Conception d'une plateforme applicative de type " banc d'essais " dédiée à la gestion de données scientifiques orientées ingénierie des connaissances

Dino Cosmas

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE REGIONAL ASSOCIE DE RHONE ALPES

MEMOIRE

présenté en vue d'obtenir

le DIPLOME D'INGENIEUR CNAM

SPECIALITE : SCIENCES & TECHNOLOGIE INFORMATIQUE

OPTION : CYC12 - INGENIEUR SYSTEMES D'INFORMATION

par

Dino COSMAS

Conception d'une plateforme applicative de type « banc d'essais » dédiée à la gestion de données scientifiques orientées ingénierie des connaissances

Soutenu le 11 juin 2012

JURY

PRESIDENT : Monsieur Christophe PICOULEAU

MEMBRES : Monsieur Bertrand DAVID
Monsieur Claude GENIER
Monsieur Olivier CHAMPALLE
Monsieur Pierre-Antoine CHAMPIN

Remerciements

Mes remerciements vont tout d'abord à Monsieur Olivier CHAMPALLE, qui m'a donné l'opportunité, conjointement à l'élaboration de sa thèse, de réaliser le projet d'ingénieur que soutient ce mémoire. Notre coopération n'était à priori pas évidente, tant nos expériences professionnelles respectives et nos motivations étaient différentes, mais nous avons su nous organiser pour travailler ensemble, « entre CNAMIEN ».

Je tiens aussi à adresser mes sincères remerciements à l'ensemble des intervenants pédagogiques du CNAM et plus particulièrement à Monsieur Bertrand DAVID et Madame Eléonore GONDEAU qui depuis plusieurs années ont œuvré pour m'aider à atteindre mes objectifs.

Mes remerciements vont aussi à tout le laboratoire LIRIS et plus particulièrement à l'équipe Silex où j'ai pu rencontrer et partager des idées, des questions ainsi que de nouvelles théories. Je nommerai au sein de cette équipe Alain MILLE, Pierre-Antoine CHAMPIN, Yannick PRIE, Karim SEHABA mais aussi Bruno, Olivier, François, Romuald, Kreshnik et tant d'autres qui ne m'en voudront pas de ne pouvoir tous les citer mais, sans qui ce travail n'aurait pas eu la même saveur.

De plus, je tiens à remercier le jury par avance pour sa présence, son jugement et ses conseils avisés qui permettront de valider, critiquer et améliorer mon travail.

Enfin, il est évident que je ne remercierai jamais assez ma famille, mes parents et Sophie pour leur indéfectible soutien moral, culinaire et affectif ; tous ont contribué à l'élaboration de ce mémoire.

Table des matières

Introduction

Dans le cadre de la formation professionnelle et du transfert de savoir-faire, les situations de simulation sont particulièrement bien adaptées et rependues[1]. L'emploi des simulateurs dit « pleine échelle » est en particulier très présent. Le but des simulateurs pleine-échelle est de reproduire de manière réaliste une situation de travail dans toutes ses dimensions. La situation ainsi reproduite « autorise une fidélité maximale avec la situation professionnelle de référence[2] ». Les simulateurs pleine échelle prennent place dans les milieux dangereux et/ou pouvant entraîner des risques importants tant pour les personnes que pour les matériels. Il n'est donc pas rare de les retrouver dans le domaine de la conduite de centrale nucléaire, du pilotage d'avion de ligne ou de celle de char d'assaut[3]. Leurs avantages nombreux autorisent de réelles mises en situations en permettant notamment de faire évoluer le scénario de simulation très facilement en faisant intervenir des pannes par exemple. Ces simulations ont pour but de tester et/ou de faire évoluer les connaissances d'une ou de plusieurs personnes apprenantes dans un contexte identique à celui de leur métier quotidien. Le simulateur pleine échelle est piloté par un à plusieurs formateurs chargés de créer, de superviser le déroulement de la situation de simulation et d'observer, et d'analyser le comportement des stagiaires dans un but de débriefing et de retour d'expérience.

Afin d'aider à cette expertise d'observation, l'expert peut utiliser des outils basés sur la théorie de la trace modélisée. L'ensemble des termes rattachés à cette théorie (M-Trace, modèle de M-Trace, observé, transformation) permet une organisation, une analyse et un partage de connaissance de manière structuré.

L'équipe SILEX du département « Données, Connaissances, Services » est un pôle « découverte de connaissances »[4] à l'intérieur du laboratoire LIRIS. C'est au sein de celle-ci que des travaux poussés sur la théorie de la trace modélisée sont effectués et ont servi de base principale à l'élaboration du projet décrit dans ce mémoire.

Le sujet de ce mémoire a été conçu par Olivier CHAMPALLE, doctorant au sein de l'équipe SILEX. Son sujet de thèse était : « *Instrumentation d'environnements informatiques pour l'apprentissage humain intégrant des simulateurs : capitaliser à partir des expériences d'utilisation pour faciliter l'adaptation aux contraintes de formation* ». Suite à une étude d'implémentations existantes relatives à ce sujet, puis à la conception et aux tests de l'application, nous avons pu conforter et illustrer les problématiques traitées au sein de la thèse.

---

[1] (Hetzner S, et al...) Adult Self-regulated Learning through Linking Experience in Simulated and Real World: A Holistic Approach. *In the Proceedings of the 6th European conference on Technology enhanced learning: towards ubiquitous Learning*, 2011, pp 166-180).

[2] [Pastré 2005] Apprendre par la résolution de problèmes : le rôle de la simulation. Apprendre par la simulation : de l'analyse du travail aux apprentissages professionnels. 2005

[3] [Joab 2000] La prise en compte de la dynamique dans la conception d'un système de formation fondé sur la simulation. Simulation et formation professionnelle dans l'industrie. 2000

[4] http://liris.cnrs.fr/presentation/organigramme-du-liris

Chapitre 1

Contexte et état de l'art

1.1 Contexte de réalisation

L'élaboration de ce projet s'est faite intégralement au laboratoire le LIRIS au sein de l'équipe SILEX. Le sujet a été conçu par Olivier CHAMPALLE en liaison avec le sujet de la thèse qu'il a effectué d'avril 2009 à avril 2012. Le laboratoire ayant eu de nombreuses thèses dans des domaines proches, des applications en relation avec le prototype que j'ai conçu ont été à prendre en compte avant de proposer une méthodologie à suivre pour le bon déroulement et le succès de ce projet.

1.1.1 Equipe SILEX du LIRIS

Le LIRIS (Laboratoire d'InfoRmatique en Image et Systèmes d'information) travaille dans le domaine des Sciences et Techniques de l'Information et de la Communication. Il est associé au CNRS[5] avec le label UMR 5205. L'institut d'attachement principal est INS2I[6]. L'institut d'attachement secondaire est INSIS[7]. Le LIRIS a également une convention avec l'ISCC[8] du CNRS.

Regroupant environ 280 personnes, dont près de 110 chercheurs et enseignants-chercheurs, le LIRIS a cinq tutelles : le CNRS, l'INSA[9] de Lyon, l'Université Claude Bernard Lyon 1, l'Ecole Centrale de Lyon et l'Université Lumière Lyon 2, ainsi que des sites où se déroulent des activités à La Doua, Ecully et Bron.

Les activités du laboratoire sont regroupées dans les deux départements thématiques : "Image" et "Données, Connaissances, Services" dont fait partie l'équipe SILEX.

L'objectif de l'équipe SILEX est de construire des théories, des formalismes, des méthodes et des outils intégrant le caractère intrinsèquement dynamique des connaissances exploitées dans les environnements informatiques ouverts, dans l'objectif de concevoir des systèmes co-apprenants.

---

[5] CNRS : Centre National de la Recherche Scientifique
[6] INS2I : Institut des Sciences de l'information et de leurs Interactions
[7] INSIS : Institut des Sciences de l'Ingénierie et des Systèmes
[8] ISCC : Institut des Sciences de la communication
[9] INSA : Institut National des Sciences Appliquées

Trois thèmes s'articulent pour atteindre cet objectif :

1. l'ingénierie de l'expérience tracée,
2. la personnalisation des Environnements Informatiques d'Apprentissage Humain,
3. les systèmes adaptatifs pour la co-construction d'activités interactives, en particulier dans leur application aux situations de handicap.

Les travaux s'inscrivent dans le cadre plus général de l'ingénierie des connaissances et se déclinent de manière originale pour gérer le caractère dynamique de la connaissance.

1.1.2 Conditions de réalisation

Le travail de stage entre dans le cadre d'un projet de recherche mené par l'équipe SILEX en partenariat avec l'Unité de Formation Production Ingénierie d'EDF. Ce projet porte sur l'instrumentation d'Environnements Informatiques pour l'Apprentissage Humain (EIAH) intégrant des simulateurs pleines échelles pour la formation et l'évaluation des personnels dédiés à la conduite des centrales du groupe EDF. En particulier, l'accent est mis sur l'utilisation des traces d'interaction comme outils d'aide à l'observation des apprenants en situation de formation et/ou d'évaluation.

En effet, au cours des sessions de formation / évaluation, les personnels apprenants doivent répondre à des situations conçues pour faire appel à des connaissances considérées comme acquises ou en cours d'acquisition. Ils sont ainsi amenés à réaliser un *transitoire*[10] consistant à appliquer un ensemble de procédures pour permettre le passage du simulateur d'un état initial à un état final connu et attendu. Un *transitoire* est considéré comme validé lorsque les actions des personnels apprenants répondent à un ensemble *d'observables*[11] attendus[12].

Aspect essentiel des sessions de formation et d'évaluation sur simulateur de conduite, l'observation de l'activité individuelle et collective des stagiaires forme le cœur du retour d'expérience et de l'amélioration des connaissances de conduite. Tout l'enjeu du projet de recherche était de collecter et d'exploiter l'historique des actions des apprenants sur les simulateurs sous la forme de traces d'interaction modélisées à des fins de retour d'expérience rapide et assisté par l'informatique.

1.1.2.1 Différents rôles identifiés

A la fois chef de projet, ingénieur d'étude, et développeur de cette plateforme applicative, je devais être capable de réaliser toutes les différentes démarches attendues par ces différents métiers au sein de l'élaboration d'un projet informatique.

A savoir :

- L'analyse des besoins d'Olivier CHAMPALLE (et par son intermédiaire des utilisateurs potentiels),

[10] Transitoire : Action de faire passer le simulateur d'un état physique E(0) à un instant T0 à un état attendu et connu E(n) à Tn.
[11] Observable : action ou activité observé possédant un type et ayant des attributs temporels
[12] Utilisation des traces d'interaction comme outils d'aide à l'observation sur simulateur. O. Champalle. Dans 3èmes Rencontres Jeunes Chercheurs en EIAH, RJC-EIAH 2010, ATIEF ed. Lyon. pp. 101-106. 2010. http://liris.cnrs.fr/Documents/Liris-4636.pdf

- L'analyse de l'existant, nous mettons alors en évidences les forces et faiblesses des applications existantes au sein du laboratoire approchant le sujet que nous voulons informatiser,
- La création du prototype conforme à des spécifications fonctionnelles et techniques réalisées aux préalables,
- Prévoir la documentation et les tests permettant à la fois d'aider l'utilisateur à s'approprier l'application et de justifier sa robustesse.

1.1.2.2 Résultats attendus

Les résultats attendus étaient susceptibles d'évoluer en fonction du déroulement du projet. Néanmoins, certains livrables incontournables peuvent être listés ainsi :

- Cahier des charges, spécifications fonctionnelles,
- Réflexion sur le choix des technologies de conception,
- Conception de l'application en adéquation avec les outils du laboratoire et de l'équipe SILEX.

Documentation exhaustive de manière à faciliter la capitalisation et la réutilisation de la plateforme applicative.

1.1.2.3 Planning prévisionnel

Ce travail devait permettre de valider des hypothèses scientifiques propres à l'équipe SILEX dans le domaine de l'Ingénierie des connaissances et plus particulièrement dans celui de l'aide à l'observation à base de traces modélisées. Le temps imparti à la réalisation de ce projet était de 9 mois. Au cours de ce temps, 4 phases ont eu lieu.

La première phase, d'une durée de 3 semaines, a permis de recueillir les besoins du projet. Elle avait pour but de permettre une intégration dans l'équipe SILEX et de mieux cerner les tenants et aboutissants du projet. C'est durant cette phase qu'a été délimité le périmètre du projet et qu'a été étudiée la réutilisabilité des précédents projets réalisés au sein de l'équipe SILEX.

La deuxième phase, d'une durée d'un mois, a été la phase d'organisation et de planification de la réalisation du développement applicatif du projet. C'est alors qu'a été déterminé le planning du projet, les jalons des différentes étapes et les livrables attendus à chacun de ces jalons.

Venait ensuite la phase de conception et de mise en place de la plateforme applicative. La fiabilité des approches scientifiques qui étaient défendues dans la thèse d'Olivier CHAMPALLE devait être validée par la réalisation de cette plateforme. La fiabilité de cette dernière était en fonction de la vitesse et de la qualité d'avancement des développements.

Enfin, la phase finale, la réalisation du mémoire retraçant l'histoire de ce projet.

1.2 Théorie de la trace modélisée

L'Environnement Informatique pour l'Apprentissage Humain (EIAH) étudié dans la thèse d'olivier CHAMPALLE, était de type simulateur pleine échelle, dédiés à la formation et au perfectionnement des agents de conduite de centrale nucléaire (*figure 1*).

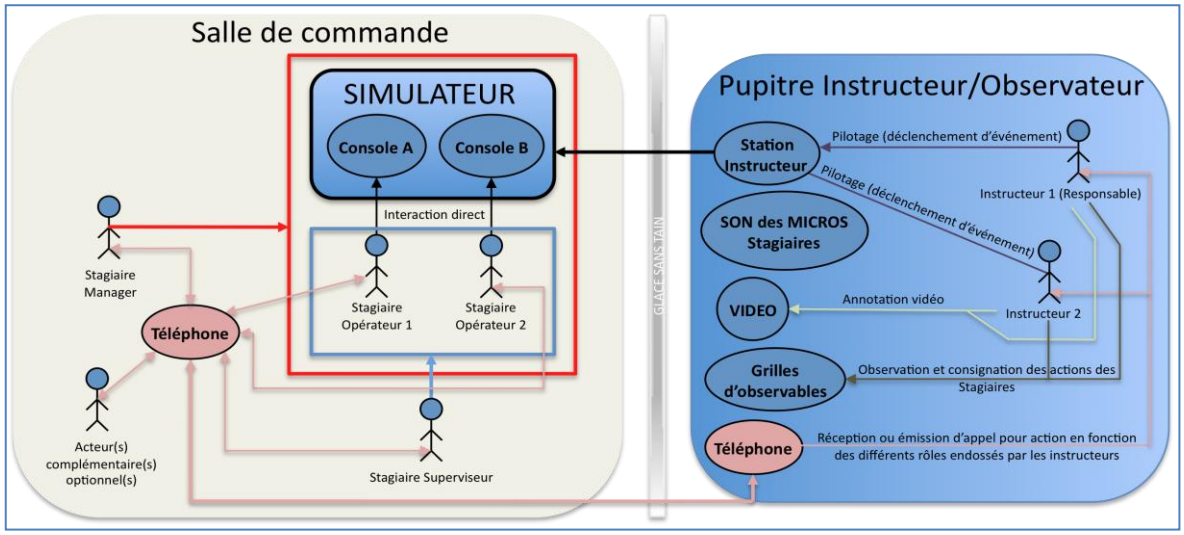


Figure 1 : Simulateur pleine échelle de conduite de systèmes industriels complexes

Ces séances de formation et/ou de perfectionnement sont primordiales pour la formation et le maintien des compétences des opérateurs.

Comme le montre la *figure 1*, durant la séance de simulation, les formateurs (un ou deux suivant les types de formation) ont la charge de piloter la simulation et d'observer, par différents moyens, les réalisations des stagiaires.

La tâche de ces derniers consiste à réaliser le transitoire. Il s'agit de faire passer le simulateur d'un état physique initial e0 à un instant t0 à un état final en à tn. Pour cela, les stagiaires doivent conduire l'installation selon des attendus organisés en familles d'objectifs pédagogiques. Parmi ces objectifs, on trouve : connaître et surveiller l'installation, conduire l'installation, assurer la continuité de service, travailler en équipe, etc.

Une session de simulation (de formation ou de perfectionnement) comporte plusieurs phases. Parmi ces phases, on trouve :

1) Accueil et briefing des stagiaires : les formateurs présentent le scénario de simulation aux opérateurs stagiaires ainsi que le rôle de chacun d'eux (superviseur, chef d'exploitation, manager, etc.).
2) Réalisation du transitoire : pour cela, les opérateurs doivent appliquer les procédures relatives au scénario et aux différentes situations rencontrées durant la simulation,
3) Analyse de la séance de simulation : les formateurs mettent en commun leurs observations (annotations et des grilles d'observations) pour préparer la phase de débriefing. Il s'agit de vérifier si les attendus ont été réalisés (ou non) ainsi que les actions permettant d'appréhender les difficultés rencontrées par les stagiaires lors de l'exercice, ainsi que les raisons de ces réussites (ou de ces échecs). Durant cette phase, les formateurs peuvent visualiser les données collectées par le simulateur telles que la vidéo, la téléphonie, les journaux de bord, etc.
4) Débriefing de la séance de simulation : durant cette phase, les formateurs rendent compte de leurs observations aux stagiaires. Le but étant de répondre aux situations critiques et d'apporter une aide aux stagiaires pour surmonter les difficultés rencontrées.

Le simulateur est piloté en temps réel par les formateurs. Parallèlement les interactions des formateurs, de celles des stagiaires ainsi que les informations permettant de suivre les changements d'état du simulateur sont stockées dans des journaux de bords.

Au terme de la simulation, les journaux de bords peuvent être conservés sous forme de log permettant de retracer l'ensemble de l'activité de la session à posteriori.

Ces journaux de bords reflètent différentes observations (numériques si elles proviennent des interactions sur le simulateur, manuelles si elles proviennent des annotations des formateurs) et représente ainsi une trace de l'activité de simulation au cours du temps.

Dans sa thèse, Olivier CHAMPALLE associe les informations contenues dans les journaux de bords à des observations de bas niveaux qu'il est possible de manipuler sous le forme de « trace modélisée » afin d'assister les formateurs dans l'observation et l'analyse de l'activité de simulation.

Par définition, une trace modélisée est un ensemble *d'observés* temporellement situés. On appelle *observé* toute information structurée issue d'une observation. Dans le cadre de recherche d'Olivier CHAMPALLE, les observés sont générés à partir de l'observation :

- des interactions entre les utilisateurs (stagiaires et formateurs) et le simulateur pleine-échelle
- les changements d'état du simulateur dus au programme de la simulation en lui-même permettant de simuler la « vie » de la centrale nucléaire.

Formellement *(figure 2)*, tout observé possède un type, défini dans le *modèle de trace*, ainsi qu'une estampille temporelle dans la M-Trace. Un observé peut potentiellement être en *relation* avec d'autres observés de la même M-Trace au travers de relations dont le type est défini par le modèle de la M-Trace. En fonction de son type un observé peut avoir un ensemble d'attributs/valeurs qui le caractérise. Le *modèle de trace* définit ainsi l'ensemble des types d'observés et des types de relations qui composent la trace. On appelle *trace modélisée*, M-Trace, une structure de données composées d'observés et accompagnée de son modèle de trace.

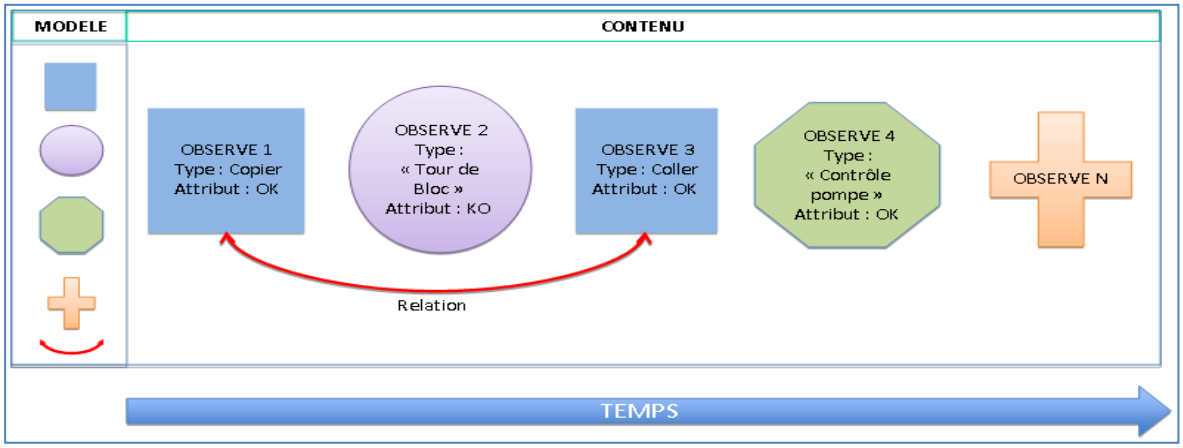


Figure 2 : Exemple simplifié d'une trace modélisée ou M-Trace

Les traces sont gérées par un Système de Gestion de Base de Trace (SGBT)[13] chargé de la gestion et du stockage des M-Traces (gestion des droits, base de données...) ainsi que de leurs transformations. Les transformations sont particulièrement utiles quand des interprétations de traces sont nécessaires. Une transformation de M-Trace permet de construire une nouvelle M-Trace cible à partir d'une M-Trace source selon des règles de transformation *(figure 3)*.

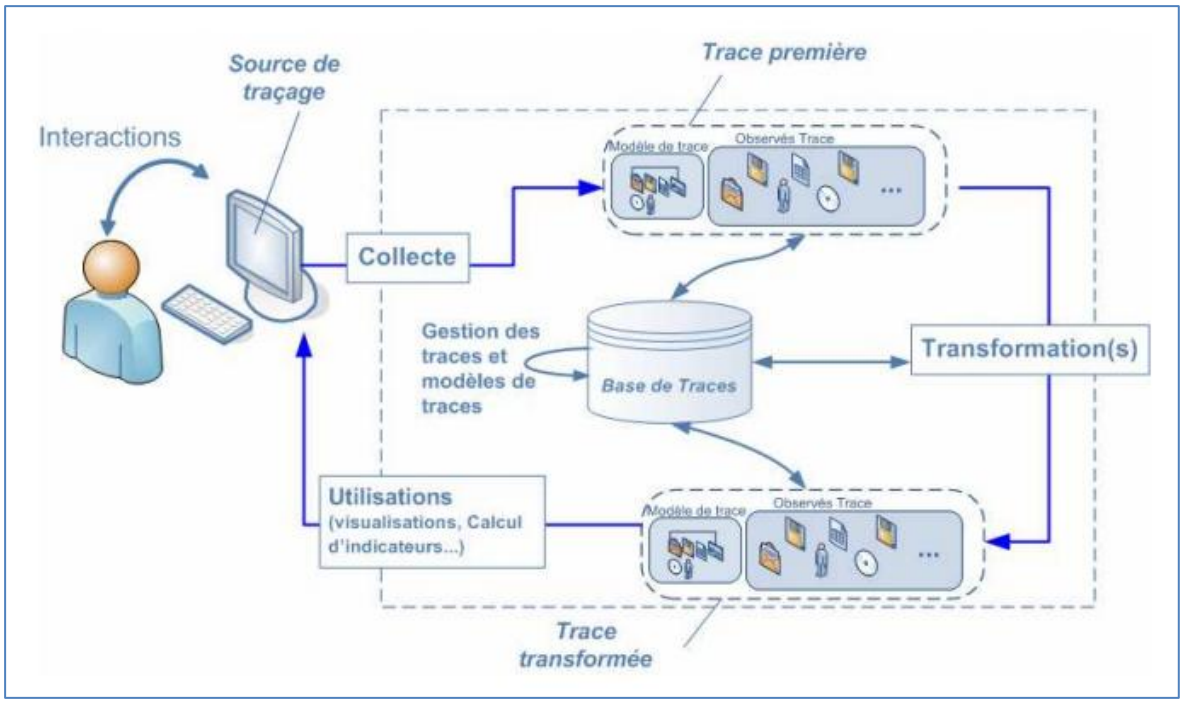


Figure 3 : Théorie de la trace modélisée

[13] SETTOUTI L.S. (2011). Systèmes à Base de traces modélisées - Modèles et langages pour l'exploitation des traces d'Interactions. Thèse de Doctorat en Informatique. Université Claude Bernard Lyon 1. http://liris.cnrs.fr/Documents/Liris-4984.pdf

1.3 Etude de l'existant

L'équipe SILEX, ayant accueilli tous les ans des stages de master ainsi que des doctorants, possédait des plateformes logicielles en lien direct avec mon sujet de mémoire. De plus, de nouveaux langages et outils de programmation que nous avons pu identifier ont été utilisés. Enfin, l'équipe était en mesure de nous fournir un environnement matériel en adéquation avec nos besoins liés à la bonne réalisation du projet de stage. Nous pouvions donc commencer à évaluer les différents développements en cours ou passés.

1.3.1 Abstract

Abstract ("Analysis of Behavior and Situation for menTal Representation Assessment and Cognitive acTivitiy modeling")[14] est un atelier logiciel de modélisation de l'activité à partir de traces. Ce système est composé d'un système de collecte, d'un système de manipulation des traces permettant de les transformer et de les visualiser, ainsi que d'un éditeur d'ontologie permettant de définir les modèles de trace, d'un système de documentation et d'indexation. Bien que la réalisation de cet atelier ait pour contexte l'analyse de la conduite automobile, il a été conçu de manière générique pour permettre l'analyse de toute activité basée sur la théorie de la trace modélisée. L'architecture d'Abstract comporte 3 systèmes principaux.

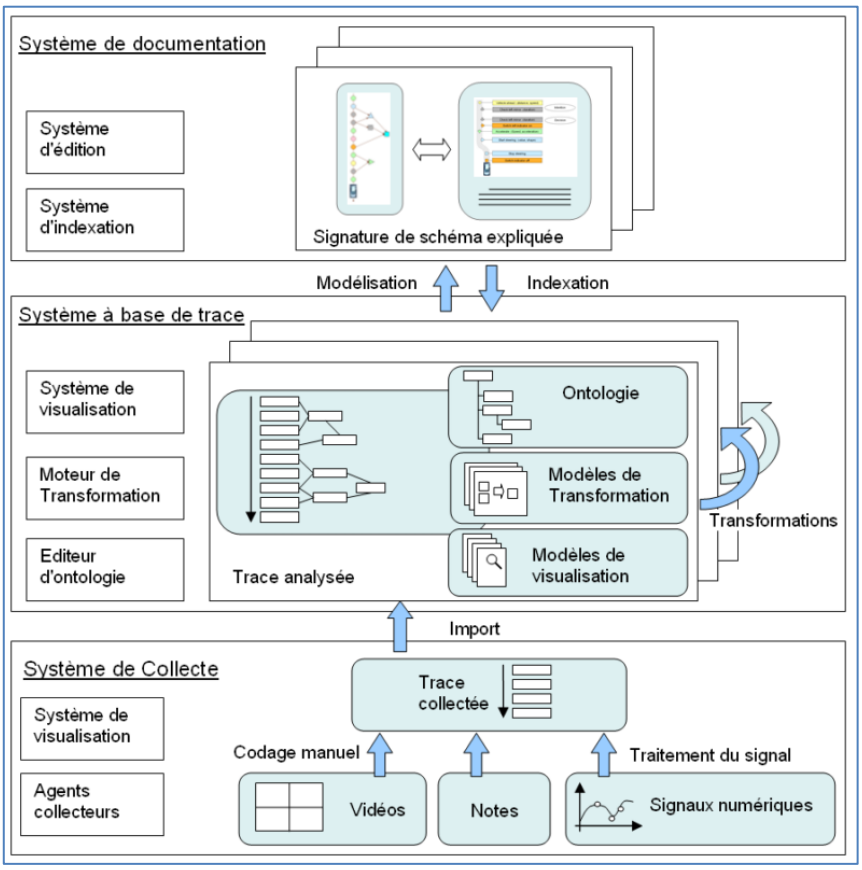


Figure 4 : Architecture générale d'Abstract

[14] Thèse Olivier Georgeon : http://liris.cnrs.fr/Documents/Liris-4201.pdf

Comme on peut le voir sur la figure 4, il y a tout d'abord le **système de collecte** permettant de récupérer au sein du système les observations de différentes natures ; vidéo, notes et signaux numériques.

Suite à l'import des données issues de la collecte, intervient le **système à base de trace**. Ce système va permettre d'analyser les différentes données à travers différentes modélisations. Il est aussi possible d'effectuer des transformations, permettant d'abstraire les données brutes et de permettre une interprétation humaine.

Enfin intervient le **système de documentation**, permettant l'interprétation des données grâce à une visualisation structurée.

Un exemple de visualisation est présenté sur la *figure 3.*

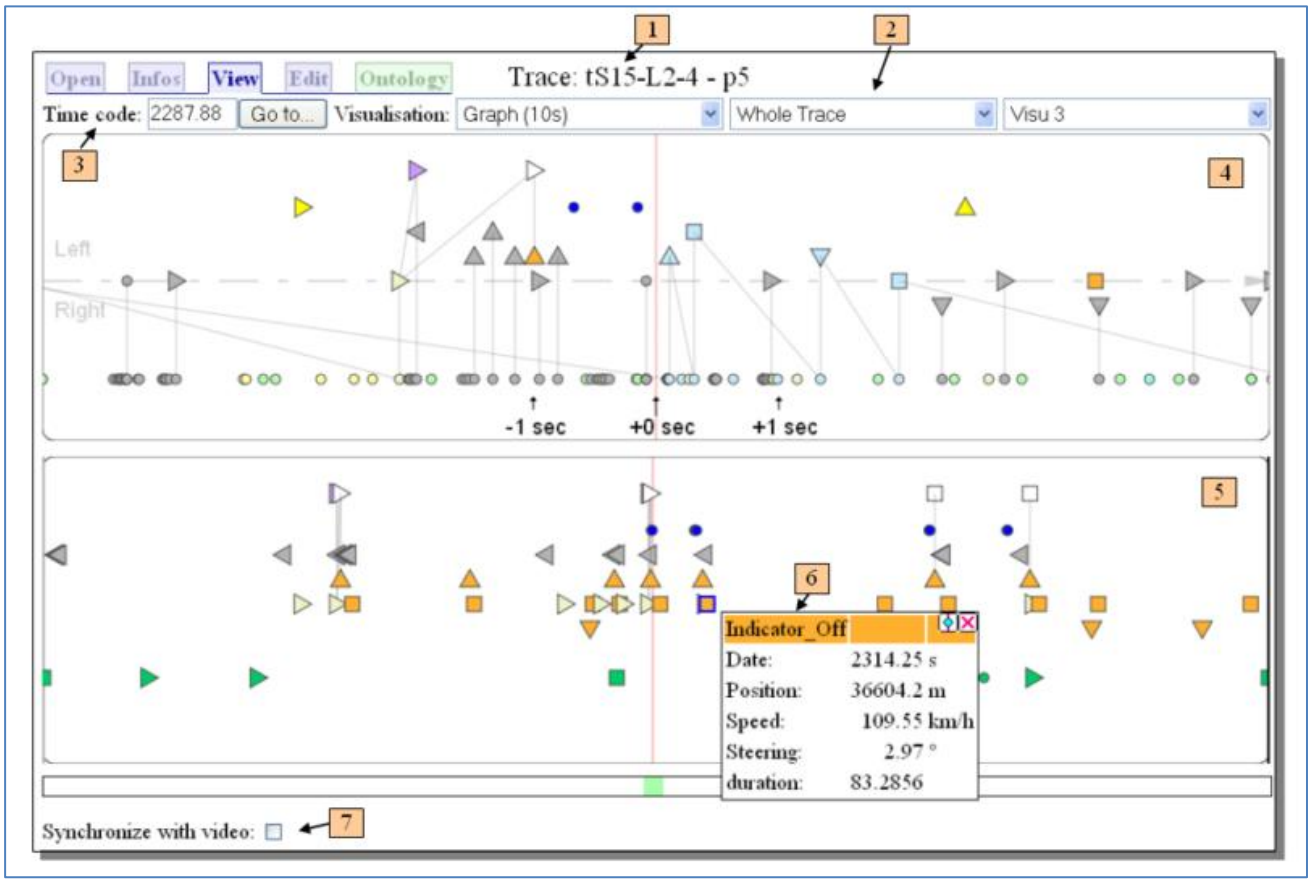


Figure 5 : Visualisation de séquences transformées

1. Consultation de la référence de la trace et de la séquence transformée.
2. Sélection des modules de visualisation dans des listes déroulantes. Chacun correspond à une feuille de style XSL particulière. Trois modules peuvent être visualisés simultanément à l'écran.
3. Consultation et saisie du time code : La valeur indiquée correspond au time code à l'emplacement du curseur dans les modules de visualisation (ligne rouge verticale). On peut saisir un time code et cliquer sur "Go to" pour s'y positionner.
4. Exemple de module de visualisation ayant une plage de temps de 10 secondes.
5. Exemple de module de visualisation de la trace complète, avec seulement les observés de haut niveau affichés.
6. Affichage des propriétés par un clic sur les observés. Cette fenêtre offre une icône sur laquelle on peut cliquer pour centrer la visualisation sur cet observé.

7. Synchronisation de la trace avec la vidéo. Quand cette case est cochée, la visualisation se synchronise avec l'outil de visualisation vidéo. La trace avance au fur et à mesure que la vidéo est jouée.

Pour produire ces séquences transformées, Abstract contient un outil permettant la saisie et l'exécution de ces transformations.

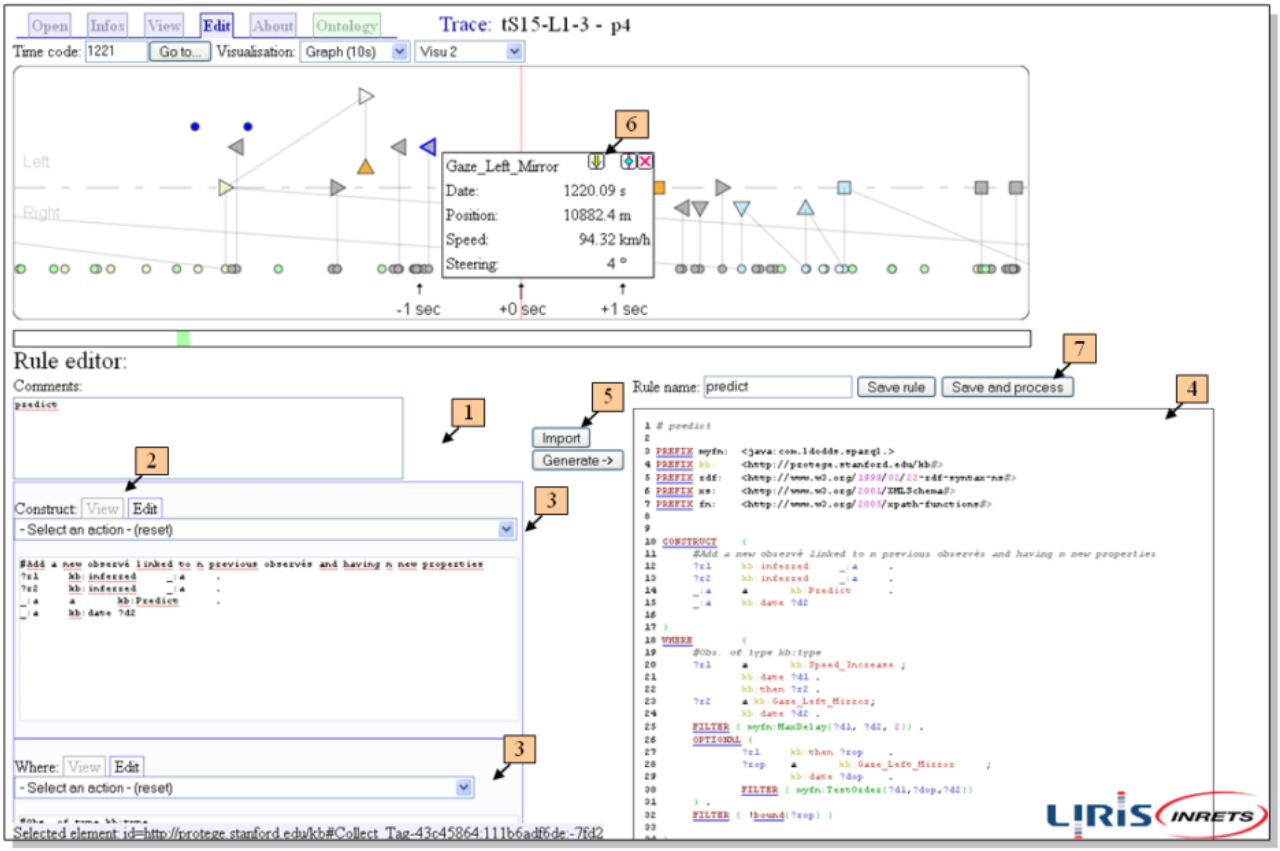


Figure 6 : Outil de transformation

1. Saisie d'une transformation SPARQL en trois parties : la partie commentaire, la partie CONSTRUCT et la clause WHERE.
2. Visualisation de la transformation SPARQL. Ce mode offre une coloration syntaxique et permet de cliquer sur les mots clés pour obtenir une aide contextualisée. Cela permet aux utilisateurs novices de se former à SPARQL.
3. La sélection de modèles de clauses CONSTRUCT ou WHERE dans des listes déroulantes.
4. Visualisation du fichier transformation en entier. Cette visualisation offre la numérotation des lignes. Cela facilite la correction des erreurs de syntaxe, car le numéro d'une éventuelle ligne erronée est renvoyé par le système quand la requête est appliquée.
5. Import d'une requête existante à partir d'un fichier.
6. Import d'une requête à partir d'un observé vu dans la trace. Ce bouton permet d'importer la requête qui a généré l'observé sélectionné afin de s'en servir comme modèle pour générer un autre observé.
7. Enregistrement et application de la transformation sur la séquence transformée courante. Lorsque la transformation est appliquée avec succès, une séquence transformée fille est générée. L'utilisateur est informé du nombre d'observés générés par la requête. La séquence transformée générée devient la séquence transformée courante.

1.3.2 SBT-IM

L'utilisation des indicateurs dans les EIAH reste difficile car nécessite non seulement la conception d'indicateurs intéressants pour le suivi, l'animation et l'évaluation d'une activité d'apprentissage, mais aussi une mise en œuvre technique (collecte des événements, élaboration des indicateurs, etc.) mobilisant des compétences dépassant souvent celles des concepteurs et utilisateurs de ces indicateurs (concepteurs de cours, enseignants, tuteurs, apprenants).

Afin de rendre cette tâche plus facile, il fallait développer un outil apportant un cadre général reposant sur la collecte et la modélisation de traces d'interaction selon un méta-modèle générique (théorie de la trace modélisée) et sur la gestion de transformations explicites des traces collectées pour obtenir les informations nécessaires au calcul, explicite également, des indicateurs.

L'outil SBT-IM[15] est une implémentation particulière de ce principe pour l'appliquer aux cours développés avec Moodle[16].

L'outil permet d'une part de concevoir les modèles et d'autre part de calculer les instances conformément à la conception. Des services de visualisation des traces et des indicateurs complètent l'outil pour faciliter la conception comme l'utilisation des indicateurs.

