EPSI - 2010

Rapport projet IA

Conception et mise en œuvre d’un générateur de systèmes experts

Morgan Beau
Nicolas Courazier

Sommaire

Cahier des charges 3

Présentation générale 4

Analyse et modélisation 6

Le moteur d'inférence 7

Sérialisation et formats de sauvegardes 9

Visualisation des résultats 12

Conclusion 14

Cahier des charges

Le générateur de systèmes à base de connaissances que l'on nous demande de réaliser est un générateur de systèmes à base de règles de production écrites dans la logique des propositions. Ce générateur doit intégrer les composantes suivantes :

- Un moteur d'inférences
- Un éditeur de projets
- Un éditeur de règles
- Un éditeur de faits

Autres composantes optionnelles (relations entre les règles, pas à pas, trace du raisonnement, etc.)

Présentation générale

Nous avons nommé notre projet AIR car nous voulions donner une identité à ce projet. Nous avons choisi l'écologie, les énergies renouvelables, etc…

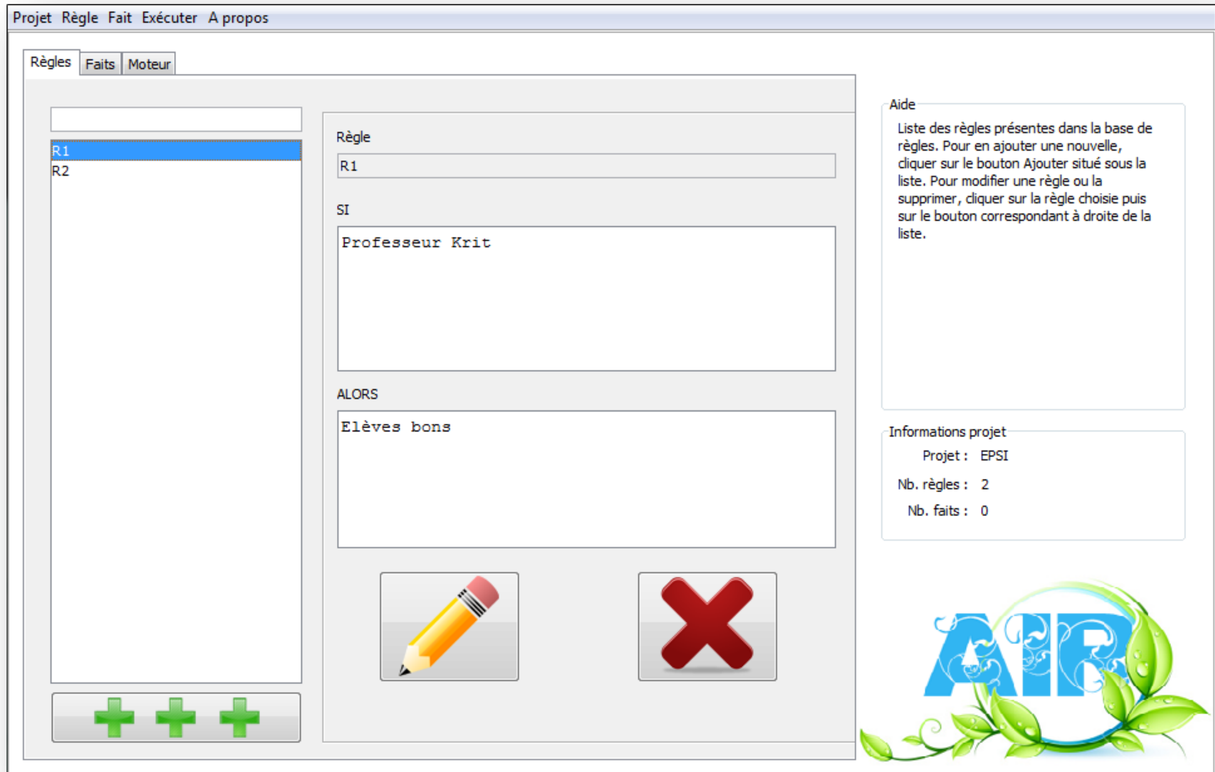
AIR est un générateur de système expert classique.

