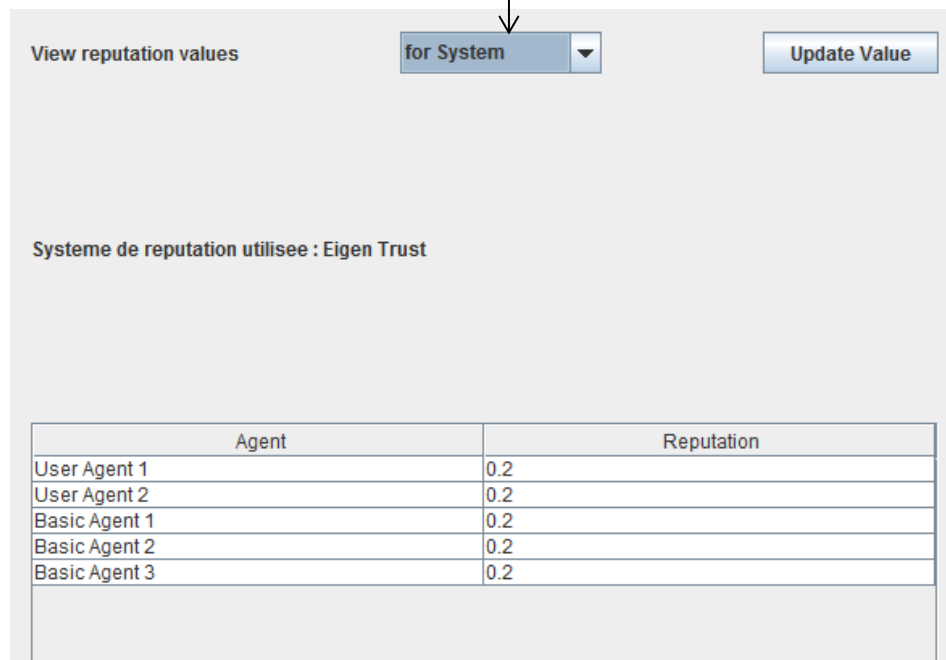
Reorganize Graph

Figure 2

## *Onglet 2 : Résultats du système de réputation.*

Le second onglet est en charge de l'affichage de données calculées par le système de réputation. On peut alors y sélectionner la vision du système, offrant pour chaque agent une valeur de réputation globale, ou un agent en particulier, affichant alors un vecteur de confiance avec la prise en considération des témoignages des autres agents.

Sélection de la vision : Voir les valeurs de réputation à l'échelle du système ou pour un agent donné.



View reputation values

for System ▼

Update Value

Systeme de reputation utilisee : Eigen Trust

Agent	Reputation
User Agent 1	0.2
User Agent 2	0.2
Basic Agent 1	0.2
Basic Agent 2	0.2
Basic Agent 3	0.2

Valeur de réputation  
par agent.

### *Onglet 3 : Modifications des relations entre agents.*

Le dernier onglet est en charge de l'affichage et de la modification des comportements des différents agents. L'utilisateur aura la possibilité de sélectionner un agent, et de voir l'ensemble de ses relations (connexion avec un autre agent) et de ses stratégies (comportement vis-à-vis du système concernant l'autre agent). En plus de l'affichage, si l'agent en question est modifiable par l'utilisateur, il peut alors ajouter ou supprimer des relations ou des stratégies, ou modifier les existantes.

Les modifications ainsi réalisées s'afficheront en temps réel sur l'onglet d'affichage du réseau, mais la prise en compte dans les calculs du système de réputation ne se fera que lors de la mise à jour, déclenchée par le bouton sur l'onglet d'affichage relatif au système.

View Agent : User Agent 2 ▼ ← Sélection de l'agent voulu.

Agent	Trust	Behavior	Value associate
User Agent 1			
Basic Agent 1	0.2	Honest	
Basic Agent 2			
Basic Agent 3			

Table des relations avec les autres agents, permettant la modification des comportements associés

Application des modifications.

