SENHAJI Ismail
SERIN Kévin
SIMAR Jérémy

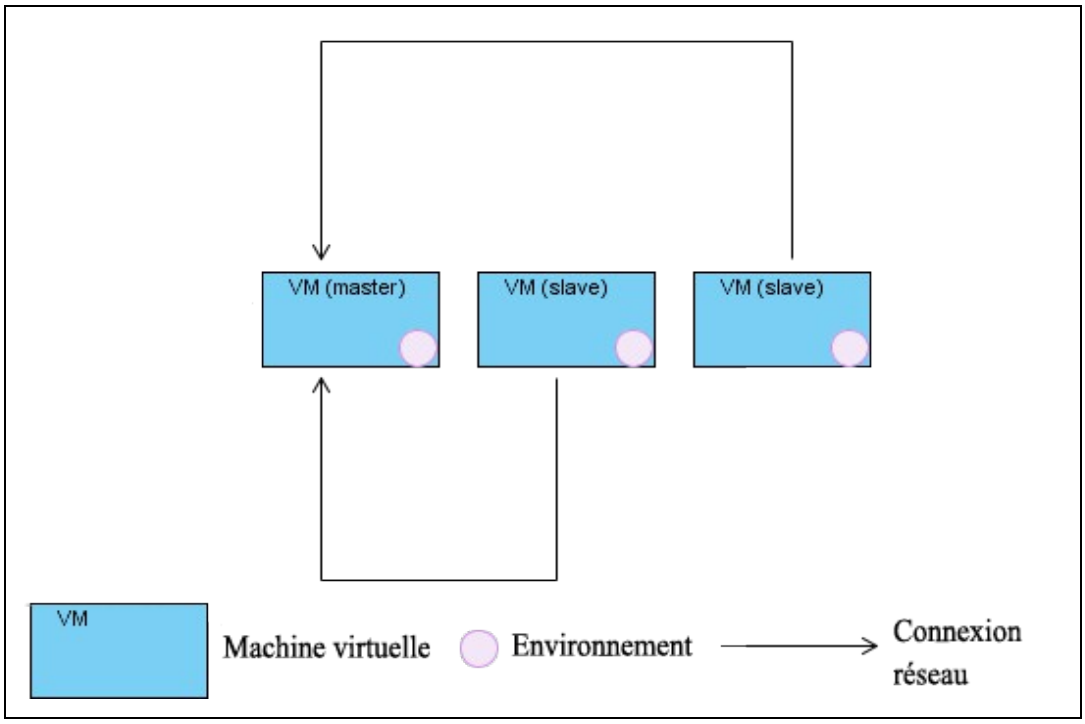
GitHub : https://github.com/exasky/RobotsBoxes

Devoir Maison
Système Multi-Agents
– Rendu –

Sommaire

I. Gestion de la répartition

Vue d'allocation représentant la répartition des machines virtuelles



Exigences :

ENF 12. Votre architecture doit être décrite le plus précisément possible (vues C&C, module et allocation).

ENF 17. Il doit être possible de répartir l'exécution du système.

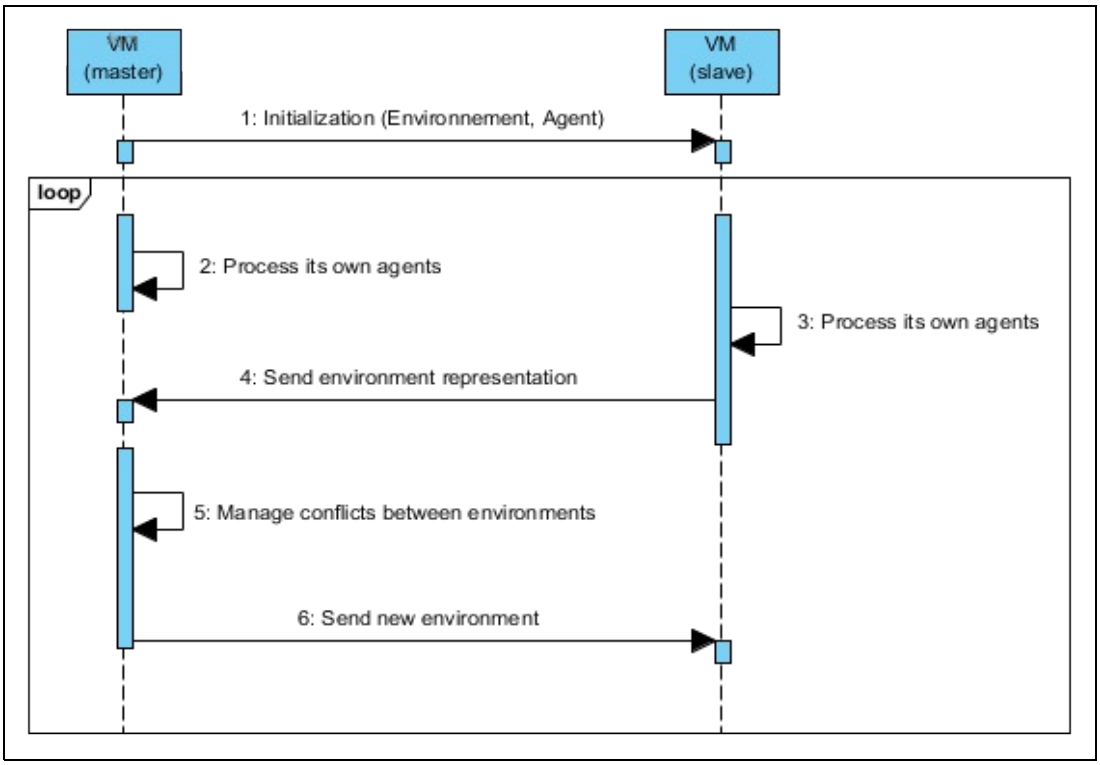
Justifications :