Il propose les interfaces permettant de :

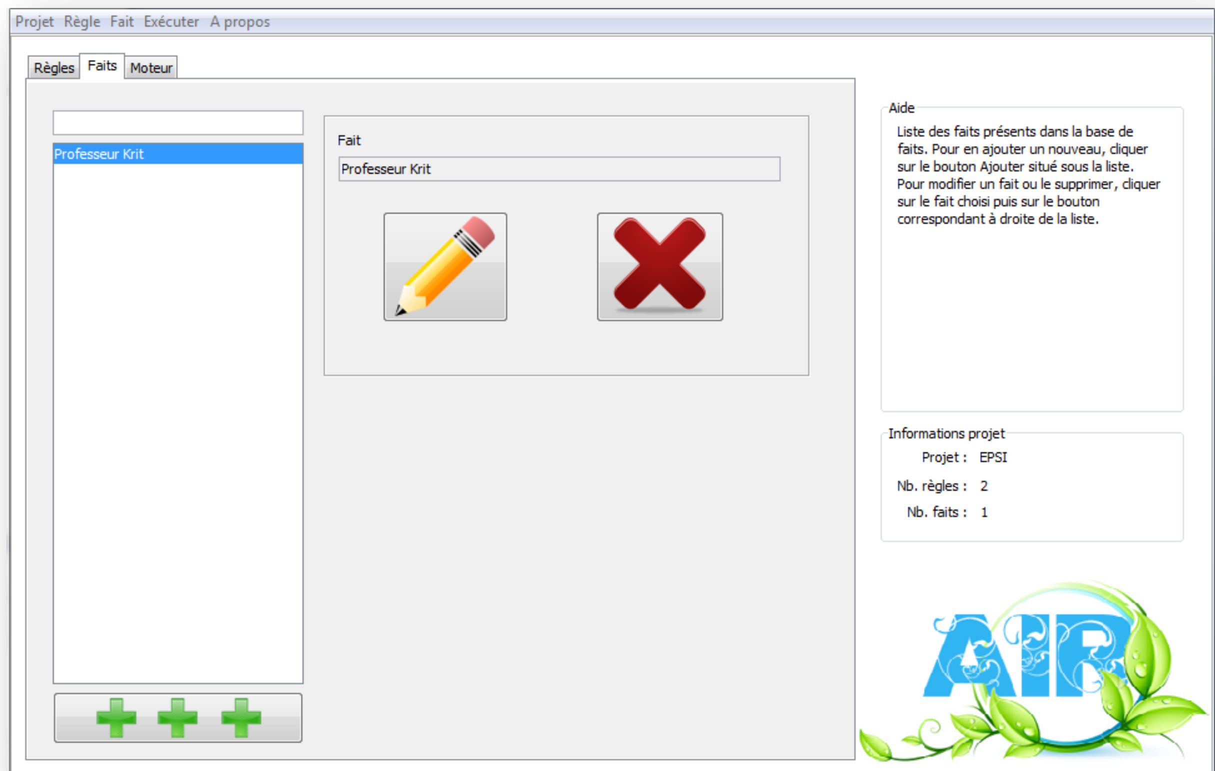
- Gérer un projet : création, sauvegarde, ouverture d'un projet sauvegardé précédemment ;
- Gérer une base de faits : ajouter de nouveaux faits, sauvegarder la base existante, en charger une ancienne, modifier un fait existant, vider la base ;
- Gérer une base de règles : ajouter de nouveaux règles, sauvegarder la base existante, en charger une ancienne, modifier une règle, vider la base
- Exécuter le moteur d'inférence et en afficher le résultat ;

L'utilisateur est guidé via une aide dynamique consultable en permanence sur l'interface.

