Projet
Systèmes Multi-Agents
Architecture Logicielle

L'objectif de ce projet est de vous mettre en situation. Il s'agit de mettre en œuvre vos connaissances des systèmes multi-agents pour prendre en compte un certain nombre d'exigences, fonctionnelles et non fonctionnelles, et de proposer une architecture logicielle à base de composants pour les résoudre.

Le résultat attendu est une architecture logicielle pour implanter des systèmes multi-agents (et en particulier le cas d'étude ci-dessous). Les livrables demandés sont :

1. Le code source de votre solution sous la forme d'un lien vers une archive.
2. Un document d'architecture détaillé qui décrit précisément comment chaque exigence est prise en compte.

Pour vous aider ou pour approfondir vos connaissances sur le type de SMA que nous vous proposons d'implanter, vous avez à disposition un certain nombre de ressources sur le site Moodle dédié à cet enseignement (http://moodle.univ-tlse3.fr/course/view.php?id=2113). Par ailleurs, un forum a été définir pour poser des questions et chercher de l'aide de la part de vos collègues ou d'enseignants/chercheurs du domaine.

Pour la réalisation de votre solution, vous serez assistés pendant les 10 séances présentielles restantes.

Date limite proposée de dépôt des livrables sur le dépôt Moodle : jeudi 2 juin 2016 à 23h55.

Cas d'étude

Pour rendre concret la conception et le développement qui vous sont demandés, voici un exemple de SMA que vous devrez implanter pour valider vos résultats.

Il s'agit d'un système de résolution de contraintes. Par exemple, il peut s'agir de l'énigme d'Einstein ou d'un problème d'emploi du temps.

Inspiré de [Picard 05] : Les enseignants et les étudiants doivent trouver des partenaires, des créneaux horaires et des salles pour donner ou recevoir des enseignements. Chaque acteur possède des contraintes concernant ses disponibilités ou des équipements nécessaires. De plus, un enseignant peut ajouter ou retirer des contraintes à n'importe quel moment de la résolution via une interface adaptée. Une telle application nécessite clairement de l'adaptation et de la robustesse. Le système doit être capable de s'adapter aux perturbations environnementales (modifications de contraintes) et ne pas calculer de nouvelles solutions, depuis le début, à chaque changement.

Exigences non fonctionnelles

Voici un certain nombre d'exigences.

ENF 1. (5) Pour des facilités de débogage, il doit être possible de contrôler l'exécution du système (mettre en pause, pas à pas , vitesse plus ou moins rapide).

ENF 2. Il doit être possible de changer la politique de scheduling du système.

ENF 3.

ENF 4. (5) De sa création jusqu'à son suicide, un agent répète un cycle d'exécution composé de trois opérations séquentielles Percevoir-Décider-Agir.

ENF 5. (1) Un agent peut créer d'autres agents.

ENF 6. (2) Un agent peut se suicider mais ne peut détruire un autre agent.

ENF 7. (5) Il doit être possible de visualiser l'exécution du système.

ENF 8. (8) Il doit être possible de visualiser l'état du système et/ou d'agents sélectionnés.

ENF 9. (8) Le système de visualisation doit être le plus découplé possible du reste du système.

ENF 10. (3) Le langage de programmation doit être Java.

ENF 11. (8) Le langage de description des composants et des assemblages doit être SpeADL.

ENF 12. (5) Vous devez quantifier le nombre de classes et de lignes de code écrites (et pas générées).

ENF 13. (13) La réutilisabilité/généricité de votre solution doit être évaluable objectivement, mesurée et la plus forte possible.

ENF 14. (5) La paramétrisation de votre solution pour le cas d'étude doit être complètement spécifiée.

ENF 15. (8) Votre architecture doit être décrite le plus précisément possible (vues C&C, module et allocation)

ENF 16. (8) Il doit être possible d'obtenir une trace d'une exécution d'un agent sous forme de log.

ENF 17. (8) Il doit être possible de rejouer une exécution d'un scenario avec un état initial donné, y compris en présence de phénomènes aléatoires.

ENF 18. (8) Il doit être possible de persister l'état du système.

ENF 19. (21) Il doit être possible de répartir l'exécution du système.