Nous avons pris en compte la notion de répartition dès le début afin de pouvoir concevoir une architecture prenant en compte cette contrainte importante (l'implémentation pourra ne pas être faite par la suite).

Nous avons décidé qu'en cas de répartition, les agents pourraient être répartis sur plusieurs machines virtuelles JAVA différentes. L'une des machines, serait considérée comme la machine maître et serait en charge de synchroniser les ressources partagées (l'environnement principalement).

Chaque machine virtuelle possède une représentation complète de l'environnement mais ne contrôle qu'une partie des agents. A la fin de chaque cycle d'action des agents, les machines esclaves renvoie leur version de l'environnement à la machine maître afin que cette dernière fusionne les différentes modifications pour donner l'environnement du prochaine cycle.

Diagramme de séquence UML de la communication entre la machine maître et les machines esclaves



Exigences :

ENF 12. Votre architecture doit être décrite le plus précisément possible (vues C&C, module et allocation).

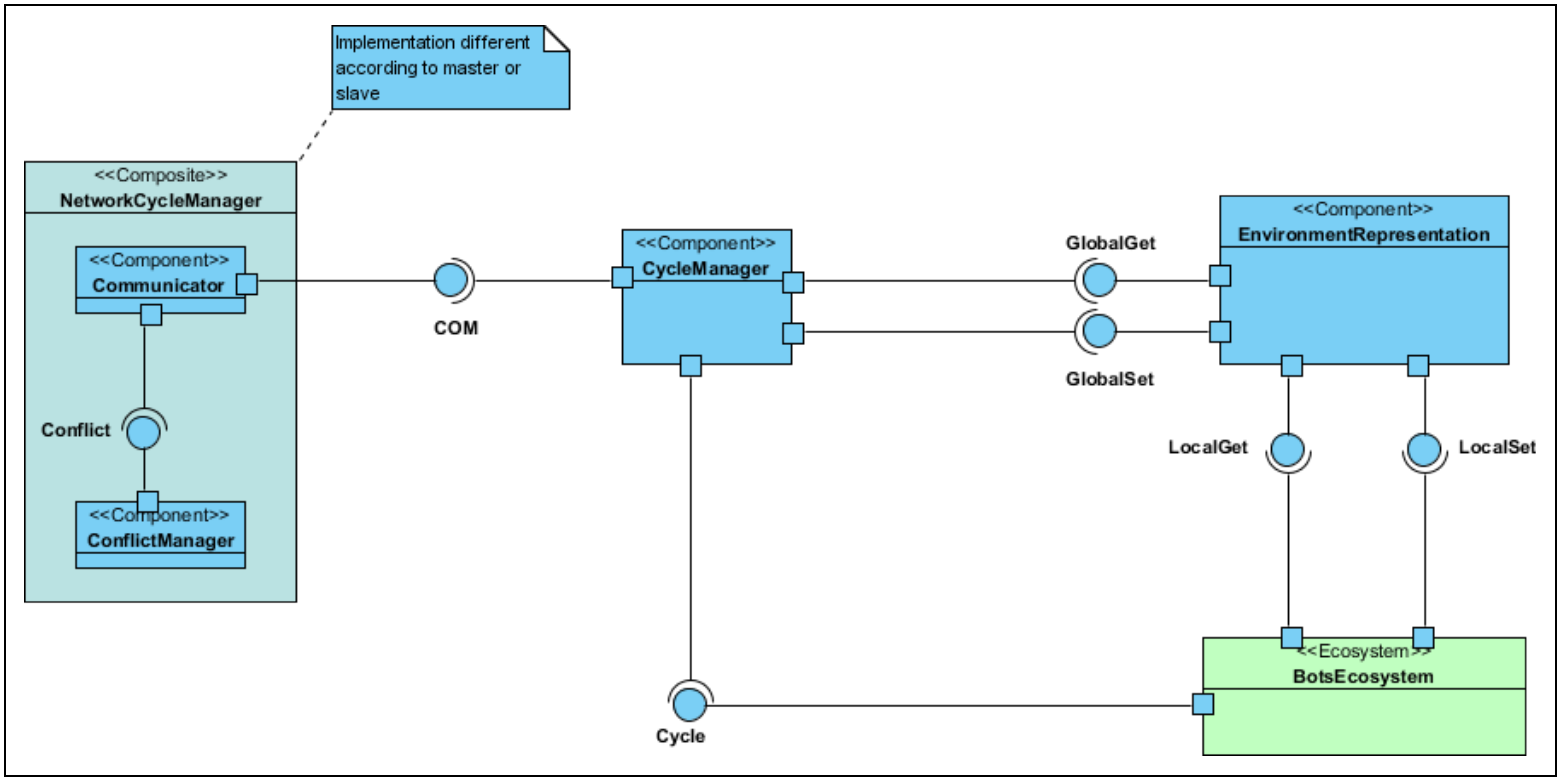
ENF 17. Il doit être possible de répartir l'exécution du système.