Voici quelques photos d'écrans de l'interface.



Fenêtre règles



Fenêtre faits

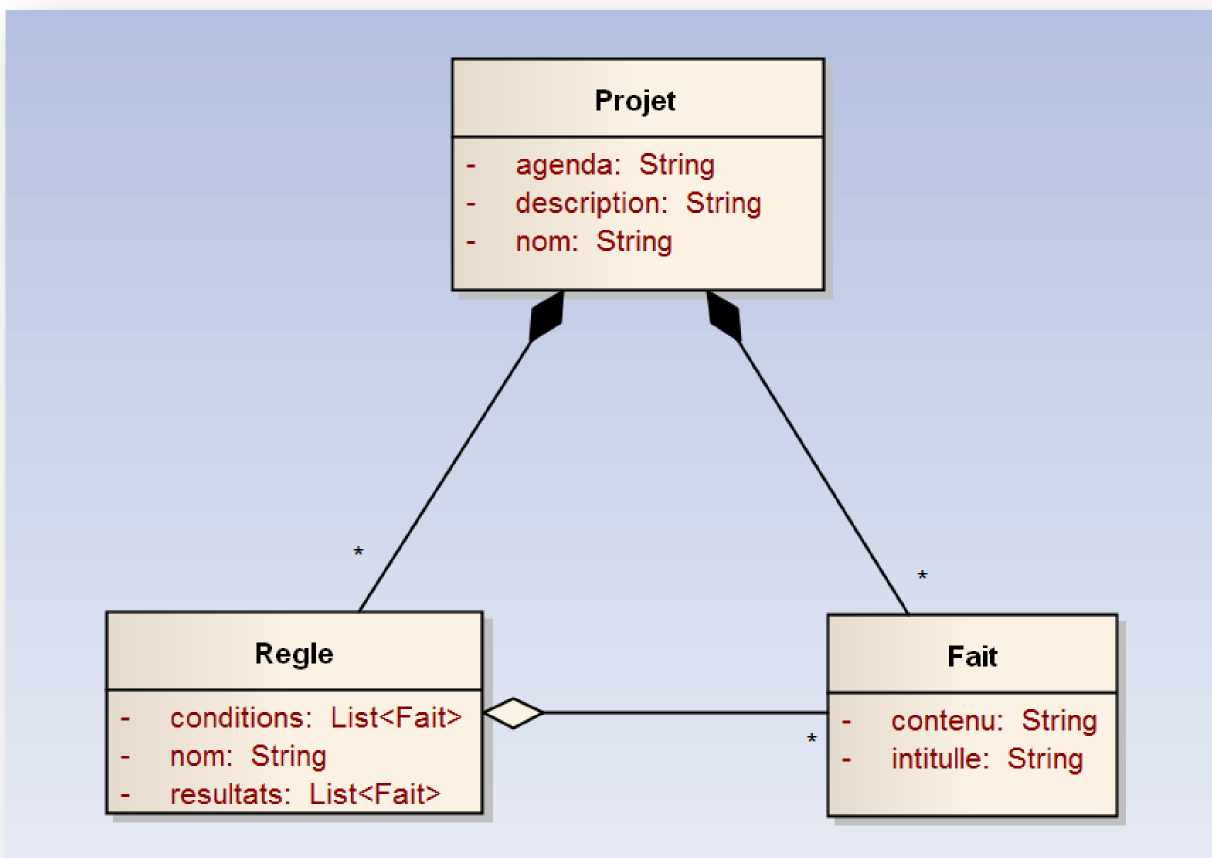
Analyse et modélisation

L'objectif est de gérer des projets. Chaque projet contient une base de règles ainsi qu'une base de fait. Chacune des ces base est représentée par une liste de faits ou de règles dans le projet.

Afin d'être déclenchée, une règle nécessite de remplir certaines conditions. Ces conditions sont la présence de faits dans la base de faits.
Une règle possède donc une liste de faits conditionnels.
De plus, lorsqu'elle est déclenchée, une règle insère de nouveaux faits dans la base. Elle possède donc une liste de faits résultats.