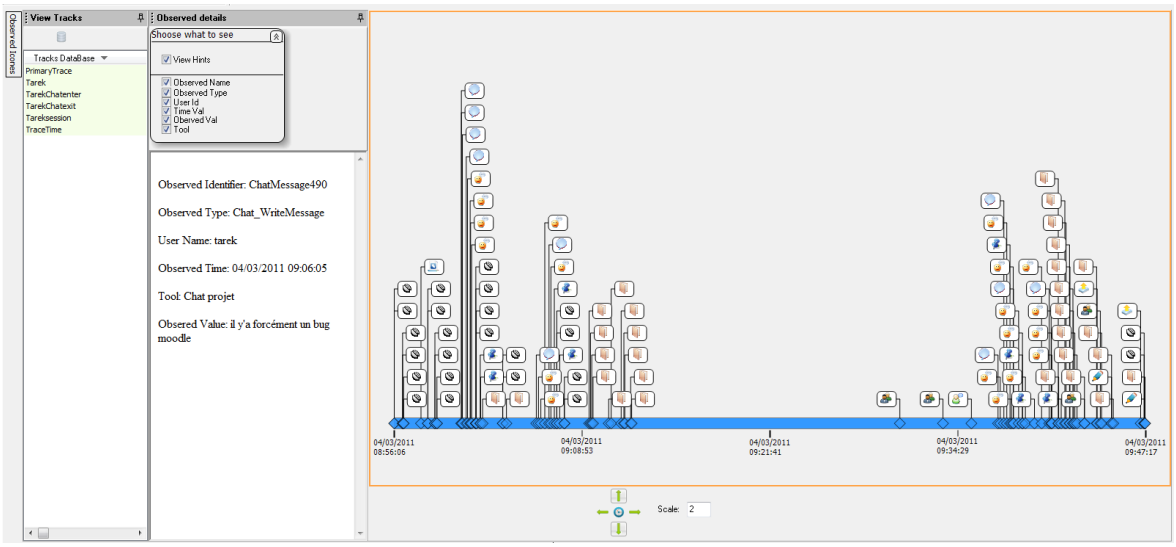


Figure 7 : Interface utilisateur SBT-IM[17]

Il nous est présenté sur cette interface utilisateur une ligne de temps avec des indicateurs de différents types. Ils sont calculés, possèdent différents types, contiennent des propriétés et sont temporalisés.

1.3.3 KTBS

Le « kernel for Trace- Based Systems[18] » est un outil développé au sein du laboratoire LIRIS par des membres de l'équipe SILEX Pierre-Antoine CHAMPIN et Françoise CONIL. Ce

[15] Thèse de Tarek DJOUAD : http://liris.cnrs.fr/Documents/Liris-5360.pdf
[16] MOODLE : Moodle est un environnement d'apprentissage libre (Learning Management System ou LMS). C'est une application web gratuite que les acteurs de l'éducation peuvent utiliser pour créer des sites d'apprentissage efficaces.
[17] Source : http://liris.cnrs.fr/tdjouad/SBTIMStudentTeacher/TBSIM/Home.html

noyau applicatif permet de générer un serveur de gestion de base de trace réifiant la théorie de la trace modélisée. En voici le diagramme de classes général :

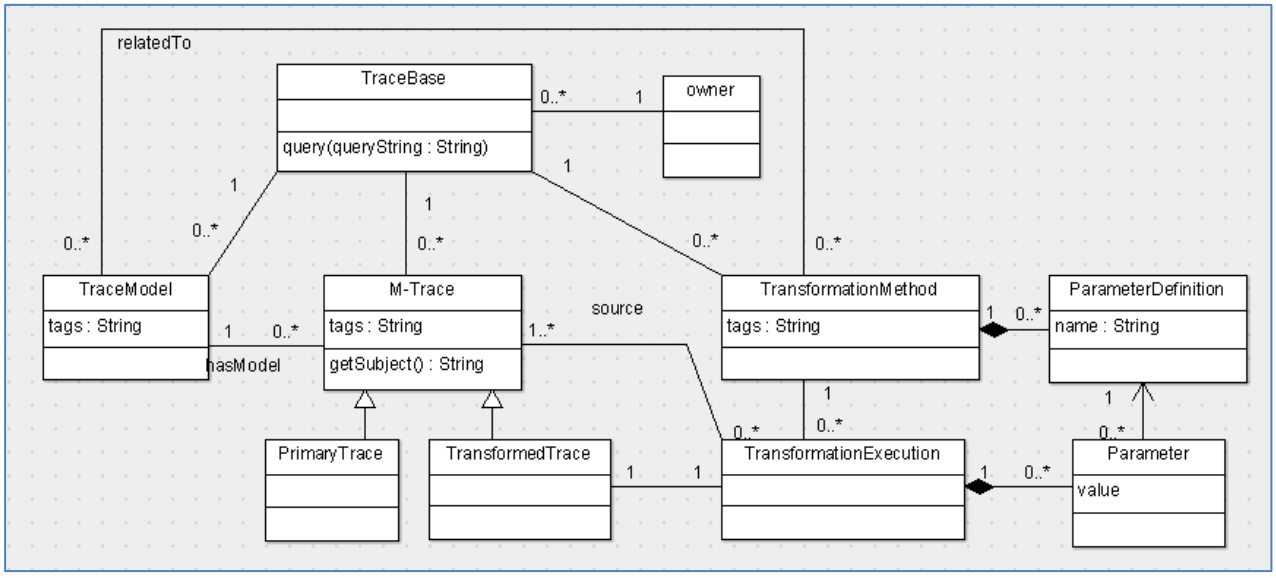


Figure 8 : Diagramme de classes général du kTBS

Le serveur kTBS lancé, une interface web permet de manipuler les ressources présentes ou d'en créer de nouvelles.

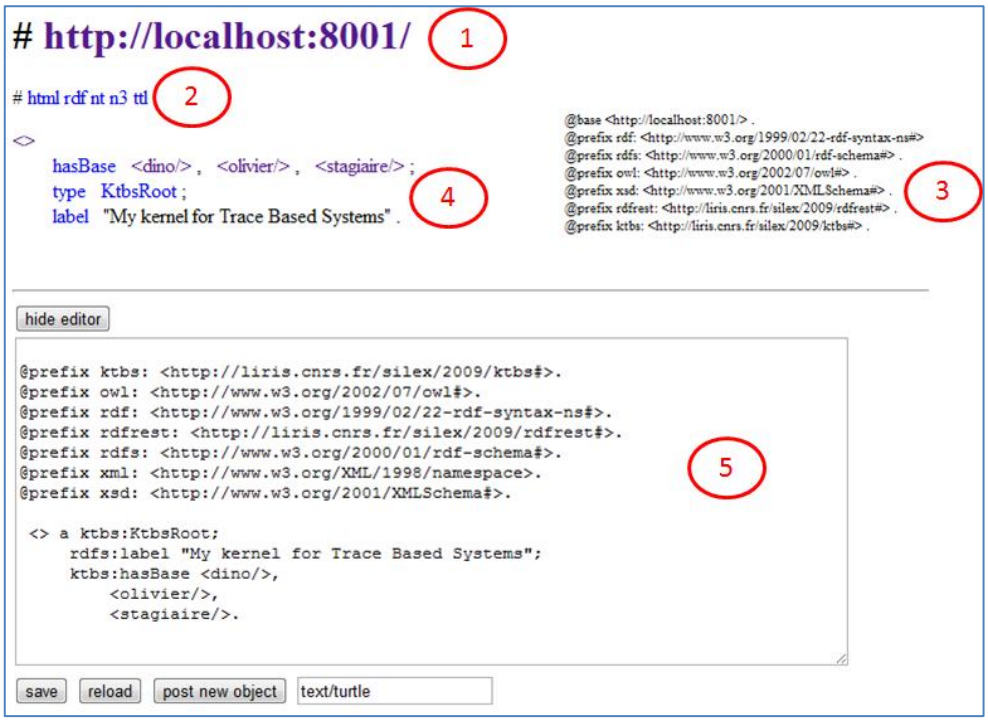


Figure 9 : Interface Web du kTBS

1. URL cliquable de la ressource consultée.
2. Liste des visualisations possibles pour la ressource.

---

[18] Source : http://liris.cnrs.fr/sbt-dev/ktbs/doc/

3. Liste des préfixes utilisé au sein du kTBS et donc de la visualisation Web.
4. Elément s de la ressource consultée, visualisation du graph RDF[19] lié à la ressource.
5. Possibilité de modifier le graphe RDF de la ressource sélectionnée, ou de créer une nouvelle ressource.

Le kTBS respecte l'architecture REST[20] ; *dans cette architecture, un composant lit ou modifie une ressource en utilisant une représentation de cette ressource. Une ressource est une chose nommable, qui peut évoluer avec le temps*. L'application de cette architecture au Web se comprend à travers quelques principes :

1. l'URI[21] est important : connaitre l'URI doit suffire pour nommer et identifier une ressource.
2. HTTP fournit toutes les opérations nécessaires (GET, POST, PUT, DELETE).
3. Chaque opération est auto-suffisante : il n'y a pas d'état.
4. Utilisation des standards hypermedia : HTML ou XML.

Cette architecture explique la forme que prennent les logs liés à l'utilisation du kTBS. En effet, lorsque l'on cherche à avoir accès à une ressource, une requête GET va être envoyée avec l'URI de la ressource désirée. En retour le kTBS renverra un statut (par exemple : 200 pour une requête correctement exécutée), ainsi que le poids en octet de la réponse.

1.3.4 Tatiana

Tatiana (Trace Analysis Tool for Interaction ANAlysts) est[22] un environnement conçu, par Grégory Dyke lorsqu'il était doctorant à l'école des mines de Saint-Etienne, pour la manipulation des divers types d'artéfacts.

Ces artéfacts peuvent être classifiés en trois espèces. **Les agrégations** résument les données correspondant à une certaine période de temps, afin de permettre la production d'indicateurs ou de statistiques. **Les enrichissements** sont des artéfacts qui ajoutent une connaissance créée par le chercheur après examen des données (ces analyses consistent en codages, annotations, liens, etc.). Les artefacts qui conservent la notion d'ordre des évènements et d'interactions temporellement situées sont nommés **rejouables**. Ce sont des objets qui peuvent être rejoués, synchronisés et enrichis. Le processus d'analyse est considéré comme une création itérative de nouveaux artéfacts (tels que les rejouables et les enrichissements), qui mettent en lumière la compréhension des données par le chercheur, ou lui permettent d'affiner cette compréhension.

[19] RDF : Resource Description Framework, modèle de graphe destiné à décrire de façon formelle les ressources Web et leurs métadonnées.
[20] REST : Architecture liée au World Wide Web, basée sur l'utilisation d'URI et de d'opération http (GET, POST, PUT, DELETE) http://fr.wikipedia.org/wiki/Representational_State_Transfer
[21] URI : Courte chaine de caractères identifiant une ressource sur un réseau et dont la syntaxe respecte la norme d'internet mise en place par le World Wide Web.
[22] Thèse de Gregory DYKE : http://dl.dropbox.com/u/223827/2009-dyke-thesis.pdf

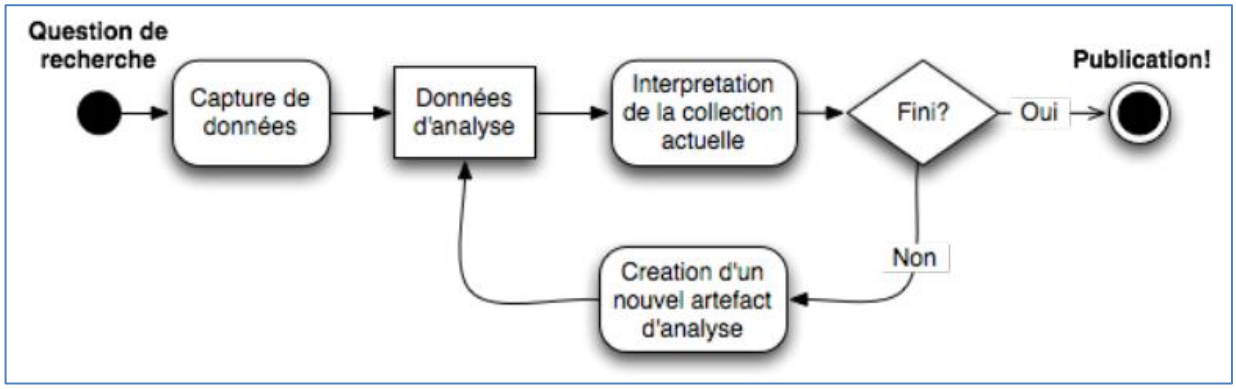


Figure 10 : Représentation graphique du modèle de traitement de Tatiana

Le logiciel Tatiana repose sur un certain nombre de concepts et composants fondamentaux (*cf. Figure 10*) et particulièrement autour de la notion de rejouable. Ces rejouables peuvent être créés automatiquement (par exemple, par importation de données extérieures ou par des transformations) ou manuellement. Une fois créés, ils bénéficient de quatre fonctionnalités : ils peuvent être **transformés, enrichis, visualisés ou synchronisés**. Afin de coordonner ces fonctionnalités, les rejouables sont enregistrés sous un format pivot interne à Tatiana, basé sur une représentation sous forme d'évènements ayant un certain nombre de propriétés.

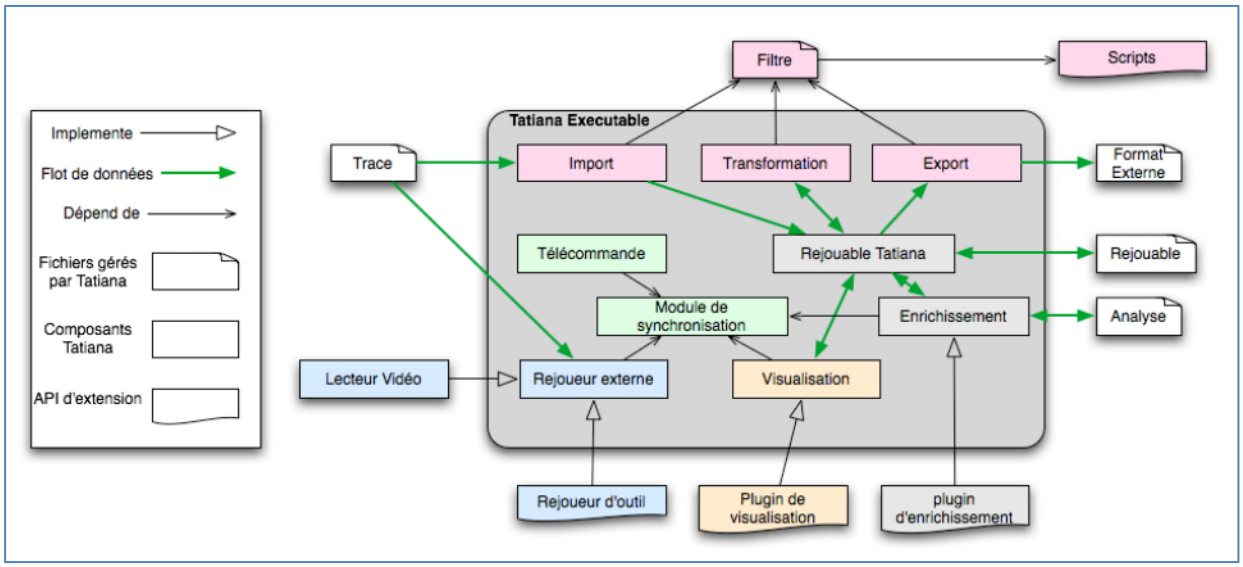


Figure 11 : Architecture de Tatiana

L'application Tatiana, est une application de type client-lourd qui permet l'affichage de différents rejouables (*cf figure 10*). Ainsi l'on peut voir, la trace d'un éditeur de texte partagé (haut gauche), transcription des dialogues (milieu gauche), unités de rédaction (haut centre), visualisation des reformulations (bas gauche), ainsi que la « télécommande » (bas, droit) permettant la synchronisation avec des outils extérieurs tels que le rejoueur DREW (haut droit) ou un afficheur vidéo (milieu droit).

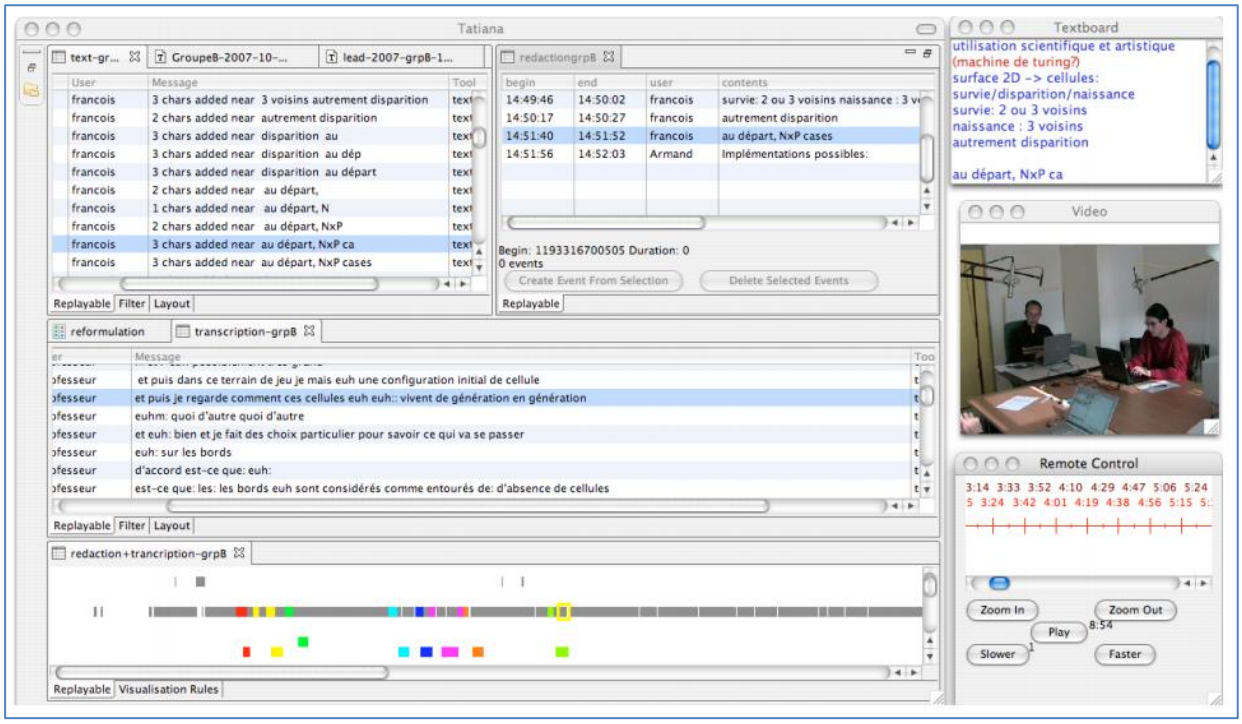


Figure 12 : Affichage de différents rejouables dans Tatiana

1.3.5 TrAVis

TrAvis (Tracking Data Analysis and Visualization platform) a[23] été développé, par Madeth MAY lorsqu'il était doctorant à l'INSA de Lyon, pour assister les acteurs d'une formation à distance (enseignants-tuteurs et apprenants) lors de l'exploitation de traces CMC. En voici (*figure 13*) son architecture globale :

[23] http://hal.archives-ouvertes.fr/docs/00/38/58/17/PDF/Jocair-Draft.pdf

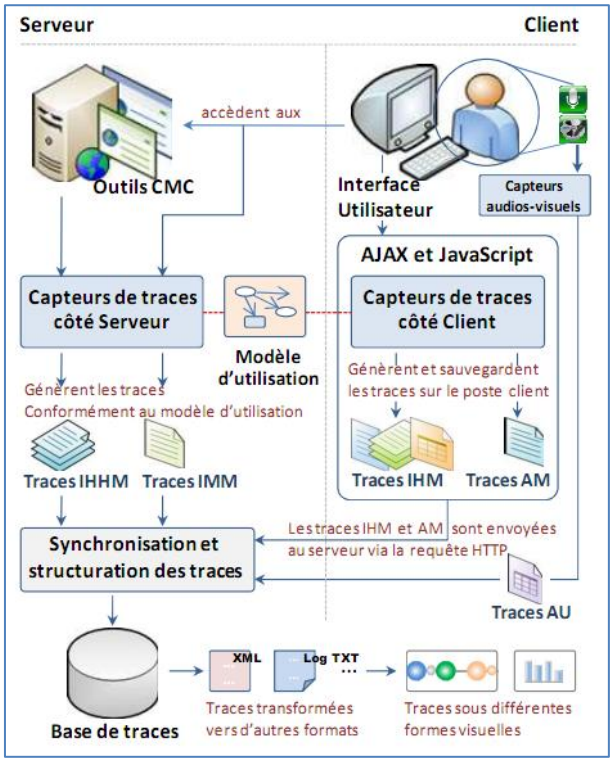


Figure 13 : Architecture globale de TrAVis

Figure 14 : Vue principale de TrAVis

La vue principale de la plateforme TrAVis (*figure 14)*, fournit un certain nombre de fonctionnalités permettant aux utilisateurs d'accéder facilement à la base de traces, d'analyser et de visualiser les traces sous différentes formes ainsi qu'à différentes échelles. TrAVis est conçue pour être accessible non seulement aux enseignants-tuteurs, mais aussi aux apprenants. Ainsi, les enseignants peuvent analyser des traces en construisant les indicateurs sur les interactions des apprenants. Quant aux apprenants, ils peuvent se servir de TrAVis comme outil réflexif donnant une visualisation globale sur leurs activités de communication en fournissant des indicateurs leur permettant de prendre conscience de leur propre comportement et de celui des autres. Dans un but de partage avec les utilisateurs d'outil CMC[24] dans les communautés académiques, TrAVis a été développée en utilisant des langages de développement Open Source comme PHP, JavaScript et AJAX. Afin de rendre TrAVis capable d'exploiter des traces CMC produites par d'autres systèmes de traçage, le système a été développé sous forme de cinq composants indépendants :

[24] CMC : (Computer Mediated Communication) ensemble de moyens informatiques au service de la communication entre humains

1. **Interface** : permet aux utilisateurs d'accéder aux fonctionnalités de TrAVis en faisant des requêtes de manière simplifiée,
2. **Traitement de données** : s'occupe d'interroger la base de traces avec les paramètres provenant du composant « interface »,
3. **Analyse de données** : extrait et calcule les indicateurs en fonction des données soumises par le composant traitement de données,
4. **Transformation de données** : transforme les indicateurs calculés en une forme précise pour la visualisation,
5. **Visualisation des données** : représente les indicateurs calculés sous différentes formes graphiques et avec différentes échelles.

1.4 Bilan de l'étude de l'existant

Les applications étudiées précédemment (cf 1.3), sont de grandes qualités et fonctionnent conformément aux attentes auxquelles elles répondent. Elles imposent néanmoins quelques contraintes.

En effet, les applications **SBT-IM** et **Tatiana** sont des applications client lourdes ce qui impose de devoir les installer sur le poste client sur lequel on veut les utiliser. **Abstract** ne permet pas une aide à la saisie de règle de transformation par un utilisateur non informaticien et ne permet pas d'afficher un corpus de trace modélisé permettant l'aide à la compréhension de plusieurs niveaux de trace. **TrAVis** est très orienté activité CMC ce qui n'est pas forcément le meilleur angle d'attaque pour étudier l'aide à l'observation.

Chapitre 2 Etude

2.1 Cahier des charges

2.1.1 Description de la demande

La demande est de créer une plateforme applicative de type « banc d'essais » dédiée à la gestion des traces d'activités provenant de sessions de formation sur simulateur pleine-échelle. Tenant compte des applications déjà présentes dans l'équipe SILEX, la plateforme devait avoir 3 principales fonctions :

1. Le stockage et la gestion des traces d'interaction (principe d'une base de données) en fonction des modèles de traces fournis par l'équipe du projet.
2. Des opérations sur les traces : requêtes et transformations de traces.
3. Couplage avec des prototypes existants au laboratoire : Visualisation des traces, analyses statistiques.

Des démarches d'analyse des besoins, d'étude fonctionnelle et de prototypage ont suivi. Il s'agissait de s'appuyer le plus possible sur les travaux et les prototypes déjà existants au sein de l'équipe SILEX. Une partie « benchmarking » a aussi été mise à profit pour déterminer des technologies à mettre en place, le tout en accord avec les besoins et l'organisation de l'équipe SILEX.

2.1.2 Contraintes

La plateforme applicative devait suivre une architecture client-serveur de manière à être accessible par un navigateur Web afin de ne nécessiter aucune installation sur les postes clients. Afin de compléter au mieux les travaux de recherche qui étaient en lien avec elle, la plateforme devait utiliser l'outil de gestion de base de trace réifiant la théorie de la trace modélisée : le kTBS. Pour permettre le partage de cette application et autoriser son utilisation dans un environnement professionnel il fallait qu'elle gère l'internationalisation et qu'elle implémente un système d'authentification des utilisateurs notamment pour déterminer différents rôles applicatifs. De plus, l'ensemble de ces développements devait utiliser au maximum des outils et langages sous licence libre. Enfin, l'ensemble de la production devait être déployable sur plusieurs environnements serveur (Windows, Unix, Macintosh) et correctement documenté.

2.2 Gestion de projet

2.2.1 Gestion des ressources

Pendant toute la durée du stage, le projet que nous avons décidé d'élaborer avec Olivier CHAMPALLE, visait à réaliser un prototype logiciel utilisé pour la découverte et la capitalisation de la connaissance issue de simulateur pleine échelle. Afin de réaliser au mieux ce projet nous avons établi les différentes ressources inhérentes à ce projet : humaines et techniques.

2.2.1.1 Ressources humaines

Les principales personnes impliquées au sein de ce projet sont issues de l'équipe SILEX. Elles prennent part de manière active à la réflexion et la réalisation de ce projet. Nous pouvons ainsi identifier :

Amaury BELIN : doctorant en informatique. Il poursuit une thèse dont le sujet est *Assistance à l'activité intellectuelle dans des environnements numériques - Application à la lecture active multimédia*. Nous avons réfléchi ensemble sur des réponses ergonomiques aux problèmes de construction de requête de données issues de simulateur pleine-échelle.

Olivier CHAMPALLE : doctorant en 2ème année de thèse, il est le principal investigateur de ce projet. En effet, ce prototype apporte une implémentation des questions de recherche auxquelles Olivier répond au sein de sa thèse. Il est à la fois la maitrise d'ouvrage, le directeur du projet et un testeur du projet.

Pierre-Antoine CHAMPIN : maître de conférences en informatique, il est à l'origine de la création d'un système de gestion de base de données de traces modélisées appelé kTBS.

Damien CRAM : docteur en informatique, il a réalisé l'API ktbs4j permettant l'accès au KTBS par une application développée en Java.

Tarek DJOUAD : doctorant en informatique dans le domaine des Connaissances et Systèmes Complexes (CoSyCo). Il est l'auteur du logiciel SBT-IM permettant le calcul d'indicateur dans la plateforme d'apprentissage Moodle[25].

Olivier GEORGEON : docteur en sciences cognitives et principal initiateur de l'application Abstract. Par le biais de l'élaboration de cette application, il a été confronté à des problématiques proches de celles qui émergent de l'élaboration du prototype et est donc une ressource expérimentée tant sur le développement que sur les questions de recherches liées au développement d'un tel prototype logiciel.

Karim SEHABA : maître de conférences en informatique, ses intérêts de recherche portent sur le comportement adaptatif des systèmes interactifs. Il est un des encadrant d'Olivier CHAMPALLE.

2.2.1.2 Ressources techniques

Afin d'organiser les différentes étapes liées à l'élaboration du projet, j'ai proposé à Olivier CHAMPALLE différents outils liés à l'organisation de projet que j'ai pu rencontrer au cours de mon expérience professionnelle.

[25] Une licence est dite libre si les conditions d'utilisation qu'elle définit respectent les droits d'utiliser, d'étudier, de modifier, de dupliquer ou de diffuser l'œuvre sur laquelle porte la licence.

Tel **Subversion**[26] pour gérer le code source de l'application, permettant le travail collaboratif, l'établissement de différentes versions applicatives et facilitant les processus de sauvegardes. Au cours du projet nous avons dû utiliser la technologie **GIT**[27] car les applications KTBS et ktbs4j sont hébergées sur des dépôts GIT.

L'utilisation d'une application web de suivi des bugs **Trac**[28] nous a permis, de saisir les retours de recette applicative, de mettre en corrélation les développements de correctifs avec les bugs qu'ils corrigent et de suivre la progression du projet.

Afin de faciliter l'échange et le travail collaboratif de documents nous utilisions la plateforme **Dropbox**[29] permettant de partager des fichiers de manière simple et conviviale. **Google documents** a aussi pu être utilisé en fonction des besoins.

De plus, un **client FTP** a été utilisé pour les transferts de fichiers éventuels entre nos plateformes et le serveur applicatif.

La plateforme de développement était celle présente sur le matériel qui nous a été fourni c'est-à-dire un **ordinateur portable** sous **Windows7** avec l'environnement de développement intégré **Eclipse**.

Néanmoins l'accès à d'autres systèmes étant possible au sein de l'équipe, le déploiement de l'application a été testé sur plateforme Macintosh, Linux (Ubuntu) et Windows.

2.2.2 Gestion du temps

La durée de ce projet est conditionnée par la durée du stage et est donc de 9 mois à temps plein. Pour ma part, je travaillai dans un bureau d'ordinaire réservé aux doctorants, et je ne suis allé qu'une fois sur le site de la centrale du Bugey alors qu'Olivier CHAMPALLE y était 50 à 60% de son temps. Il était aussi possible de faire du télétravail notamment grâce au fait que nous ayons eu des ordinateurs portables et une connexion avec les serveurs du laboratoire.

2.2.3 Méthodologie

La méthodologie que nous avons choisie de mettre en place pour la conduite de ce projet était le développement agile avec une succession de cycle en V *(figure 15)*.

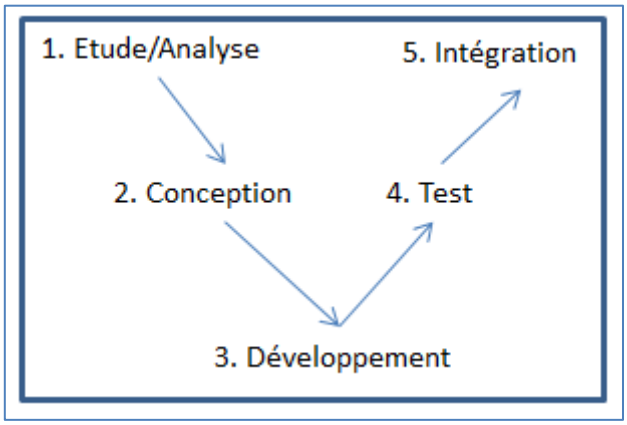


Figure 15: Modèle de conduite de projet de développement "Cycle en V "

[26] http://subversion.tigris.org/
[27] http://fr.wikipedia.org/wiki/Git
[28] http://trac.edgewall.org/
[29] http://www.dropbox.com

L'avantage de ce modèle est de pouvoir facilement travailler en mode incrémental ; ainsi l'on peut aisément concevoir un noyau applicatif comportant un nombre restreint de fonctionnalité et l'augmenter au fur et à mesure des validations utilisateurs. Ce modèle est idéal pour gérer un projet qui ne peut être spécifié dans son ensemble car fluctuant avec le temps et devant produire rapidement une application fonctionnelle. En effet, les problématiques auxquelles doit répondre l'outil applicatif sont des problématiques de recherche et ne sont pas forcément immuables dans le temps. C'est pourquoi, être capable d'informatiser une situation simple dans un premier temps puis ajouter des fonctionnalités au fur et à mesure de l'avancé des réponses liées aux problématiques de recherche, nous a décidé à adopter cette méthodologie.

Afin de nous aider à la conduite de projet nous nous sommes appuyés sur le logiciel Trac. En effet celui-ci nous a permis de fixer des jalons temporels et fonctionnels liés à l'application et à la poursuite des développements. De plus, il était possible d'intégrer à la plateforme de développement un plug-in (Mylyn[30]) permettant d'interagir avec ce logiciel. Ainsi chaque développement était en relation avec une tâche prédéfinie et chaque correction était en relation avec un dysfonctionnement répertorié.

En faisant référence à la section 1.4 « Bilan de l'étude de l'existant », nous pouvions partir du principe que nous n'étions pas les premiers à travailler sur la problématique très large et complexe qu'est l'aide à l'observation, ainsi même que dans l'idée d'exploiter le concept de trace modélisée pour y faire face.

Néanmoins, compte tenu de la spécificité de la problématique de recherche d'Olivier CHAMPALLE et de nos besoins et objectifs, notre angle d'approche était différent de ce qui avait déjà été fait et c'est pourquoi nous avons choisi de réaliser un nouveau développement.

[30] http://www.eclipse.org/mylyn/

Chapitre 3 Réalisation

3.1 Moyens mis en œuvre

3.1.1 Outils de développement

Pour des raisons de praticité la plateforme de développement dédiée à la création du projet était sous environnement Windows. En effet, l'ordinateur portable fourni par le laboratoire possèdait cet environnement et c'est celui que j'ai le plus communément utilisé professionnellement.

Pour la réalisation de différentes plateformes logicielles déjà existantes, différents langages de programmation ont été utilisés : PHP, Delphi, HTML, XML et JavaScript principalement. Pour ma part ayant une expérience de plusieurs années de développement avec le langage Java mon choix se porta naturellement sur ce langage. Ce choix a été consolidé par le fait qu'une API Java d'accès au SGBT KTBS était en cours de développement au sein du laboratoire.

En ce qui concerne le choix du système de gestion permettant de gérer les bases de traces de l'application, mon choix s'est naturellement porté sur le KTBS ; étant une implémentation réifiant la théorie de la trace modélisée et développée au sein du laboratoire.

L'environnement de développement Intégré (IDE) était Eclipse Galileo (3.6). Cet environnement est la référence pour les développements Web utilisant les langages de programmation au sein de la plateforme J2EE[31]. Il existe aussi d'autres éditeurs logiciels pour programmer des plateformes J2EE, mais ceux-ci sont moins modulaires qu' Eclipse et possèdent une communauté d'utilisateurs plus faible. De plus, je possède une expérience de plus de 5 ans de développement J2EE sous cet environnement.

3.1.2 Bibliothèques de patterns et langages

Après que nous ayons choisi l'environnement de développement, il convenait de choisir les bibliothèques à utiliser. En effet, de multiple framework[32] liés au développement sous plateforme J2EE existent et permettent des développements plus rapides et plus qualitatifs.

Le framework de présentation Struts[33] en sa version 2 implémente le principe Modèle-Vue-Contrôleur[34]. Celui-ci permet un découpage architectural et modulaire rependu au sein des

[31] J2EE : est une spécification pour la technique Java de Sun plus particulièrement destinée aux applications d'entreprise.

[32] Framework : Représente un ensemble de plusieurs classes offrant des fonctionnalités diverses telles que l'authentification, l'accès aux bases de données, la gestion de formulaires web etc…

développements Web. Il est alors aisé de séparer les données, des traitements de la présentation et permet ainsi une modularité accrue. En effet, il devient alors possible de modifier l'une de ces trois parties en impactant au minimum les deux autres.

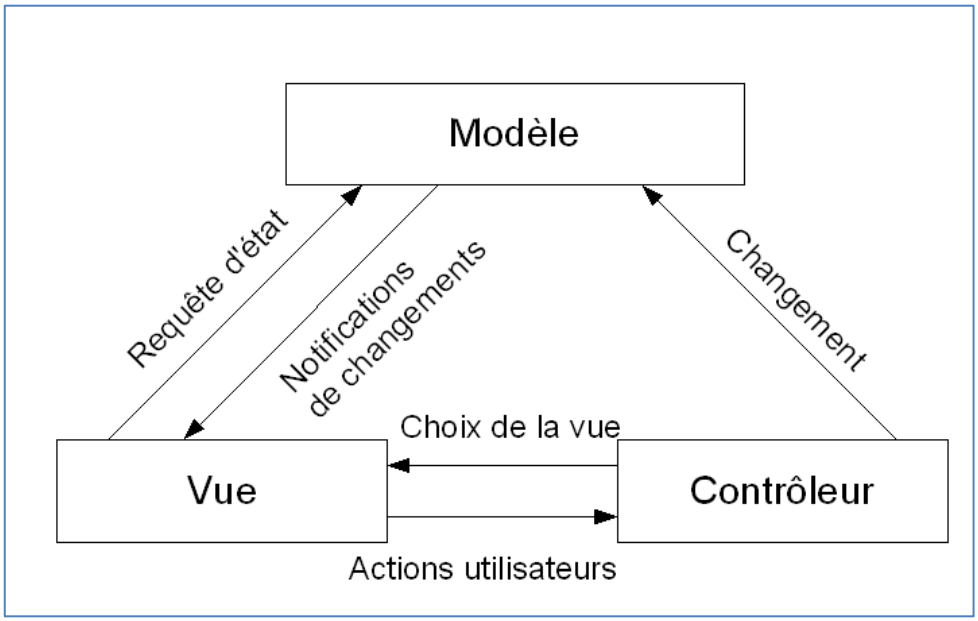


Figure 16 : Interactions MVC

Afin d'augmenter ce modèle architectural, notamment au niveau de la partie « Vue », la technologie Ajax[35] par le biais du langage JQuery[36], a été utilisée de manière à inclure de la dynamique côté client Web. En effet, les avantages liés à l'utilisation de ce genre de bibliothèque sont nombreux :

1. génération de code JavaScript compatible multi-navigateur client web,
2. appel synchrone et asynchrone possible entre l'interface et le serveur applicatif,
3. de nombreuses fonctionnalités déjà implémentées,
4. une forte communauté d'utilisateurs et de développeurs.

Concernant l'accès à la base de données, une API cliente développée en JAVA permet de simplifier la communication entre l'application et le système de gestion des données. Cette API nommée ktbs_4j[37] fournit l'ensemble des méthodes et objets permettant de manipuler les ressources stockées dans le KTBS.

Enfin pour la présentation graphique des données, les langages XML, XSL et SVG ont été utilisés.

---

[33] Struts2 : Framework maintenu et développé par la communauté apache servant au développement d'application web J2EE.
[34] MVC : le Modèle-Vue-Contrôleur est une méthode de conception organisant les IHM d'une application logicielle. Les IHM sont alors divisées en un modèle de données, une vue (présentation, interface utilisateur) et un contrôleur (logique de contrôle, gestion des événements, synchronisation), chacun ayant un rôle précis dans l'interface.
[35] Ajax : Asynchronous Javascript and XML, méthode de développement de site Web basée principalement sur les technologies Dom, XMLHttpRequest et Javascript.
[36] JQuery : bibliothèque Javascript Libre contenant de fortes fonctionnalités Ajax
[37] Annexe 2 : Guide kTBS API ktbs4j Sparql 1.1_4j

3.1.3 Serveur applicatif

Différents serveurs applicatifs pour environnement J2EE existent ; citons notamment Tomcat[38], JBoss[39] et Websphere Application Server (WAS)[40]. Considérant la taille du projet ainsi que les exigences liées à son exécution, un serveur de servlet[41] comme Tomcat suffisait amplement aux besoins d'hébergement et d'exécution de l'application que nous souhaitions réaliser.

3.2 D3KODE

L'élaboration d'un nom pour l'application que nous allions développer impliqua l'ensemble des acteurs du projet. Il fallait trouver un nom qui soit évocateur de ce que celle-ci faisait, c'est pourquoi nous nous sommes rapidement dirigés vers un acronyme. En effet cela nous offrait la possibilité d'utiliser plusieurs mots pour décrire et caractériser notre réalisation. De plus, nous avions comme objectif de donner un sens à cet acronyme autant en français qu'en anglais. Regroupant l'ensemble des termes inhérent à notre réflexion nous avons collégialement décidé de l'appeler : D3KODE.