Justifications :

Chaque machine travaille avec les agents locaux qu'elle contrôle et effectue un cycle d'action pour tous les agents qu'elle contrôle. A la fin de chaque cycle, il est nécessaire de gérer les conflits pouvant avoir eu lieu sur les différentes machines (puisque l'environnement est partagé). C'est la machine maître qui s'en charge et qui renvoie le nouvel environnement à utiliser à tous les esclaves afin de commencer un nouveau cycle.

Ce système permet de répartir la charge de travail s'il y a de nombreux agents tout en évitant une trop grosse quantité de communications. En effet, on ne communique qu'une seule fois à la fin de chaque cycle, et non pas une fois à chaque action d'un agent. La difficulté résidera dans la gestion de potentiels conflits sur l'environnement. Dans notre cas, la problème paraît suffisamment simple pour envisager une gestion de conflits sans problème.

Vue composants et connecteurs général



Exigences :

ENF 12. Votre architecture doit être décrite le plus précisément possible (vues C&C, module et allocation).

ENF 17. Il doit être possible de répartir l'exécution du système.

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

ENF 10. La ré-utilisabilité/généricité de votre solution doit être évaluable objectivement, mesurée et la plus forte possible.

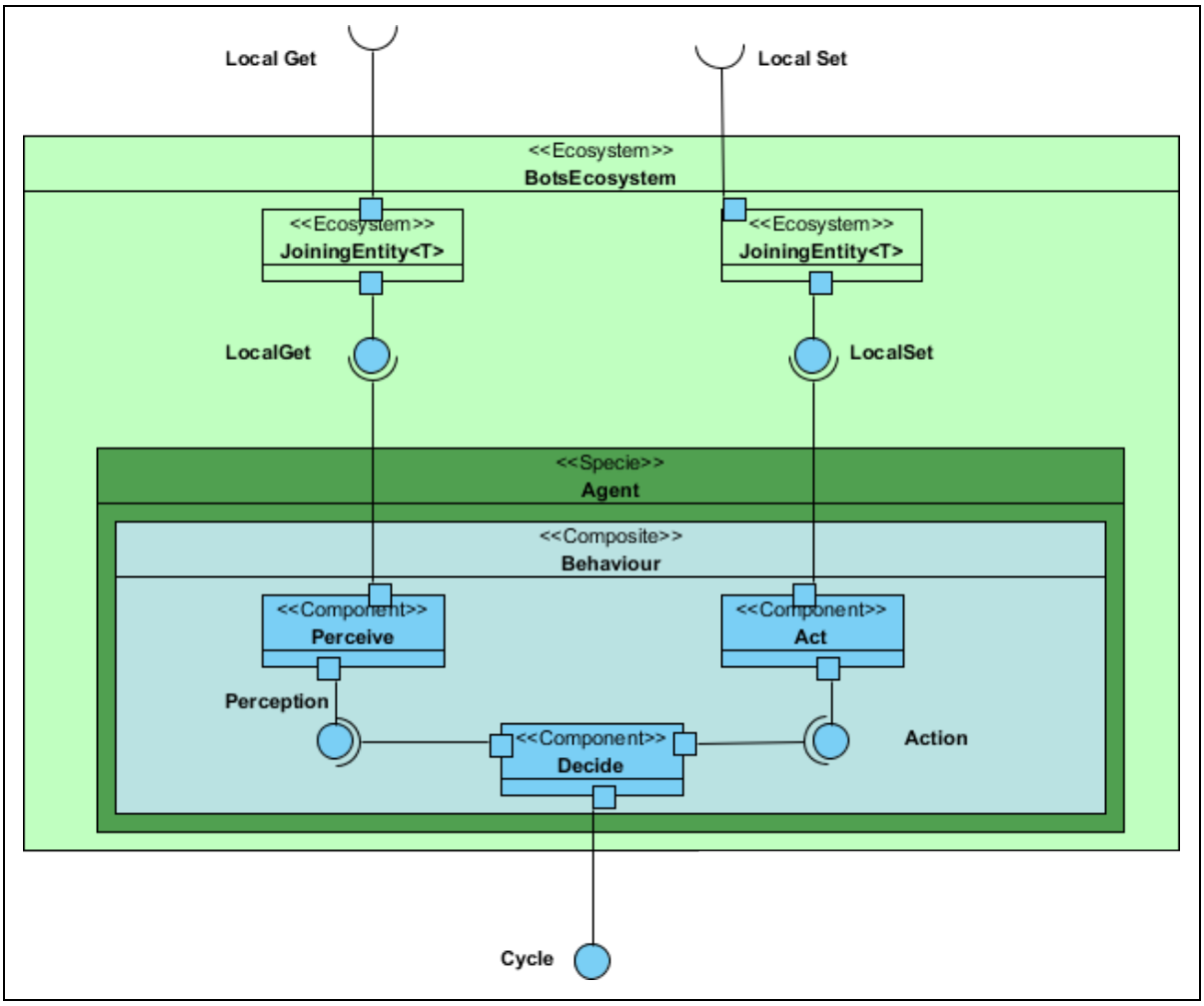
Justifications :