Qu'en au fait, il est constitué d'une intitulé et d'un contenu. L'intitulé seul peut servir à identifier un fais mais un contenu de longueur ou d'un format quelconque est également accepté.

On obtient donc le diagramme de classe simplifié suivant :



En fonctions de nos connaissances des langages de programmation, nous avons choisi Java dans sa version 5 pour sa portabilité et ses nombreuses librairies.

Le moteur d'inférence

Nous avons implémenté un algorithme de chaînage avant d'ordre 0.
Voici son fonctionnement global :

Tant que de nouveaux fait son ajoutés
    Pour toutes les règles applicables
        Si les nouveaux faits ne sont pas déjà présents dans la base de faits
            On applique la règle en insérant les faits déduis
        Fin Si
    Fin pour
Fin tant que

Nous avons optimisé l'algorithme en écartant les règles déjà déclenchées du fonctionnement du moteur de la manière suivant.

Règles à traiter = toutes les règles du projet
Tant que de nouveaux fait son ajoutés
    Pour toutes les règles applicables **parmi les règles à traiter**
        Si les nouveaux faits ne sont pas déjà présents dans la base de faits
            On applique la règle en insérant les faits déduis
        Fin Si
        Règles à traiter = règles à traiter – règles appliquées
    Fin pour
Fin tant que

Afin de respecter les itérations, c'est-à-dire ne pas déclencher une règle dont un des faits conditionnels à son déclenchement a été déduit dans la même itération, nous insérons les nouveaux faits à la fin de chaque itération.

Règles à traiter = toutes les règles du projet
Tant que de nouveaux fait son ajoutés
    Pour toutes les règles applicables parmi les règles à traiter
        Si les nouveaux faits ne sont pas déjà présents dans la base de faits
            Nouveaux faits = application des règles applicable
        Fin Si
        Règles à traiter = règles à traiter – règles appliquées
    Fin pour
    Base de faits = base de faits + nouveaux faits
Fin tant que

Afin de générer le graphe de visualisation, nous devons créer dynamiquement la liste des règles déclenchées.

Règles à traiter = toutes les règles du projet
Tant que de nouveaux fait son ajoutés
    Pour toutes les règles applicables parmi les règles à traiter
        Si les nouveaux faits ne sont pas déjà présents dans la base de faits
            Nouveaux faits = application des règles applicable
            Règles déclenchées = Règles déclenchées + règle en cours d'application
        Fin Si
        Règles à traiter = règles à traiter – règles appliquées
    Fin pour
    Base de faits = base de faits + nouveaux faits
Fin tant que

C'est à partir de cette liste de règles stockée au niveau du projet que le graphe est généré.

De la même manière, l'agenda est renseigné tout au long de l'exécution du moteur.

Règles à traiter = toutes les règles du projet
Tant que de nouveaux fait son ajoutés
    Pour toutes les règles applicables parmi les règles à traiter
        Si les nouveaux faits ne sont pas déjà présents dans la base de faits
            Nouveaux faits = application des règles applicable
            Règles déclenchées = Règles déclenchées + règle en cours d'application
        Fin Si
        Règles à traiter = règles à traiter – règles appliquées
    Fin pour
    Base de faits = base de faits + nouveaux faits
    Agenda = agenda + « insertion du fait X par la règle Y »
Fin tant que

Sérialisation et format de sauvegarde

Afin de faciliter l'utilisation d'AIR, nous avons intégré un système de sauvegarde d'un projet, de la base de fait ou de la base de règle.
Le format choisi pour son universalité est le format XML.
Pour cela, nous avons utilisé l'API XStream. Celle-ci est simple d'utilisation et d'une performance et d'une stabilité éprouvée.

Voici son fonctionnement.