Cet acronyme signifie :

***D**efine, **D**iscover, and **D**isseminate **K**nowledge from **O**bservation to **D**evelop **E**xpertise*

(Définir, Découvrir et Partager la Connaissance d'Observation pour Développer l'Expertise).

3.2.1 Spécifications fonctionnelles

Même si normalement, elles devaient être réalisées avant même le début des développements, les spécifications fonctionnelles de D3KODE ont été réalisées au fur et à mesure de l'avancée des itérations de construction du projet. Elles couvraient l'intégralité des fonctionnalités de l'application, que ce soit le chargement, le traitement ou la représentativité des données.

3.2.1.1 Chargement de données

D3KODE ne permet pas de traiter des données en temps réel mais à postériori. En effet, les simulateurs de formation sont nombreux et variés, et leurs données potentiellement exploitables peuvent être de différentes natures, ne permettant pas forcément un interfaçage aisé. De plus un traitement en temps réel, imposerait des contraintes techniques lourdes n'entrant pas dans le cadre de ce projet. Enfin, avant toute étude sur des connaissances, il convient de travailler de manière empirique et donc à base d'évènements déjà produits. Il fallait donc créer une interface permettant de charger les données issues de simulateurs au sein de l'application.

Les données issues des simulateurs sont qualifiées de **ressources**. Elles possèdent un type qui caractérise ses types d'attributs. Ces types d'attributs peuvent être multiples mais chaque ressource doit avoir au moins des attributs de temporalité (début et fin). C'est alors que l'on va pouvoir qualifier ces ressources d'**observé** et leur types de **type d'observé**. L'ensemble des

[38] Tomcat : Moteur de servlet incorporant un serveur Web
[39] JBoss Application Server : serveur d'applicationJ2EE libre écrit en Java
[40] WAS : Plate-forme applicative regroupant serveurs d'applications et de multiples outils de développement et de déploiement.
[41] Une Servlet est une classe Java qui permet de créer dynamiquement des données au sein d'un serveur HTTP

types d'observés et des types d'attributs possibles et identifiables lors d'une simulation seront regroupés au sein d'un **modèle**. L'ensemble des observés représente la **trace modélisée d'activité** (M-Trace d'activité) qui sera donc en adéquation avec le **modèle de trace** précédemment défini. On qualifie la trace d'activité résultante directe de la simulation de **trace modélisée première ou M-Trace première**. Il faut donc deux IHM[42] : l'une pour la création de modèles de trace et une autre pour la création de M-Trace première respectant un des modèles de trace.

3.2.1.2 Création de modèle de transformation

Suite à la création des modèles de traces D3KODE doit permettre de spécifier des **modèles de transformation**. Une transformation est un mécanisme qui permet d'effectuer des **opérations** sur une M-Trace d'activité ou première en vue de permettre une interprétation plus fine de l'activité qui a eu lieu. Différentes opérations peuvent être réalisées :

1. Filtrage temporel d'une M-Trace : consiste à créer une M-Trace composée des observés de la M-Trace source filtrés par des bornes de début et de fin.

   Exemple : Créer une M-Trace composée des observés ayant eu lieu entre 12h05 et 12h15.

2. La fusion de deux M-Trace : consiste à créer une M-Trace regroupant les observés des traces fusionnées ; cela implique que celles-ci possèdent le même modèle de M-Trace.

   Exemple : Créer une M-Trace composée de 2 M-Traces ayant le même modèle de M-Trace.

3. La création en fonction de critères complexes : consiste à créer une M-Trace ne possédant pas le même modèle de M-Trace que la M-Trace source. Les observés créés dans la M-Trace cible seront créés par calcul(s) appliqué(s) sur les observés de la M-Trace source.

   Exemple : Créer une M-Trace contenant des acquittements d'alarme. Un acquittement d'alarme peut être créé si une même alarme a eu son statut passé de *activé* à *non-activé* dans un intervalle de temps inférieur à 5 minutes.

Le modèle de transformation doit donc spécifier le modèle de M-Trace source sur lequel il va s'appliquer ainsi que le modèle de M-Trace cible de la trace résultante à la transformation. Les IHM permettant de créer ces modèles de transformations doivent guider au maximum l'utilisateur non expert en informatique.

3.2.1.3 Transformation de M-Trace

A partir des modèles de transformations et d'une M-Trace d'activité il est alors possible d'exécuter une transformation de M-Trace. Cette opération demande donc un modèle de transformation, une M-Trace source et un libellé pour désigner la M-Trace cible. Les calculs inhérents à cette transformation sont effectués par le système de gestion à base de trace (le KTBS). A la suite de cette transformation, le résultat, présenté sous la forme de nombre d'observé produit, est retourné à l'utilisateur.

3.2.1.4 Représentation graphique de M-Trace et de transformations

D3KODE doit être en mesure de fournir une IHM de représentation graphique des M-Trace d'activités ainsi que des transformations ayant permis de les construire. Cette interface

[42] IHM : Interface Homme-Machine qualifie l'ensemble des composants graphiques utilisés par un utilisateur pour utiliser une application.

doit permettre de faire figurer de manière intelligible l'ensemble des observés de la **totalité** de la M-Trace ou **d'une portion temporelle** de celle-ci. La figuration des observés doit être paramétrable et intelligible. L'ensemble des attributs des observés doivent aussi pouvoir être consultés en plus du libellé qui caractérise chacun des observés. De plus un mécanisme de zoom doit être implémenté de manière à permettre une lecture correcte d'un corpus de M-Trace.

1.2.2 Architecture technique et fonctionnelle de D3KODE

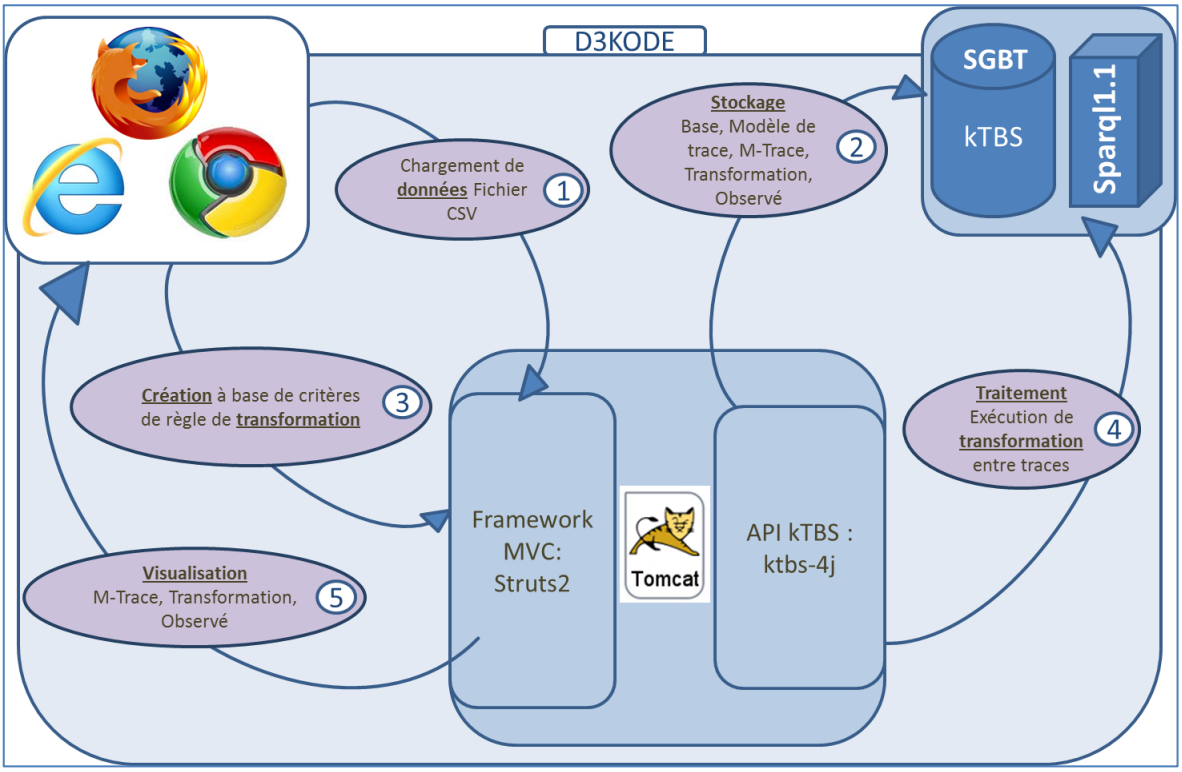


Figure 17 : Architecture de la plateforme D3KODE

D3KODE permet 5 principales fonctionnalités identifiées dans le cahier des charges :

1. Chargement de données au format fichier CSV. Le format CSV permet une structuration simple des données séparées par des retours à la ligne ou des virgules.
2. Stockage des ressources liées à la théorie de la trace modélisée et utilisées dans l'application
3. Création de transformation aidée par une Interface Home Machine de « type calculatrice »
4. Traitement de l'exécution des requêtes de transformation sur les traces stockées dans le kTBS
5. Visualisation des traces d'activité de manière graphique et paramétrable

3.2.3 Implémentations clés

Au cours du développement de D3KODE, différentes implémentations ont dû être mises en œuvre pour produire une application Web répondant aux exigences du cahier des charges. Nous détaillons ici les implémentations clés de l'application.

3.2.3.1 Gestion des composants de la trace modélisée

Comme précisé dans les spécificités fonctionnelles, D3KODE doit fournir les IHM permettant de construire les modèles de M-Trace. Pour ce faire nous avons choisi d'utiliser une

représentation (*figure 18*) en arbre (1) pour la visualisation des modèles. Chaque élément de l'arbre est cliquable et permet d'accéder à 2 formulaires de saisie : l'un de modification (2) l'autre d'ajout (3).

L'arbre possède un élément racine (*Root*) qui correspond au serveur KTBS. Cet élément racine contient la liste des bases contenues sur le serveur ; elle porte l'identifiant de la personne connectée à l'application (*dino*). A l'intérieur de cette base, sont contenus les modèles de M-Trace (*modèle de niveau 1*). Le modèle de M-Trace contient les types d'observés (ici : *Applique_NI5FS11*) avec leurs types d'attributs (*hasBegin* etc…). Parmi les 2 formulaires ; il y a le formulaire de détail de la ressource sélectionnée et le formulaire d'ajout d'un élément plus bas hiérarchiquement que celui sélectionné ; il a donc 3 cas possibles :

1. Si l'on sélectionne la base, la ressource fille est un modèle de trace.
2. Si l'on sélectionne un modèle de trace la ressource fille est un type d'observé.
3. Si l'on sélectionne un type d'observé la ressource fille est un type d'attribut.

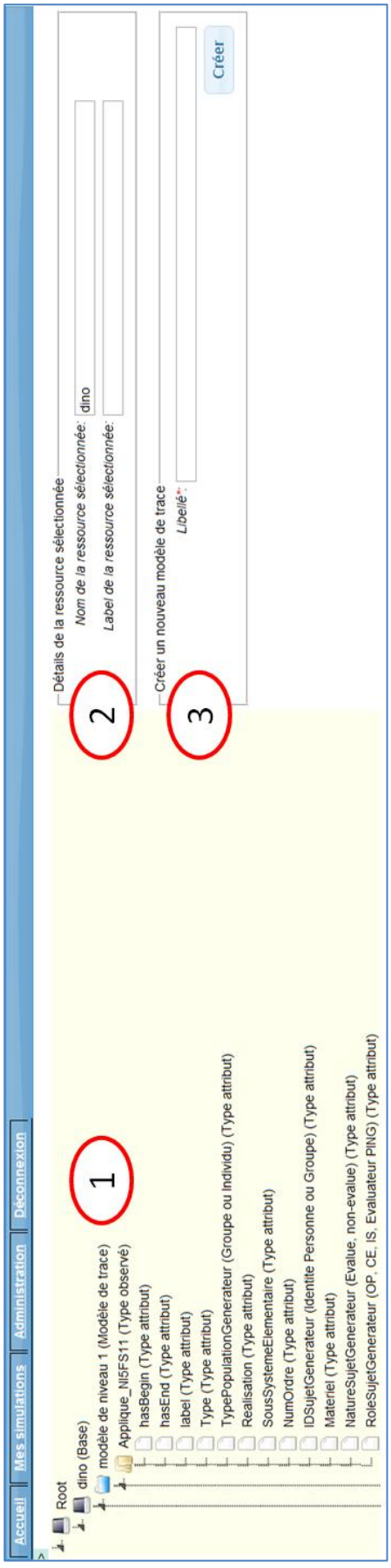


Figure 18 : IHM modèle de M-Trace

L'on peut ainsi créer/modifier l'ensemble des modèles de M-Trace et leurs constituantes. Mais il est aussi possible de créer le contenu d'un modèle de M-Trace par chargement d'un fichier au format CSV comme le montre la *figure 19*.

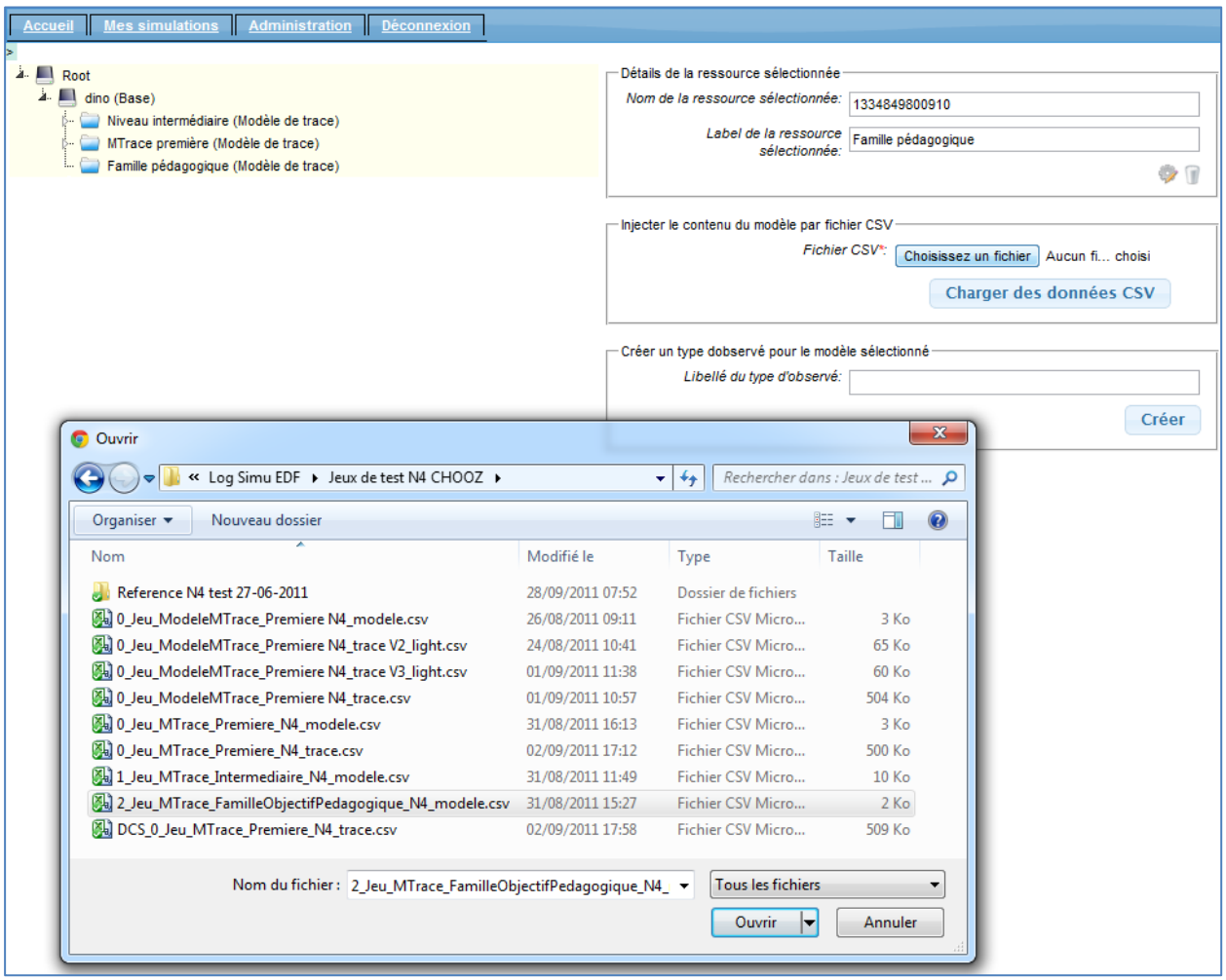


Figure 19 : Ajout de type(s) d'observé(s) par traitement de fichier CSV

Le format CSV permet une vision tabulaire simple (structurellement parlant) de données. Chaque donnée est séparée de la suivante par un « ; » ou par un retour à la ligne.

```
ObselType;Telephonie;NumeroOrigineAppel;STRING;True;NumeroReceptionAppel;STRING;True;TempsS
ObselType;ActionOperateur;IDSujetGenerateur (Identite Personne ou Groupe);STRING;False;Type
ObselType;EvenementAlarme;IDSujetGenerateur (Identite Personne ou Groupe);STRING;False;Type
ObselType;EvenementLogique;IDSujetGenerateur (Identite Personne ou Groupe);STRING;False;Typ
ObselType;EvenementProcedure;IDSujetGenerateur (Identite Personne ou Groupe);STRING;False;T
ObselType;ActionInstructeur;IDSujetGenerateur (Identite Personne ou Groupe);STRING;False;Ty
ObselType;RepereInstructeur;IDSujetGenerateur (Identite Personne ou Groupe);STRING;False;Ty
ObselType;Invalidites;IDSujetGenerateur (Identite Personne ou Groupe);STRING;False;TypePopu
MetaData;Simulateur;STRING;True;;;;;;;;;;;;;;;;;;;;;;;;;;
MetaData;NumTranche;INTEGER;True;;;;;;;;;;;;;;;;;;;;;;;;;;
MetaData;Formation;STRING;True;;;;;;;;;;;;;;;;;;;;;;;;;;
MetaData;Cursus;STRING;True;;;;;;;;;;;;;;;;;;;;;;;;;;
MetaData;Categorie;STRING;True;;;;;;;;;;;;;;;;;;;;;;;;;;
MetaData;NomSeance;STRING;True;;;;;;;;;;;;;;;;;;;;;;;;;;
MetaData;Phase;STRING;True;;;;;;;;;;;;;;;;;;;;;;;;;;
MetaData;NiveauMTrace;STRING;True;;;;;;;;;;;;;;;;;;;;;;;;;;
MetaData;DateDebut;DATETIME;True
```

Figure 20 : Extrait de contenu CSV d'un modèle de M-Trace

Au sein de l'application D3KODE (*figure 20*) voici la structure qui a été retenue :

1ère colonne : **type** de la ressource (ellipse bleue) → soit un **obselType** (type d'observé) soit **MetaData** (métadonnée du modèle de trace).

2ème colonne : **libellé** de la ressource (ellipse rouge)

Triplet de colonne suivant : **Nom de l'attribut**, **Type de l'Attribut**, **Obligation de présence** (ellipse verte) pour le type d'observé et seulement **type** et **obligation de présence** (ellipse jaune) pour les métadonnées.

3.2.3.2 Chargement de M-Trace première

Une fois les modèles de M-Trace effectués, il est possible de charger au sein de D3KODE les journaux de bords issus du (des) simulateur(s). Ces journaux sont sous forme de fichiers CSV formatés comme le montre la *figure 21*.

```
;;ActionInstructeur;;IDSujetGenerateur (Identite Personne ou Groupe);TypePopulationGenerateur (Groupe ou Individu);RoleSujet
27/06 11:04:35.96;27/06 11:04:35.96;ActionInstructeur;"ouvrir fenetre ""jdb"" 4   operateur";-;-;-;-
27/06 11:05:01.16;27/06 11:05:01.16;ActionInstructeur;"ouvrir fenetre ""JDB Autre Seance"" 5";-;-;-;-
27/06 11:05:13.20;27/06 11:05:13.20;ActionInstructeur;"abandon   ""JDB Autre Seance""";-;-;-;-
27/06 11:05:46.11;27/06 11:05:46.11;ActionInstructeur;"ouvrir fenetre ""jdb"" 7   reperes";-;-;-;-
27/06 11:05:56.41;27/06 11:05:56.41;ActionInstructeur;"ouvrir fenetre ""jdb"" 8   alphanum";-;-;-;-
27/06 11:10:00.32;27/06 11:10:00.32;ActionInstructeur;"abandon   ""Journal de bord Invalidite Alphanum - Seance courante""";
27/06 11:10:01.62;27/06 11:10:01.62;ActionInstructeur;"abandon   ""Journal de bord Repere Instructeur - Seance courante""";-
27/06 11:10:02.52;27/06 11:10:02.52;ActionInstructeur;"abandon   ""Journal de bord Actions Operateur - Seance courante""";-;
27/06 11:10:07.12;27/06 11:10:07.12;ActionInstructeur;gel;-;-;-;-
27/06 11:10:17.32;27/06 11:10:17.32;ActionInstructeur;charger init S092;-;-;-;-
27/06 11:15:02.50;27/06 11:15:02.50;ActionInstructeur;continu;-;-;-;-
27/06 11:15:46.53;27/06 11:15:46.53;ActionInstructeur;"ouvrir fenetre ""Synoptique"" 35   ""Synoptique systeme GEV numero 1"
27/06 11:15:55.73;27/06 11:15:55.73;ActionInstructeur;"ouvrir fenetre ""Liste des pannes du procede"" 36   repere ""ALT006_P
27/06 11:18:12.53;27/06 11:18:12.53;ActionInstructeur;"activer panne ""ALT006_PS""    taux ""val=0.000""";-;-;-;-
27/06 11:31:11.05;27/06 11:31:11.05;ActionInstructeur;"desactiver panne ""ALT006_PS""    taux ""val=0.000""";-;-;-;-
27/06 11:31:13.35;27/06 11:31:13.35;ActionInstructeur;"abandon   ""Liste des pannes du procede""";-;-;-;-;;;;;
27/06 11:50:26.41;27/06 11:50:26.41;ActionInstructeur;gel;-;-;-;-;;;;;
```

Figure 21 : Extrait de contenu CSV pour la création d'une trace modélisée

On peut détailler le contenu de ce fichier CSV comme suit :

Tout d'abord, la structure du type d'observé que l'on veut charger (rectangle bleu) : **2 colonnes vides** (correspondant à la date de début et à la date de fin), le nom du type d'observé et la liste des noms des types d'attribut du type d'observé.

Ensuite vient l'ensemble des lignes représentant chacune un observé. Chacune commence par **deux dates** (rectangle rouge) ; date de début et date de fin (au format DD/MM HH:mm:ss.SSS ou YYYY/MM/DD HH:mm:ss.SSS), suivi du **nom du type d'observé** (rectangle vert) et enfin de la liste **des valeurs de ses attributs** (rectangle jaune).

Une fois les modèles de M-Traces spécifiés il est possible de créer une M-Trace première (*figure 22*).

Pour cela il suffit d'utiliser un formulaire permettant de saisir :

1. le **libellé** pour la trace modélisée,
2. le **fichier CSV** utilisé pour injecter les observés,
3. **le modèle** auquel la trace correspond
4. **éventuellement** un intervalle de date pour le début et la fin de la M-Trace (par défaut seront récupérées la date de début la plus petite et la date de fin la plus grande).

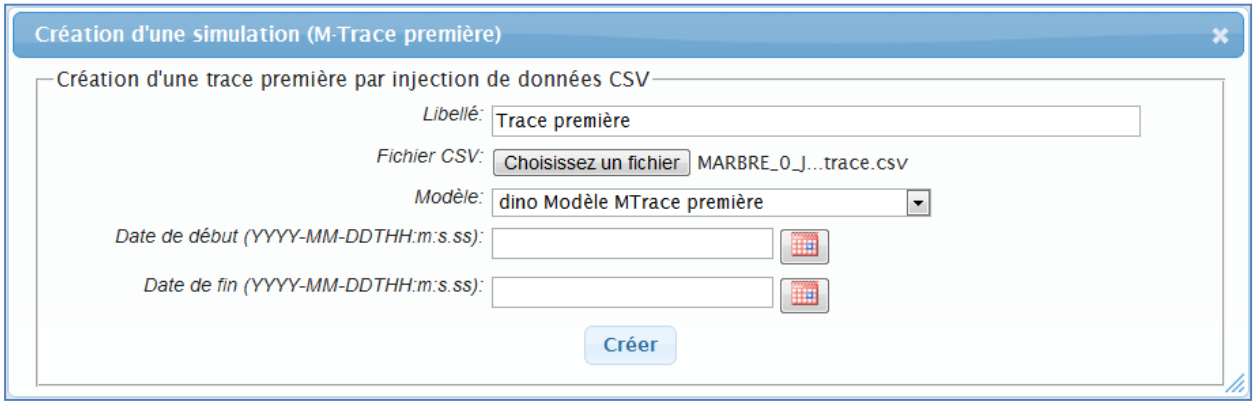


Figure 22 : Création d'une M-Trace par chargement de fichier CSV

Après clique sur le bouton **Créer** *(figure 22)* l'ensemble des lignes contenues dans le fichier CSV sera converti en observé et inséré dans la M-Trace créée *(figure 23)*.

	Date de création	Date de début	Date de fin	Libellé	Modèle de Trace	Liste types d'obsel
0	lun., 27 févr. 2012 14:30:45,847	mer., 27 juin 2012 11:00:09,005	mer., 27 juin 2012 12:00:20,008	Trace première	Modèle MTrace première	309 EvenementAlarme 127 Invalidites 4049 EvenementLogique 27 ActionInstructeur 63 ActionOperateur 1 RepereInstructeur 33 EvenementProcedure

Figure 23 : M-Trace créée par import de fichier CSV

Il est alors présenté sous forme tabulaire :

1. Un **indice** de ligne.
2. Une **date de création** qui correspond à la date à laquelle a été créée la M-Trace première.
3. Une date de début et de fin qui correspondent :
    a. Soit aux **dates de début du premier** observé et **date de fin du dernier observé**
    b. Soit aux dates qui ont été **choisies par l'utilisateur pour l'import du fichier CSV** ; la date début peut à ce moment-là être antérieur à la date de début du premier observé et la date de fin postérieur à la date de fin du dernier observé.
4. Un **libellé de M-Trace**, il servira, lors de la représentation graphique d'un corpus de trace, à identifier une trace par rapport à une autre.
5. Le **nom et le lien vers le modèle de trace** auquel la M-Trace correspond, il est alors possible si l'on suit ce lien d'arriver à la définition en arbre du modèle de trace sélectionné *(figure 24)*.
6. La **liste des différents types d'observés** présents à l'intérieur de la M-Trace avec leur fréquence d'apparition.

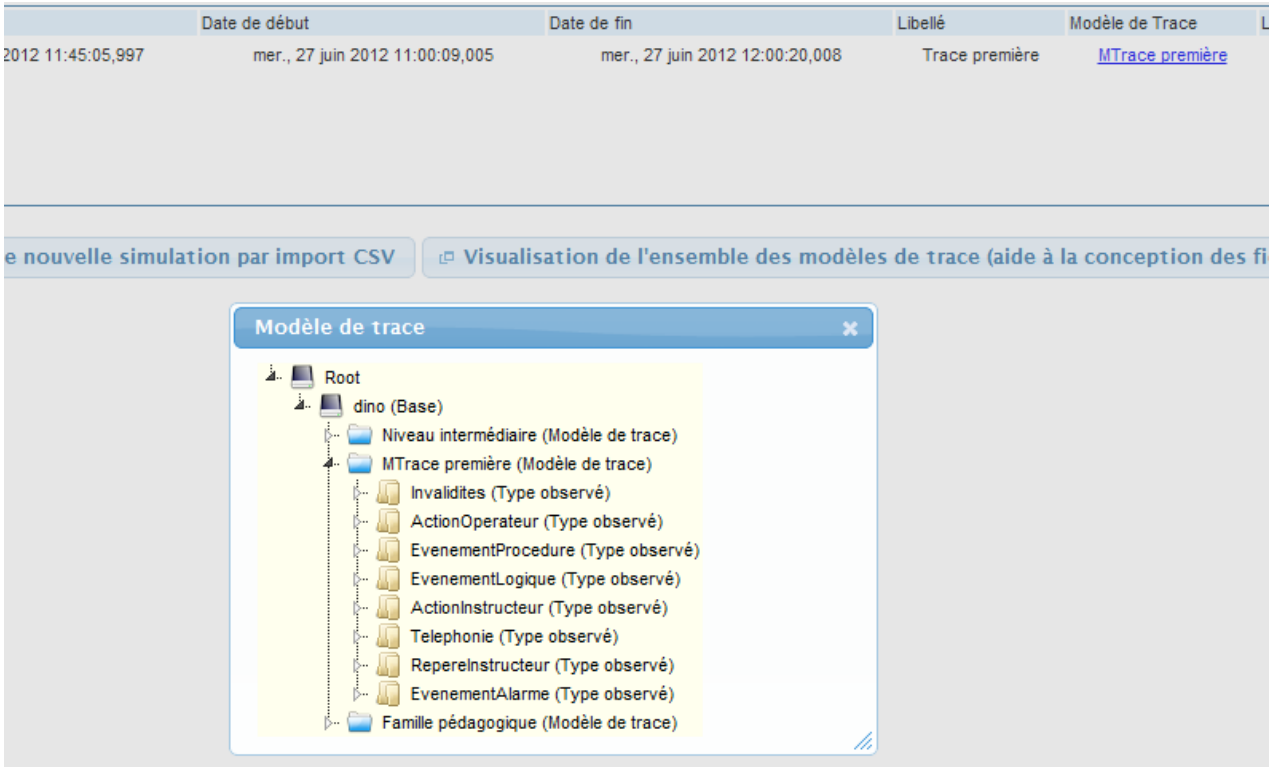


Figure 24 : Pop-up modèle de trace

Donc à ce stade, nous avons des modèles de traces qui décrivent les constituantes d'une M-Trace et des M-Traces qui contiennent des observés conformes à un modèle de trace. Intéressons-nous alors à la construction de modèle de transformations.

3.2.3.3 Construction de modèle de transformation

Notre but étant de faire émerger la connaissance à partir d'activités tracées, il convient de fournir un moyen permettant d'interpréter l'expérience tracée. Pour cela la théorie de la trace modélisée propose un mécanisme de transformation de M-Trace. Ce mécanisme basée sur un ensemble de règles permet de construire de nouveaux observé à partir d'un ou plusieurs observés. L'on pourra par exemple produire un observé « prise en compte alarme A15B25 »

lorsqu'on a trouvé un observé « alarme A15B25 activé » (A1), « alarme A15B25 » (A2) désactivé et « action opérateur sur alarme A15B25 » (AO1) dans l'ordre temporel correcte. Cet ordre est définis par le fait que l'ordre d'apparition temporelle de ces 3 observés soit ainsi : A1 – AO1 – A2 et ce dans un intervalle de temps définis : 250 secondes par exemple. La transformation permet donc à partir d'observés contenus dans une M-Trace et respectant un modèle de M-Trace de produire de nouveaux observés dans une nouvelle M-Trace ayant un modèle de M-Trace identique ou différent.

Au sein de D3kode cela se traduit par la gestion de modèle de transformation incorporant des modèles de règles.

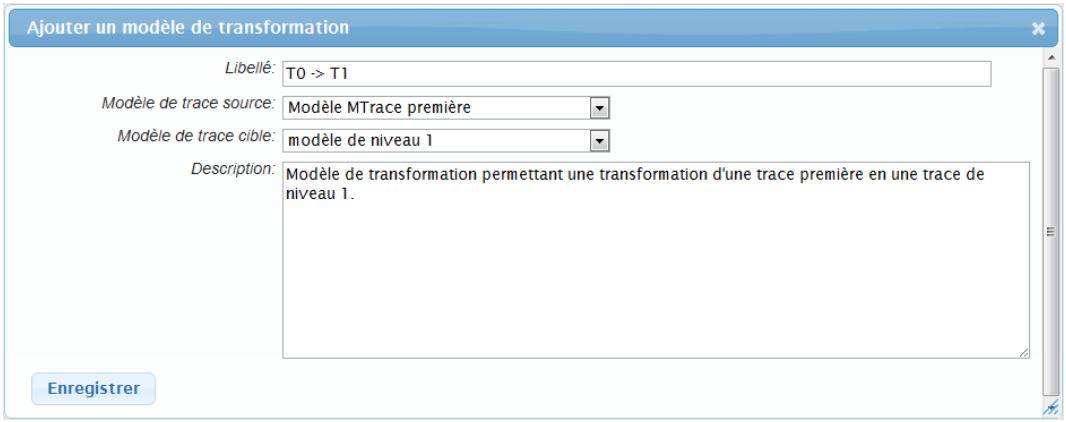


Figure 25 : Création d'un nouveau modèle de transformation

Lors de la création d'un modèle de transformation *(figure 25)* sont renseignés :

1. Le **libellé** du modèle de transformation.
2. Le **modèle de la M-Trace en source** : la transformation **portera** sur une M-Trace qui aura ce modèle de M-Trace.
3. Le **modèle de la M-Trace cible** : la transformation **produira** une M-Trace qui aura ce modèle de M-Trace.
4. Une **description** du modèle de transformation.

L'IHM permettant de gérer la création d'un nouveau modèle de transformation de M-Trace permet aussi de les lister *(figure 26)*.

Figure 26 : Liste de modèles de transformation

Un ensemble d'information pour chaque modèle de transformation est alors consultable :

1. Le libellé du modèle de transformation
2. Le nom du modèle de M-Trace accepté en tant que source
3. La liste des modèles de règle que composent le modèle de transformation
4. Le nom du modèle de M-Trace cible qui sera celui de la M-Trace produite par l'exécution de cette transformation

5. Un bouton d'action permettant de modifier le modèle de transformation
6. Un bouton d'action permettant d'ajouter/modifier les modèles de règles inclus dans le modèle de transformation
7. Un bouton d'action/indicateur permettant d'enregistrer le modèle de transformation au sein du KTBS
    a. L'icône cadenas fermé indique que le modèle de transformation ou que l'une des règles n'a pas été modifié depuis le dernier enregistrement au sein du KTBS
    b. L'icône du cadenas ouvert indique que le modèle de transformation au sein de D3kode est différent de celui enregistré dans le KTBS ; il est alors cliquable et permet l'enregistrement.
8. Un bouton d'action portant l'icône d'une poubelle permettant la suppression du modèle de transformation avec l'ensemble des règles associées.

3.2.3.4 Construction de règle de transformation

Lors de la création d'une règle au sein d'un modèle de transformation, 2 possibilités sont offertes à l'utilisateur. La *figure 27* présente la création d'une règle à partir d'un formulaire de création nécessitant de remplir le libellé et la description d'une règle de transformation.

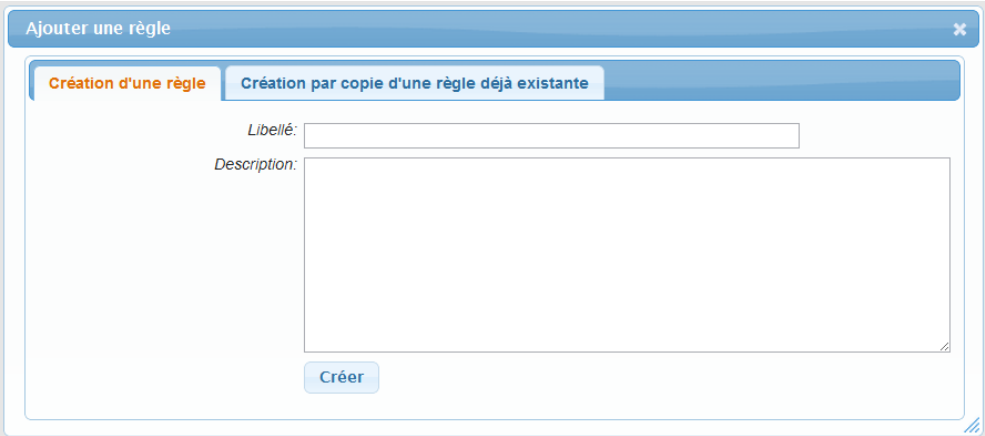


Figure 27 : Création d'une règle de transformation

Mais il est aussi possible de créer une règle à partir d'une précédente. L'IHM propose alors une liste de règles pouvant être dupliquée pour être utilisée au sein de la transformation que nous considérons. En effet les règles proposées ont été définies au sein de modèle de transformation ayant les mêmes modèle de M-Trace source et cible que le modèle de transformation que nous sommes en train de créer.

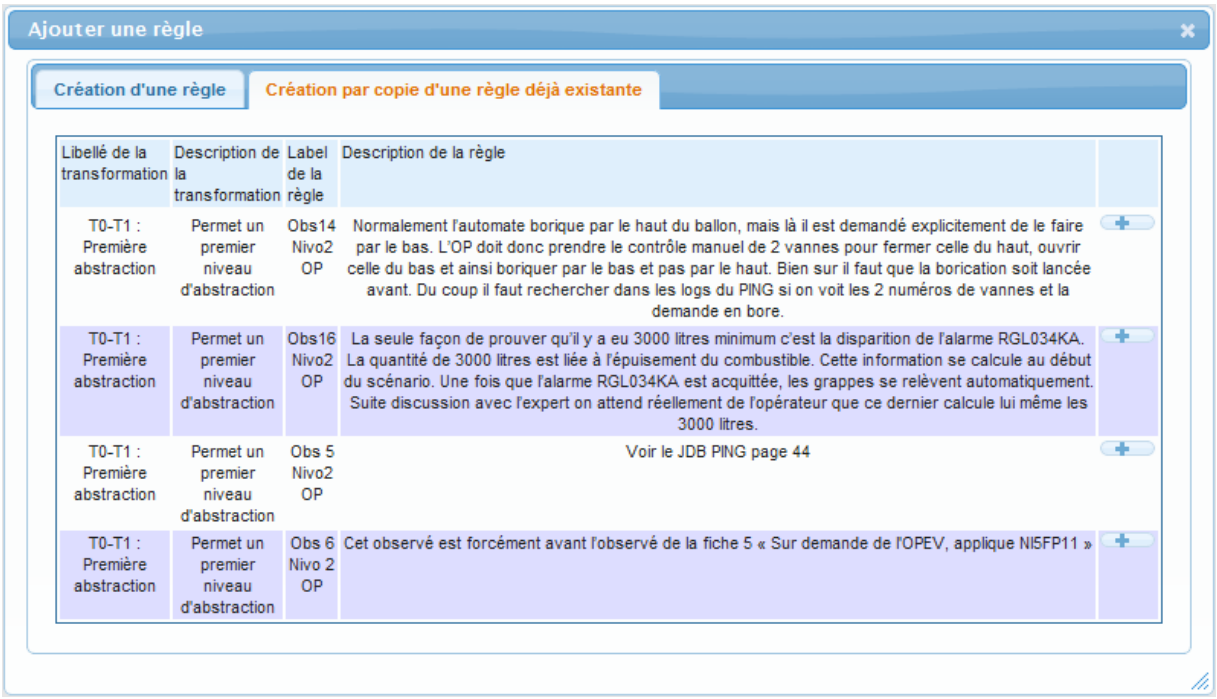


Figure 28 : Création par copie d'une règle déjà existante

Lors de la création d'une règle par le formulaire de création, un squelette de règle (*figure 29*) est produit. Le libellé de la règle permet d'identifier la règle en cours d'élaboration et un ensemble de bouton permet de guider l'utilisateur dans la création.

Figure 29 : Patron de règle initialisé

Les 2 principales constituantes de la règle sont identifiées par des cadres nommés : « Partie construction » et « Partie sélection ». Un bouton d'action dans chacune de ces parties permet de faire apparaitre une liste de choix de type d'observés sélectionnable. Pour la partie **création** (*figure 30*), la liste des types d'observés a été construite à partir du **modèle de M-Trace cible** défini dans le modèle de transformation.