Un composant local (CycleManager) est chargé de gérer le cycle des agents et se charge de retransmettre le résultat de l'environnement sur le réseau si l'application est repartie.

L'environnement fournit quant à lui plusieurs interfaces afin d'y accéder et de le modifier. Des interfaces globales, qui sont ici utilisées pour consulter et mettre à jour tout l'environnement entre chaque cycle. Des interfaces locales sont utilisées pour les agents, puisque ces derniers n'ont qu'une vision local de leur environnement.

II. Comportement des agents

Vue composants et connecteurs des agents



Exigences :

ENF 12. Votre architecture doit être décrite le plus précisément possible (vues C&C, module et allocation).

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

ENF 10. La ré-utilisabilité/généricité de votre solution doit être évaluable objectivement, mesurée et la plus forte possible.

ENF 1. De sa création jusqu'à son suicide, un agent répète un cycle d'exécution composé de trois opérations séquentielles Percevoir-Decider-Agir.

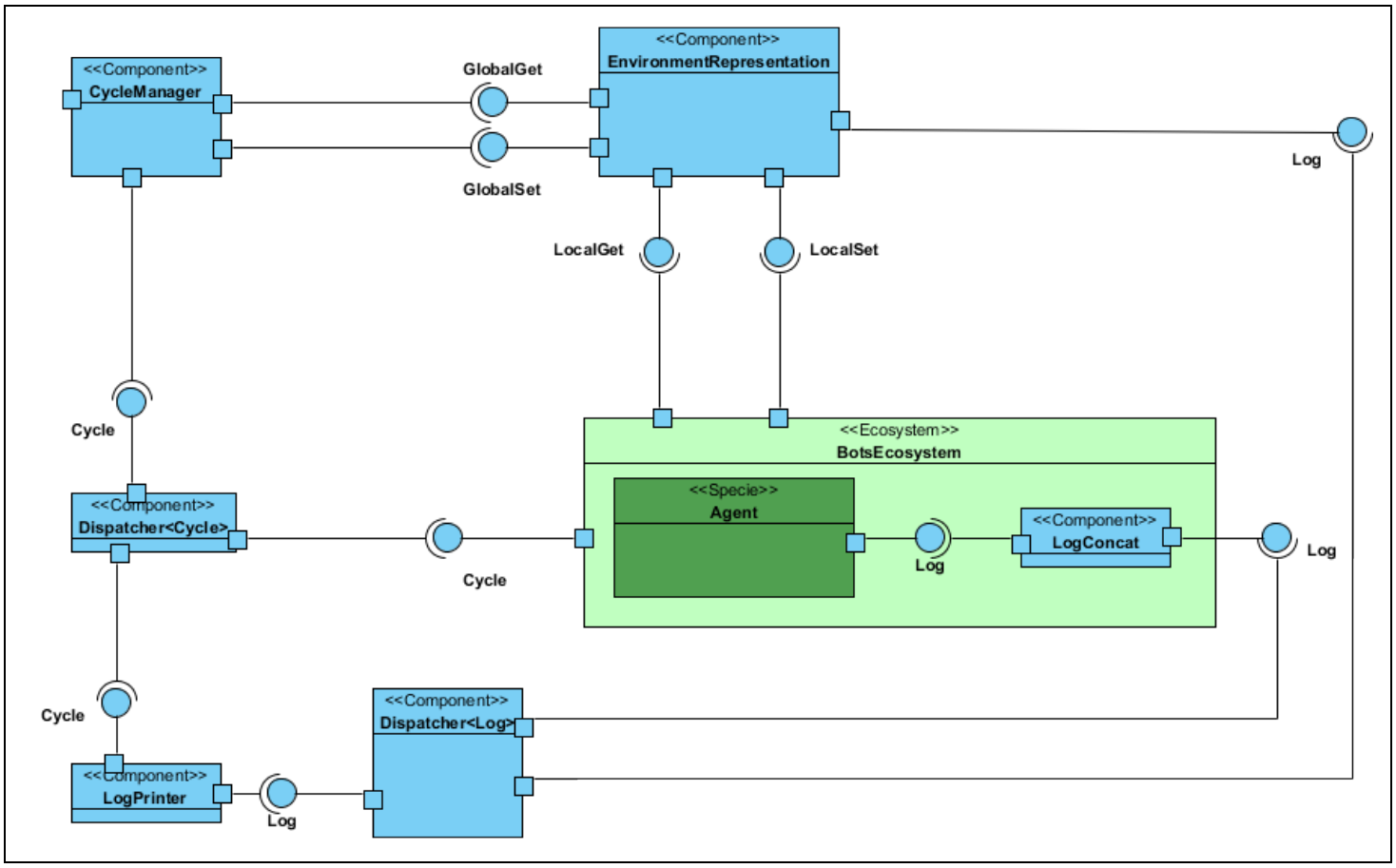
Justifications :

