Visualisation

Valvassori Moïse

Fri Jun 27 07:38:41 2003

Résumé

Ceci n'est encore qu'un **brouillon**.
Ce document décrit le service de visualisation de Fungus.

Table des matières

1 Déclaration d'une Visualisation

ToDO Avec /Sans Mycelium

2 Flux d'exécution

Nous allons décrire différents flux d'exécution liés aux visualisations.

2.1 Création d'une visualisation

Nous partons d'un tag `<visualisation>` qui peut contenir des paramètres.

Lors du traitement du tag, on récupère les paramètres[1] de la visualisation et on les stocke dans un table de hachage. Ensuite, on appelle `addVisualisation` le service de visualisation.

Dans le service de visualisation, on trouve un `Container` et un `DataCollector`. Ces objets sont donnés par la classe `VisualisationManager`. Le container est fixé par l'environnement d'execution (Frame AWT, Module de visualisation pour la GUI). Le `DataCollector` est donnée par une instance de `VisualisationFactory`. Actuellement, il existe deux *factory* : `NullFactory` et `VisualisationModule`.

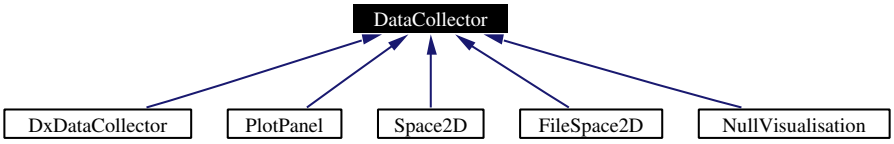

FIG. 1 – Héritage de `DataCollector`

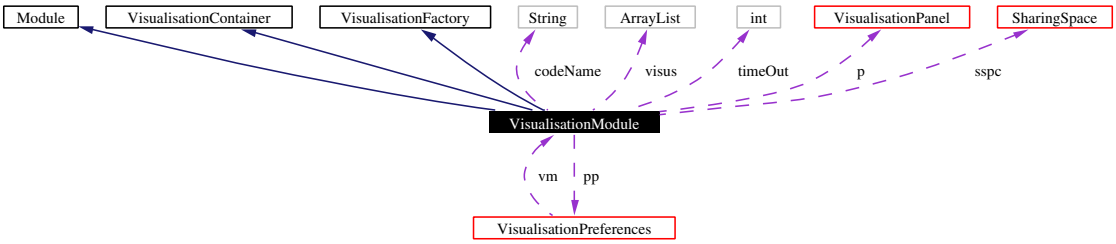

FIG. 2 – Collaboration de `VisualisationModule`

[1] Les paramètres ne sont pas directement lié à la visualisation. Ils dépendent du type de visu.

`VisualisationModule` utilise le fichier de propriété `fungus.prop` pour trouver la bonne visualisation. Le `DataCollector` est inséré dans le container.

On passe les paramètres de la visualisation au `DataCollector`. Puis, on démare la visualisation.

2.2 Processus de Visualisation

Il existe deux modes de visualisation :
- toutes les données sont visualisables
- on ne visualise que des échantillons de données

2.2.1 dirigé par `Collectable`

`DataCollector` envoie un référence de lui même au `Collector` (`sendDataCollector`). Le `Collector` peut alors envoyer ses données.

Cette méthode est utilisé pour faire des visualisations synchronisés avec les agents. Si on envoie toutes les données, elles pourront être traité en direct.

Normalement, si le `Collector` ne possède pas de référence au `DataCollector`, on n'envoie pas de données. On utilise peut être la méthode dirigée par le `DataCollector`.

2.2.2 dirigé par `DataCollector`

Le `DataCollector` demande les données quand il en a besoin. Il fait cela en utilisant la méthode `getData`.

En général, on attend dans un *thread*. Quand le *thread* se reveille, il demande les données.

On peut aisément réaliser des visualisations qui se rafraîchissent à intervalles réguliers et pas forcement de manière synchrone.

3 Quelques `DataCollector`

3.1 `Space2D`

3.2 `DxDataCollector`

4 Ancienne Doc

On visualise des «*parties*» d'agents ou de groupes d'agents.