Figure 30 : Choix du type d'observé à construire

Pour la partie **sélection** (*figure 31)*, la liste des types d'observés a été construite à partir **du modèle de M-Trace source** du modèle de transformation.

Figure 31 : Choix du type d'observé à rechercher

La composition d'une règle peut être étudiée à partir de la *figure 32* ci-après. Lors du choix du type d'observé dans la partie construction (1), il sera ajouté pour la partie création un tableau (2) comportant l'ensemble des type d'attribut (3) relié au type d'observé choisi. En effet, c'est grâce à cela que l'utilisateur pourra spécifier l'observé que la règle créera si elle trouve des observés répondant aux critères de recherches définis dans la partie sélection. La partie sélection permet à la fois de choisir les types d'observés (4) à rechercher mais aussi l'ensemble des contraintes (5) à cette recherche. 2 boutons d'action aident à cette saisie. Le premier (7) permet d'afficher une fenêtre (*figure 35*) permettant d'ajouter à la partie sélection un filtre de sélection. Le second (6) permet d'apporter la négation des critères déjà spécifiés (*figure 36*). Enfin, 2 autres boutons (8) d'actions placés dans la partie inférieure droite de chaque règle permettent de modifier la règle (libellé et description) ou de la supprimer (icône poubelle).

Figure 32 : Différentes constituantes d'une règle

Le bouton d'action permet de faire apparaitre une fenêtre (*figure 33*) d'aide à la saisie pour la valorisation de chaque attribut de l'observé que l'on va construire.

Figure 33 : Modification de la valeur d'un attribut de l'observé construit

Cette fenêtre est ainsi découpée :

1. La valorisation en cours de construction. Au fur et à mesure que l'utilisateur navigue dans cette fenêtre et clique sur les items, il construit à l'instar d'une calculatrice l'expression qui sera utilisée pour la valorisation de l'attribut considéré.
2. Le choix des types de valorisation. Comme on peut le voir au sein de la *figure 32* l'utilisateur a choisi de rechercher 3 observés de type « ActionOperateur ». Il est donc possible de valoriser l'attribut de l'observé construit par une valeur présente au sein de ces observés. Il est aussi possible d'utiliser des opérateurs (*figure 34)* pour construire des expressions calculatoires. Et enfin, l'on peut affecter une constante textuelle typée (*figure 35*).
3. La précision sur le type de valorisation. Pour chaque type de valorisation possible, il faut préciser son choix ; un attribut pour un type d'observé, un opérateur pour une opération et écrire une valeur pour une valeur textuelle.
4. Enregistrement de cette valorisation au sein de la règle. Enfin une fois que l'expression est conforme à ce que l'on souhaite on peut la valider et cela renseignera la case du tableau de l'attribut valorisé.

Figure 34 : Construction d'un filtre de sélection, choix d'un opérateur

De manière analogue, l'on peut ajouter des filtres de sélection à la partie sélection de la règle. La fenêtre d'ajout de filtre de sélection (*figure 35)* est similaire à celle de modification d'attribut. Mais elle n'aura pas pour but de permettre la valorisation de champ, ce seront des filtres tels que ceux présenté dans la *figure 32* (5).

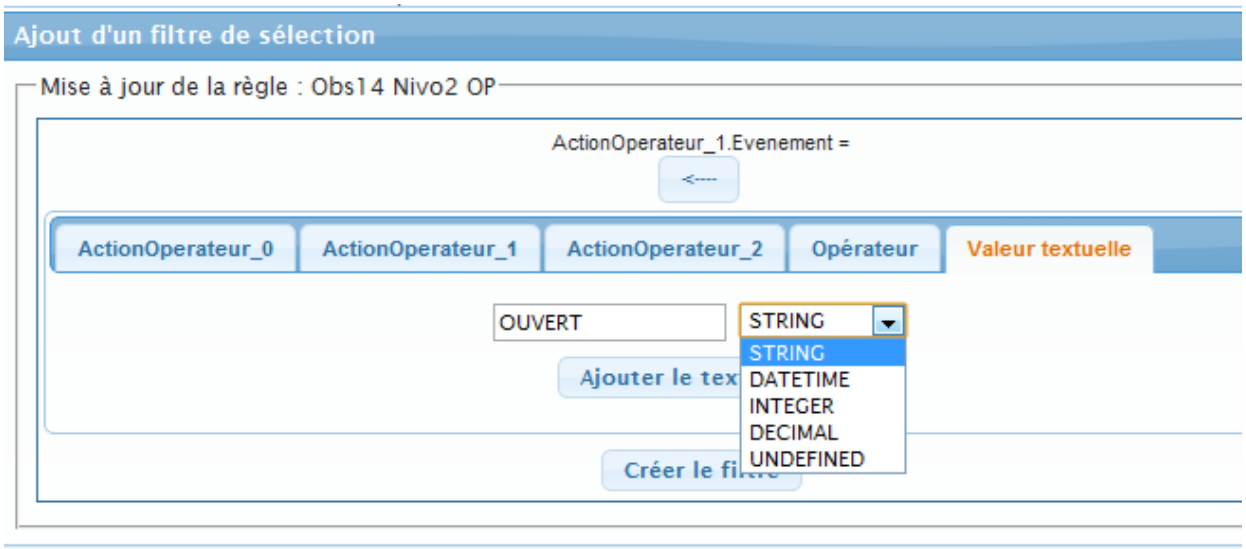


Figure 35 : Construction d'un filtre de sélection, choix de l'opérande

Si l'on peut établir une règle capable de construire un observé lorsque l'on détecte la présence d'autres, il est aussi possible de construire un observé sur l'absence d'autres observés. Pour cela il suffit d'apporter la négation à la partie sélection (*figure 35*).

Figure 36 : Négation de la partie sélection

Une fois la règle construite, celle – ci peut être enregistrée dans le KTBS par enregistrement du modèle de transformation auquel elle appartient. D3KODE permet de construire le graphe RDF correspondant à chacune des règles que l'utilisateur à créées graphiquement précédement.

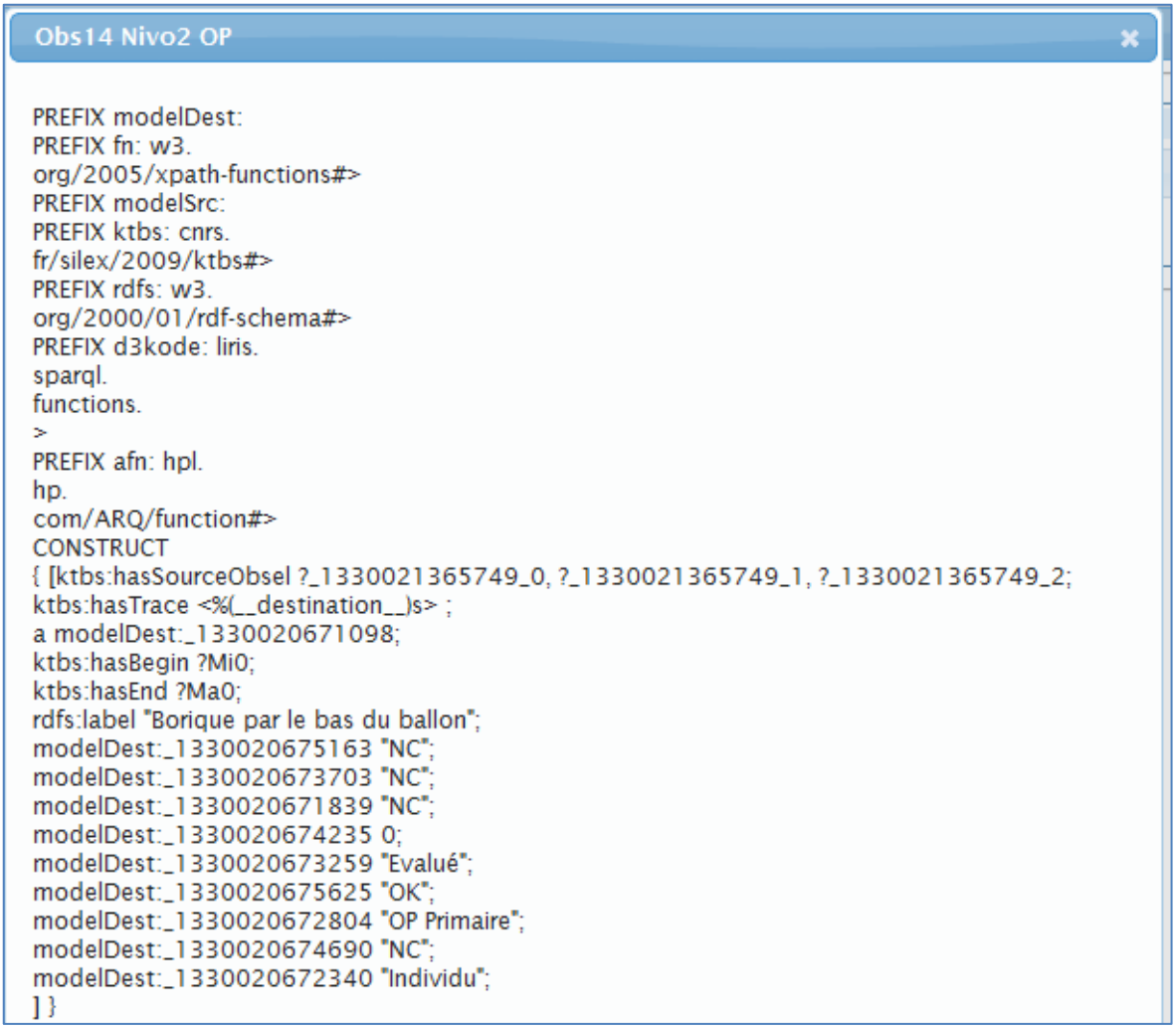


Figure 37 : Règle au format RDF : partie construction

Comme le montre l'ensemble *des figures 37 - 38 - 39* une simple règle (*figure 32*) représente un graphe RDF assez conséquent et difficilement lisible pour un œil non-expert. Néanmoins, l'on peut identifier (*figure 37*) la partie construction, la partie sélection (*figure 38*) et enfin la partie filtre et calcul (*figure 39).*

```
WHERE
{ ?_1330021365749_0 a modelSrc:_1330021365749;
modelSrc:_1330021366192 ?_13300213661920;
ktbs:hasBegin ?hasBegin0;
ktbs:hasEnd ?hasEnd0;
modelSrc:_1330021365774 ?_13300213657740;
rdfs:label ?label0;
modelSrc:_1330021366088 ?_13300213660880;
modelSrc:_1330021365871 ?_13300213658710;
modelSrc:_1330021366044 ?_13300213660440;
modelSrc:_1330021365834 ?_13300213658340;
modelSrc:_1330021365924 ?_13300213659240;
modelSrc:_1330021366136 ?_13300213661360;
modelSrc:_1330021365803 ?_13300213658030.
?_1330021365749_1 a modelSrc:_1330021365749;
modelSrc:_1330021366192 ?_13300213661921;
ktbs:hasBegin ?hasBegin1;
ktbs:hasEnd ?hasEnd1;
modelSrc:_1330021365774 ?_13300213657741;
rdfs:label ?label1;
modelSrc:_1330021366088 ?_13300213660881;
modelSrc:_1330021365871 ?_13300213658711;
modelSrc:_1330021366044 ?_13300213660441;
modelSrc:_1330021365834 ?_13300213658341;
modelSrc:_1330021365924 ?_13300213659241;
modelSrc:_1330021366136 ?_13300213661361;
modelSrc:_1330021365803 ?_13300213658031.
?_1330021365749_2 a modelSrc:_1330021365749;
modelSrc:_1330021366192 ?_13300213661922;
ktbs:hasBegin ?hasBegin2;
ktbs:hasEnd ?hasEnd2;
modelSrc:_1330021365774 ?_13300213657742;
rdfs:label ?label2;
modelSrc:_1330021366088 ?_13300213660882;
modelSrc:_1330021365871 ?_13300213658712;
modelSrc:_1330021366044 ?_13300213660442;
modelSrc:_1330021365834 ?_13300213658342;
modelSrc:_1330021365924 ?_13300213659242;
modelSrc:_1330021366136 ?_13300213661362;
modelSrc:_1330021365803 ?_13300213658032.
```

Figure 38 : Règle au format RDF : partie sélection

```
FILTER
( (regex(?_13300213659240,?_13300213659241) ) )
FILTER
( (regex(?_13300213659241,?_13300213659242) ) )
FILTER
( (regex(?_13300213659242,"RCV") ) )
FILTER
( (?_13300213660440 = 1 ) )
FILTER
( (?_13300213660441 = 155 ) )
FILTER
( (?_13300213660442 = 175 ) )
FILTER
( (regex(?_13300213660880,"KJ-") ) )
FILTER
( (regex(?_13300213660882,"VB-") ) )
FILTER
( (regex(?_13300213660881,"VP-") ) )
FILTER
( (regex(?_13300213661360,"C") ) )
FILTER
( (regex(?_13300213661361,"R") ) )
FILTER
( (regex(?_13300213661362,"R") ) )
FILTER
( (regex(?label0,"AUTOMATE APPOINT") ) )
FILTER
( (regex(?label1,"ISOLEMENT LIGNE DE DILUTION") ) )
FILTER
( (regex(?label2,"ISOLEMENT LIGNE D APPOINT") ) )
FILTER
( (?hasEnd0 < ?hasEnd1 ) && (?hasEnd1 < ?hasEnd2 ) )
FILTER
( (regex(?_13300213661920,"DDE BORICATION") ) )
FILTER
( (regex(?_13300213661921,"OUVERT") ) )
FILTER
( (regex(?_13300213661922,"FERME") ) )
LET
(?Mi0:=d3kode:Min(?hasBegin0,?hasBegin1,?hasBegin2) )
LET
(?Ma0:=d3kode:Max(?hasEnd0,?hasEnd1,?hasEnd2) ) }
```

Figure 39 : Règle au format RDF : partie filtre et calcul

3.2.3.5 Création d'une M-Trace d'activité

Il est possible de créer des M-Trace d'activités à partir de l'exécution d'une transformation sur l'ensemble des observés d'une M-Trace. L'IHM de construction de M-Trace d'activité (*figure 40*) permettant cela se décompose en 2 parties :

1. Un tableau rappelant l'ensemble des M-Traces pouvant éventuellement être source d'une transformation. Ce tableau contient :
    a. la date de création de la M-Trace,
    b. le libellé de celle-ci,
    c. le nombre d'observé qu'elle comporte,
    d. la liste des modèles de transformations applicables (c'est-à-dire ceux qui ont en modèle de M-Trace source le modèle de M-Trace de la M-Trace courante),
    e. la liste des M-Traces intermédiaires, c'est-à-dire la liste des M-Traces générées par la transformation : une par règle
    f. le nom de la M-Trace résultante de la transformation
    g. le nombre d'observé généré par la transformation

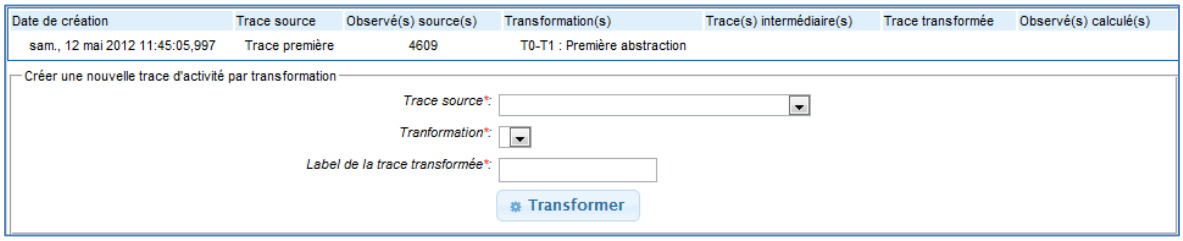


Figure 40 : Formulaire de construction de trace d'activité

2. Un formulaire permettant la création d'une M-Trace d'activité résultante de l'exécution d'un modèle de transformation sur une M-Trace *(figure 41)*. Lors de la sélection de la trace source, la liste des modèles de transformation possible se mets à jours de manière à proposer que les modèles de transformation qui ont en modèle de M-Trace source le modèle de la M-Trace sélectionnée précédemment. Enfin il faut saisir le libellé de la M-Trace que l'on va créer.

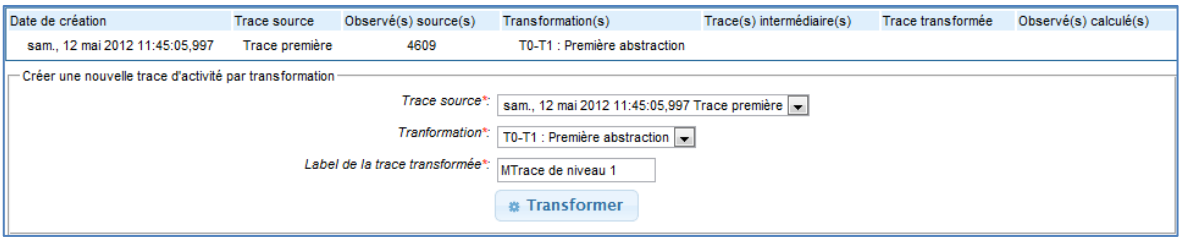


Figure 41 : Création d'une trace d'activité par transformation

Une fois la transformation exécutée, une M-Trace est créée pour chaque règle que compose le modèle de transformation (*figure 42*). Il est alors possible d'exécuter, de manière analogue, une autre transformation sur la M-Trace générée précédemment et produire un enchainement de M-Trace.

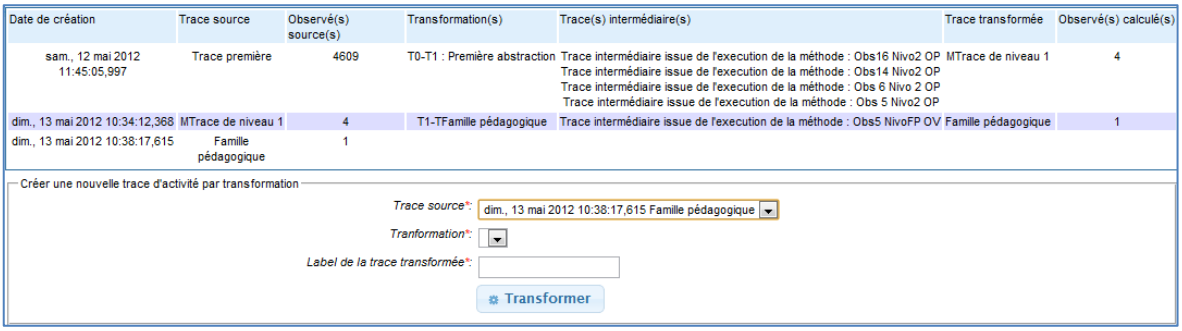


Figure 42 : Résultat d'exécution de la transformation

3.2.3.6 Représentation graphique

Lorsque l'on accède à l IHM qui permet la visualisation du corpus de de M-Trace, un tableau récapitulant l'ensemble des M-Traces d'activités est présenté (*figure 43*). Ce tableau permet d'avoir une vision succincte du corpus de M-Trace de l'activité courante.

Id kTBS	Date de création	Libellé		Date de début	Date de fin	Modèle de trace	Nombre d'observé(s)	Nombre de trace(s) calculée(s)
1336815905997	12-05-2012 11h45	Trace première		27-06-2012 11h00:09.005	27-06-2012 12h00:20.012	MTrace première	4609	5
_1336889813094	13-05-2012 08h16	MTrace de niveau 1		27-06-2012 11h21:35.010	27-06-2012 11h43:46.013	Niveau intermédiaire	4	2
___1336889813094__1336815866764	12-05-2012 11h44	Trace intermédiaire issue de l'execution de la méthode : Obs14 Nivo2 OP		27-06-2012 11h28:59.012	27-06-2012 11h29:33.009	Niveau intermédiaire	1	0
___1336889813094__1336889720609	13-05-2012 08h15	Trace intermédiaire issue de l'execution de la méthode : Obs 6 Nivo 2 OP		27-06-2012 11h21:35.010	27-06-2012 11h21:35.010	Niveau intermédiaire	1	0
___1336891314966__1336891206704	13-05-2012 08h40	Trace intermédiaire issue de l'execution de la méthode : Obs5 NivoFP OV		27-06-2012 11h21:35.010	27-06-2012 11h43:46.013	Famille pédagogique	1	0
___1336889813094__1336840906945	12-05-2012 18h41	Trace intermédiaire issue de l'execution de la méthode : Obs16 Nivo2 OP		27-06-2012 11h43:46.013	27-06-2012 11h43:46.013	Niveau intermédiaire	1	0
___1336889813094__1336858419158	12-05-2012 23h33	Trace intermédiaire issue de l'execution de la méthode : Obs 5 Nivo2 OP		27-06-2012 11h21:48.007	27-06-2012 11h21:48.007	Niveau intermédiaire	1	0
_1336891314966	13-05-2012 08h41	Famille Pédagogique		27-06-2012 11h21:35.010	27-06-2012 11h43:46.013	Famille pédagogique	1	0

Figure 43 : Corpus de M-Trace, vision tabulaire

Accolé à ce tableau, un formulaire (*figure 44*) permet de lancer la création graphique du corpus de M-Trace.

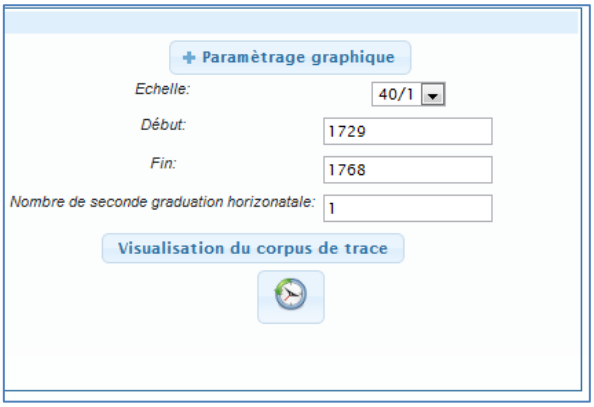


Figure 44 : Gestion de la représentation graphique du corpus de M-Trace

1. Le bouton d'action « paramétrage graphique » permet (*figure 45*) de gérer la configuration graphique et spatiale des observés au sein de la M-Trace
2. L'échelle de représentation, représentant le ratio entre le nombre de pixel et le nombre de seconde considérée (40/1 signifie qu'une seconde sera représentée par 40pixels).
3. La date de début de la représentation graphique exprimée en seconde
4. La date de fin de la représentation graphique exprimée en seconde
5. L'intervalle de graduation de la ligne de temps
6. Le bouton d'action « Visualisation du corpus de trace » permet de lancer la création graphique du corpus de M-Trace (*figure 45*).
7. Enfin le bouton d'action symbolisant l'historique permet d'afficher la liste des précédentes représentations graphiques (*figure 49*).

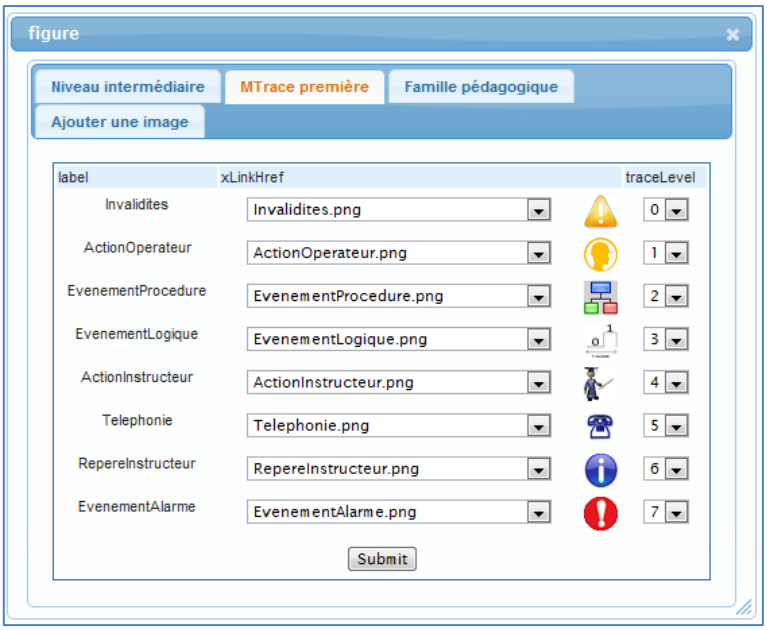


Figure 45 : Configuration de la représentation graphique et spatiale des observés des M-Traces

La fenêtre de paramétrage de configuration graphique et spatiale des observés propose un choix de représentation graphique (icône) ainsi qu'un choix de positionnement vertical au sein de la représentation graphique de la M-Trace. En effet en fonction du nombre de type d'observé du modèle de M-Trace, l'utilisateur pourra choisir à quel niveau (vertical) la représentation de cet observé sera faite.

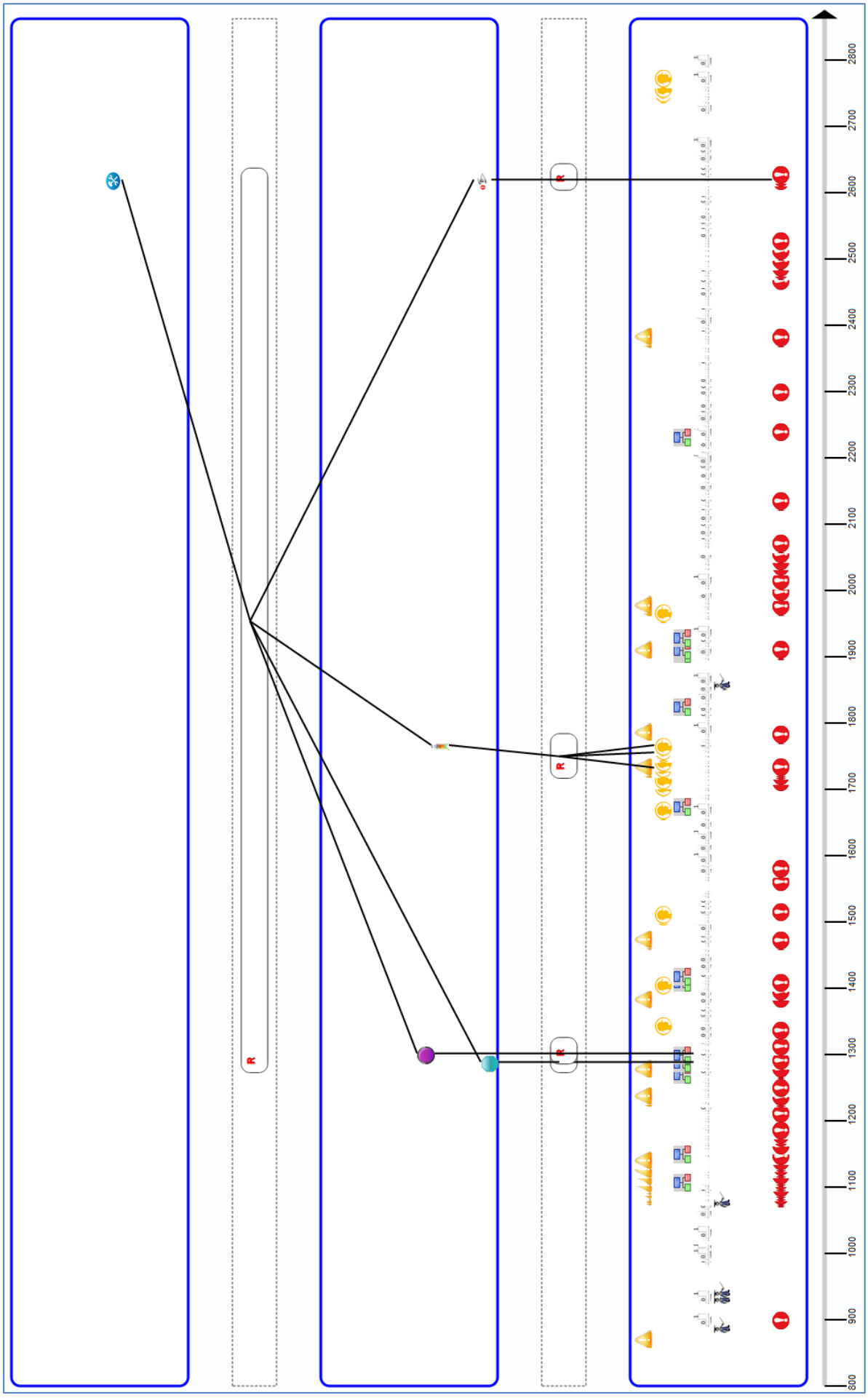


Figure 46: Représentation graphique du corpus de M-Trace

La *figure 46* représente graphiquement un corpus de M-Trace ; nous pouvons lire cette image interractive de bas en haut. Tout d'abord une ligne de temps graduée comme paramétrée dans le formulaire de création graphique : 100 secondes entre chaque graduation. Puis la M-Trace première délimité par un rectangle bleu. Elle contient des observés placés horizontalement par leur apparition temporelle, verticalement par leur paramétrage et ont chacun une représentation graphique différente. Entre 2 représentation de M-Trace est représenté par un rectangle de pointillés noirs une transformation. Celle-ci contient de plus petits restangle symbolisant une règle. La largeur de la règle est calculé suivant la date de début la plus petite de la liste d'observé source et la date de fin la plus grande de cette même liste. Enfin des traits noirs plein relient les observés sources et cibles à la règle qui les a construits/sélectionnés. Afin d'affinner la lecture de ce graphique il est

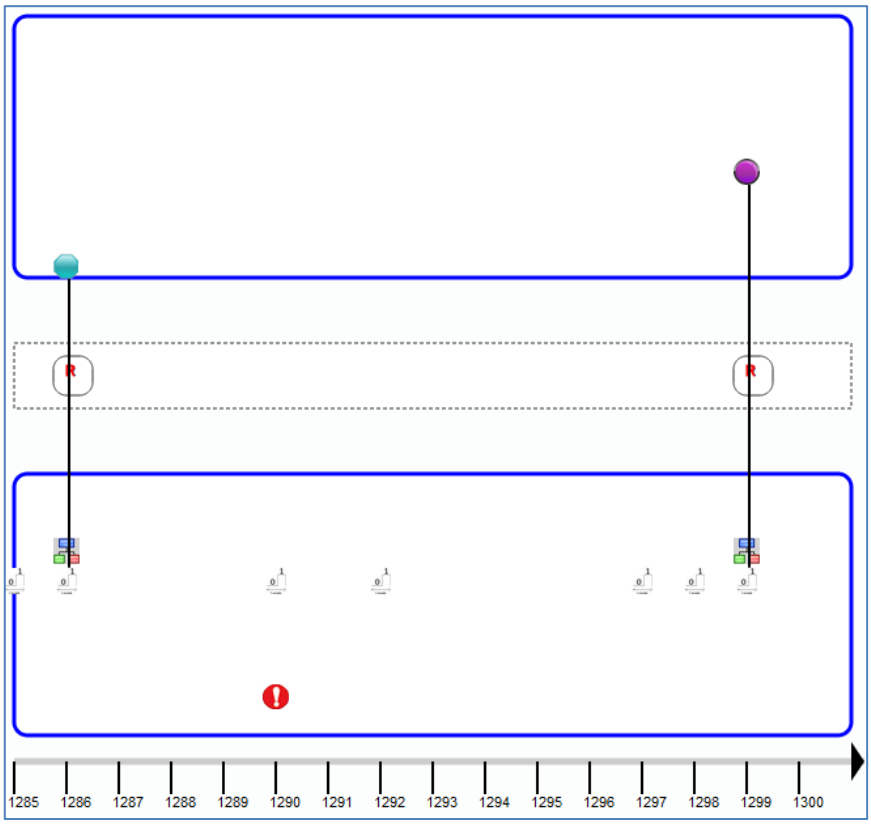


Figure 47: Zoom temporel sur une transformation

possible, grâce à l'échelle et aux bornes de début et de fin liées à la représentation graphique de concentrer le graphique sur un endroit particulier (*figure 47)*. Les deux règles générant des observés de différentes nature étaient superposées dans la première représentation (*figure 46)* mais sont au sein de la *figure 47* bien dissociées.

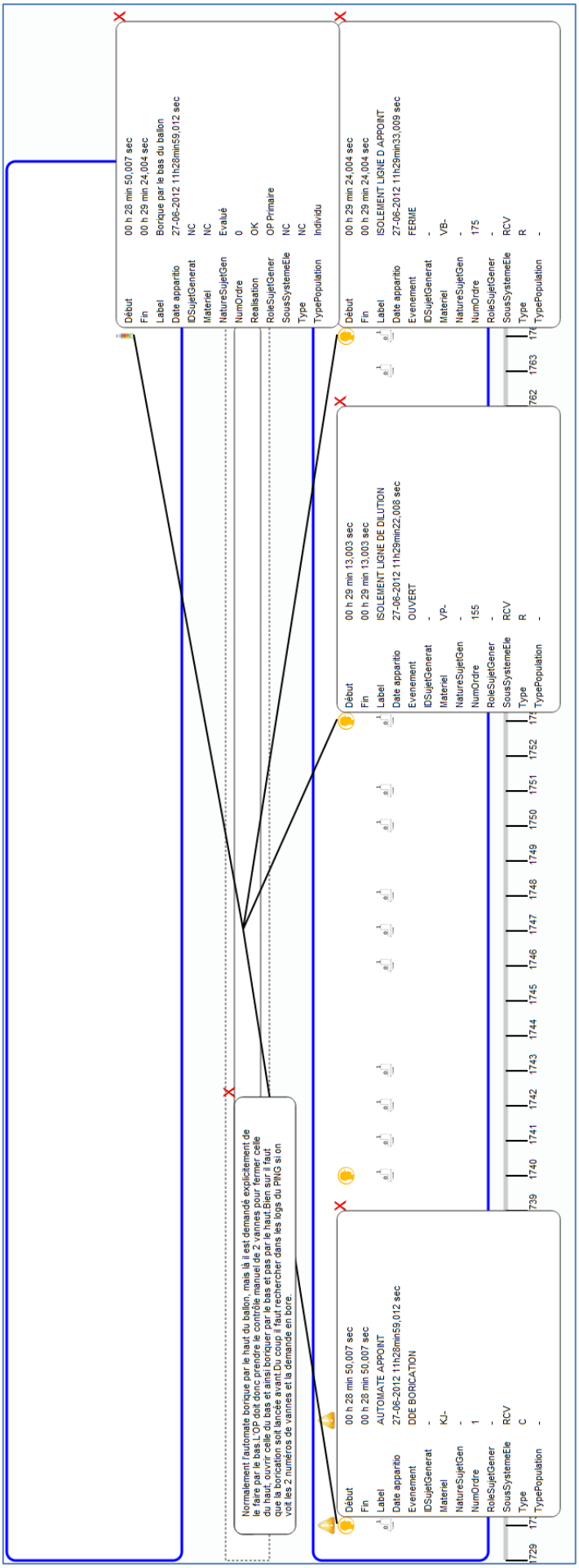


Figure 48 : Détails des attributs d'observés, et des règles

De plus, comme le montre la *figure 48* les observés et les règles sont cliquables. Apparait alors pour les règles la description de chacune. En ce qui concerne les observés, apparait la liste des attributs de chacun. Enfin, l'historique de création de représentation graphique est consultable au sein d'un tableau les listant. L'on peut ainsi retrouver par ordre de création, l'ensemble des représentations graphiques (fichier SVG) avec comme renseignement leurs bornes de début et de fin, l'échelle utilisée pour leur visualisation ainsi que le nombre de transformations qu'elles comportaient.

Liste des visualizations existantes

Date de création	Nom du fichier	Début	Fin	Echelle	Nombre de transformation
sam., 12 mai 2012 20:58:00,020	1336815905997_1710_1800_10.0_1.svg	1710	1800	10	1
sam., 12 mai 2012 21:04:44,165	1336815905997_1700_1900_10.0_1.svg	1700	1900	10	1
sam., 12 mai 2012 21:42:14,329	1336815905997_1_5000_10.0_1.svg	1	5000	10	1
sam., 12 mai 2012 22:10:46,043	1336815905997_1_5000_0.75_1.svg	1	5000	0.75	1
sam., 12 mai 2012 22:31:10,711	1336815905997_1725_1770_40.0_1.svg	1725	1770	40	1
sam., 12 mai 2012 22:31:10,711	1336815905997_1728_1766_20.0_1.svg	1728	1766	20	1
sam., 12 mai 2012 22:31:10,711	1336815905997_1728_1766_40.0_1.svg	1728	1766	40	1
sam., 12 mai 2012 22:31:10,712	1336815905997_1710_1790_10.0_1.svg	1710	1790	10	1
sam., 12 mai 2012 22:31:10,712	1336815905997_1725_1770_10.0_1.svg	1725	1770	10	1
sam., 12 mai 2012 23:19:57,039	1336815905997_1701_1901_10.0_1.svg	1701	1901	10	1
sam., 12 mai 2012 23:36:28,996	1336815905997_5_4500_0.1_1.svg	5	4500	0.1	1
sam., 12 mai 2012 23:47:17,122	1336815905997_2_5200_0.5_1.svg	2	5200	0.5	1
dim., 13 mai 2012 00:47:17,245	1336815905997_1100_2700_0.75_1.svg	1100	2700	0.75	1
dim., 13 mai 2012 08:46:40,309	1336815905997_0_5069_0.25_2.svg	0	5069	0.25	2
dim., 13 mai 2012 08:51:45,344	1336815905997_0_3809_0.25_2.svg	0	3809	0.25	2

Page 1 sur 2 15

Figure 49 : Historisation des précédentes représentations graphiques

Chapitre 4 Déploiement et usages

4.1 Déploiement de l'application

D3kode est donc un environnement applicatif regroupant :

1. une application web J2EE déployée sur un serveur web J2EE
2. une base de données d'utilisateur hsqldb
3. une base de M-Trace le KTBS

Lors d'une installation de d3kode, ces briques applicatives doivent être installées et paramétrées. Mais que l'on soit dans un environnement de développement ou de production ces actions vont comporter quelques différences.

4.1.1 Environnement de développement

L'environnement logiciel principal de développement est l'IDE[43] Eclipse. Il faut donc télécharger et installer cet atelier logiciel à partir de http://www.eclipse.org/downloads/. Plusieurs versions packagées sont proposées au téléchargement ; la version pour développeur J2EE sera préconisée pour le développement de d3kode (*figure 50*).

D3kode étant une application Web, il faut la déployer sur un serveur Web sachant interpréter les servlet. Tomcat a été choisi pour réaliser cette tâche, il est possible comme décrit par Lars Vogel dans ce tutoriel : http://www.vogella.com/articles/EclipseWTP/article.html de l'installer, le configurer et le gérer à travers Eclipse (*figure 51)*.

[43] IDE : Integrated Development Environment (ou environnement de développement intégré) est un programme regroupant un ensemble de fonctionnalités dédiées à l'aide pour la création logicielle. Parmi ces fonctionnalités citons : la coloration syntaxique, l'aide contextuelle, la génération de code ou la présence d'un débogueur.