Afin de permettre une meilleure ré-utilisabilité des composants, un agent possède un comportement générique (avec simplement des interfaces d'accès à l'environnement).

Dans notre cas, notre comportement sera découper en 3 parties : percevoir, décider, agir, qui sera un comportement particulier d'agents.

III. Gestion de la trace

Vue composants et connecteurs de la gestion de la trace



Exigences :

ENF 12. Votre architecture doit être décrite le plus précisément possible (vues C&C, module et allocation).

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

ENF 10. La ré-utilisabilité/généricité de votre solution doit être évaluable objectivement, mesurée et la plus forte possible.

ENF 14. Il doit être possible d'obtenir une trace d'une exécution d'un agent sous forme de log.

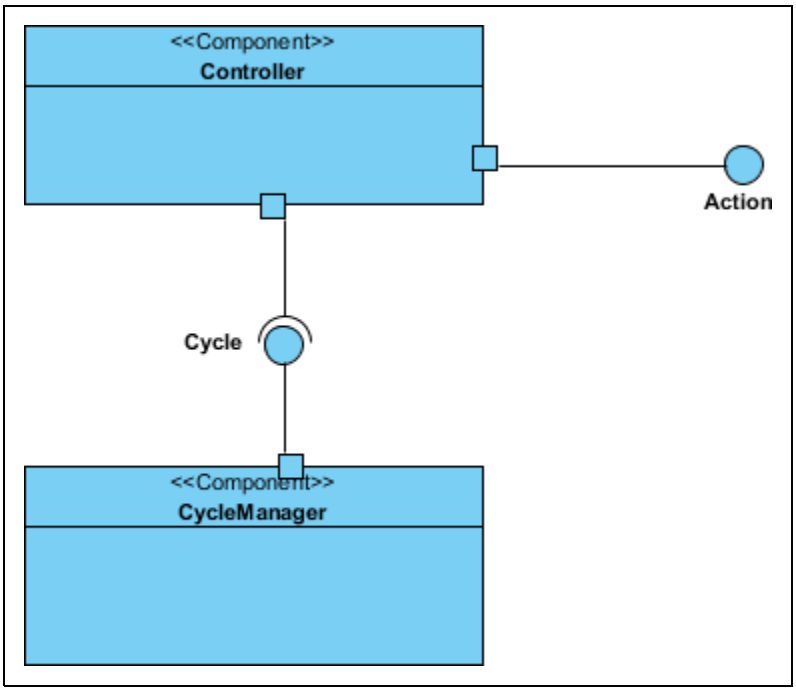
Justifications :

La gestion de la trace ne doit pas être quelque chose d'obligatoire. Il serait donc maladroit que les composants requiert un logger. Il faut donc inverser le contrôle : les composants fournissent un service de trace qui permet d'obtenir l'état du composant, et un logger se connecte aux composants dont il désire l'état. Ceci nécessite la mise en place d'un petit dispatcher afin de se connecter à plusieurs services fournis.

De plus, le composant d'affichage a besoin de savoir à quel moment il doit logger, c'est pour cela qu'il doit lui aussi être connecté au gestionnaire de cyles.

IV. Contrôle et paramétrisation du système

Vue composants et connecteurs de la gestion du contrôle de l'exécution



Exigences :