Dans un premier temps il est nécessaire d'indiquer à XStream le mapping de la classe à sérialiser. Pour cela, on utilise l'annotation @XStreamAlias en lui passant en paramètre le nom de l'attribut tel qu'il sera stocké dans le fichier XML.

```
@XStreamAlias("projet")
public class Projet {

    @XStreamAlias("nom")
    private String nom;

    @XStreamAlias("description")
    private String description;

    @XStreamAlias("faits")
    private List<Fait> listeFaits;

    @XStreamAlias("declanchement")
    private List<Regle> declanchementRegles;

    @XStreamAlias("agenda")
    private String agenda;

    @XStreamAlias("regles")
    private List<Regle> listeRegles;
```

Mapping de la classe Projet

Ensuite, il faut initialiser XStream. En effet, pour dé-sérialiser un objet, XStream a besoin de connaitre le mapping de sa classe. Pour cela, il est indispensable qu'il ait auparavant sérialisé un objet de cette classe.
En forçant la sérialisation d'un objet Projet, on s'assure que XStream aura connaissance de la structure de cette classe mais aussi des classes Fait et Regle car celles-ci sont contenues sous forme de listes dans la classe Projet.

```
private void initXstream(){
    this.xstream = new XStream(new DomDriver());
    xstream.autodetectAnnotations(true);
    xstream.toXML(new Projet());
}
```

Une fois ces étapes réalisées, on peut sérialiser et dé-sérialiser ces objets.

Exemple de sérialisation :

```
String xml = xstream.toXML(projet);
```

Exemple de dé-sérialisation :

```
String xml = Functions.ouvrirFichier();
projet = (Projet)xstream.fromXML(xml);
```

Pour un projet contenant deux règles et un fait, voici le résultat sous forme de fichier XML :

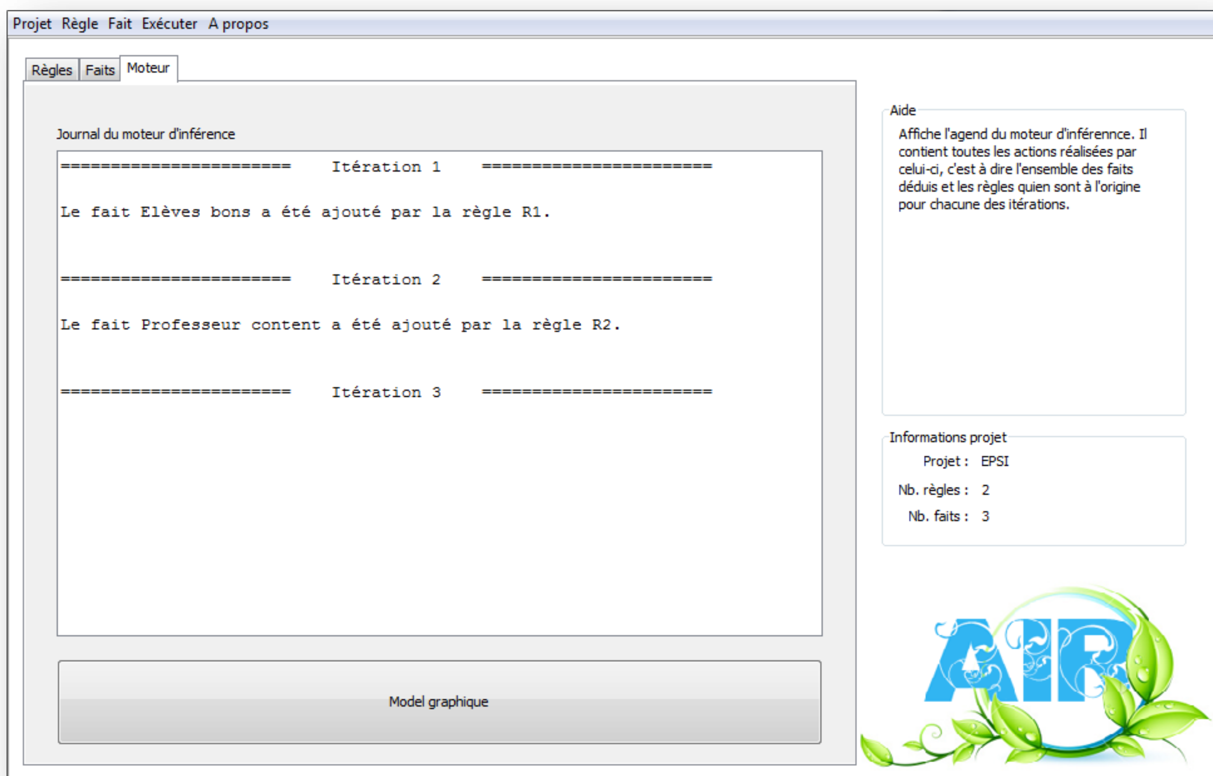
```xml
<projet>
  <nom>Professeur</nom>
  <description>Projet de demonstration.</description>
  <faits>
    <fait>
      <intitule>Professeur</intitule>
      <contenu>Krit</contenu>
    </fait>
  </faits>
  <declanchement/>
  <agenda></agenda>
  <regles>
    <regle>
      <nom>R1</nom>
      <si>
        <fait>
          <intitule>Professeur</intitule>
          <contenu>Krit</contenu>
        </fait>
      </si>
      <alors>
        <fait>
          <intitule>Eleves</intitule>
          <contenu>Bons</contenu>
        </fait>
      </alors>
    </regle>
    <regle>
      <nom>R2</nom>
      <si>
        <fait>
          <intitule>Eleves</intitule>
          <contenu>Bons</contenu>
        </fait>
      </si>
      <alors>
        <fait>
          <intitule>Professeur</intitule>
          <contenu>Content</contenu>
        </fait>
      </alors>
    </regle>
  </regles>
</projet>
```