Listes des types de visualisations connues :

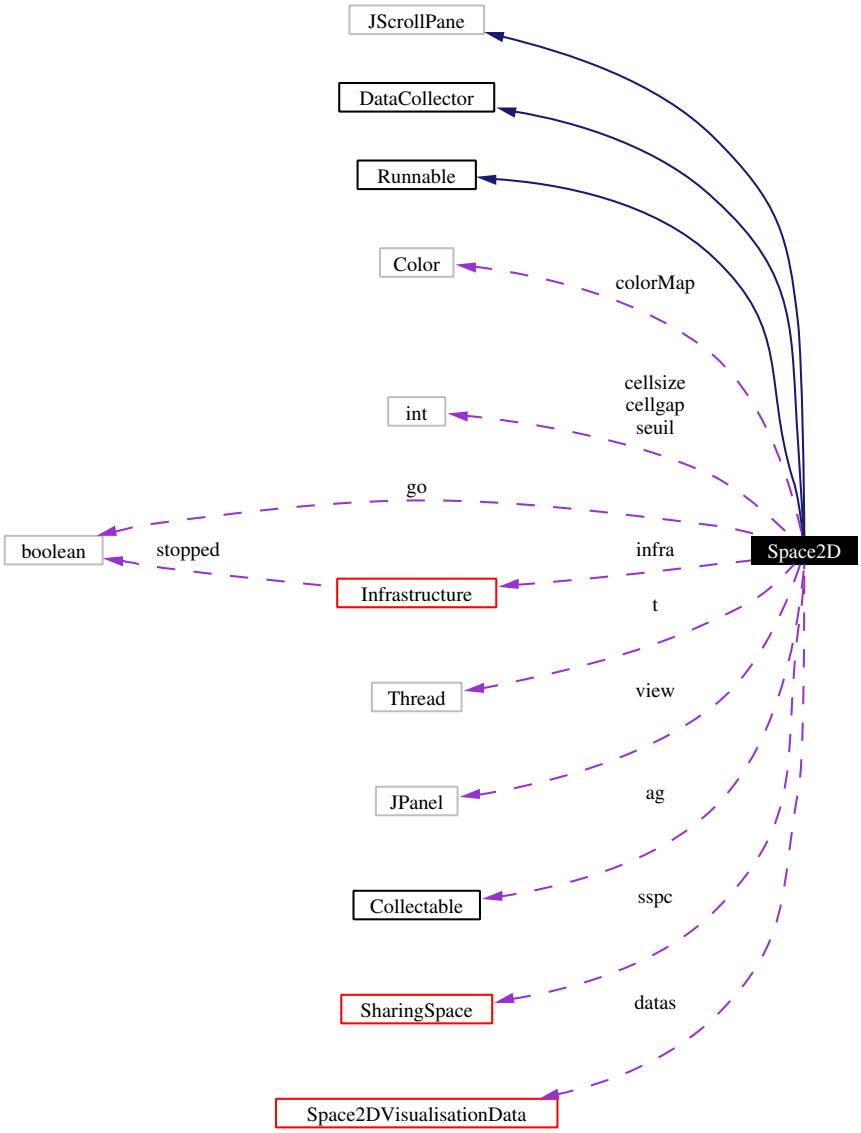

FIG. 3 – Collaboration de `Space2D`

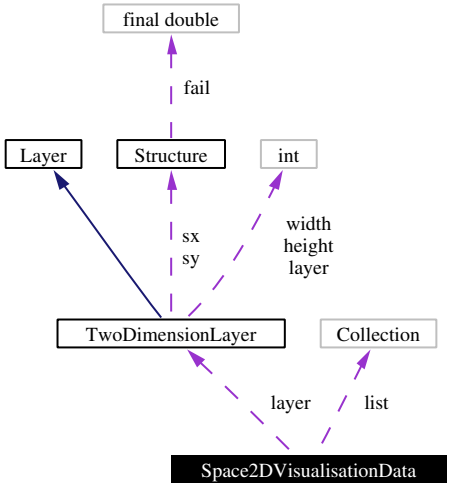

FIG. 4 – Collaboration de `Space2D`

int2Dplot

simple graphe en 2D. Les entrées sont des valeurs entières.

reconu par `fungus.gui.module.VisualisationModule` et tracé par `fungus.gui.modu`

Quelques idées dans [FC02].

Références

[FC02] Jean-Marie Favre and Humberto Cervantes. Visualisation of component-based software and component-based software visualisation tools. In *VISSOFT'02*, Paris, 2002.