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

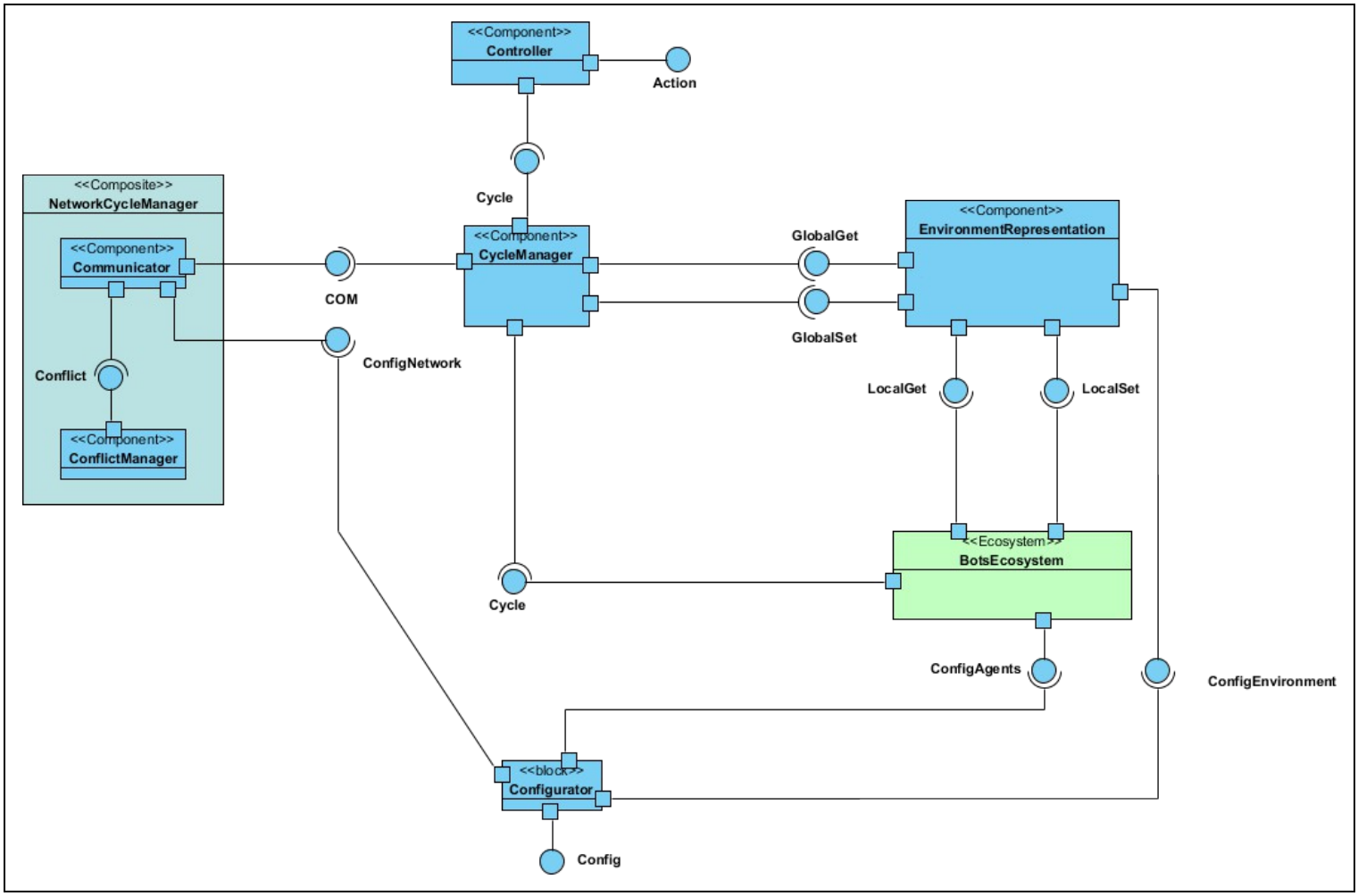
ENF 12. Votre architecture doit être décrite le plus précisément possible (vues C&C, module et allocation).

ENF 13. Il doit être possible de contrôler l'exécution du système (mettre en pause, pas à pas, vitesse plus ou moins rapide).

Justifications :

L'ajout d'un composant de contrôle permet de manière simple de gérer le flot d'exécution du système en appelant quand il le faut les services fournis par l'interface Cycle. L'interfcae Action permet de choir d'effectuer un cycle, ou bien d'effectuer tous les cycles à la suite de manière plus ou moins rapide.

Vue composants et connecteurs de la gestion de la configuration



Exigences :

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

ENF 11. La paramétrisation de votre solution pour le cas d'étude doit être complètement spécifiée.

ENF 12. Votre architecture doit être décrite le plus précisément possible (vues C&C, module et allocation).