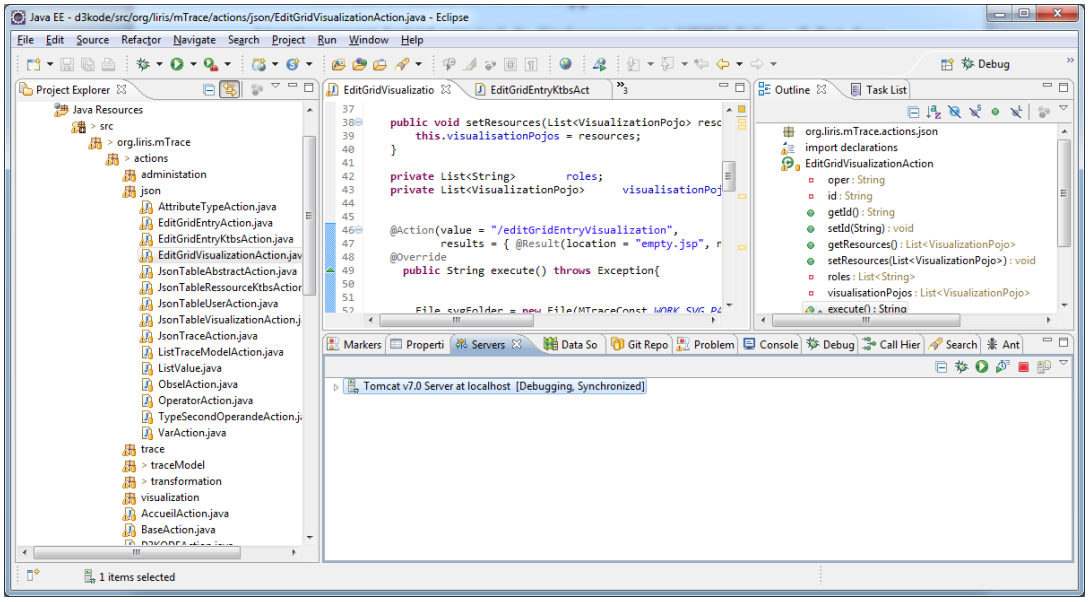


Figure 50 : L'atelier logiciel Eclipse

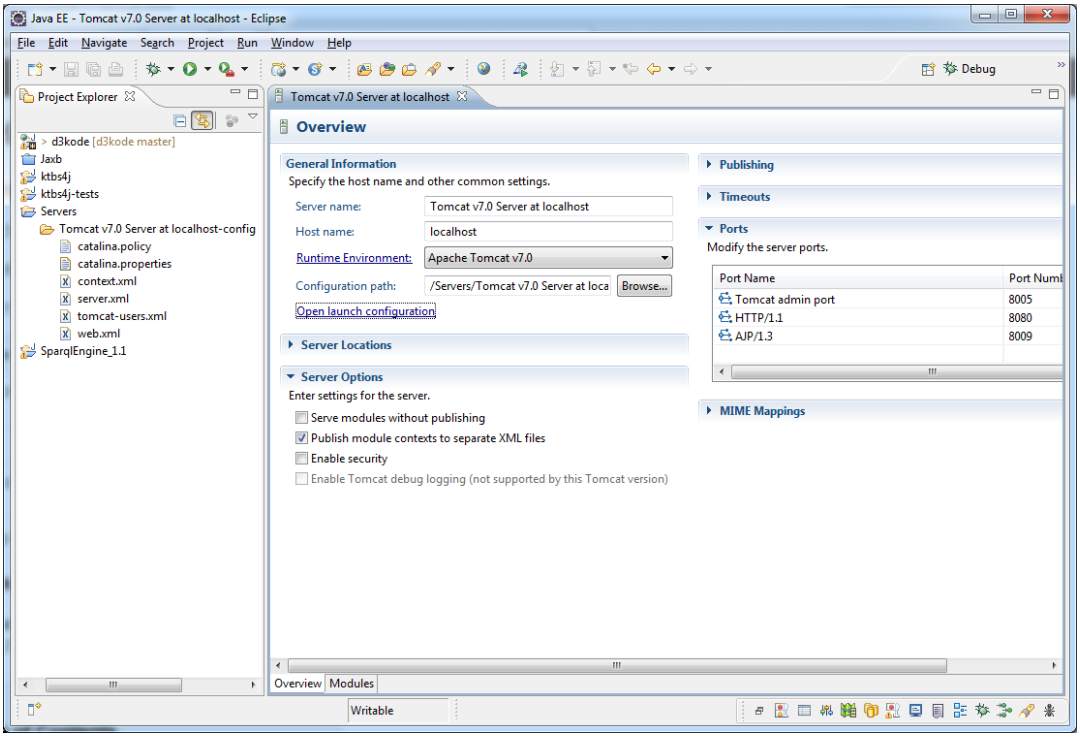


Figure 51 : Gestion serveur Web au sein d'Eclipse

Plusieurs projets composent d3kode, il faut récupérer les codes sources via les différents gestionnaires de codes sources pour pouvoir compléter la plateforme de développement.

Pour les projets **d3kode, ktbs4j** et **SparqlEngine_1.1**, ils sont hébergé sur un gestionnaire public de code source : https://github.com. Ce gestionnaire utilise le logiciel de gestion de versions décentralisé Git[44]. Ce logiciel libre, crée par Linus Torvalds[45] à l'avantage de

[44] http://git-scm.com/

[45] Linus Torvalds ingénieur en informatique américano-finlandais, il créa en 1991 le noyau Linux.

ne pas reposer sur un serveur décentralisé (contrairement à SVN[46] par exemple) et d'indexer les fichiers d'après leur somme de contrôle calculé avec la fonction SHA-1.

Par contre à l'heure actuelle le projet **KTBS**, maintenu par Pierre-Antoine Champin et Françoise Conil, n'a pas été complètement migré sous github et conserve donc ses sources hébergées sous evironnement SVN à l'adresse : https://svn.liris.cnrs.fr/sbt-dev/ktbs-rest-impl/trunk.

Afin de récupérer les sources de ces différents projets, les plugins subversive pour SVN et Egit pour GIT, permettent de synchroniser Eclipse avec les gestionnaires de code source. Il sera alors possible d'effectuer via l'atelier logiciel toutes les opérations que permet un gestionnaire de code source ; faire des versions, charger, mettre à jour etc…

Enfin la librairie exécutable hsqldb.jar (http://hsqldb.org/) doit être ajoutée au chemin de compilation du serveur web et une instance serveur de ce gestionnaire de base de données.

```
C:\cygwin\home\Dinous>java -cp hsqldb.jar org.hsqldb.Server -database.0 file:d3kodeDB\d3kodeDB -dbname.0 d3kodeDB
[Server@40e9e799]: [Thread[main,5,main]]: checkRunning(false) entered
[Server@40e9e799]: [Thread[main,5,main]]: checkRunning(false) exited
[Server@40e9e799]: Startup sequence initiated from main() method
[Server@40e9e799]: Could not load properties from file
[Server@40e9e799]: Using cli/default properties only
[Server@40e9e799]: Initiating startup sequence...
[Server@40e9e799]: Server socket opened successfully in 5 ms.
[Server@40e9e799]: Database [index=0, id=0, db=file:d3kodeDB\d3kodeDB, alias=d3kodedb] opened sucessfully in 319 ms.
[Server@40e9e799]: Startup sequence completed in 325 ms.
[Server@40e9e799]: 2012-05-12 07:41:44.401 HSQLDB server 2.2.5 is online on port 9001
[Server@40e9e799]: To close normally, connect and execute SHUTDOWN SQL
[Server@40e9e799]: From command line, use [Ctrl]+[C] to abort abruptly
```

Figure 52 : Console serveur hsqldb

Donc, pour avoir un environnement de développement d3kode il suffit de :

1. Installer l'IDE eclipse
2. Installer le serveur de servlet Tomcat par le biais d'Eclipse
3. Installer les plugins pour l'accès aux fonctionnalités de gestion de code source SVN et GIT
4. Récupérer le code source des projets d3kode, ktbs4j et Sparql_Engine sous github
5. Récupérer le code source du système de gestion à base de traces : KTBS sous le svn du laboratoire Liris
6. Récupérer la librairie auto-exécutable du serveur de base de données hsqldb

4.1.2 Environnement de production

Les livrables inhérents à l'application d3kode sont :

- **D3kode.war** : la designation de "war" signifie **W**eb **A**pplication **AR**chive. C'est un format de fichier JAR (**J**ava **AR**chive) pouvant contenir l'ensemble des fichiers nécessaire à une application Web à base de servlet. C'est ce fichier qui sera déployé sur le serveur de servlet Tomcat.
- **KTBS** : Le serveur de gestion de base de trace KTBS est une arborescence de répertoire et de fichier nécessitant un compilateur python pour s'exécuter. Le KTBS est alors exécuté, respectant les principes de l'architecture REST.

[46] SVN : SubVersioN est un logiciel de gestion de version distribué sous licence Apache. Ses créateurs s'appuie fortement sur les concepts de CVS, mais en revoyant son implémentation.

- **Ktbs4j.jar** : Ce projet créé originalement par Damien CRAM, est une API[47] fournissant un ensemble de classes Java permettant la communication entre une application web (d3kode) et un serveur de gestion de base de trace (ktbs) via une architecture REST.
- **Sparql_engine_1.1.jar** : Cette librairie permet l'exécution de requête SPARQL version 1.1. A l'instar du SQL[48] pour les bases de données relationnelles, le SPARQL permet d'accéder et d'interroger des bases de données RDF. La recommandation officielle du W3C considère la version 1.0 du langage mais prochainement la recommandation devrait passer en 1.1. Malheureusement l'implémentation SPARQL qui a été faite au sein du KTBS suit la recommandation 1.0 et n'est pas suffisante pour les besoin de requêtage de d3kode. C'est pourquoi nous avons conçu Sparql_engine_1.1 qui permet de traiter une partie des futures nouvelles recommandations SPARQL.
- **Hsqldb.jar** et arborescence base de données : d3kodeDB.log, d3kodeDB.properties, d3kodeDB.script. Le serveur de base de données et l'API d'accès à la base de données de d3kode d3kodeDB utilise la bibliothèque hsqldb.jar. C'est pourquoi avant d'exécuter le serveur d'application, il faut veiller à ajouter au chemin d'exécution du serveur Web cette librairie. Le fichier d3kodeDB.log contient l'ensemble des traces d'exécutions loguées lié à l'exécution de la base de données. Le fichier d3kodeDB.properties est utilisé pour spécifier les caractéristiques de la base de données, comme sa version par exemple. Enfin le fichier d3kodeDB.script permet de loguer l'ensemble du SQL qui a été utilisé sur la base de données.

En conclusion, pour installer l'application Web d3kode, il faut déployer d3kode.war sur le serveur Web Tomcat. Lancer le KTBS et le serveur de base de données hsqldb. Et enfin utiliser un client web pour accéder à l'application.

4.2 Cas d'utilisation

Différents type d'utilisateur peuvent se connecter à D3kode. Ces différents types d'utilisateurs s'identifient par un nom d'utilisateur et un mot de passe. Cette identification, lorsqu'elle réussit permet de récupérer en base de données les droits utilisateurs au sein de D3kode.

4.2.1 Formateur expert

L'utilisateur ayant pour rôle « formateur expert » possède l'ensemble des droits sur D3kode. Il peut donc accéder à l'ensemble des fonctionnalités de l'application comme nous le montre la *figure 53*. D'un point de vue fonctionnel, l'utilisateur expert sera celui qui initialisera les modèles de traces au sein de D3kode, créera les modèles de transformations et de règles qui pourront servir de références au formateur novice.

Au sein de D3kode, l'utilisateur ayant pour rôle « formateur expert » a donc accès à l'ensemble des possibilités offertes par l'application. Notamment la partie administration (*figure 54*) et la gestion (création, modification, suppression) des modèles de transformation.

[47] API : Application Programming Interface. Une interface de programmation permet l'interaction des programmes les uns avec les autres.

[48] SQL : Structured Query Language. Langage informatique normalisé servant à effectuer des opérations (sélection, mise à jour, suppression) sur des bases de données.

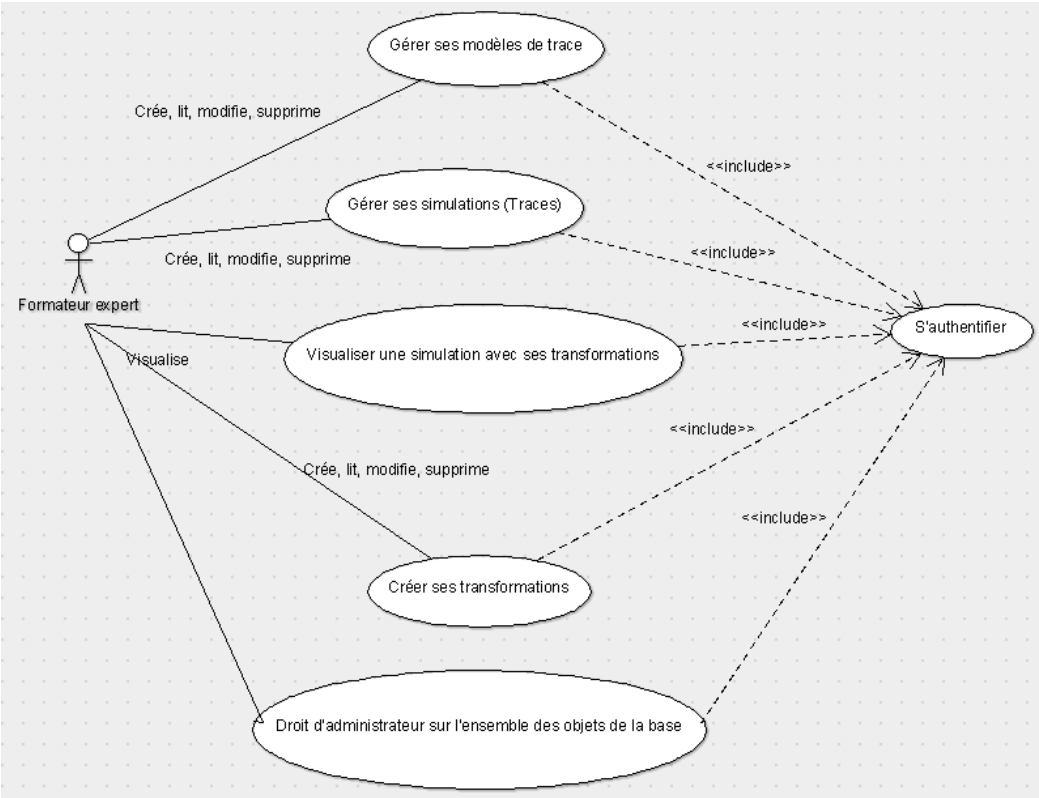


Figure 53 : Cas d'utilisation au sein de D3kode d'un formateur expert

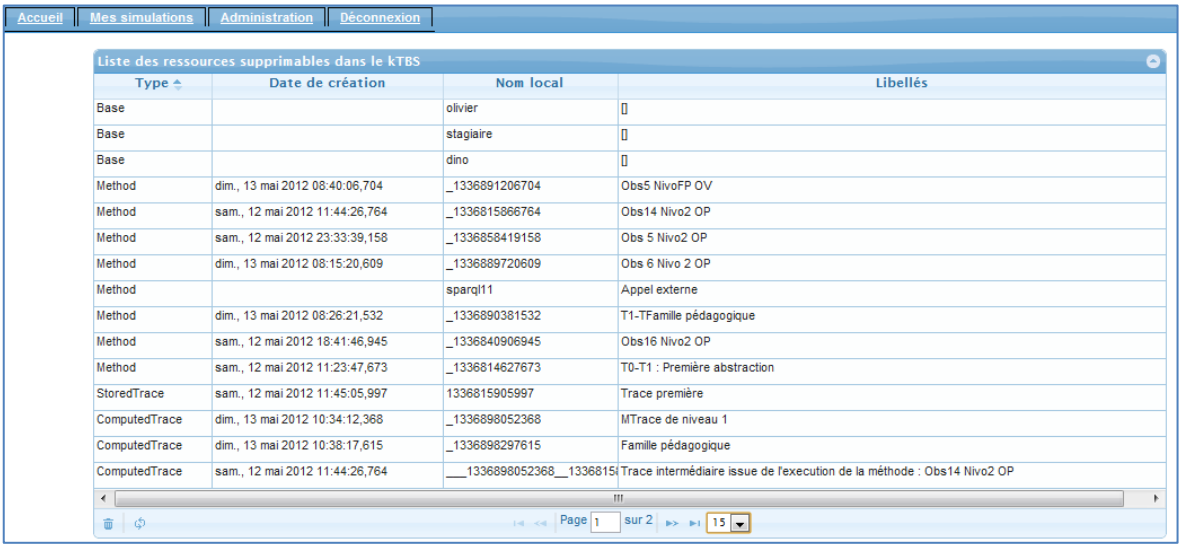


Figure 54 : Accès à la partie administration de D3kode

4.2.2 Formateur novice

Le formateur novice accède à l'application avec presque autant de droits que l'utilisateur expert sauf qu'il ne peut gérer que ses propres M-Traces, modèles de M-Trace, modèles de transformations et de règles (*figure 55*). Il peut néanmoins utiliser et consulter les objets propres à l'utilisateur expert comme par exemple les modèles de M-Trace (*figure 56*) ou les modèles de transformations.

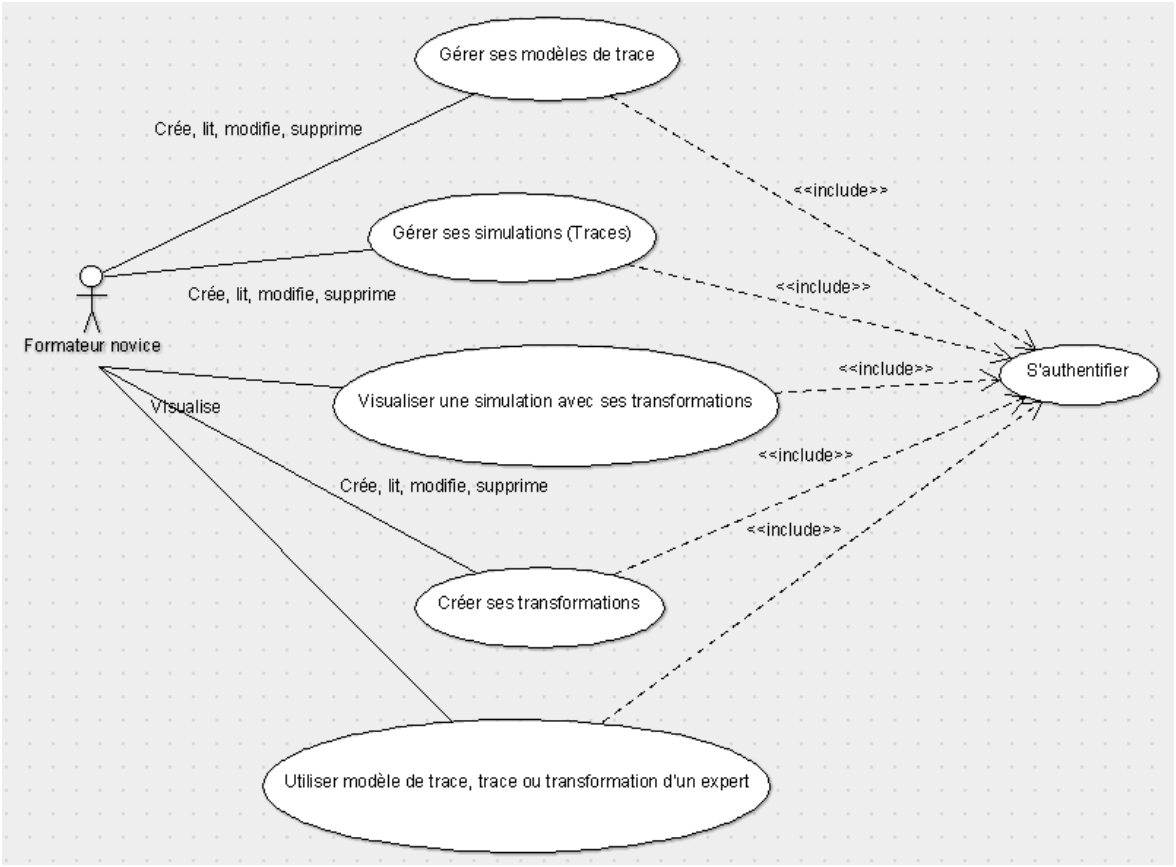


Figure 55 : Cas d'utilisation au sein de D3kode d'un formateur novice

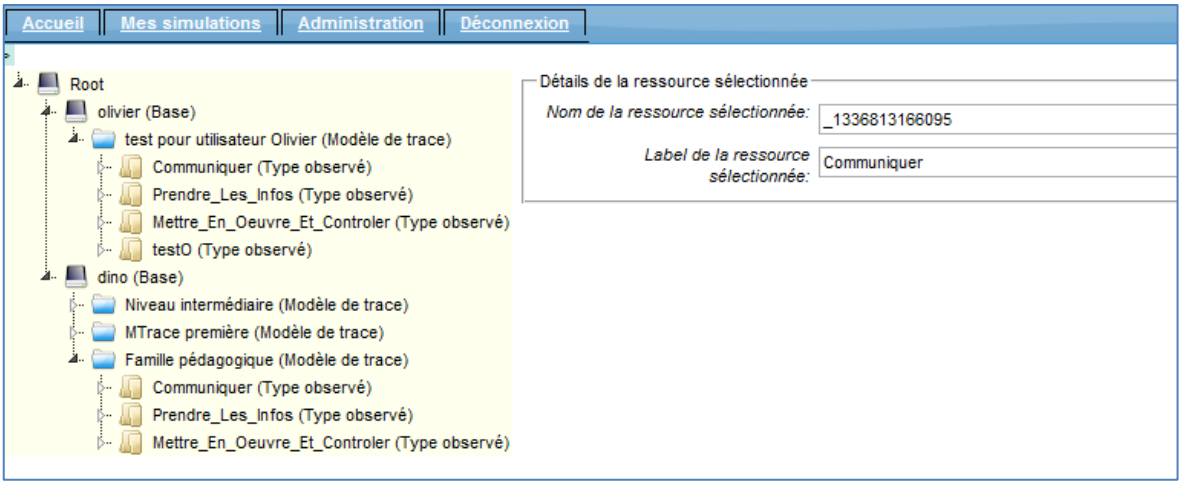


Figure 56 : Visualisation de l'ensemble de modèle de M-Trace

4.2.3 Stagiaire

Lorsqu'un utilisateur ayant pour rôle « stagiaire » se connecte à l'application, il n'a qu'un ensemble de possibilité au sein de D3kode assez restreint. Il ne peut par exemple pas créer de modèle de M-Trace ou de modèle de transformation ou accéder à la partie administration du site (*figure 55*). Par contre, il lui est possible de charger sa M-Trace d'activité (*figure 56*) et d'appliquer différentes transformations pour voir son corpus de M-Trace.

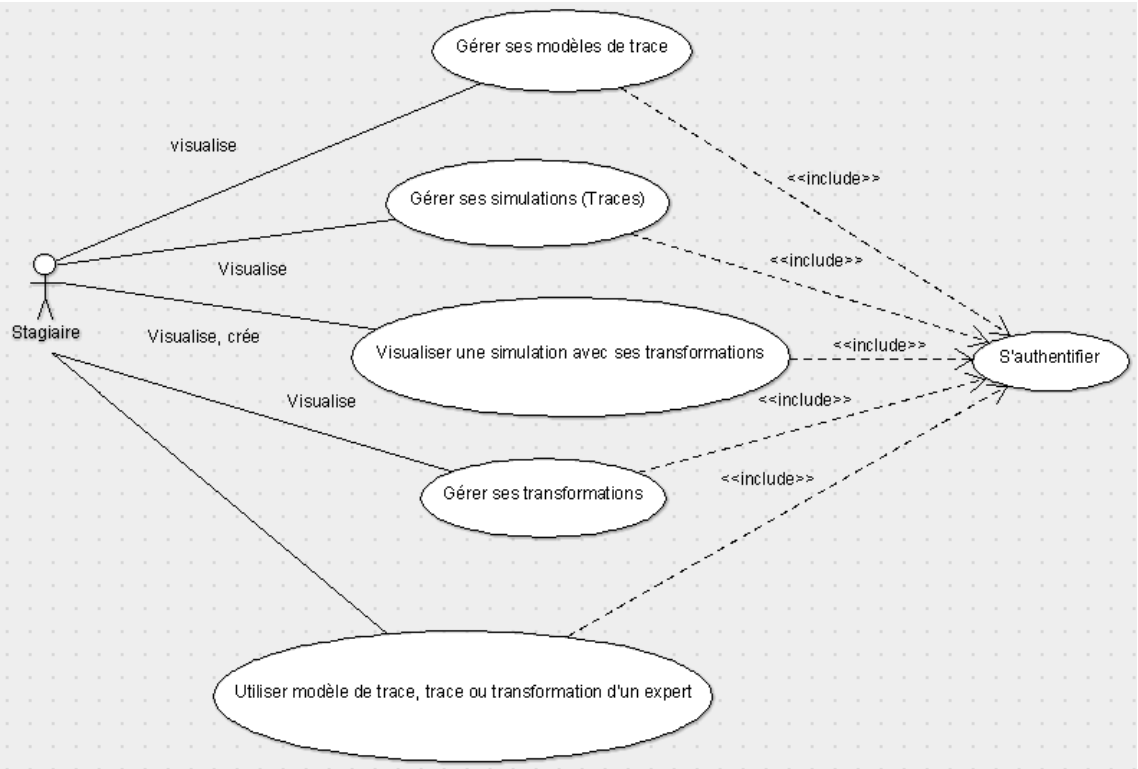


Figure 57 : Cas d'utilisation au sein de D3kode d'un stagiaire

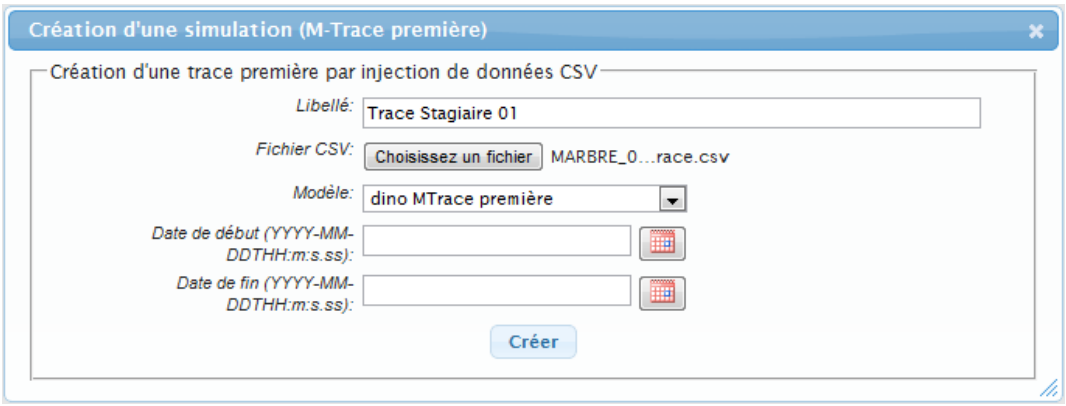


Figure 58 : Chargement M-Trace première d'un stagiaire

Conclusion & perspectives

Au cours de mes 9 mois de stages au sein de l'équipe SILEX du LIRIS, j'ai eu l'opportunité de manipuler des concepts innovants d'ingénierie des connaissances dans le cadre de recherche portant sur l'aide à l'observation et l'analyse de l'activité appliquée à un contexte de formation professionnelle et de transfert de savoir-faire.

En particulier mon travail a pris forme au travers de la thèse d'Olivier CHAMPALLE dont l'enjeu portait sur l'utilisation du concept de « trace modélisée » à des fins de modélisation et de transfert de l'expertise d'observation dans les simulateurs pleine échelle de conduite de centrale nucléaire du groupe EDF.

Tout au long de mon stage, mon objectif a été d'étudier et de comprendre les besoins du projet de recherche afin de proposer et concevoir un ensemble de solution technologiques permettant d'y répondre. Ainsi, suite à une étude d'implémentations existantes relatives à ce sujet, puis à la conception et aux tests d'une application spécifique, j'ai pu conforter et illustrer les problématiques traitées au sein du projet.

Résultats de mon engagement dans l'équipe SILEX, l'application D3kode, que j'ai conçue, est une plateforme applicative N-Tiers capable d'exploiter des données sous la forme de trace modélisée. Ces données sont conformes à un modèle paramétrable et peuvent être sources de transformations permettant d'abstraire l'information qu'elles contiennent pour en dégager des représentations de plus haut niveau. D3kode permet alors d'obtenir une représentation graphique, réifiant la théorie de la trace modélisée, d'un corpus de traces d'activités. Cette représentation graphique est navigable horizontalement et verticalement. Dans la dimension horizontale, les informations observées sont visualisées par rapport à leur apparition dans le temps. La dimension verticale met en évidence les relations entre les niveaux d'abstraction permettant d'analyser plus finement l'activité.

L'intérêt et la solidité de D3kode permettent d'envisager un test « grandeur nature » des propositions théoriques du projet de recherche. Un protocole d'évaluation a donc été défini en ce sens et une évaluation écologique sur un simulateur pleine échelle du Groupe EDF est en cours d'organisation.

D3kode est une application reconnue au sein de l'équipe SILEX et dont l'utilisation est dès à présent envisagée par d'autres acteurs de l'équipe dans un but de visualisation et d'analyse de corpus de traces provenant d'activité diverses (site web, engagement en serious game,…). En cela D3kode prouve sa parfaite adéquation avec l'enjeu de son cahier des charges : l'exploitation et la manipulation du concept de trace modélisée.

D3kode est une application dont j'ai pensé l'architecture comme « évolutive ». Elle permet d'envisager différents types de perspectives ; technologiques, fonctionnelles et liées à la recherche.

Tout d'abord sur un plan technologique, il serait bon d'outiller D3kode avec une architecture de test. En effet ce qui n'était pas réellement possible initialement pour ce projet, car manquant de spécifications fixes, l'est à l'heure actuelle, car se basant sur un existant tangible. Il conviendrait donc de porter une réflexion sur les tests unitaires automatisés (JUnit) ainsi que sur des tests fonctionnels (Selenium). De plus la mise en place d'une plateforme d'intégration continue (Hudson) allié à un outil de gestion de l'automatisation de production (Maven) permettrait d'augmenter la qualité des développements et des tests. Tous ces outils peuvent aisément être intégrés à l'atelier logiciel Eclipse.

Ensuite d'un point de vue fonctionnel, une réflexion plus aboutie sur la gestion de la matrice rôles d'utilisateurs / droit applicatif pourrait être menée. Une gestion paramétrable des rôles applicatifs et de l'ensemble des droits d'accès à chaque fonctionnalité ainsi qu'une connexion avec un annuaire d'utilisateur (ldap) seraient appréciées dans un contexte industriel. De plus, l'ergonomie générale de l'application (qui pourrait être relative à un utilisateur ou un

rôle fonctionnel applicatif) est fonctionnelle mais mériterait aussi une réflexion de la part d'un ergonome, aidé par un ou des utilisateurs finaux. La gestion de création de requêtes a suscité un travail de conception graphique conséquent afin d'être intuitive et la plus guidée possible. Néanmoins une piste d'amélioration graphique pourrait être apportée pour abstraire un peu plus la théorie de la trace modélisée et être plus facile d'accès.

Enfin, des questions de recherches sur la signification et l'implémentation « d'observés négatifs » ou de « non observation » pourraient être envisagées. En effet, il est une chose de savoir quoi créer quand on recherche des actions d'activités précises, il en est une autre de savoir quoi créer lors du cas contraire.

Bibliographie

HETZNER S, STEINER CM, DIMITROVA V, et al, 2011, Adult self-regulated learning through linking experience in simulated and real world: a holistic approach, *EC-TEL'11 Proceedings of the 6th European conference on Technology enhanced learning: towards ubiquitous learning*, pp. 166--180

PASTRE P, 2005. De l'analyse du travail aux apprentissages professionnels. Editions Octarès, Toulouse, 363p

AUZENDE O, JOAB M, MOINARD C, 2000, La prise en compte de la dynamique dans la conception d'un système de formation fondé sur la simulation". In *Simulation et formation professionnelle dans l'industrie*, pp. 61--94.

Organigramme du LIRIS. In : Université Claude Bernard Lyon 1. Accueil, [en ligne]. Disponible sur http://liris.cnrs.fr/presentation/organigramme-du-liris (consulté le 02/02/2011)

CHAMPALLE O, 2010, Utilisation des traces d'interaction comme outils d'aide à l'observation sur simulateur [en ligne], *Dans 3èmes Rencontres Jeunes Chercheurs en EIAH, RJC-EIAH 2010, ATIEF* ed. Lyon. pp. 101-106. Disponible sur http://liris.cnrs.fr/Documents/Liris-4636.pdf. (Consulté le 05/02/2011)

SETTOUTI L.S., 2011, Systèmes à Base de traces modélisées - Modèles et langages pour l'exploitation des traces d'Interactions [en ligne]. Thèse de Doctorat en Informatique. Université Claude Bernard Lyon 1. 256 p. Disponible sur : http://liris.cnrs.fr/Documents/Liris-4984.pdf. (Consulté le 15/02/2011)

GEORGEON O. 2008, Analyse de traces d'activité pour la modélisation cognitive : Application à la conduite automobile [en ligne], Université Lumière Lyon 2. 322 p. Disponible sur : http://liris.cnrs.fr/Documents/Liris-4201.pdf. (Consulté le 7/02/2011).

DJOUAD T. 2011, Ingénierie des indicateurs d'activités à partir de traces modélisées pour un Environnement Informatique d' Apprentissage Humain [en ligne], Université de Constantine, 182 p. Disponible sur : http://liris.cnrs.fr/Documents/Liris-5360.pdf .(Consulté le 10/12/2011)

DJOUAD T., TBS-IM: Trace Bases System for Indicators calculus in Moodle, [en ligne] disponible sur http://liris.cnrs.fr/tdjouad/SBTIMStudentTeacher/TBSIM/Home.html (consulté le 03/02/2011)

CHAMPIN P-A., kTBS : a kernel for Trace-Based Systems, [en ligne]. Disponible sur : http://liris.cnrs.fr/sbt-dev/ktbs/doc/ . (Consulté le 07/02/2012)

DYKE G. 2009, Un modèle pour la gestion et la capitalisation d'analyses de traces d'activités en interaction collaborative, Ecole Nationale Supérieur des Mines, 280 p. Disponible sur : http://dl.dropbox.com/u/223827/2009-dyke-thesis.pdf . (Consulté le 17/02/2011)

MAY M, GEORGE S., PREVOT P,. 2011, TrAVis to Enhance Self-Monitoring in Online Learning Supported by Computer-Mediated Communication Tools. *International Journal of Computer Information Systems and Industrial Management Applications (IJCISIM)*, Vol. 3 No. 4, pp. 623-634. Disponible sur : http://www.mirlabs.org/[...]/Paper70.pdf. (Consulté 05/03/2011)

Table des annexes

Annexe 1
Guide utilisateur D3kode

1. Installation & Déploiement

1.1. Installation java

D3KODE nécessite qu'une JVM java soit installée sur le serveur où il est déployé.

La commande, tapée dans une invite de commande DOS ou Unix, java -version permet d'obtenir la version de la JVM installée. Celle-ci doit être au moins de version 1.6. Si tel n'était pas le cas il convient de mettre à jour (téléchargement Java).

1.2. Installation tomcat

Afin de pouvoir déployer l'application Web D3KODE il faut installer un serveur J2EE. D3KODE est développé et optimisé pour le serveur applicatif TOMCAT 7.0. (téléchargement tomcat 7.0) Le répertoire de base de tomcat sera appelé $HOME_TOMCAT dans la suite de ce document. Veiller à mettre la librairie Xerces (xercesImpl-2.7.1.jar) dans le répertoire lib de Tomcat. En effet un bug connu dans la librairie interne de Tomcat obligé à ajouter celle-ci.

1.3. Déploiement war

L'application D3KODE est packagée dans un fichier nommé mTraceProject.war et doit être copié dans le répertoire $HOME_TOMCAT/webapps. Il sera déployé lors du lancement du serveur applicatif avec la commande :

Sous Windows : ***$HOME_TOMCAT/bin/startup.bat***
Sous Unix : ***sh $HOME_TOMCAT/bin/startup.sh***

Pour arrêter le serveur les commandes sont :

Sous Windows : ***$HOME_TOMCAT/bin/shutdown.bat***
Sous Unix : ***sh $HOME_TOMCAT/bin/shutdown.sh***

Pour créer le fichier « war » il suffit de faire un export à partir d'Eclipse sur le projet d3kode.

1.4. Installation ktbs

Le kTBS est le Système de Gestion de Base de Trace (SGBT) utilisé par D3KODE. Il est écrit en python et demande un environnement particulier pour s'exécuter. (*cf documentation kTBS*)

Sous windows :

1. Installer cygwin qui est un logiciel permettant d'obtenir un environnement Unix sous Windows.

2. Installer Subversion comme expliqué à cette adresse. Il conviendra de vérifier que les packages, python 2.6, Mingw-gcc-core et Mingw-gcc-g++ sont installés à travers le setup de cygwin. Il faut aussi installer setuptools-0.6c11-py2.6 afin de pouvoir installer la librairie de gestion de RDF et Sparql rdflib.

a. Pour installer setuptools, à partir d'une invite de commande Unix : sh setuptools-0.6c11-py2.6.egg
b. Pour installer rdflib : décompresser l'archive et executer à partir de cette archive ***python setup.py install***

Sous Unix :

Procédez de même que sous Windows sans l'installation de Cygwin et de Mingw.

Les sources sont à récupérer sur le serveur SVN du liris à cette adresse ; par le biais d'un client SVN pour Subversion. Puis exécuter ***svn co https://svn.liris.cnrs.fr/sbt-dev/ktbs-rest-impl/trunk*** à partir du répertoire dans lequel on veut stocker le serveur kTBS.

Afin de pouvoir exécuter des requêtes sparql 1.1 il faut également copier au même niveau que les sources du serveur kTBS le fichier **Sparql1.1_4j.jar**.

1.5. Lancement ktbs

A partir d'un terminal Unix (cygwin sous Windows), aller dans le répertoire racine du kTBS et exécuter ***./bin/ktbs --ipv4*** . Si l'on veut lancer le kTBS avec une base de données persistante : ***./bin/ktbs –ipv4 –r <URL répertoire de sauvegarde>.***

1.6. Lancement de la base de données utilisateurs/rôles

Mettre dans le répertoire « database » le répertoire **d3kodedb** et la bibliothèque **hsqldb.jar**.

Vérifier la présence de la bibliothèque **hsqldb.jar** dans le répertoire lib de tomcat (***$HOME_TOMCAT/bin***).

Lancer la base de données hsqldb, à partir du répertoire database, par la commande :

```
java -cp hsqldb.jar org.hsqldb.Server -database.0 file:d3kodeDB/d3kodeDB -dbname.0 d3kodeDB
```

2. Identification

2.1. Mode d'identification et paramétrage

Lors d'un accès à une des pages de l'application D3KODE il convient d'être identifié. C'est pourquoi avant tout accès à l'application, une mire d'authentification est proposée.

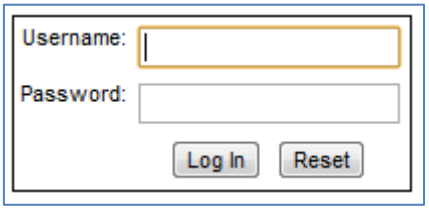


Figure 59 : Mire de login

L'utilisateur peut alors remplir un nom d'utilisateur et un mot de passe. Ceux-ci seront alors testés en fonction du paramétrage mis dans le fichier tomcat-users.xml du serveur Tomcat.

```
<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users>
  <role rolename="admin"/>
  <role rolename="tomcat"/>
  <role rolename="manager-gui"/>
  <role rolename="manager-script"/>

  <role rolename="expert"/>
  <role rolename="novice"/>
  <role rolename="stagiaire"/>
  <user username="dino" password="cosmas" roles="admin,expert,novice,stagiaire,tomcat,manager-gui,manager-script" />
  <user username="olivier" password="champalle" roles="stagiaire" />
</tomcat-users>
```

Figure 60 : Configuration des utilisateurs dans fichier tomcat-users.xml

Ainsi deux utilisateurs sont capables de s'identifier : Olivier et Dino. Olivier avec le seul rôle « stagiaire » et Dino avec l'ensemble des rôles liés à D3KODE ainsi que les rôles liés au serveur applicatif.