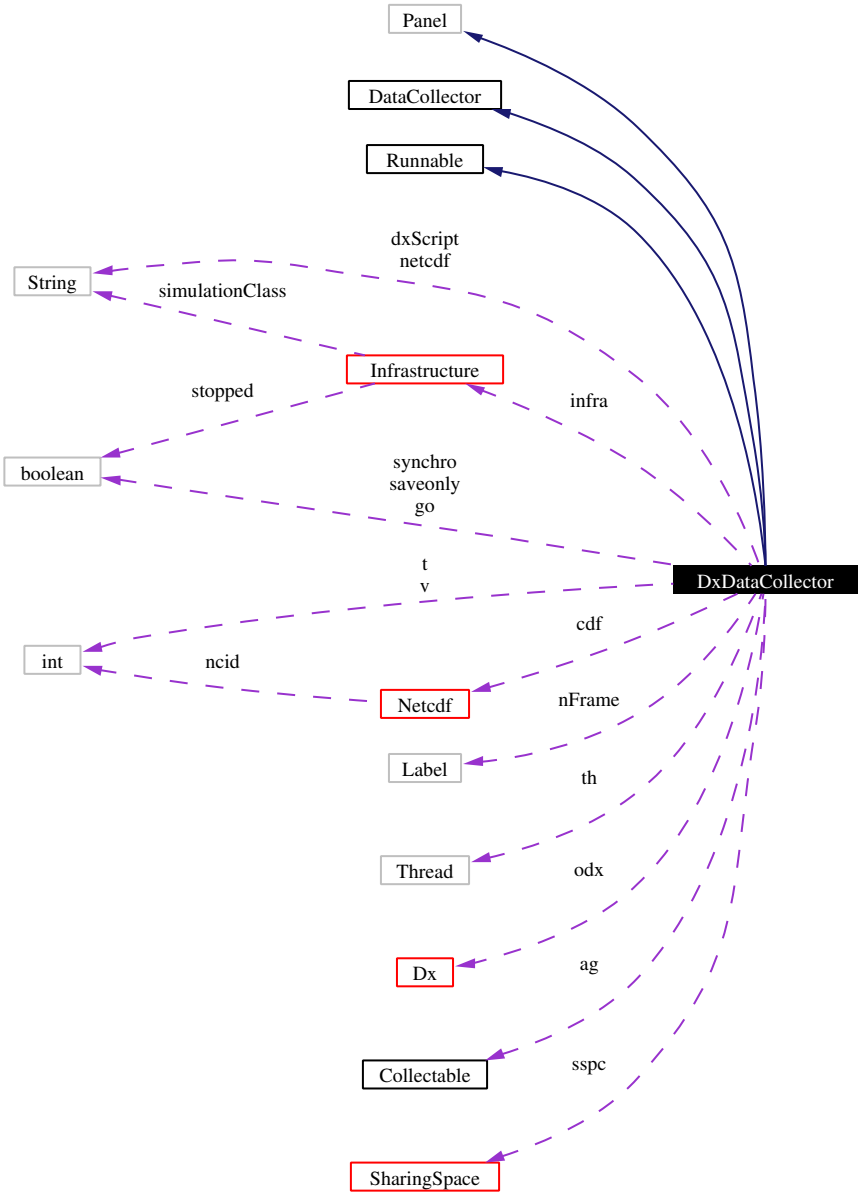

FIG. 5 – Collaboration de `DxDataCollector`

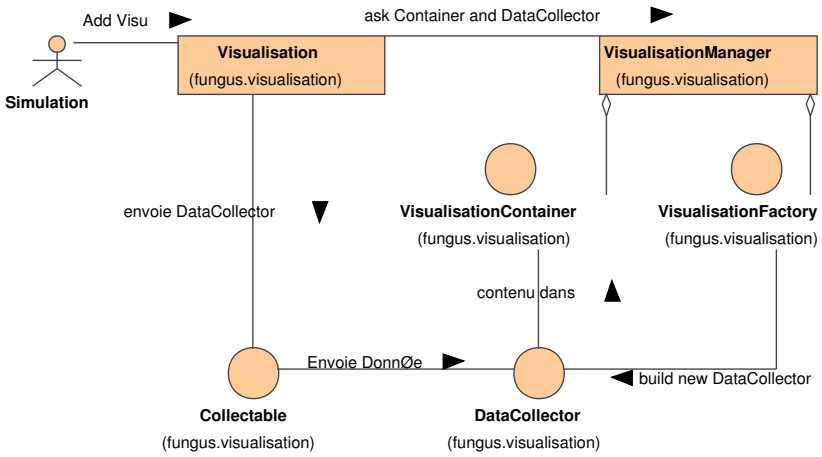

FIG. 6 – Diagrame UML de cas. Une simulation demande une visualisation sur un agent. Le service à l'aide du *Manager* fabrique un collecteur. L'agent envoie ses données au collecteur.