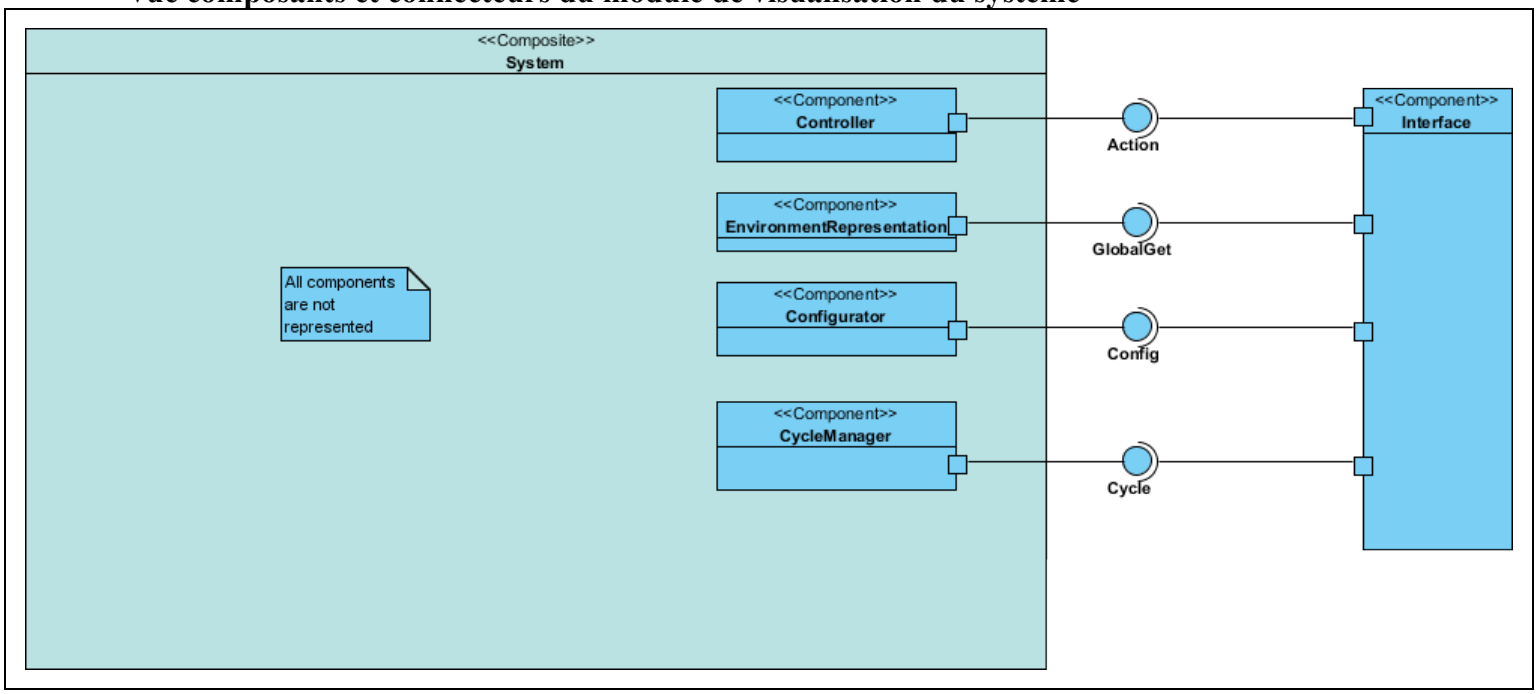
Justifications :

Ici, afin de configurer les composants, des ports fournissant des services de configuration ont été rajoutés (pour la communication réseau, l'environnent, et les agents).

Un composant permettant de gérer toutes ces configurations a également été créée pour une gestion plus aisée et centralisée de la configuration du système.

V. Visualisation du système

Vue composants et connecteurs du module de visualisation du système



Exigences :

ENF 4. Il doit être possible de visualiser l'exécution du système.

ENF 6. Le système de visualisation doit être le plus découplé possible du reste du système.

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

ENF 12. Votre architecture doit être décrite le plus précisément possible (vues C&C, module et allocation).

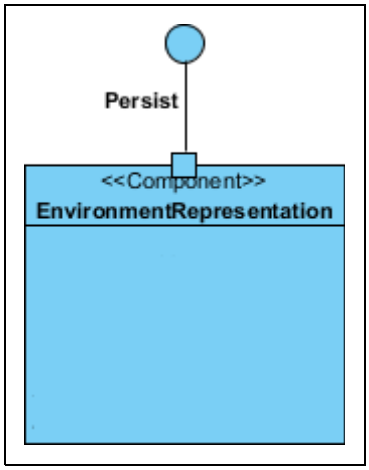
Justifications :

Un composant composite représentant l'ensemble du système sans la partie interface est réalisé afin de limiter au maximum le couplage entre le système et son interface. Les services nécessaires au contrôle du système et à sa visualisation sont bien sûr fournis par ce composite.

Un composant d'interface est quant à lui prévu afin de justement visualiser le système.

VI. Persistance

Vue composants et connecteurs de l'environnement



Exigences :

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

ENF 12. Votre architecture doit être décrite le plus précisément possible (vues C&C, module et allocation).

ENF 16. Il doit être possible de persister l'état du système.

Justifications :