Update Behavior

## Scénario

Comme prévu durant la conception, un système de scénario a été intégré à l'application pour permettre des lancements rapides. Le choix du scénario par l'utilisateur est fait via la fenêtre de configuration, au lancement de l'application. Actuellement, l'utilisateur a le choix entre un scénario vide, et 3 scénarios préconfigurés.

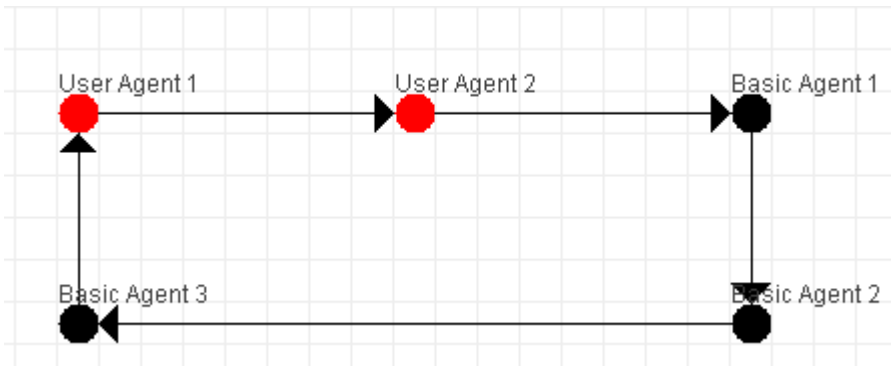
Dans l'interface actuelle, les agents sont appelés « Basic Agent » s'ils appartiennent au simulateur (donc non modifiable par l'utilisateur), et « User Agent » si l'utilisateur peut les modifier. L'objectif de l'utilisateur est de partir de ce scénario et, en ajout des agents ou modifiant les relations de ceux qu'ils contrôlent, d'augmenter la confiance du système pour chaque agent qu'il contrôle.

Les 3 scénarios préconfigurés sont présentés ci-dessous à titre purement informatif, pour permettre de voir rapidement la différence entre chaque.

### Scénario 1

Le premier scénario est composé de 5 agents, dont 2 pouvant être configurés par l'utilisateur. La particularité de ce scénario est de former un réseau circulaire.

La représentation des agents est de la forme suivante :