2.2. Droits liés à l'identification

Suite à une identification réussie, un ou plusieurs rôles sont affectés à l'utilisateur ce qui lui permettra l'accès ou non aux différentes fonctionnalités de D3KODE. L'ensemble de ces rôles est présenté à l'utilisateur sur la première page de l'application ainsi qu'en pieds de page.

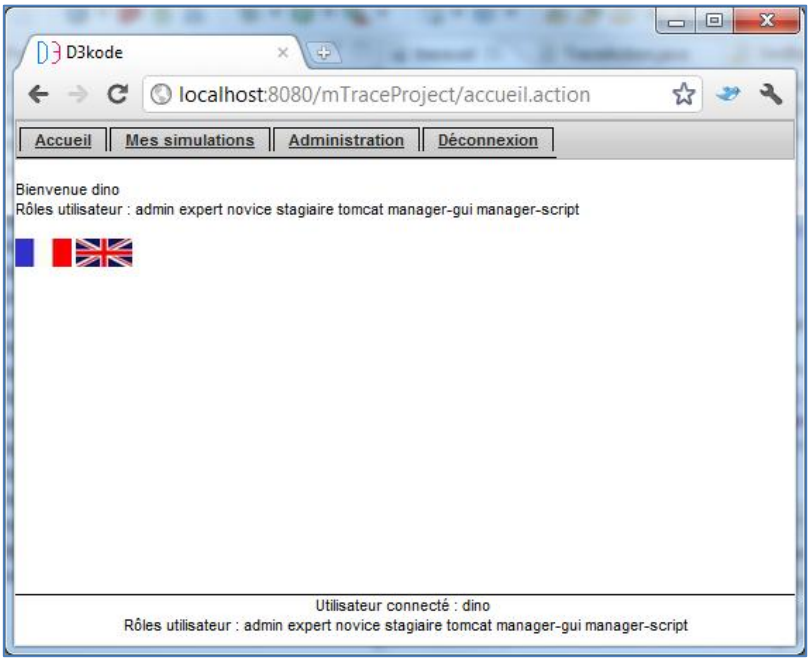


Figure 61 : Utilisateur authentifié avec rôle

3. Multilinguisme

3.1. Gestion des langues

Comme on peut le remarquer sur la *figure 61*, la langue utilisée au sein de D3KODE est paramétrable. A l'heure actuelle, seuls le Français et l'Anglais sont pris en charge mais il est possible d'ajouter n'importe quelle langue de manière assez simple.

3.2. Paramétrage

Chaque « élément texte » dans D3KODE est une clé dans un fichier de propriété et permet le multilinguisme. Le clique sur l'icône représentant le drapeau de la langue permet de fixer la variable locale de l'application et de choisir le bon fichier de langue.

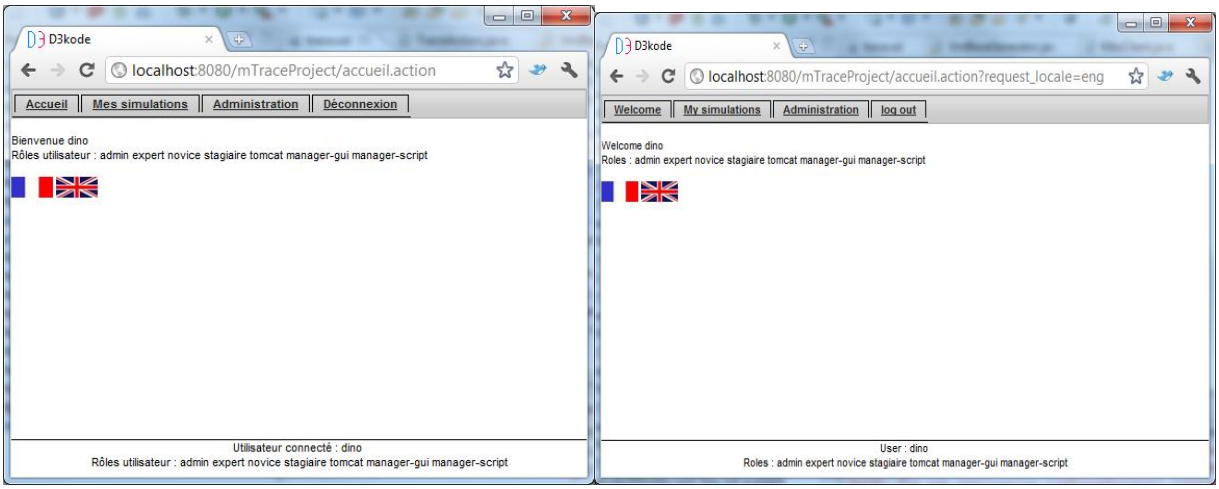


Figure 62 : Application en Français et en Anglais

```
layout.footer.user.connected=User :
welcome.user.roles=Roles :
welcome.message=Welcome
base.current.trace.model.empty=One TraceModel is requiered in current base.
upload.title=Load CSV file (extention .csv, size under 2mo)
base.manage=My simulations
layout.load.time=Loading page :
header.accueil=Welcome
header.manage=Administration
header.upload.file.csv=Upload CSV file
header.read.file.csv.inject.in.ktbs=Read CSV file &amp; injection kTBS
header.create.method.in.ktbs=Create method in kTBS
header.extract.ktbs.in.xml=Extraction kTBS in XML
header.create.file.svg.and.html=Create SVG file &amp; HTML
header.execute.method.in.ktbs=Create M-Trace
header.load.svg=SVG visualization
header.create.method.in.ktbs.where=Selection part of rule
header.create.method.in.ktbs.construct=Construction part of rule
header.logout=log out
```

```
layout.footer.user.connected=Utilisateur connecté :
welcome.user.roles=Rôles utilisateur :
welcome.message=Bienvenue
layout.load.time=Chargement de la page en :
base.current.trace.model.empty=La base de travail courrante ne comporte pas de modèle de trace
base.manage=Mes simulations
header.accueil=Accueil
header.manage=Administration
header.upload.file.csv=Chargement fichier CSV
header.create.method.in.ktbs.where=Création de la partie sélection d''une règle
header.create.method.in.ktbs.construct=Création de la partie construction d''une règle
header.create.transformation.in.ktbs=Création de transformation dans kTBS
header.read.file.csv.inject.in.ktbs=Lecture fichier CSV &amp; injection kTBS
header.extract.ktbs.in.xml=Extraction kTBS en XML
header.create.file.svg.and.html=Création fichier SVG &amp; HTML
header.load.svg=Chargement SVG
header.execute.method.in.ktbs=Création d''une M-Trace
header.logout=Déconnexion
```

Figure 63 : Correspondance clé-valeur pour multilinguisme

4. Gestion modèle de Trace

4.1. Principe

Le but de cette partie de D3KODE est de permettre la création, modification, suppression et consultation des ressources : Modèle de trace, Type d'observé et Type d'attribut. Le modèle de trace contient l'ensemble des types d'observé et des types d'attribut qui permette de définir le vocabulaire d'une famille de traces d'activité. En effet les observés qui seront au sein d'une trace auront pour type un des types définis dans le modèle de trace, et pour attribut un ensemble typé parmi la liste des types d'attributs.

4.2. Ajout d'un modèle de trace

L'ajout d'un modèle de trace se fait tout d'abord par le choix d'un libellé. Puis par l'ajout de types d'observés et de types d'attributs soit par le biais de l'IHM soit par injection de données au format CSV.

4.2.1. Traitement IHM

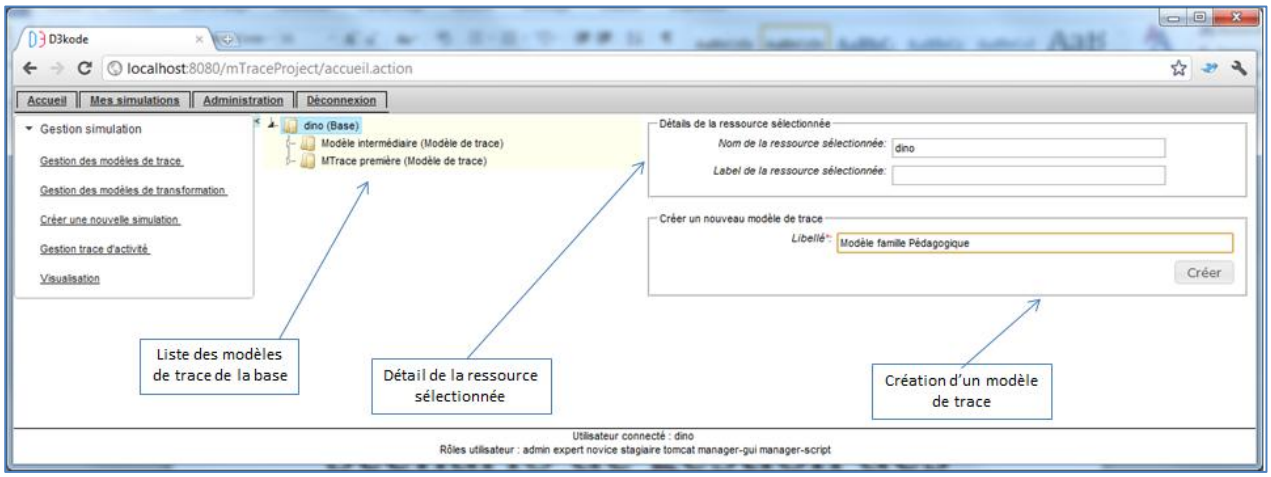


Figure 64 : Sélection d'une base dans la gestion des modèles de trace

ors de la sélection de la base (qui porte le nom de la personne connectée/identifiée) la liste des modèles de trace présents sur le SGBT apparaît. L'on peut alors voir les détails de la ressource sélectionnée et créer une ressource « fille » à la ressource sélectionnée.

3 cas sont possibles :

- Si l'on sélectionne la base, la ressource fille est un modèle de trace.
- Si l'on sélectionne un modèle de trace la ressource fille est un type d'observé.
- Si l'on sélectionne un type d'observé la ressource fille est un type d'attribut.

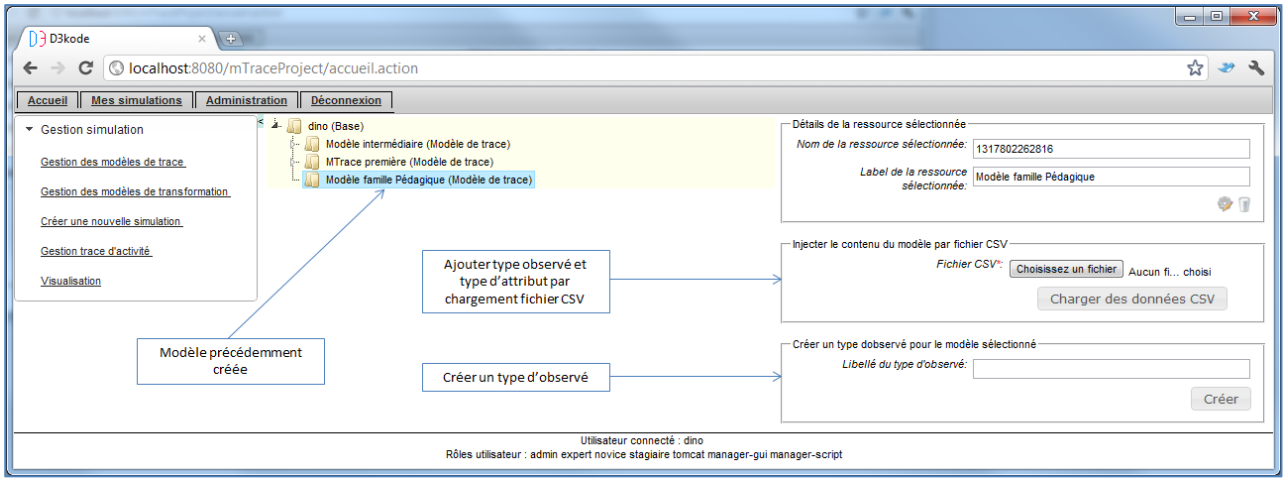


Figure 65 : Choix de l'ajout de type d'observé dans le modèle de trace

Suite à la sélection du modèle de trace *(Figure 65)*, il est alors possible de créer un type d'observé directement par l'IHM de D3KODE.

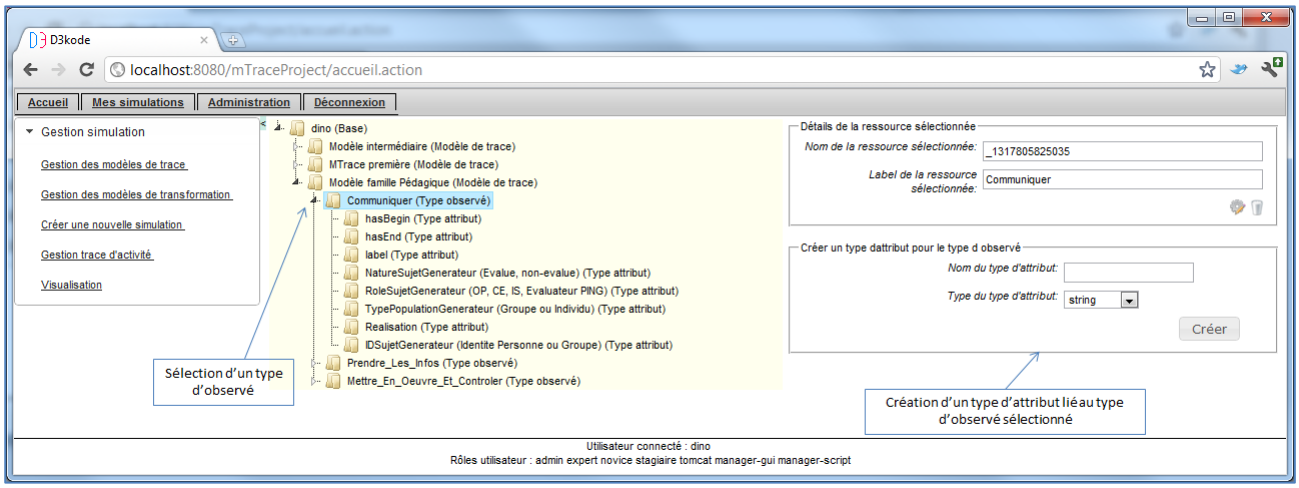


Figure 66 : Création d'un nouveau type d'attribut par l'IHM

Suite à la sélection d'un type d'observé *(Figure 66)* il est possible de créer un type d'attribut en spécifiant son libellé et son type.

4.2.2. Traitement par import de fichier CSV

Comme vu dans la *figure 65* il est possible de charger un fichier au format CSV pour créer les type d'observés et types d'attributs d'un modèle de trace. Le clique sur le bouton « Choisissez un fichier » fait apparaitre une fenêtre classique de recherche de fichier et permet de sélectionner un fichier au format CSV de la machine cliente dont les spécifications sont présentées au sein de la *figure 67*.

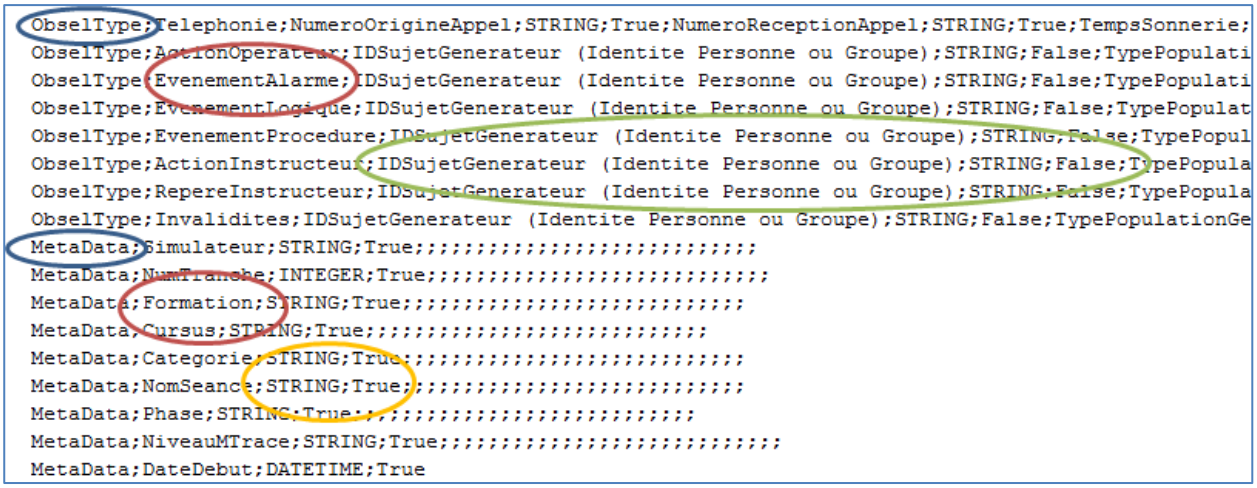


Figure 67 : Extrait de contenu CSV d'un modèle de trace

Le format CSV permet une vision tabulaire simple (structurellement parlant) de données. Chaque donnée est séparée de la suivante par un « ; » ou par un retour à la ligne. Au sein de l'application D3KODE voici la structure qui a été retenue :

1ère colonne : **type** de la ressource (ellipse bleue) → soit un **obselType** (type d'observé) soit **MetaData** (métadonnée du modèle de trace).

2ème colonne : **libellé** de la ressource (ellipse rouge)

Triplet de colonne suivant : **Nom de l'attribut**, **Type de l'Attribut**, **Obligation de présence** (ellipse verte) pour le type d'observé et seulement **type** et **obligation de présence** (ellipse jaune) pour les métadonnées.

5. Gestion de trace injectée

5.1. Spécification de fichier CSV

Une fois que l'on a créé le(s) modèle(s) de trace(s) l'on peut ajouter les données d'une trace ; un fichier au format CSV peut être chargé. Il contient l'ensemble des observés qui seront ajoutés à la trace modélisée au sein du SGBT.

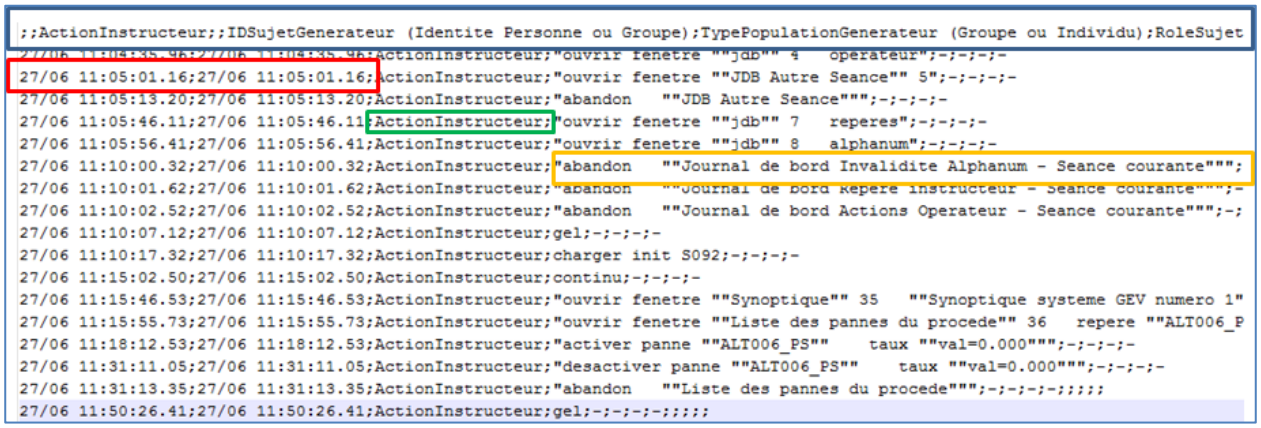


Figure 68 : Extrait de contenu CSV pour création d'une trace modélisée

Il convient en premier lieu de rappeler la structure du type d'observé que l'on veut charger (rectangle bleu). Il y a **deux colonnes vides** correspondant en fait à date de début et date de fin (*figure 68*) puis **le nom du type d'observé** et enfin la liste **des noms des types d'attribut** du type d'observé.

Ensuite vient l'ensemble des lignes représentant chacune un observé. Chacune commence par **deux dates** (rectangle rouge) ; date de début et date de fin (au format DD/MM HH:mm:ss.SSS ou YYYY/MM/DD HH:mm:ss.SSS), suivi du **nom du type d'observé** (rectangle vert) et enfin de la liste **des valeurs de ses attributs** (rectangle jaune).

5.2. Injection des données

Une fois ce fichier constitué ou récupéré il est possible de créer au sein de D3KODE une trace modélisée par injection de données.

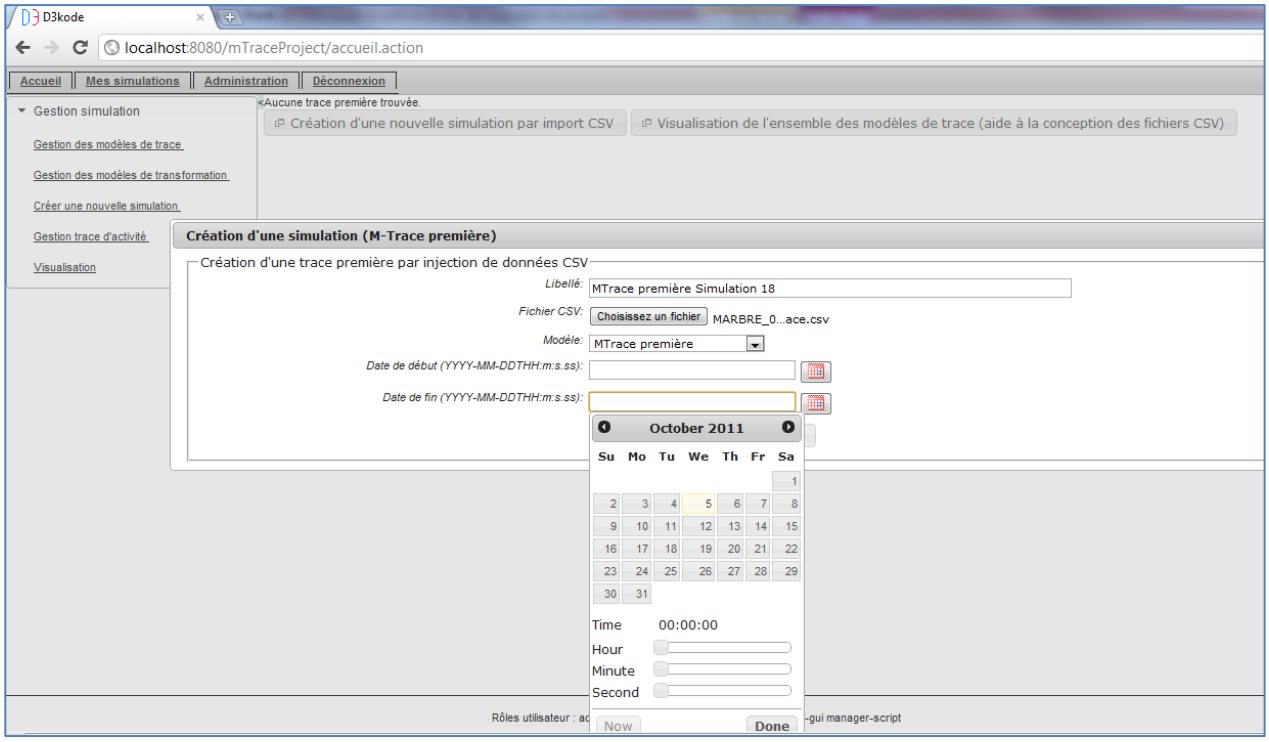


Figure 69 : Création d'une trace modélisée

Il convient de renseigner au minimum : un **libellé** pour la trace modélisée, le **fichier** utilisé pour injecter les observés, et **le modèle** auquel la trace correspond. Par défaut seront récupérées la date de début la plus petite et la date de fin la plus grande pour borner la trace modélisée, mais il est possible d'en spécifier d'autre. Après clique sur le bouton **Créer** l'ensemble des lignes contenues dans le fichier CSV sera converti en observé et inséré dans la trace modélisée crée.

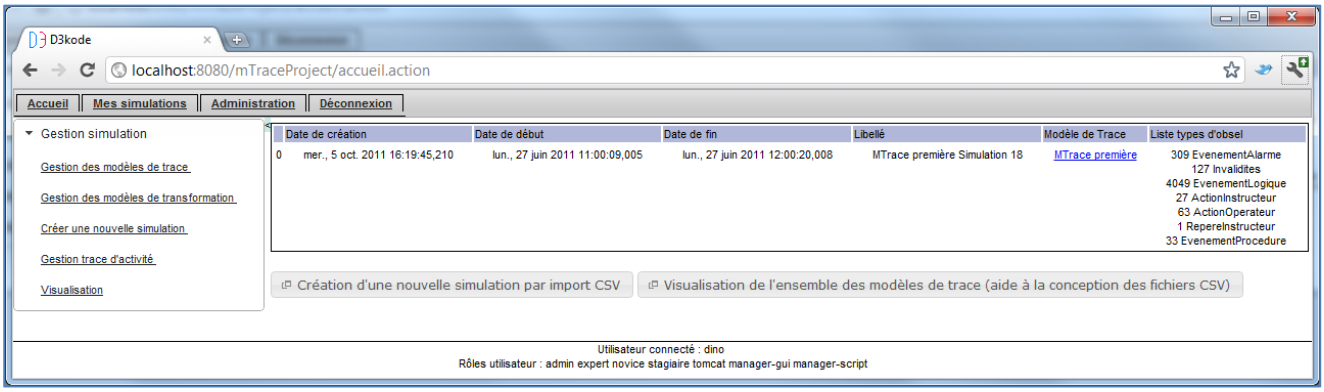


Figure 70 : Trace première injectée

Suite à la création de la trace modélisée, les informations sur sa date de création, son intervalle de date de début/fin, son libellé, son modèle de trace ainsi que quelques statistiques sur les types d'observés qu'elle comprend sont présentés.

5.3. Aide à la création de fichier CSV pour la création de trace

Afin d'aider à la conception de fichier CSV, il est possible sur clique du bouton « Visualisation de l'ensemble des modèles de trace (aide à la conception des fichiers CSV) de visualiser l'ensemble des types d'attributs de l'ensemble des types d'observés de l'ensemble des modèles de traces.

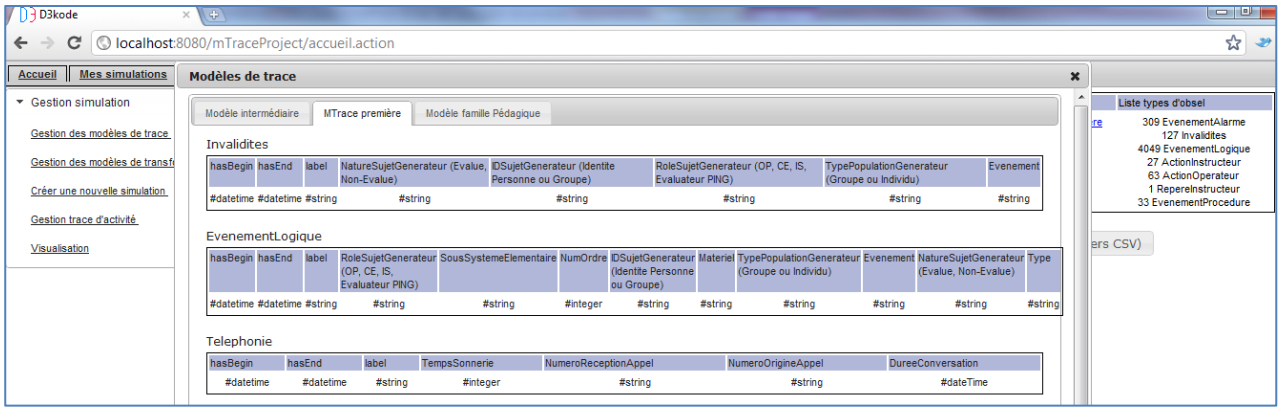


Figure 71 : Aide à la conception CSV pour trace modélisée

Remarque : les trois premières colonnes ne sont pas à spécifier dans le fichier CSV pour la définition de la liste des attributs d'un type d'observé. En effet tout observé se doit d'avoir une date de début, une date de fin et un libellé.

6. Gestion des transformations

6.1. Définition

Une transformation est un mécanisme basée sur un ensemble de 1 à N règles permettant de créer une nouvelle trace à partir d'une trace source. Chaque règle s'exécute sur l'ensemble de la trace à la recherche d'observé correspondant à ses critères de sélection et créera un observé correspondant à ses critères de construction.

6.2. Création d'une transformation

Une transformation possède un libellé, un modèle de trace source, un modèle de destination et un ensemble de règle. La trace sur laquelle sera appliquée cette transformation devra avoir pour modèle le modèle de trace source de la transformation. La trace résultante de la transformation aura pour modèle la trace cible de la transformation.

Plus précisément, chaque règle de la transformation va produire une trace intermédiaire (qui aura pour modèle le modèle de destination de la transformation) et l'ensemble de ces traces sera fusionné pour produire la trace transformée résultant de la transformation.

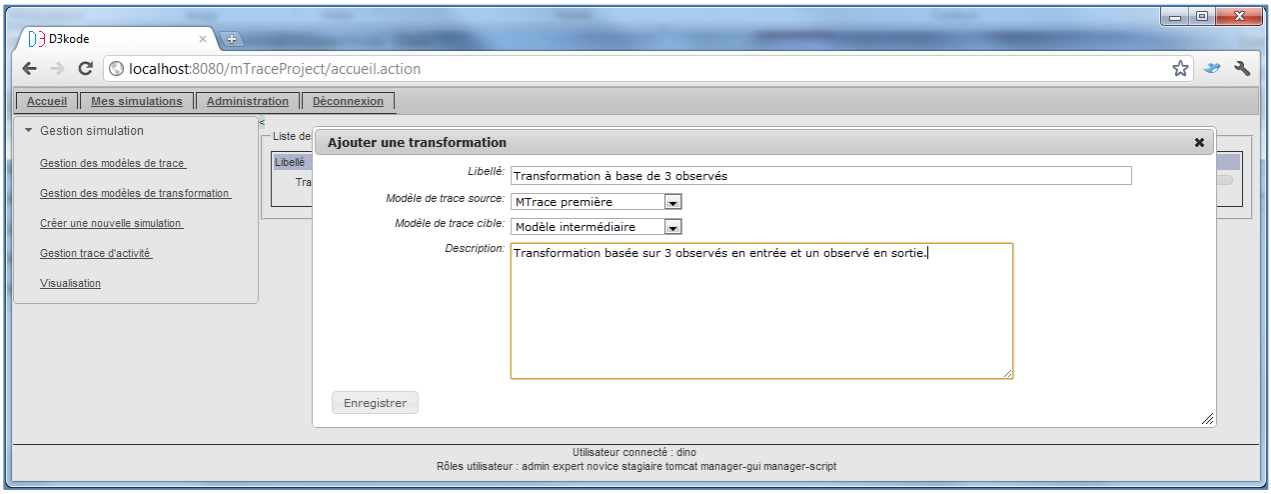


Figure 72 : Création d'une transformation

Suite au clique sur Enregistrer, la transformation est initialisée au sein de D3KODE mais pas encore au niveau du SGBT. Il est alors possible de modifier les propriétés de la transformation, lui ajouter des règles et la supprimer. Une fois qu'au moins une règle lui aura été ajoutée, elle pourra être enregistrée dans le SGBT.

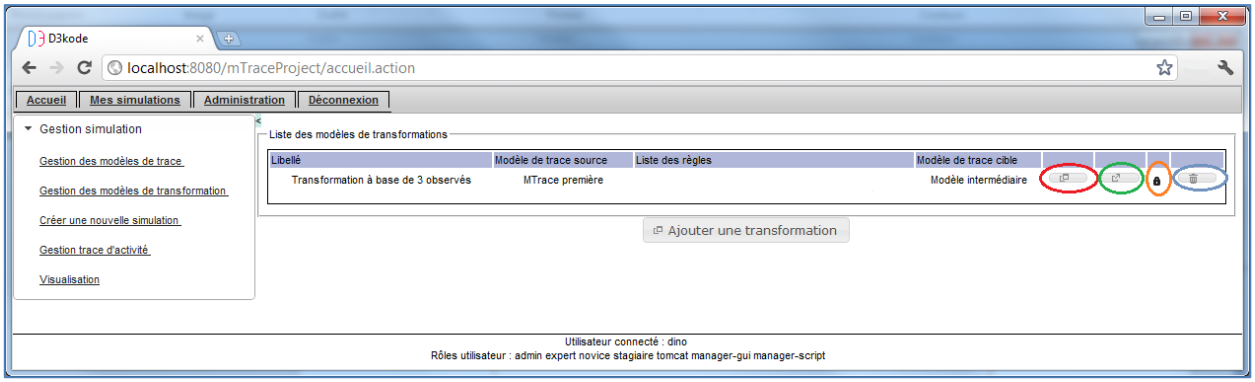


Figure 73 : Transformation initialisée

L'ellipse rouge indique la possibilité de modifier les attributs de la transformation, comme son libellé, ses modèles de trace en entrée et en sortie.

L'ellipse verte indique la possibilité d'ajouter des règles à cette transformation.

L'ellipse orange indique un cadenas fermé, ce qui signifie que la transformation ne peut être enregistrée dans le SGBT ou qu'elle existe déjà.

L'ellipse bleue permet de supprimer la transformation.

6.3. Création de règle

La création d'une règle demande à renseigner tout d'abord un libellé et une description.

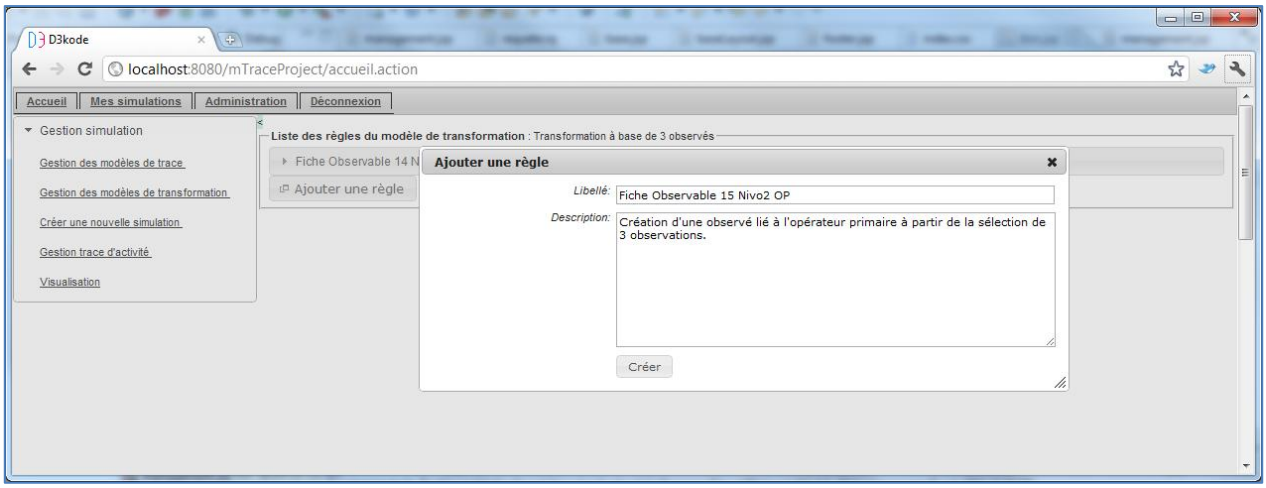


Figure 74 : Création d'une règle

Puis la construction d'une règle s'effectue en 2 principales étapes : la partie construction et la partie condition. La partie construction va spécifier le type d'observé qui sera créé alors que la partie condition spécifiera la liste des critères à respecter pour la construction.

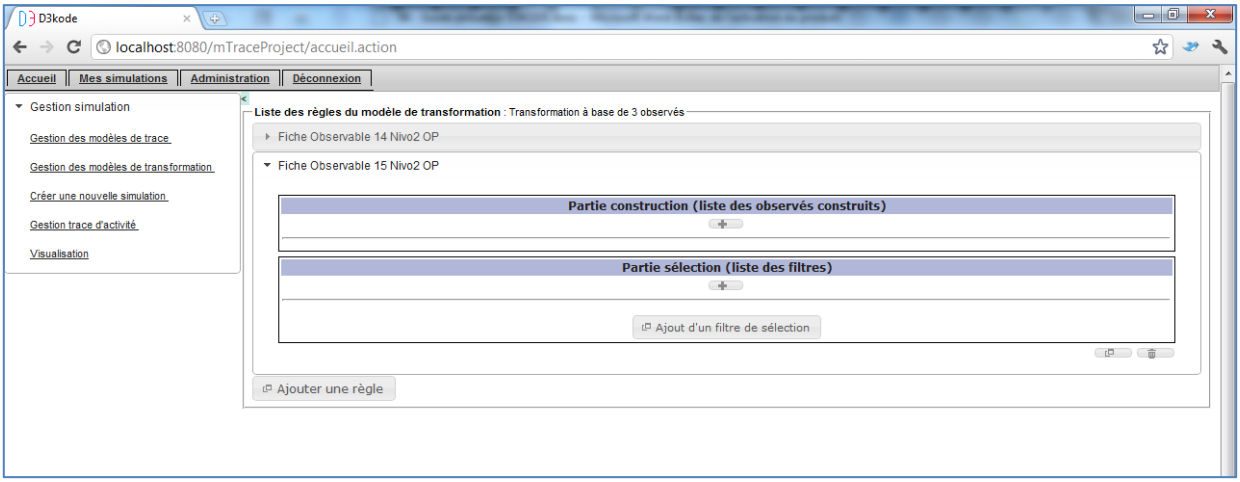


Figure 75 : Spécification d'une règle

Pour chaque partie un bouton + permet d'ajouter des instances de types d'observés. Deux autres boutons d'action se trouvent en bas à droite de la fenêtre. Ils permettent de modifier les attributs libellés et description de la transformation ou de la supprimer.