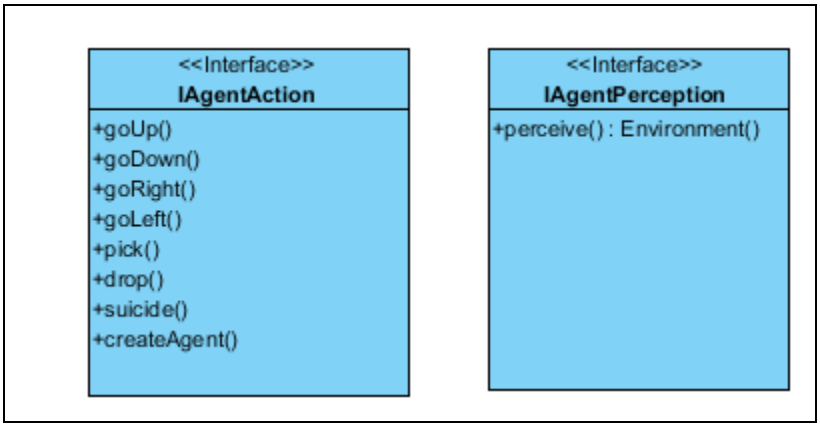
La totalité des informations permettant de visualiser le système se trouve dans la représentation de l'environnement. Le persister reviendrait donc à sauvegarder l'état courant du système. Ainsi, afin de respecter l'exigence n°16, un service de persistance à été ajouter au composant de l'environnement. Celui ci n'est pas utilisé dans notre système (puisque qu'il ne sera pas utilisé) mais il serait facile de le faire.

VII. Implémentation des agents

Plusieurs implémentations des agents ont été faites : une implémentation 'random' afin de pouvoir tester l'exécution du système et deux autres implémentations dont le fonctionnement sera détaillé.

1. Interfaces

Ici sont les interfaces auxquelles les agents ont accès. Elles sont présentées ici afin de montrer les actions réalisables par un agent :



2. Première implémentation avec vérification de blocage

Cette implémentation des agents, classe AgentDecisionFarFromAreasImpl, se base sur le fait que les agents vont chercher les boîtes vu qu'ils connaissent leurs position de base.

Si l'agent ne porte pas de boîte, il va se diriger vers la zone de stockage des boîtes.
S'il en porte une, il va se diriger vers le dépôt.

Il peut arriver des conflits au niveau des couloirs, pour cela les agents ne portant pas de boîte regardent devant eux pour voir s'il y a une situation de blocage: typiquement, si l'agent voit que devant lui il se trouve deux agents, l'agent va essayer de reculer.

La résolution du problème peut être un peu longue parfois mais réussit à chaque fois.

3. Deuxième implémentation réactive

Dans cette implémentation, les agents réagissent simplement avec leur perception locale et la connaissance de la position des sites.

Chaque agent essaye d'aller vers le point du site qui lui est le plus proche. En cas de gène, il cherche simplement à faire le déplacement qui l'éloignera le moins possible de son objectif. Lorsqu'il doit faire un déplacement qui l'éloigne de son objectif (parce qu'il n'a pas le choix), l'agent définira ce déplacement comme son déplacement 'préféré' lorsqu'il n'a pas le choix.

Ainsi, lors d'un blocage à l'intérieur des couloirs, cela donne l'impression que les agents voulant aller dans une direction poussent ceux voulant aller dans l'autre. En réalité, les agents vont naturellement choisir un déplacement 'préféré' et le passage va finir par se dégager.

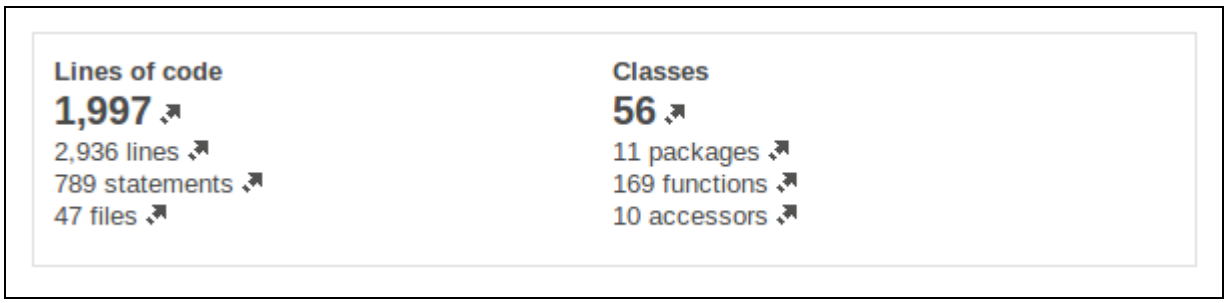
Cette implémentation à l'avantage de débloquer les couloirs assez rapidement, cependant, à la fin de l'exécution les derniers agents ne rapportent pas leur caisse au dépôt puisqu'il ont besoin d'autres agents pour les pousser à changer leur trajectoire. (Je pense qu'on appelle cela une situation de non-coopération ;)).

4. Quantification du code

Nombre de lignes de code totales : 2 936

Nombre de lignes écrites (sans commentaires, imports, ..) : 1 997

Rapport sonar :



Matrice de traçabilité des exigences

Chapitres Exigences	I	II	III	IV	V	VI	VII
ENF 1		X					
ENF 2							X
ENF 3							X
ENF 4					X		
ENF 5							
ENF 6					X		
ENF 7							
ENF 8	X	X	X	X	X	X	
ENF 9							X
ENF 10	X	X	X				
ENF 11				X			
ENF 12	X	X	X	X	X	X	
ENF 13				X			
ENF 14			X				
ENF 15							
ENF 16						X	
ENF 17	X						