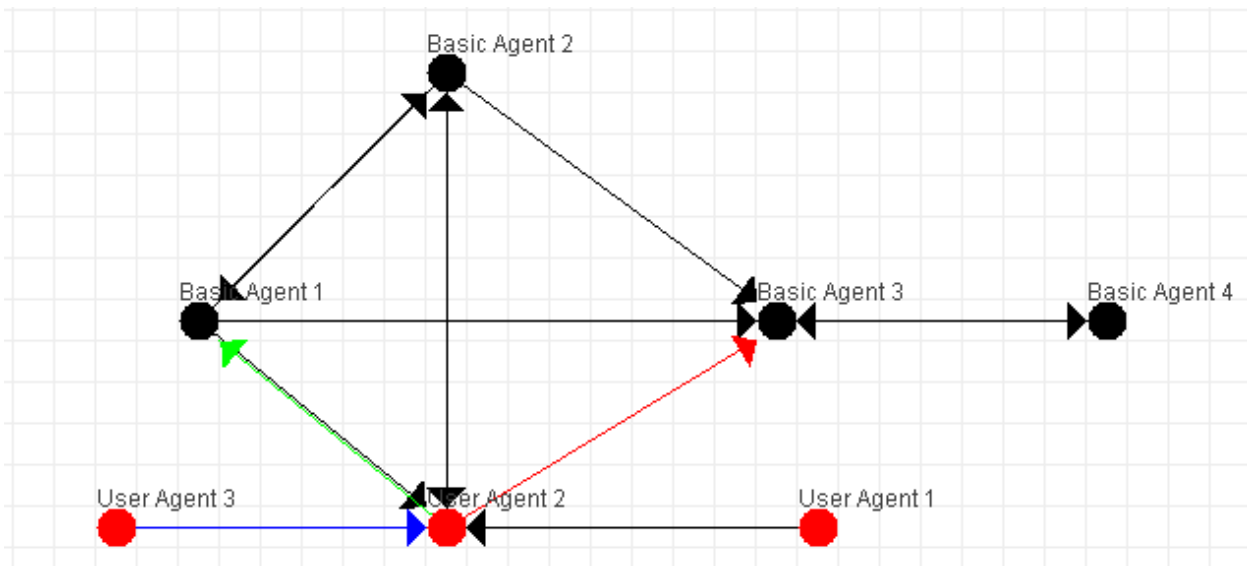


Les relations sont configurées comme suit :

Agent Source	Agent Cible	Confiance (sur 1)	Nombre de feedbacks	Evaluation des feedbacks (sur 100)	Comportement
Basic Agent 1	Basic Agent 2	0.6	10	60	Honnête
Basic Agent 2	Basic Agent 3	0.5	12	50	Honnête
Basic Agent 3	User Agent 1	0.4	6	40	Honnête
User Agent 1	User Agent 2	0.3	3	30	Honnête
User Agent 2	Basic Agent 1	0.2	2	20	Honnête

## Scénario 2

Le second scénario possède 7 agents, dont 3 pouvant être modifiés par l'utilisateur.

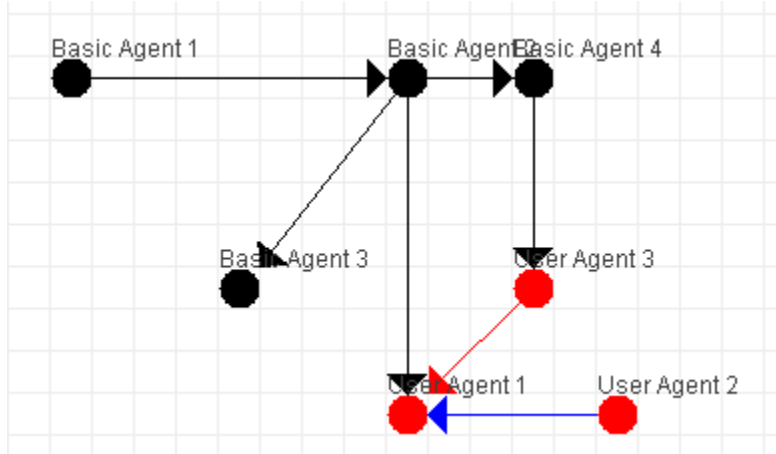


Les relations sont ici configurées comme suit :

Agent Source	Agent Cible	Confiance (sur 1)	Nombre de feedbacks	Evaluation des feedbacks (sur 100)	Comportement
Basic Agent 1	Basic Agent 2	0.8	10	80	Honnête
Basic Agent 1	Basic Agent 3	0.5	2	50	Honnête
Basic Agent 1	User Agent 2	0.3	5	30	Honnête
Basic Agent 2	Basic Agent 1	0.8	4	80	Honnête
Basic Agent 2	Basic Agent 3	0.5	3	50	Honnête
Basic Agent 2	User Agent 2	0.3	1	30	Honnête
Basic Agent 3	Basic Agent 4	0.5	10	50	Honnête
User Agent 1	User Agent 2	0.5	1	50	Honnête
User Agent 3	User Agent 2	0.5	10	50	<b>Promotion :</b> Confiance + 0.4 et Evaluation des feedbacks + 40
User Agent 2	Basic Agent 1	0.5	1	50	<b>Ignore</b>
User Agent 2	Basic Agent 2	0.5	1	50	Honnête
User Agent 2	Basic Agent 3	0.5	1	50	<b>Calomnie :</b> Confiance - 0.3 et Evaluation des feedbacks - 30

### Scénario 3

Le dernier scénario est composé 7 agents, dont 3 modifiables par l'utilisateur.  
La particularité de ce dernier est de ne pas avoir de cycle.