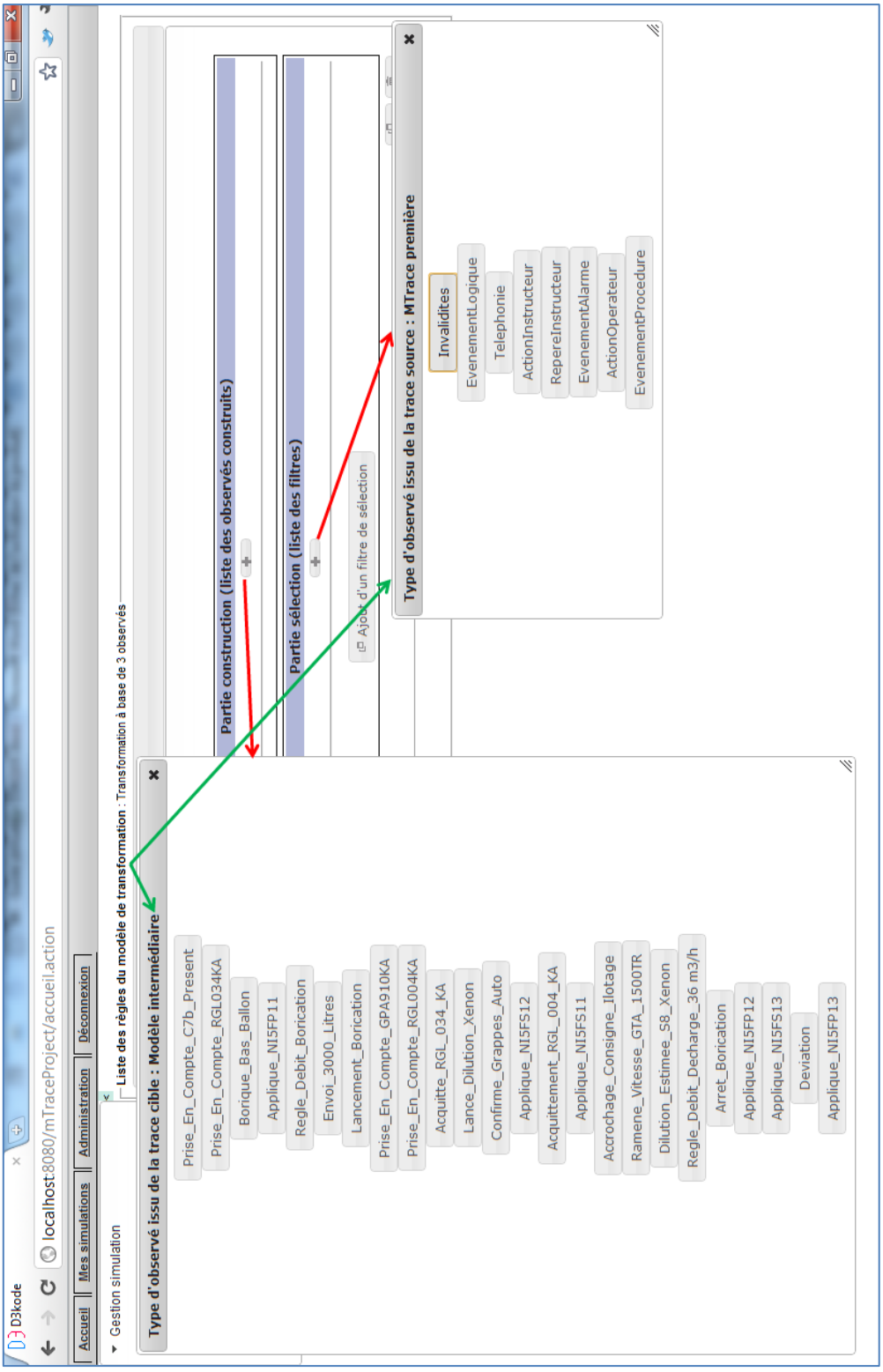


Figure 76 : Choix des instances de type d'observé en entrée et en sortie

6.3.1. Création de la partie condition d'une règle

Afin de paramétrer les critères de sélection d'une règle il faut tout d'abord sélectionner les types d'observés sur lesquels vont porter les critères de sélection.

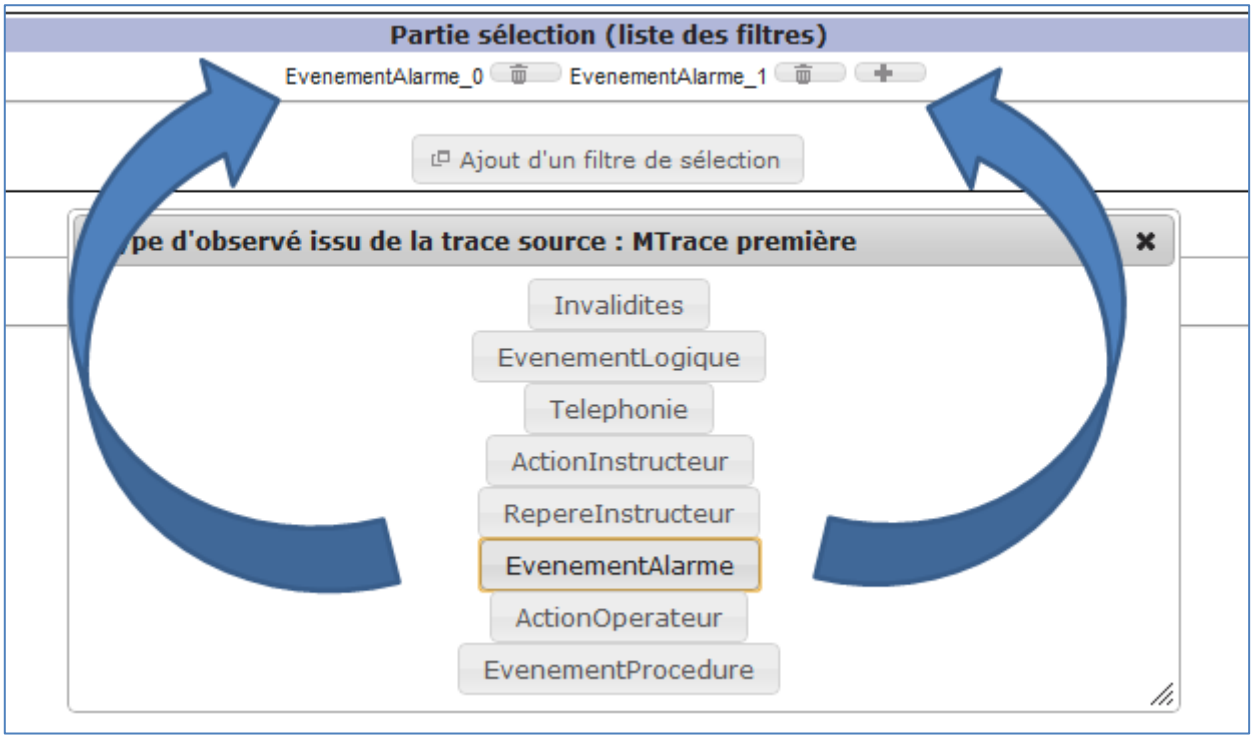


Figure 77 : Ajout d'instance de type d'observé dans la partie condition

Ainsi l'on vient de spécifier que cette règle se basera sur deux alarmes. Il a suffi de cliquer deux fois sur le bouton « EvenementAlarme » pour créer les deux instances EvenementAlarme_0 et EvenementAlarme_1. Suite à cette sélection il est possible de spécifier les critères sélections basé sur les attributs de ces instances de types d'observés.

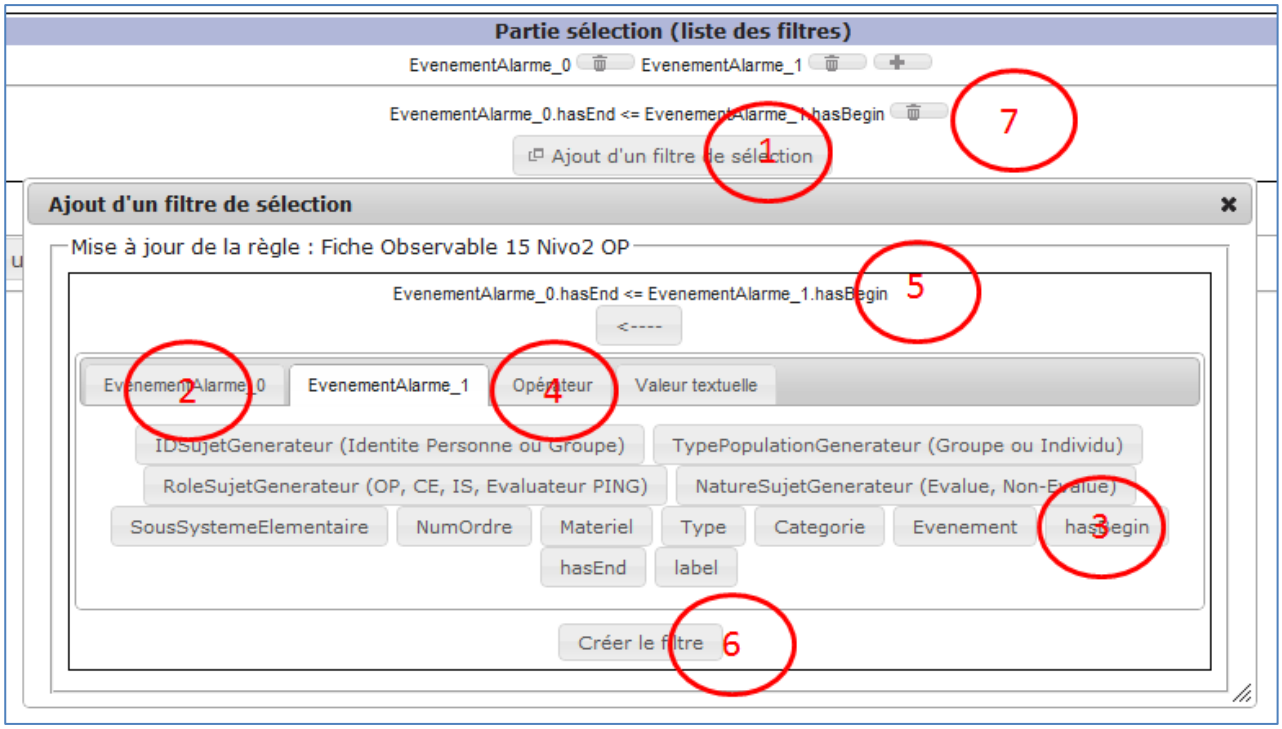


Figure 78 : Création d'un critère (temporel) de sélection

Liste des étapes pour l'ajout d'un critère de sélection dans la partie condition :

1. Clique sur le bouton d'ajout de filtre de sélection. La fenêtre de type « calculette » apparait.
2. Sélectionner parmi les onglets présent celui de l'observé sur lequel l'on choisira le premier opérande de l'expression.
3. Sélectionner l'attribut permettant de spécifier pleinement l'opérande. Il est aussi possible de saisir une valeur textuelle typée.

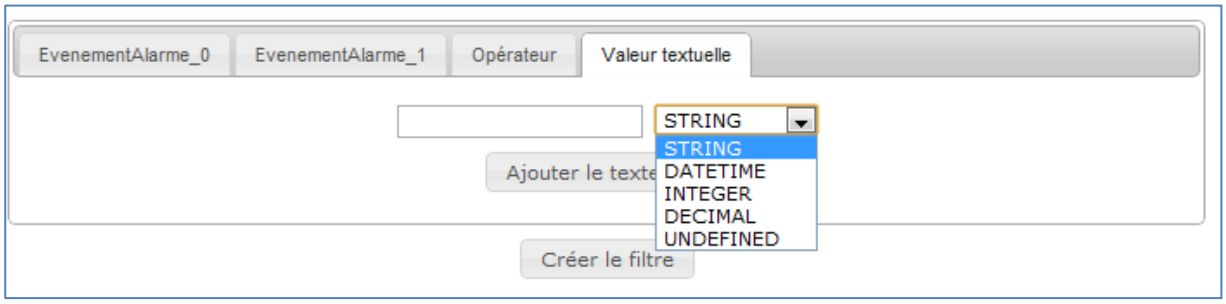


Figure 79 : Saisir d'une valeur textuelle typée

4. Sélectionner l'onglet « Opérateur » afin de choisir l'opérateur de l'expression.

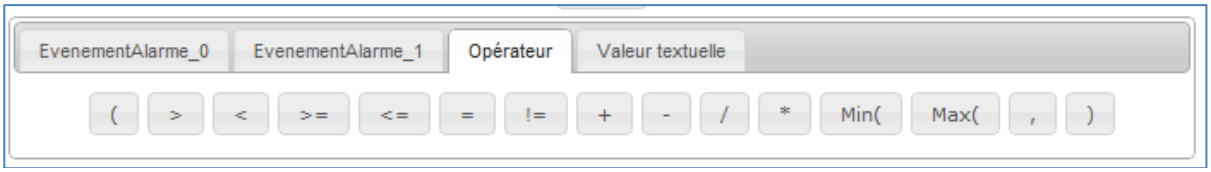


Figure 80 : Liste des opérateurs

5. On peut observer que l'expression se construit dynamiquement
6. Une fois l'expression finie, il suffit de cliquer sur «Créer le filtre » pour ajouter ce nouveau filtre à la liste des filtres de la partie condition de la règle.
7. Les filtres peuvent être supprimés par clique sur l'icône « poubelle »

Après plusieurs itérations de ces étapes, voici un exemple d'ensemble de filtre.

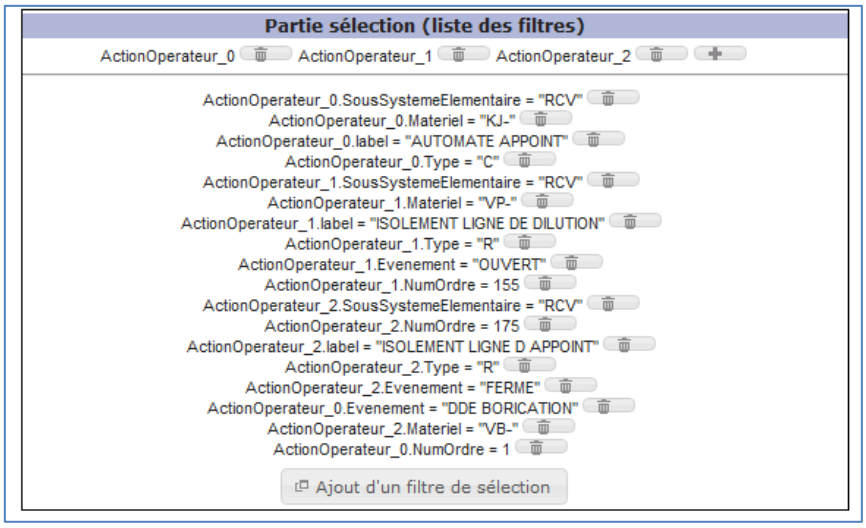


Figure 81 : Ensemble de filtre de la partie condition d'une règle

6.3.2. Création de la partie construction d'une règle

La partie construction permet de spécifier le ou les observés que l'on va créer en fonction des observé sélectionné dans la partie condition. A l'instar de la partie condition, il est possible par clique sur le bouton « + » d'ajouter des instances de types d'observés dans la partie construction. Un tableau est alors présenté à l'utilisateur permettant d'affecter une valeur ou une variable à chaque attribut de cette instance de type d'observé.

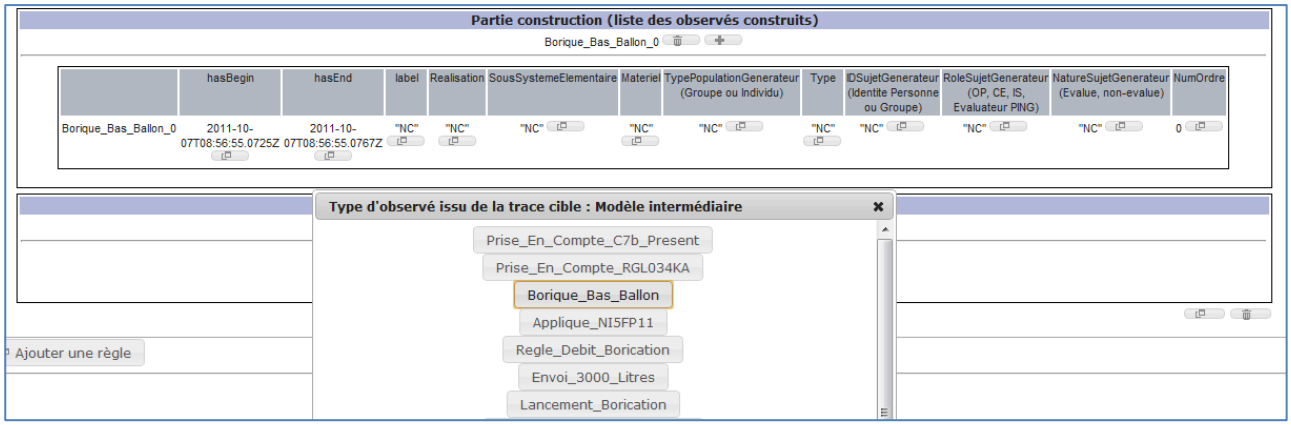


Figure 82 : Partie construction d'une règle

Ainsi les champs sont initialisés par une valeur par défaut conforme à leur type (Datetime, String, Integer). A l'aide du bouton icône présent dans chaque cellule de chaque attribut il est possible d'affecter une valeur, aidé par une fenêtre IHM similaire à ce qui a été présenté précédemment (*Figures 20-22*).

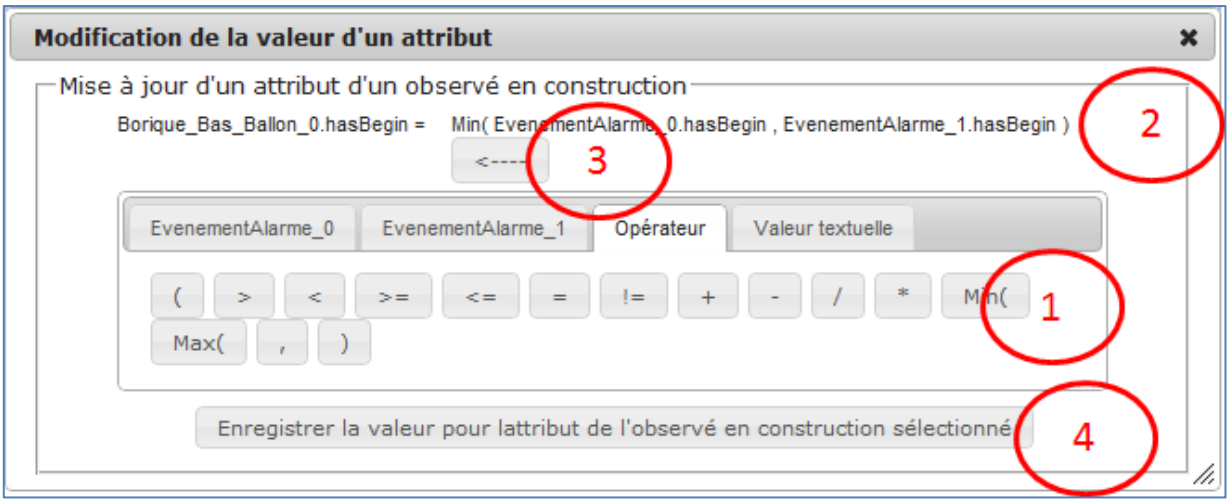


Figure 83 : Modification d'un attribut de l'observé construit

Différentes étapes sont à effectuer pour réaliser l'expression qui sera interprétée pour affecter la valeur de l'attribut sélectionné.

1. Choisir opérateur ou opérande à travers les onglets et listes d'attributs proposés
2. Constater l'élaboration en temps réel de l'expression
3. Supprimer le dernier opérande ou opérateur inséré
4. Enregistrer l'expression

L'expression signifie ici que l'on souhaite affecté comme valeur de début à l'observé « Borique_Bas_Ballon » la valeur minimale entre les deux valeurs de début des deux « EvenementAlarme » sélectionnés.

En procédant de même pour l'ensemble des attributs de l'observé que l'on veut construire, l'on peut obtenir ceci :

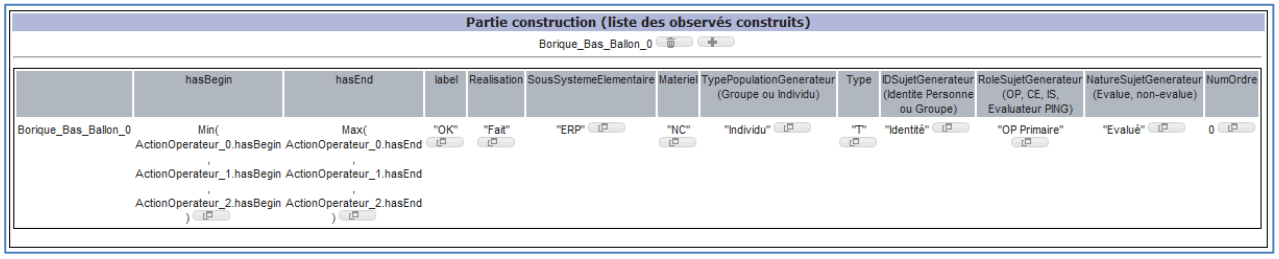


Figure 84 : Partie construction complète

Une fois la règle finie, il est possible, au travers la « Gestion des modèles de transformation » d'enregistrer la transformation au sein du SGBT. Elle sera alors utilisable au sien de D3KODE.

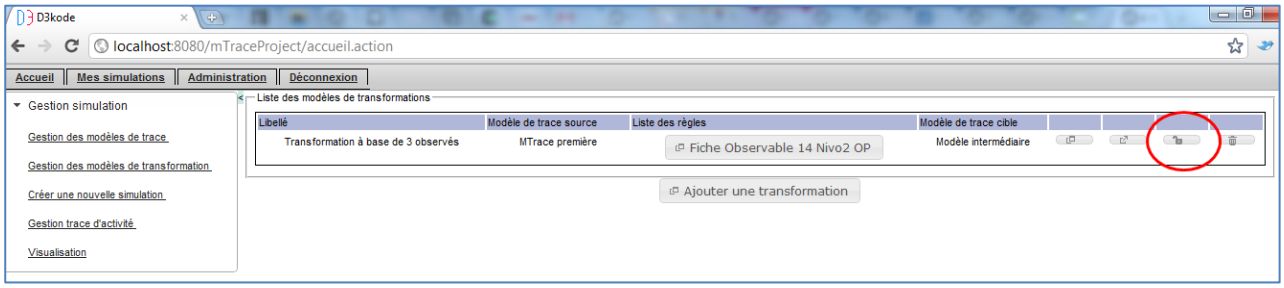


Figure 85 : Enregistrement dans SGBT du modèle de transformation

La liste des règles contient une à N règle. Le cadenas ouvert signifie que la transformation présente dans D3KODE n'est pas identique à celle présente dans le SGBT. Lors du clique sur le cadenas la transformation, ainsi que l'ensemble des règles qui la compose seront enregistrés dans le SGBT.

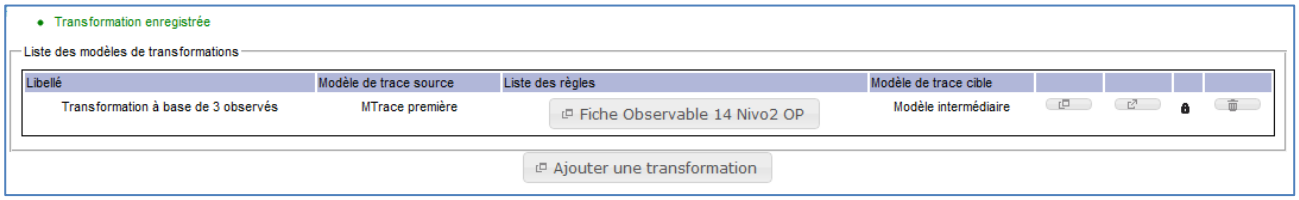


Figure 86 : Transformation enregistrée dans le SGBT

7. Gestion de trace calculée

7.1. Exécution d'une transformation

A travers l'interface de gestion des traces calculées l'on peut créer de nouvelles traces d'activités à partir d'une trace source sur laquelle on applique une transformation.

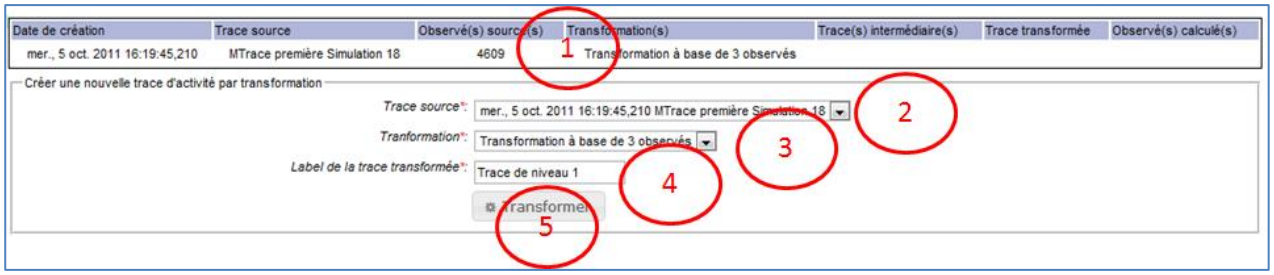


Figure 87 : Création d'une trace transformée

Premièrement, un tableau présentant l'ensemble des traces pouvant servir de source à une transformation est présent. Ensuite un cartouche permet de choisir la trace source, la transformation à exécuter puis le libellé que l'on souhaite affecter à la trace résultante de l'exécution de la transformation. Il est à noter que la liste des transformations présentées est fonction de la trace source. C'est-à-dire que le modèle de trace de la trace source doit correspondre avec le modèle de trace source de la transformation. Enfin, valider le formulaire par clique sur le bouton d'action « transformer » ; le processus de transformation sera lancé.

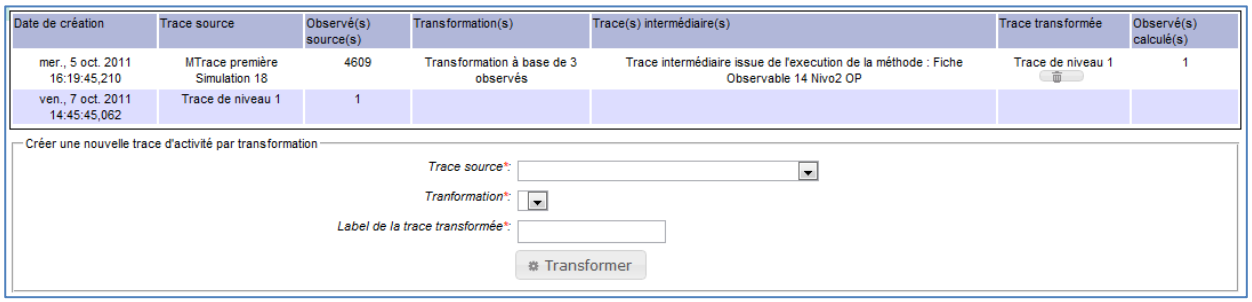


Figure 88 : Trace calculée créée

L'exécution de la transformation a créé une nouvelle trace comportant un observé. Cette nouvelle trace peut alors à son tour être source d'une nouvelle transformation. Elle possède comme modèle de trace le modèle de trace cible déclaré dans le modèle de transformation. Si l'on modifie la transformation qui a permis la création de cette trace calculée, celle-ci sera automatiquement répercutée.

8. Gestion de la visualisation d'un corpus de trace

Suite à la création de trace par injection de données ou par exécution de transformation, l'on obtient un corpus de traces modélisées. Grâce à l'interface de visualisation il va être possible d'obtenir différentes représentation de ce corpus de trace. Une vision tabulaire récapitulant l'ensemble des données propres aux traces, ainsi qu'une vision par graphique SVG permettant une visualisation imagée du corpus de trace.

8.1. Visualisation tabulaire

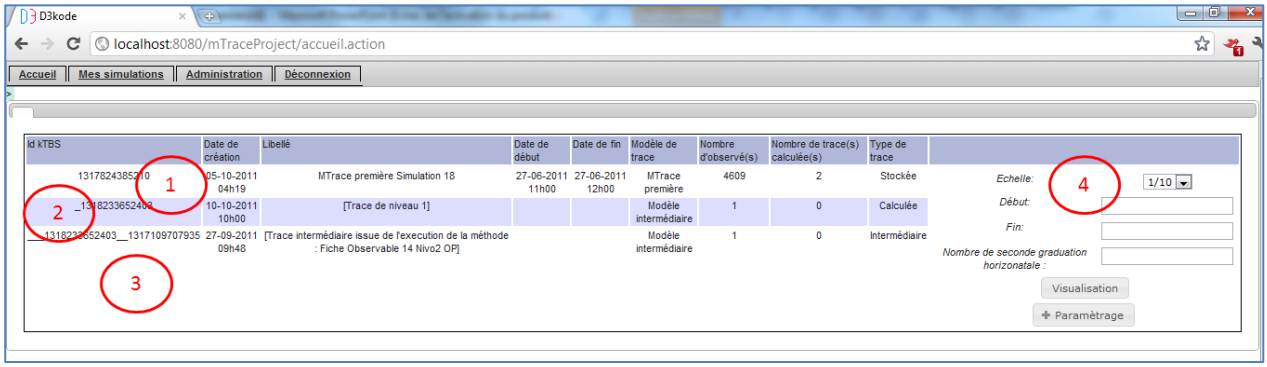


Figure 89 : Vision tabulaire d'un corpus de trace

L'ensemble des traces stockées, calculées ou intermédiaires sont présentées.

1. Une trace est considérée comme stockée à partir du moment où elle n'a pas été le résultat d'une transformation.
2. Une trace calculée est une trace qui est le résultat d'une transformation.
3. Une trace intermédiaire est une trace résultant d'une des règles définies dans la transformation.
4. Le paramétrage de la visualisation graphique

8.2. Paramétrage de la visualisation SVG

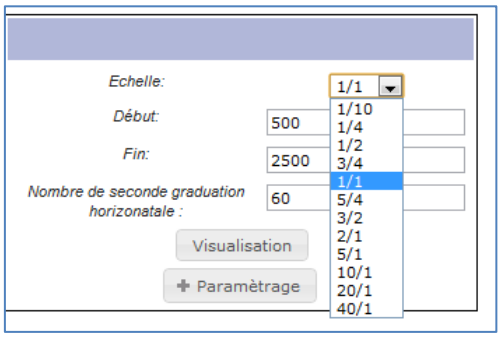


Figure 90 : Paramétrage échelle, dates et intervalle

L'échelle est la proportion entre les pixels affichés au sein du dessin SVG et la représentation d'une seconde. En échelle 1/10 le dessin sera 10 fois moins long qu'en 1/1.

La date de début est le décalage en seconde par rapport à la date de début de la trace d'où l'on veut démarrer la représentation graphique. Par défaut ce sera 0 seconde.

La date de fin est le décalage en seconde par rapport à la date de fin de la trace d'où l'on veut arrêter la représentation graphique. Par défaut ce sera 0 seconde.

Le nombre de seconde entre 2 graduations du dessin SVG permet d'affiner la précision au sein de la représentation graphique.

Enfin le bouton « Paramétrage » permet de spécifier l'iconographie liée à la représentation graphique des observés au sein de la trace.

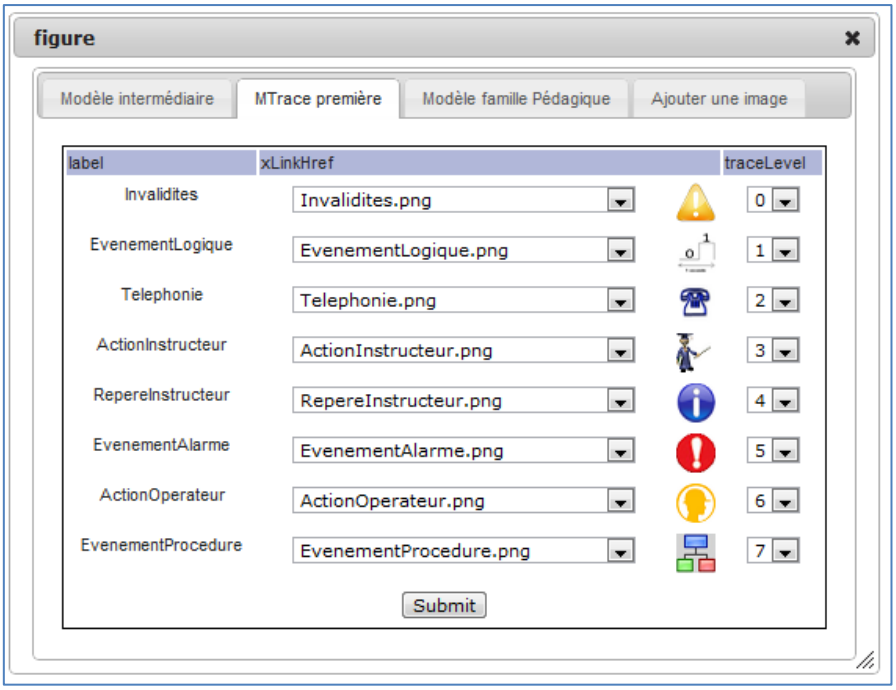


Figure 91 : Iconographie et positionnements des représentations graphiques des observés

Cette interface permet de renseigner une icône pour chaque type d'observé de chaque modèle de trace, ainsi qu'un positionnement vertical afin d'aider à la lecture du corpus de trace. Des icônes sont présentes au sein de l'application mais il est possible d'en ajouter par l'onglet « ajout d'image ».

8.3. Visualisation SVG

Une fois ces paramétrages renseigné il est possible de créer une représentation graphique du corpus de trace. Il suffit pour cela de cliquer sur le bouton « Visualisation ». Un dessin SVG est alors créé et présenté au sein du navigateur.

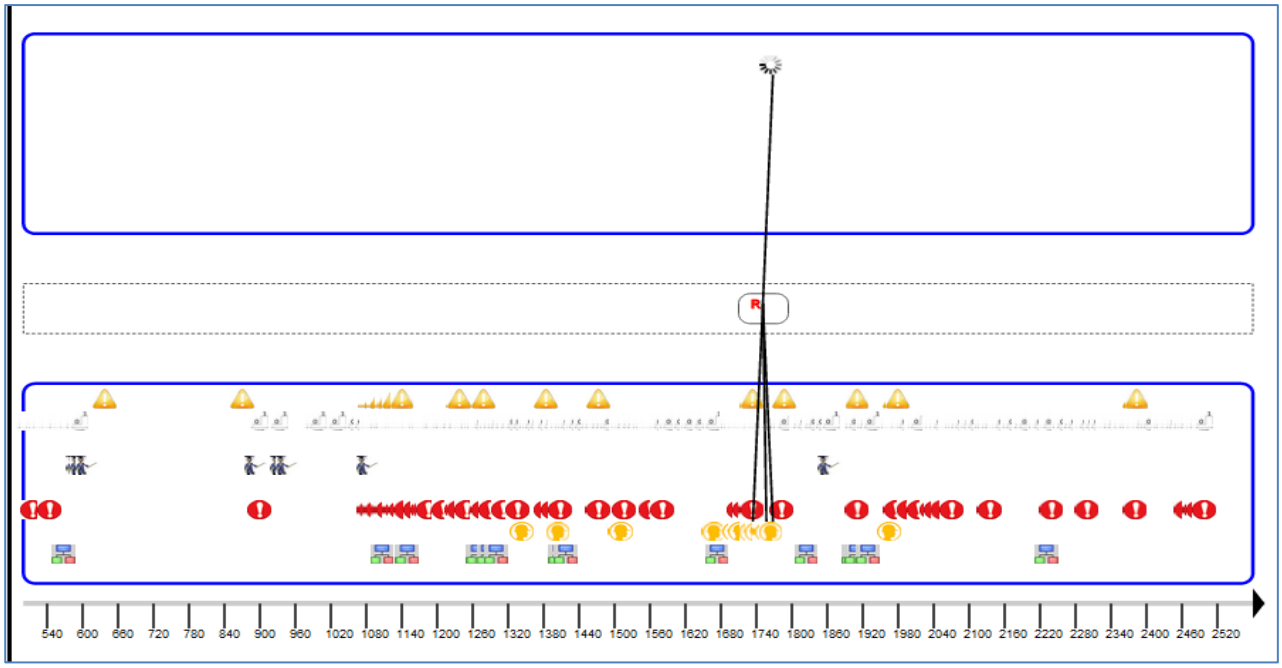


Figure 92 : Visualisation d'un corpus de traces modélisées

La trace première située à la base du dessin est bornée entre 500 et 2500 secondes. Chaque icône représente un observé et est positionnée en fonction du paramétrage. Un décalage vertical au sein de la trace est opéré afin de rendre plus lisible le dessin. Au-dessus de cette trace figure une transformation comportant une règle. Des traits relient la règle aux observés sur laquelle elle porte en entré et sortie. Afin de permettre une meilleure lisibilité l'on peut modifier l'intervalle de visualisation ainsi que l'échelle.

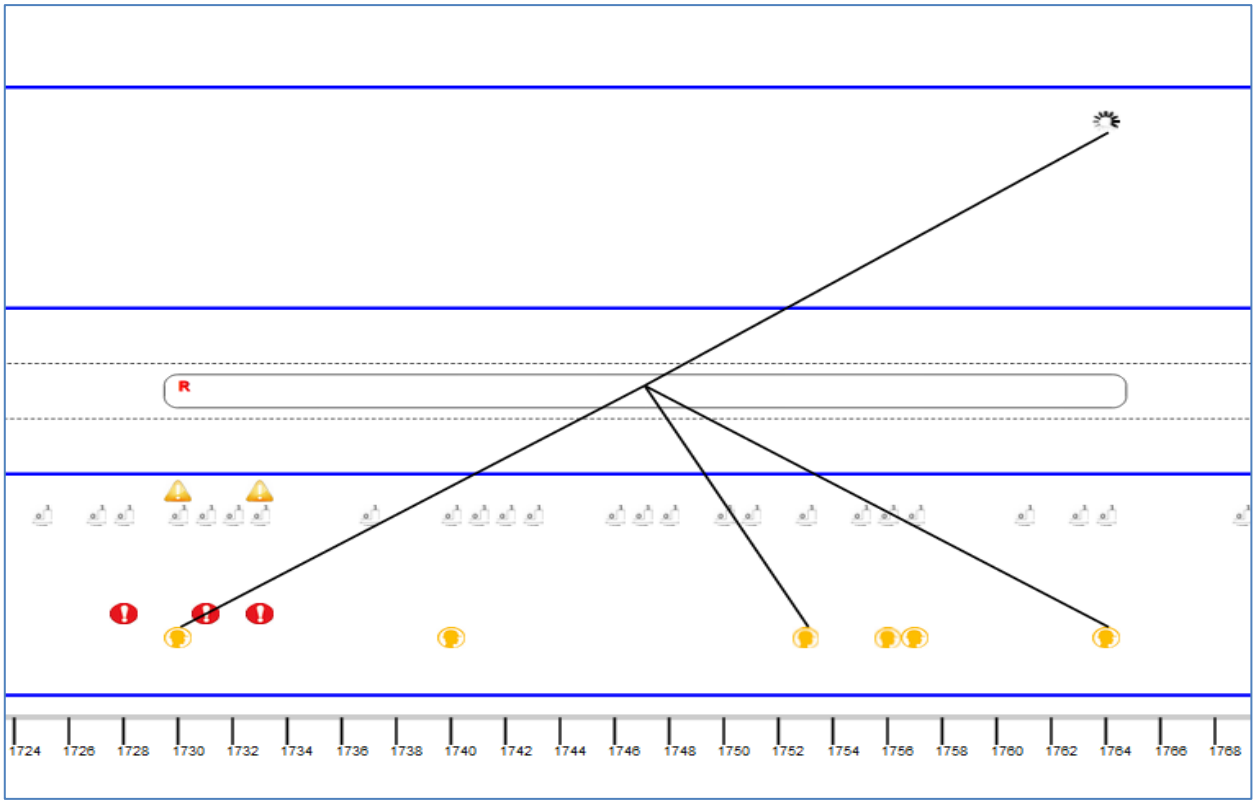


Figure 93 : Visualisation centrée sur une règle de transformation

En spécifiant un intervalle entre 1680 et 1800 secondes, une échelle à 20 et une graduation à 2 secondes, l'on peut obtenir un dessin centré sur la règle de transformation. Il est alors possible d'étudier la transformation.