Visualisations des résultats

Lorsque le moteur est exécuté, AIR génère l'agenda du projet. C'est-à-dire les différentes actions qu'il réalise sous la forme :

Le fait *X* a été ajouté par la règle *Y*.

Il précise également lors de quelle itération cette action a eu lieue.

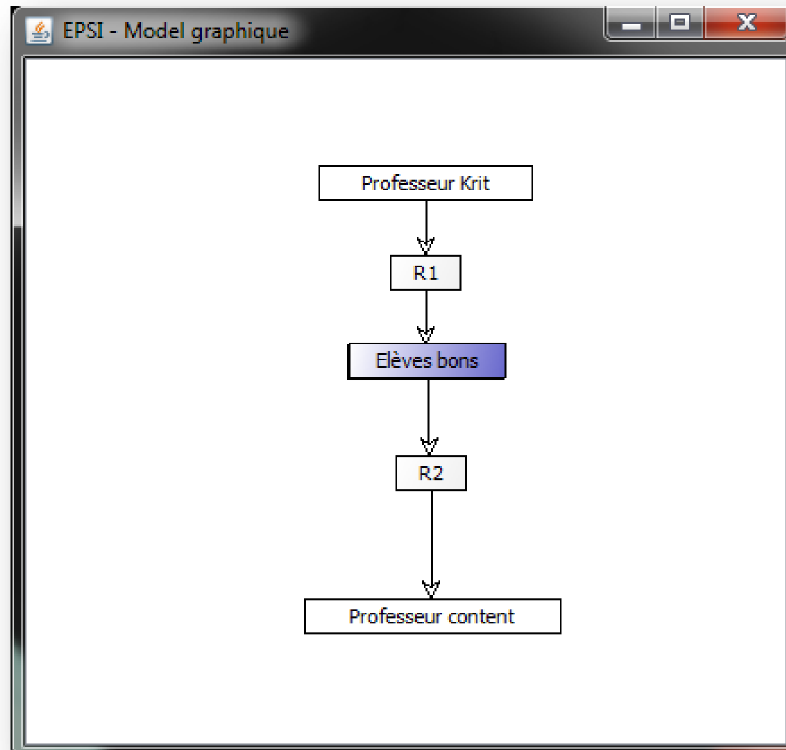


Fenêtre d'affichage de l'agenda

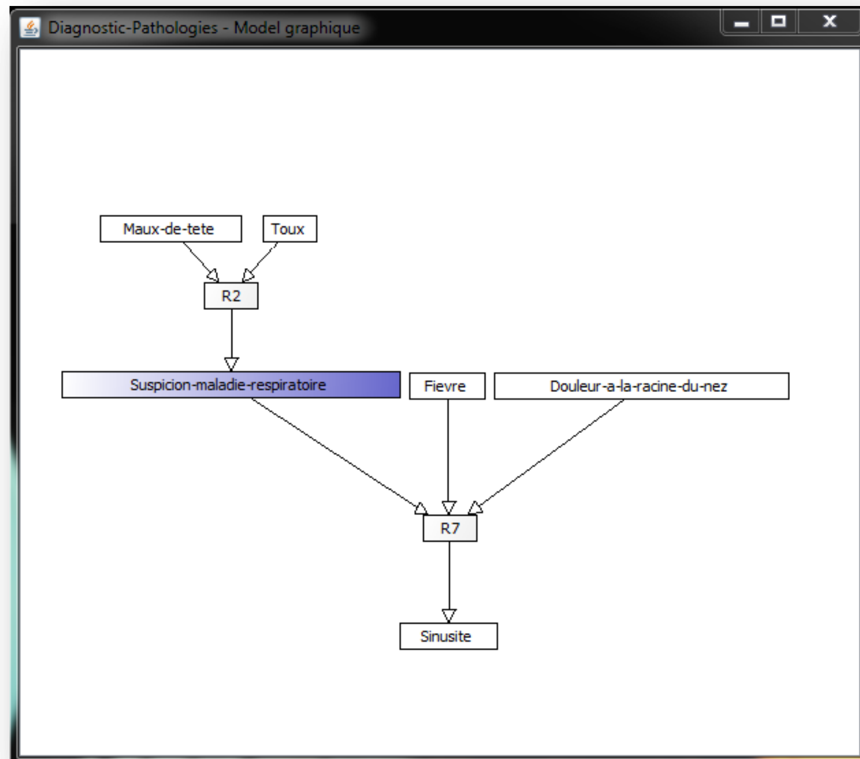
L'agenda ne générant pas un affichage agréable à lire, nous avons donc intégré la création de graphes automatique.

Chaque fait présent à plusieurs reprise dans le graphe, c'est-à-dire étant à l'origine et/ou la conséquence de plusieurs règles, sera coloré d'une couleur afin d'être facilement identifiable.

Nous avons pour cela utilisé la librairie java JGraph.



Affichage graphique d'un résultat simple



Affichage graphique d'un résultat complexe

Conclusion

Ce projet nous a permis d'assimiler beaucoup des problématiques du fonctionnement d'un moteur d'inférence en chaînage avant. Bien qu'il soit d'ordre 0, nous avons tenté d'y apporter quelques optimisations, comme le retrait des règles déjà traitées. Ce fût un vrai challenge algorithmique.

La sérialisation en XML apporte un aspect universel à AIR. Il est envisageable par le suite de charger n'importe quelle projet en uniformisant le format de sauvegarde.

Nous avons également appris à nous servir de librairies telles que JGrpah qui permet un affichage plus agréable et lisible qu'une simple console ou qu'un fichier texte.

Nous avons aussi pris beaucoup de plaisir à réaliser l'interface et la charte graphique grâce à des outils tels que Adobe Photoshop.

Dans l'avenir, nous envisageons d'intégrer un moteur d'ordre 0+ voire d'ordre 1 et d'implémenter le chaînage arrière.