Les relations sont ici configurées comme suit :

Agent Source	Agent Cible	Confiance (sur 1)	Nombre de feedbacks	Evaluation des feedbacks (sur 100)	Comportement
Basic Agent 1	Basic Agent 2	0.8	10	80	Honnête
Basic Agent 2	Basic Agent 3	0.6	12	60	Honnête
Basic Agent 2	Basic Agent 4	0.5	13	50	Honnête
Basic Agent 2	User Agent 1	0.4	16	40	Honnête
Basic Agent 4	User Agent 3	0.3	10	30	Honnête
User Agent 3	User Agent 1	0.5	2	50	<b>Calomnie :</b> Confiance - 0.3 et Evaluation des feedbacks - 30
User Agent 2	User Agent 1	0.5	10	50	<b>Promotion :</b> Confiance + 0.4 et Evaluation des feedbacks + 40

## Conclusion

L'ensemble des principaux objectifs a été rempli, tel que l'ajout d'un modèle de réseau multiagents et des systèmes de réputations Beta Reputation et EigenTrust au sein de madlib.

L'application, dont l'ensemble de la couche graphique est réalisé en utilisant Sdait, permet à la fois d'afficher les données du réseau et du système de réputation, tout en permettant la modification dynamique du réseau et du comportement de chaque agent.

Divers objectifs optionnels ont été ajoutés, certains se résumant à de simples effets graphiques pour améliorer la lisibilité, telle la coloration des arcs du réseau. Une fonctionnalité beaucoup plus complexe, celle de l'organisation automatique d'un graphe pour une représentation graphique, a également été mise en place au sein de la librairie Sdait.

Dans la librairie madib, un autre système de réputation a été mis en place, FlowTrust, pour compléter le projet après la réalisation des objectifs principaux.

Le projet peut toujours être poursuivi par l'ajout de niveau système de réputation, ou l'évolution du système vers une simulation dynamique sur un modèle complet d'échanges de services, comme présenté en début de ce rapport.

Pour conclure, j'exprimerai seulement le plaisir que j'ai eu à réaliser ce projet, me permettant de m'informer sur les problématiques de confiance dans un système multiagents, le tout dans un cadre concret.

## Bibliographie

- [1] S.D. Kamvar, M.T. Schlosser, and H. Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the Twelfth International World Wide Web Conference*, Budapest, May 2003
- [2] R. Ismail and A. Jøsang. The beta reputation system. In *Proceedings of the 15th Bled Conference on Electronic Commerce*, Bled, Slovenia, 2002.
- [3] J. Sabater and C. Sierra, Review on computational trust and reputation models. *Artificial Intelligence Review* 24, 1 (2005) 33–60.
- [4] L. Mui, M. Mohtashemi, and A. Halberstadt. A computational Model of Trust and Reputation In *35th Hawaii International Conference on System Science*, 2002
- [5] T. Vallée, G. Bonnet, and F. Bourdon. De l'utilisation des politiques de bandits manchots dans les systèmes de réputation.
- [6] G.Wang and J.Wu, FlowTrust : trust inference with network flows, *Frontiers of Computer Science in China* 5 (2) 181–194, 2011.
- [7] M. Jamali and M. Ester. Trustwalker: A random walk model for combining trust-based and item-based recommendation. In *KDD* 2009
- [8] S. Karouach and B. Dousset. Visualisation de relations par des graphes interactifs de grande taille. *ISDM: J. of Information Sciences of Decision Making*, 6(57):253-264, 2003.
- [9] T. Fruchterman and E. Reingold. Graph drawing by force-directed replacement. *Software - Practice and Experience* 21, 1991.
- [10] S. Karouach and B. Dousset Les graphes comme représentation synthétique et naturelle de l'information relationnelle de grande taille. *Workshop sur la recherche d'information : un nouveau passage à l'échelle*, associé à INFORSID'2003, (Nancy France), 3-6 juin 2003.