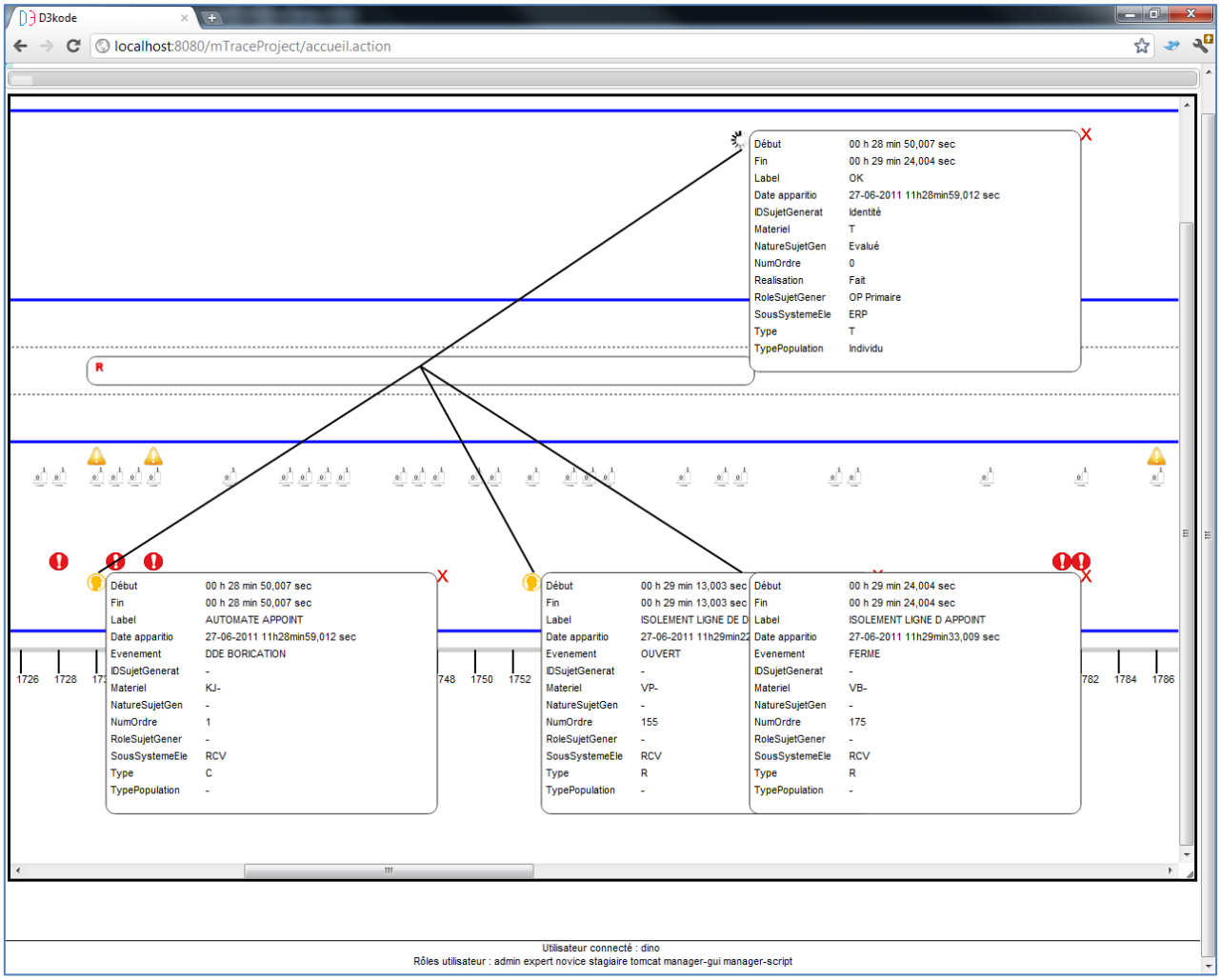
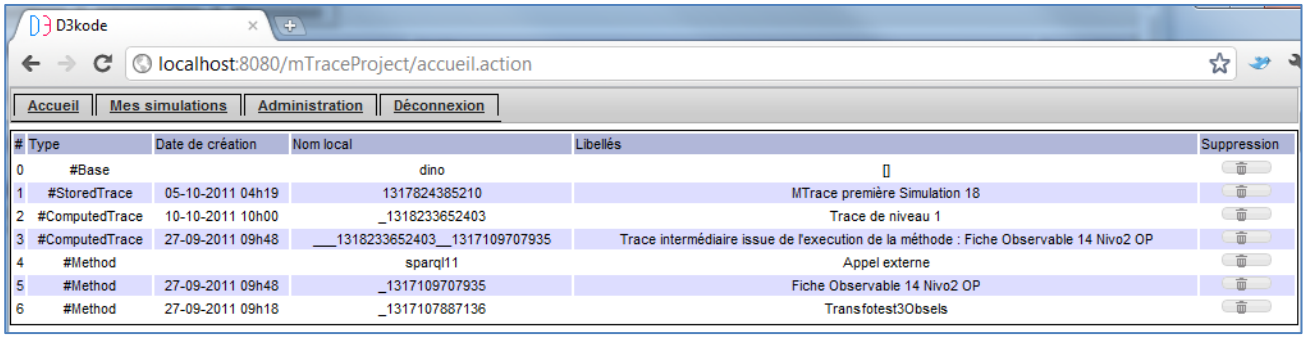


Figure 94 : Visualisation des attributs des observés

Lors du clique sur l'icône représentant un observé, une fenêtre comportant l'ensemble des attributs de l'observé est affichée. On peut sur cet exemple vérifier que la règle précédemment écrite est respectée. Une croix rouge présente en haut à droite de cette fenêtre permet de la cacher. Un mécanisme similaire est présent sur les règles au sein de la transformation ; lorsque l'on clique sur la règle une fenêtre apparait avec la description de la règle.

9. Administration

L'onglet « Administration » permet d'avoir un aperçu des ressources présentes au sein de D3KODE : les bases, les traces, les transformations et les règles. Il est alors possible de les supprimer. Ces fonctionnalités ne sont disponibles que pour les utilisateurs possédant le droit « admin ».



#	Type	Date de création	Nom local	Libellés	Suppression
0	#Base		dino	[]	
1	#StoredTrace	05-10-2011 04h19	1317824385210	MTrace première Simulation 18	
2	#ComputedTrace	10-10-2011 10h00	_1318233652403	Trace de niveau 1	
3	#ComputedTrace	27-09-2011 09h48	___1318233652403__1317109707935	Trace intermédiaire issue de l'execution de la méthode : Fiche Observable 14 Nivo2 OP	
4	#Method		sparql11	Appel externe	
5	#Method	27-09-2011 09h48	_1317109707935	Fiche Observable 14 Nivo2 OP	
6	#Method	27-09-2011 09h18	_1317107887136	Transfotest3Obsels	

Figure 95 : Interface d'administration

Annexe 2
Guide kTBS API ktbs4j Sparql 1.1_4j

1. Un SGBT : le kTBS

1.1. Présentation

Le **k**ernel for **T**race-**B**ases Systems (noyau pour les systèmes de bases de traces) est en cours de développement au sein de l'équipe SILEX dans le laboratoire LIRIS. Les principales personnes chargées du développement et de la maintenance sont Pierre-Antoine CHAMPIN et Françoise CONIL.

Ce système implémente la théorie de la trace modélisée, il apporte un système de gestion de bases de traces utilisant le langage de base du Web sémantique le RDF (Resource Description Framwork). *Le RDF est un modèle de graphe destiné à décrire de façon formelle les ressources Web et leurs métadonnées, de façon à permettre le traitement automatique de telles descriptions (Wikipedia).*

Les ressources présentes dans le kTBS sont au nombre de 9 :

1. kTBS Root : cette ressource est la base de chaque instance de kTBS, son URI[49] est donc l'URI de base de toutes les autres ressources.
2. Base : ressource permettant de regrouper un ensemble de trace ainsi que les ressources liées aux traces
3. StoredTrace : trace créée à l'extérieur du kTBS et importée dans celui-ci
4. ComputedTrace : trace calculée à partir d'une trace source et d'une requête de transformation (method)
5. Method : requête de transformation permettant de transformer une trace en une autre
6. TraceModel : modèle permettant de spécifier ce que peut contenir une trace
7. Obsel : ressource représentant un observé daté, typé (ObselType) et contenant des attributs typés (AttributeType)
8. ObselType : type d'observé affectable à des observés. Cette ressource est définie dans un modèle de trace.
9. AttributeType : type d'attribut caractérisant les attributs d'un observé. Cette ressource est définie dans un modèle de trace et est relative aux types d'observés.

Il est à noter que différentes requête de transformations sont déjà implémentées au sein du kTBS et qu'elles peuvent être étendues par d'autres requêtes de transformations. Ces requêtes (built-in methods) sont au nombre de 6 :

1. Filter : requête permettant de filtrer temporellement, les bornes étant passées en paramètres, une trace
2. Fusion : requête permettant de fusionner plusieurs traces, dont les URI sont passées en paramètre

[49] URI : Uniform Resource Identifier : est une chaine de caractère qui identifie une ressource sur un réseau physique ou abstraite et donc la syntaxe respecte la norme d'Internet mise ne place par le World Wide Web.

3. Sparql rule : requête permettant d'exécuter une requête SPARQL passée en paramètre
4. Script/Pyhton : requête permettant d'exécuter du code python
5. Super-method : requête implémentant le concept de super-method (voir ci-dessous)
6. External : requête permettant de appeler un programme externe pour son exécution

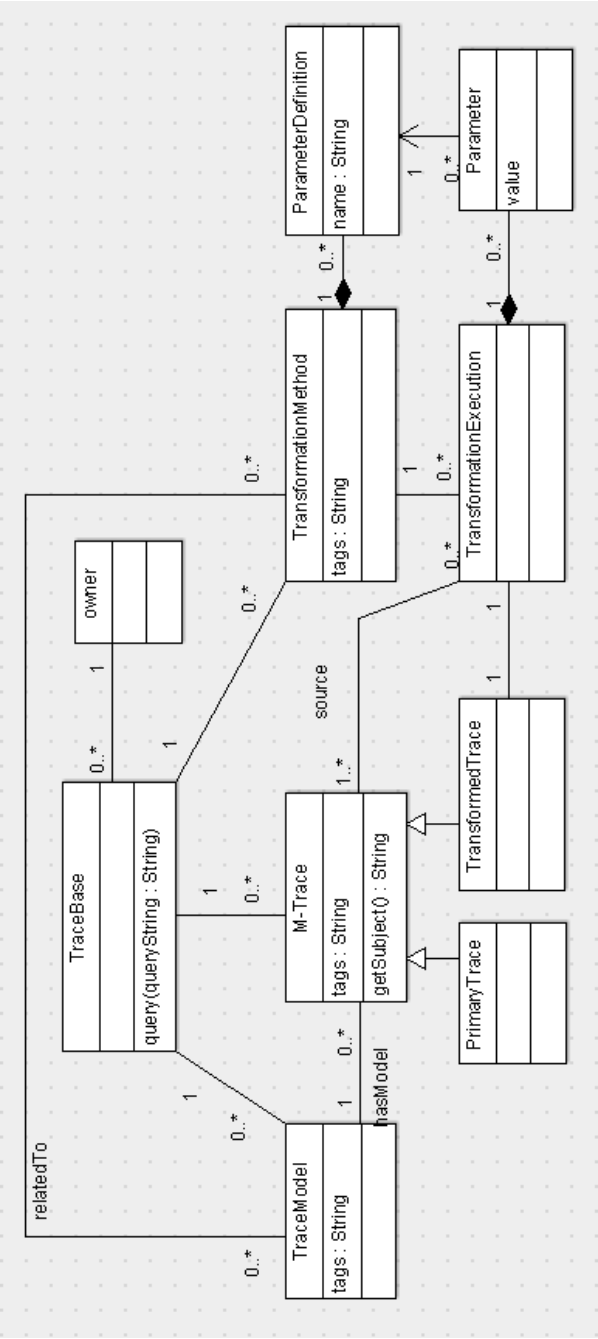


Figure 96 : Diagramme de classe général du kTBS

Le concept de super-méthode initié par le projet D3KODE, est né de l'idée qu'une transformation menant d'une trace d'activité à une autre peut être la résultante d'un ensemble de

règle. Il est alors intéressant de modéliser cet ensemble comme une super-méthode regroupant l'ensemble des exécutions des requêtes de transformations (method) en son sein. La généalogie de création de la trace résultante de la super-méthode est alors possiblement consultable.

1.2. Lancement

Sous windows : installer cygwin qui est un logiciel permettant d'obtenir un environnement Unix sous Windows. Y installer Subversion comme expliqué à cette adresse. Il conviendra de vérifier que les packages, python 2.6, Mingw-gcc-core, Mingw-gcc-g++, gcc-core et gcc-g++ sont installés dans le setup de cygwin. Il faut aussi installer setuptools-0.6c11-py2.6 afin de pouvoir installer la librairie de gestion de RDF et Sparql rdflib 2.4.

Pour installer setuptools, à partir d'une invite de commande Unix : sh setuptools-0.6c11-py2.6.egg

Pour installer rdflib : décompresser l'archive et executer à partir de cette archive ***python setup.py install***

Sous Unix procédez de même que sous Windows sans l'installation de Cygwin et de Mingw.

Les sources sont à récupérer sur le serveur SVN du liris à cette adresse ; par le biais d'un client SVN pour Subversion. Puis exécuter ***svn co https://svn.liris.cnrs.fr/sbt-dev/ktbs-rest-impl/trunk*** à partir du répertoire dans lequel on veut stocker le serveur kTBS.

Puis à partir d'un terminal Unix (cygwin sous Windows), aller dans le répertoire racine du ktsb et exécuter ***./bin/ktbs*** . Si l'on veut lancer le kTBS avec une sauvegarde fichier : ***./bin/ktbs –r <URL répertoire de sauvegarde>.***

1.2.1. Interface web

Suite au lancement du kTBS une interface graphique Web est accessible sur l'URL **http://localhost:8001/**.

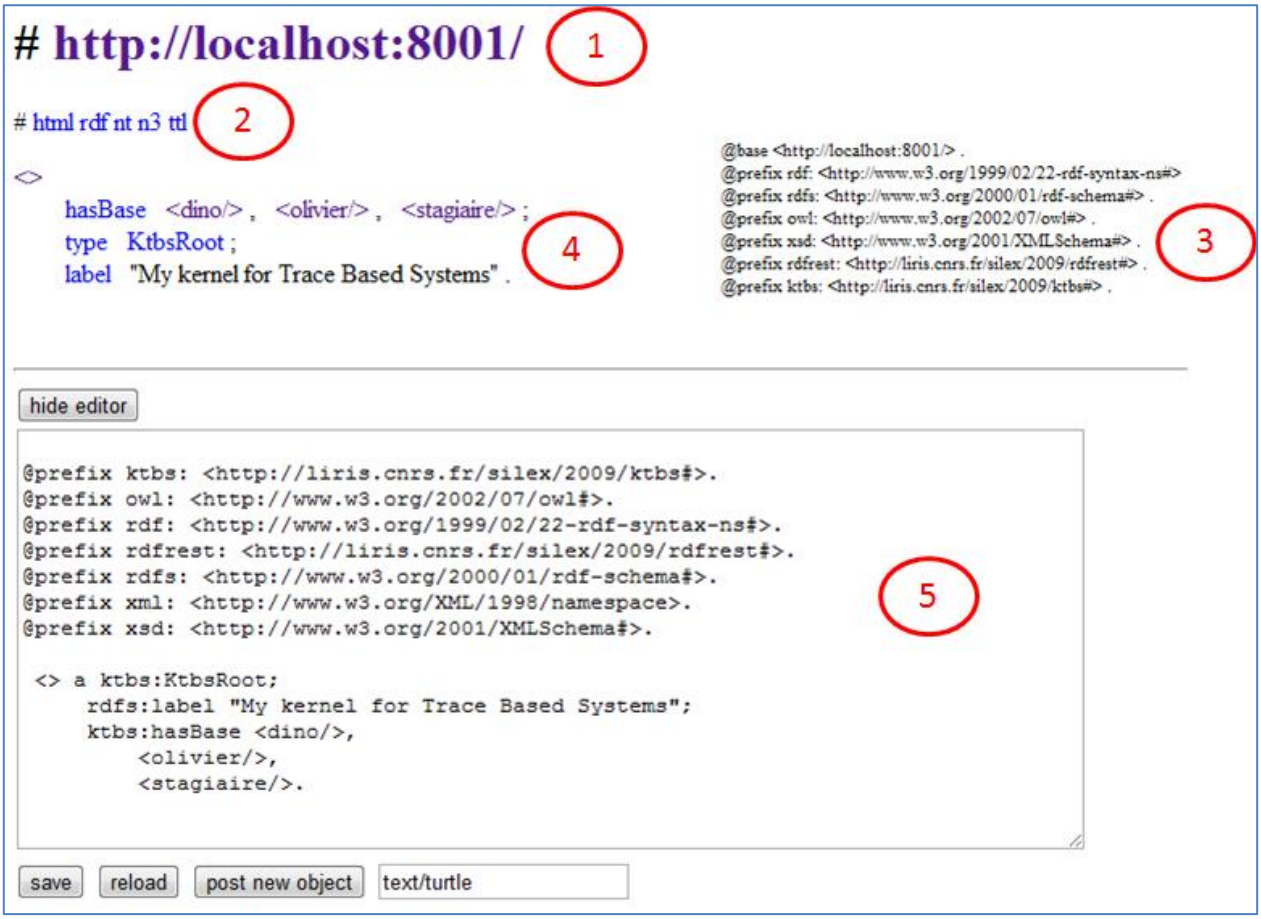


Figure 97 : Interface Web kTBS

1. URL cliquable de la ressource consultée
2. Liste des visualisations possibles pour la ressource
3. Liste des préfixes utilisé au sein du kTBS et donc de la visualisation Web
4. Eléments de la ressource consultée, visualisation du graph RDF lié à la ressource
5. Possibilité de modifier le graphe RDF de la ressource sélectionnée, ou de créer une nouvelle ressource

La figure 97 représente la ressource root de la base kTBS. Elle contient 3 bases « dino », « olivier » et « stagiaire ». Elle est de type « ktbsRoot » et a pour libellé « My kernel for Trace Based Systems ».

1.2.2. Interprétation des logs

Le kTBS respecte l'architecture REST[50] ; (source wikipedia) dans cette architecture, un composant lit ou modifie une ressource en utilisant une représentation de cette ressource. Une ressource est une chose nommable, qui peut évoluer avec le temps. L'application de cette architecture au Web se comprend sur quelques principes :

1. l'URI est important : connaitre l'URI doit suffire pour nommer et identifier une ressource
2. HTTP fournit toutes les opérations nécessaires (GET, POST, PUT, DELETE)
3. Chaque opération est auto-suffisante : le serveur ne mémorise pas l'état du client
4. Utilisation des standards hypermedia : HTML ou XML par exemple

Cette architecture explique la forme que prennent les logs liés à l'utilisation du kTBS. En effet lors que l'on cherche à avoir accès à une ressource une requête GET va être envoyée avec l'URI de la ressource désirée. En retour le kTBS renverra un statut (par exemple : 200 pour une requête correctement exécutée), ainsi que le poids en octet de la réponse.

Par exemple la ligne de log : « localhost - - [14/Oct/2011 08:30:39] "GET /dino/ HTTP/1.1" 200 18705 » Peut etre découpée ainsi :

- localhost : nom ou ip du serveur qui a lancé le kTBS
- [14/Oct/2011 08:30:39] : Date où la requête a fini de s'exécuter
- "GET /dino/ HTTP/1.1" : type de requête, URI de la ressource demandée et protocole utilisé
- 200 : numéro de statut http
- 18705 : poids en octet de la ressource demandée.

A l'instar de GET les autres opérations auront des logs similaires.

[50] Representational State Transfer : style architectural original du Web

2. API Java pour piloter le kTBS

2.1. Présentation

Afin de pouvoir manipuler les ressources présentes dans le ktBS, une API[51] a été développée par Damien CRAM en 2011 au laboratoire LIRIS dans l'équipe de recherche SILEX. Elle a pour but de fournir les interfaces et classes JAVA permettant de manipuler l'architecture REST du kTBS et donc de permettre l'ajout, la modification, la lecture et la suppression des ressources RDF stockées au sein du kTBS.

L'API gère la sérialisation de graphe RDF provenant du kTBS en objet JAVA typé. Elle gère également un niveau de cache http et applicatif. Cette gestion de cache permet d'économiser du flux réseau et du temps de sérialisation.

[51] Application programming interface : est une interface permettant l'interaction entres programmes.

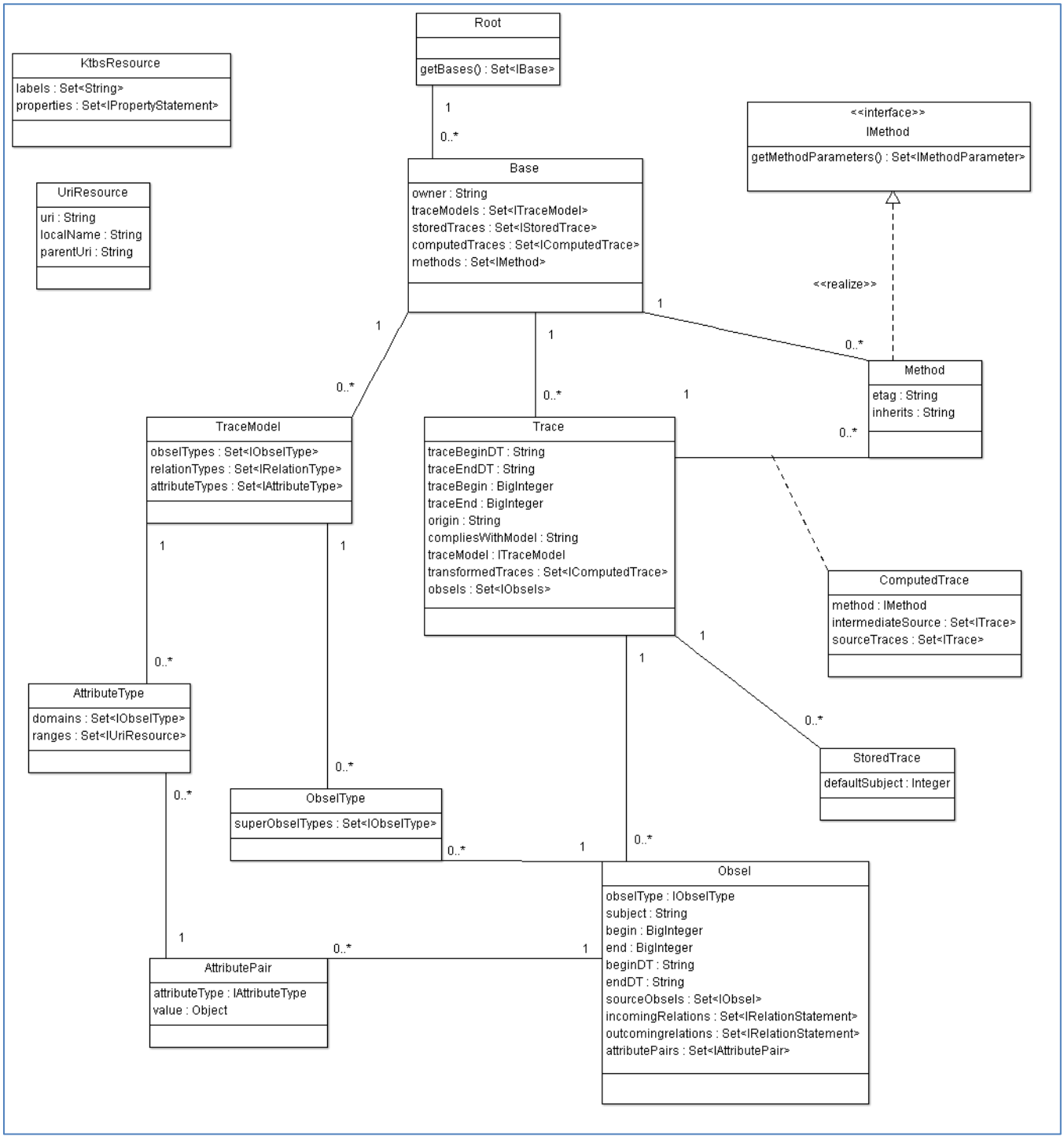


Figure 98 : Diagramme de classe de l'API ktbs4j

2.2. Exemple d'utilisation

```
// Get a KTBS client that uses REST to communicate
// with a KTBS
// the ktbs root uri is given in ktbs.root.uri
KtbsClient client = Ktbs.getRestClient();

// Display the root uri
System.out.println(client.getRootUri());

// retrieve the base1
ResourceService resourceService = client.getResourceService();
IBase base1 = resourceService.getBase("base1");
// The absolute URI would have worked as well
// resourceService.getBase("http://localhost:8001/base1/")

// display the stored traces contained in that base
for(IStoredTrace storedTrace:base1.getStoredTraces()) {
  System.out.println("------------------");
  System.out.println("Name: " + storedTrace.getLocalName());
  System.out.println("Label: " + storedTrace.getLabel());
  System.out.println("Origin: " + storedTrace.getOrigin());
  System.out.println("Default subject: " +
storedTrace.getDefaultSubject());
}

// iterate over all resources contained the base
for(IKtbsResource resource:base1) {
  System.out.println("------------------");
  System.out.println("Name: " + resource.getLocalName());
  System.out.println("Type: " + resource.getTypeUri());
  if (resource instanceof ITrace) {
       ITrace trace = (ITrace) resource;
       System.out.println(trace.getUri() + " is a trace.");
       System.out.println("Nb obsels: " + trace.getObsels().size());
  }
}

// creates a new base
// the following creates a base remotely
IBase base2 =
resourceService.getBase(resourceService.newBase("base2"));

base2.addLabel("Base créée par l'API Java 2.0");

// save the modification
resourceService.saveResource(base2, false);
```

Figure 99 : Exemple d'utilisation de l'API ktbs4j

3. Application d'exécution de SPARQL au sein du kTBS : Sparql1.1_4j

3.1. Le SPARQL

SPARQL est un langage de requêtes et un protocole pour l'accès RDF, conçu par le groupe de travail du W3C RDF Data Access.

En tant que tel, SPARQL est orienté données, en ce sens il n'effectue des recherches que sur des informations contenues dans des modèles ; il n'y a pas d'inférence dans le langage de requête lui-même. SPARQL ne fait rien d'autre que prendre la description de ce que l'application veut sous la forme d'une requête et retourne cette information sous la forme d'un ensemble de données liées ou d'un graphe RDF.

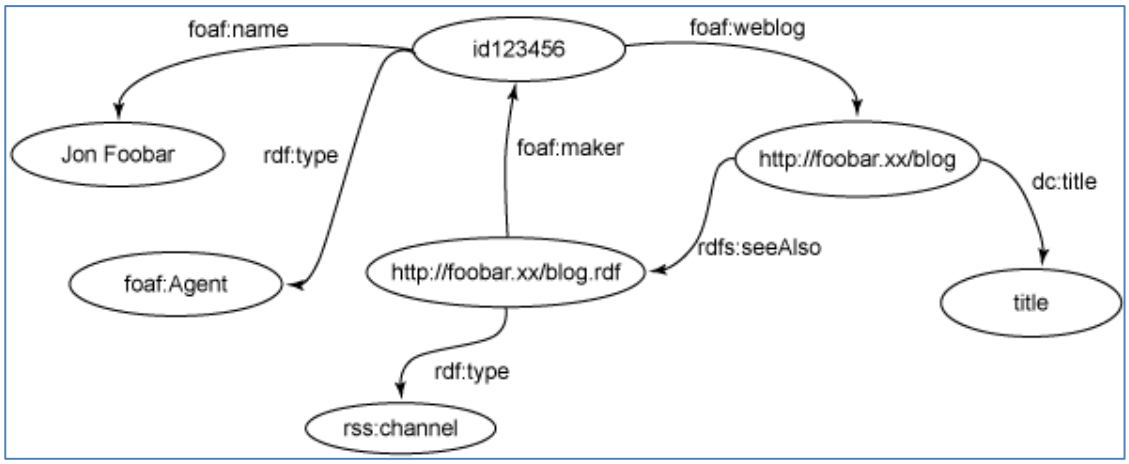


Figure 100 : Représentation d'un graph RDF (source)

Est représenté sur la figure 100, un graph simple pour un contributeur d'un blog stocké dans le fichier RDF bloggers.rdf. Si l'on veut interroger ce graph de manière à trouver « l'URL de blog de la personne qui s'appelle Jon Foobar » voici comment écrire cela en SPARQL :

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?url
FROM    <bloggers.rdf>
WHERE  {
    ?contributor foaf:name "Jon Foobar" .
    ?contributor foaf:weblog ?url .
}
```

Figure 101 : Exemple requête SPARQL (source)

Le kTBS incorpore la possibilité d'exécuter des requêtes SPARQL par le biais de la librairie rdflib2.4. Néanmoins cette version ne prend pas en charge toutes les possibilités offertes par la spécification actuelle SPARQL. Néanmoins il existe des API java (notamment JENA, ARQ) permettant la manipulation de graphe RDF et l'exécution de requête Sparql.

En utilisant les méthodes externes développées au sein du kTBS il est alors possible de lancer l'exécution d'un autre programme et de déplacer la responsabilité de l'exécution d'une requête Sparql. Ce programme pourra être utilisé en ligne de commande, acceptera des paramètres définis en entré et enverra au kTBS le graph RDF correspondant à l'exécution Sparql.

3.2. Principe de fonctionnement

```
<sparql11>
    hasParameter  """command-line=/cygdrive/c/Program\ Files\ \(x86\)/Java/jre6/bin/java -jar Sparql1.1_4j.jar "%
    (__sources__)s" "%(__destination__)s""" ,  "format=xml" ;
    inherits  external ;
    type  Method ;
    comment  "MÃ©thode temporaire visant Ã  palier l' exÃ©cution du SPARQL du kTBS."^^string ;
    label  "Appel externe"^^string .
```

Figure 102 : Méthode externe paramétrée

Lorsque le kTBS doit exécuter une méthode Sparql qui hérite d'une méthode de type externe, il enclenche l'exécution de la ligne de commande déclarée en paramètre de la ressource « method ». Est alors envoyé à l'application Sparql1.1_4j les URIs de(s) trace(s) source(s) ainsi que celui de la trace cible.

```
<_1318854083814>
    hasParameter  "model=http://localhost:8001/dino/1317107556846/" ,  """sparql= PREFIX modelDest: PREFIX fn:
    PREFIX modelSrc: PREFIX ktbs: PREFIX rdfs: PREFIX d3kode: PREFIX afn: CONSTRUCT { [ktbs:hasSourceObsel ?_1317046476410_0, ?
    _1317046476410_1, ?_1317046476410_2;ktbs:hasTrace <%(__destination__)s> ;a modelDest:_1317107637353; ktbs:hasBegin ?Mi0; ktbs:hasEnd ?Ma0;
    rdfs:label "OK"; modelDest:_1317107647567 "Fait"; modelDest:_1317107642852 "ERP"; modelDest:_1317107645195 "T"; modelDest:_1317107639359
    "Individu"; modelDest:_1317107646449 "R"; modelDest:_1317107638496 "IdentitÃ©"; modelDest:_1317107640664 "OP Primaire"; modelDest:_1317107641907
    "EvaluÃ©"; modelDest:_1317107644042 9;] } WHERE { ?_1317046476410_0 a modelSrc:_1317046476410;modelSrc:_1317046476490 ?
    _13170464764900;modelSrc:_1317046476575 ?_13170464765750;modelSrc:_1317046476667 ?_13170464766670;modelSrc:_1317046476795 ?
    _13170464767950;modelSrc:_1317046476905 ?_13170464769050;modelSrc:_1317046477027 ?_13170464770270;modelSrc:_1317046477147 ?
    _13170464771470;modelSrc:_1317046477279 ?_13170464772790;modelSrc:_1317046477419 ?_13170464774190;ktbs:hasBegin ?hasBegin0;ktbs:hasEnd ?
    hasEnd0;rdfs:label ?label0.?_1317046476410_1 a modelSrc:_1317046476410;modelSrc:_1317046476490 ?_13170464764901;modelSrc:_1317046476575 ?
    _13170464765751;modelSrc:_1317046476667 ?_13170464766671;modelSrc:_1317046476795 ?_13170464767951;modelSrc:_1317046476905 ?
    _13170464769051;modelSrc:_1317046477027 ?_13170464770271;modelSrc:_1317046477147 ?_13170464771471;modelSrc:_1317046477279 ?
    _13170464772791;modelSrc:_1317046477419 ?_13170464774191;ktbs:hasBegin ?hasBegin1;ktbs:hasEnd ?hasEnd1;rdfs:label ?label1.?_1317046476410_2 a
    modelSrc:_1317046476410;modelSrc:_1317046476490 ?_13170464764902;modelSrc:_1317046476575 ?_13170464765752;modelSrc:_1317046476667 ?
    _13170464766672;modelSrc:_1317046476795 ?_13170464767952;modelSrc:_1317046476905 ?_13170464769052;modelSrc:_1317046477027 ?
    _13170464770272;modelSrc:_1317046477147 ?_13170464771472;modelSrc:_1317046477279 ?_13170464772792;modelSrc:_1317046477419 ?
    _13170464774192;ktbs:hasBegin ?hasBegin2;ktbs:hasEnd ?hasEnd2;rdfs:label ?label2. FILTER (regex(?_13170464769050,"RCV") ) FILTER (regex(?
    _13170464771470,"KJ-") ) FILTER (regex(?label0,"AUTOMATE APPOINT") ) FILTER (regex(?_13170464772790,"C") ) FILTER (regex(?
    _13170464769051,"RCV") ) FILTER (regex(?_13170464771471,"VP-") ) FILTER (regex(?label1,"ISOLEMENT LIGNE DE DILUTION") ) FILTER (regex(?
    _13170464772791,"R") ) FILTER (regex(?_13170464774191,"OUVERT") ) FILTER (?_13170464770271 = 155 ) FILTER (regex(?_13170464769052,"RCV")
    ) FILTER (?_13170464770272 = 175 ) FILTER (regex(?label2,"ISOLEMENT LIGNE D APPOINT") ) FILTER (regex(?_13170464772792,"R") ) FILTER
    (regex(?_13170464774192,"FERME") ) FILTER (regex(?_13170464774190,"DDE BORICATION") ) FILTER (regex(?_13170464771472,"VB-") ) FILTER (?
    _13170464770270 = 1 ) LET (?Mi0:=d3kode:Min(?hasBegin0,?hasBegin1,?hasBegin2) ) LET (?Ma0:=d3kode:Max(?hasEnd0,?hasEnd1,?hasEnd2) ) }""" ;
    inherits  <sparql11> ;
    type  Method ;
    comment  "Observation de trois observÃ©s avec de multiples critÃ¨res"^^string ;
    label  "Fiche Observable 14 Nivo2 OP"^^string .
```

Figure 103 : Méthode Sparql héritant d'une méthode externe

Sparql1.1_4j va alors interroger le kTBS afin de récupérer la liste des observés inclus dans la(les) trace(s) source(s) ainsi que la méthode a exécuter. La méthode Sparql va être exécutée dans l'application Sparql1.1_4j.

Le graphe RDF résultant de cette exécution est ensuite retourné au kTBS qui l'ajoute à la « computedTrace » déclarée résultante de l'exécution de la méthode de transformation.

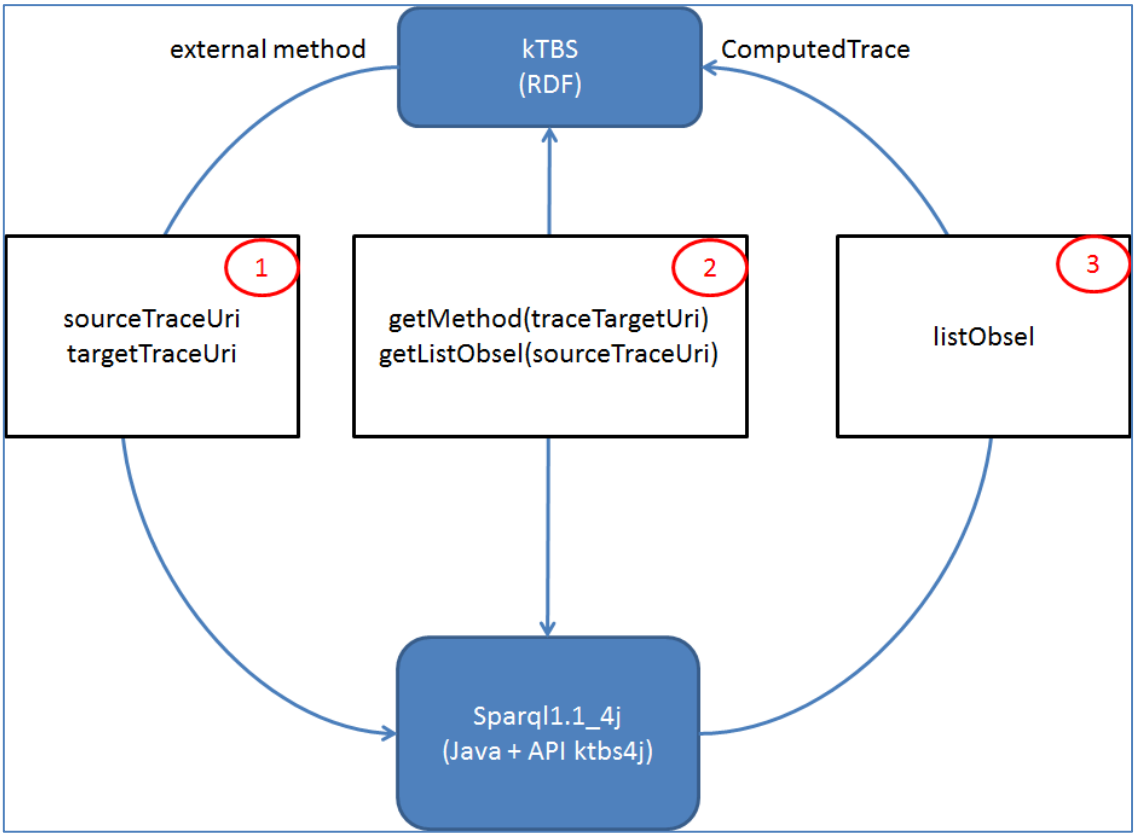


Figure 104 : Schéma de fonctionnement général de Sparql1.1_4j

Liste des figures

Conception d'une plateforme applicative de type « banc d'essais » dédiée à la gestion de données scientifiques orientées Ingénierie des connaissances

Mémoire d'Ingénieur C.N.A.M., Lyon 2012

RESUME

Alliant des questions de recherche à des réponses d'ingénierie, l'élaboration d'une application Web permettant la collecte, le stockage, le questionnement et la représentation d'observation tracée a permis de capitaliser l'expertise d'observation et ainsi d'envisager le partage de connaissance provenant d'expert en direction d'observateur novice.

L'application résultante, a été appliquée dans le contexte de l'observation de l'activité de simulation sur simulateur pleine-échelle au sein d'EDF (*Electricité de France*). Néanmoins, son élaboration a été pensée de manière modulaire et paramétrable permettant de s'adapter à n'importe quel contexte d'activité lié à l'étude d'observation basée sur la théorie de la trace modélisée.

Mots clés : Application Web, observations, construction, figuration, capitalisation de savoir, théorie de la trace modélisée.

SUMMARY

Combining research questions to answer engineering, developing a Web application for collecting, storing, questioning and observation drawn representation has enabled us to capitalize on the expertise of observation and thus to consider the sharing of knowledge between expert and novice.

The resulting application was applied in the context of the observation of full scale simulation at EDF (*Electricité de France*). However, its development has been designed in a modular and configurable to adapt to any context related to the observational study based on the theory of modelled trace.

Keywords: Web, observations, construction, figuration, capitalization of knowledge, approach based on modelled traces.