Gestion des workflows dans DIET

École normale supérieure de Lyon

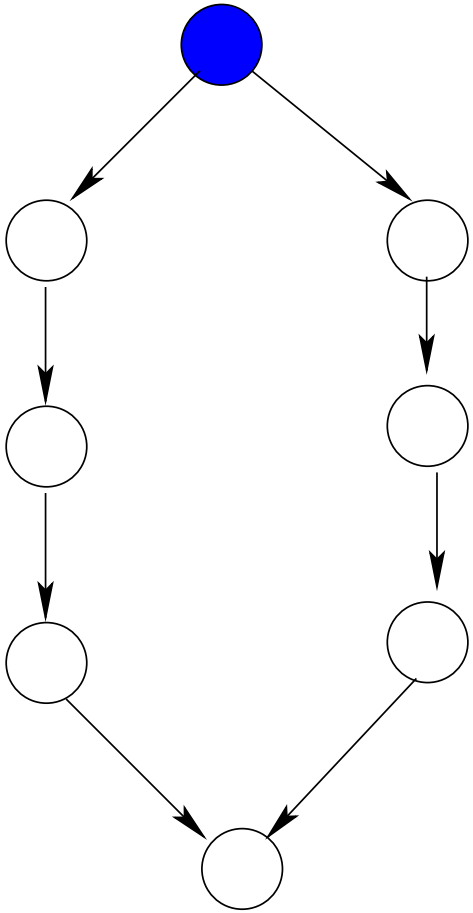
Réunion GDS
13 octobre 2006

Plan

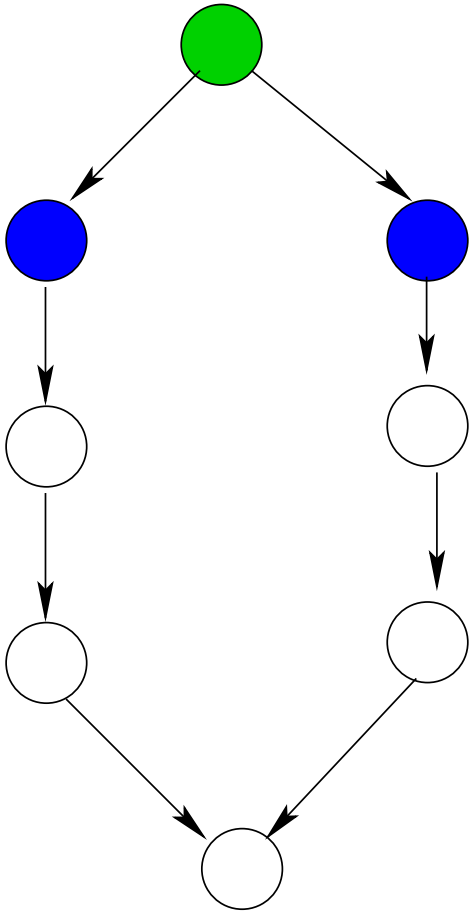
Les workflows

- Ensemble de tâches connectées
- La structure du workflow représente la relation temporelle entre ses tâches
- Généralement représenté par un DAG (*Direct Acyclic Graph*)
  - chaque nœud est une tâche
  - chaque nœud peut avoir un certain nombre de fils ou de parents à condition qu'il n'y ait pas de boucle

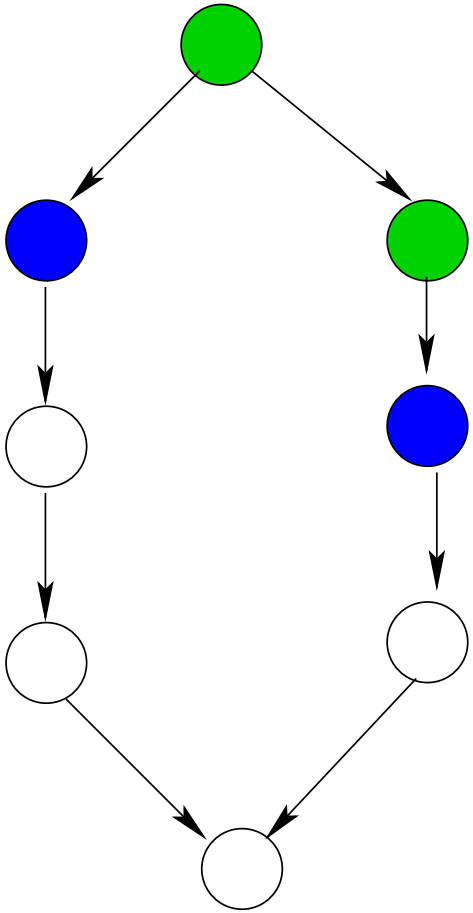
Exécution des workflows



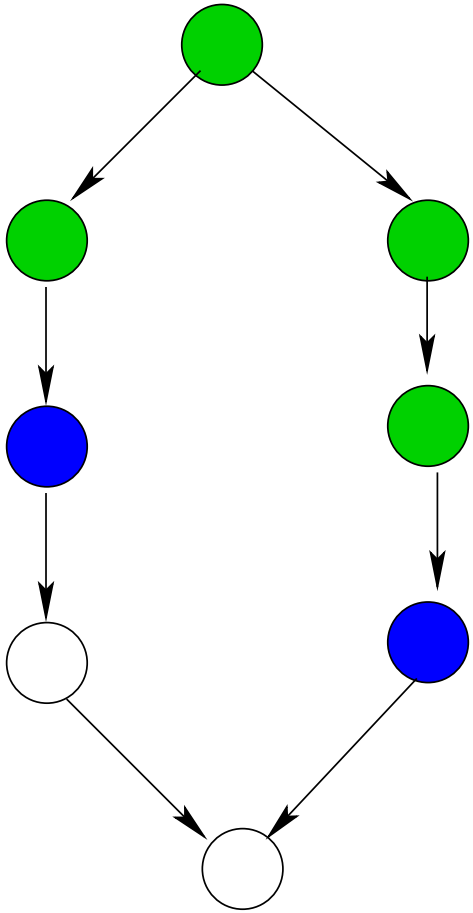
Exécution des workflows



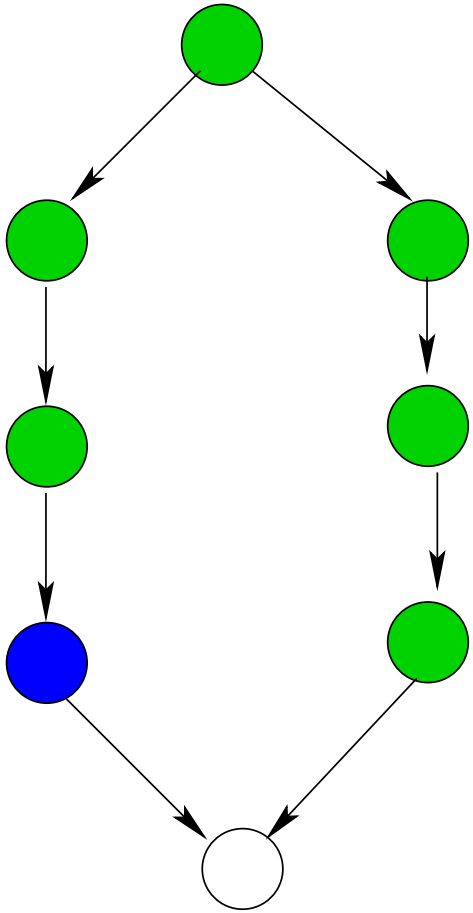
Exécution des workflows



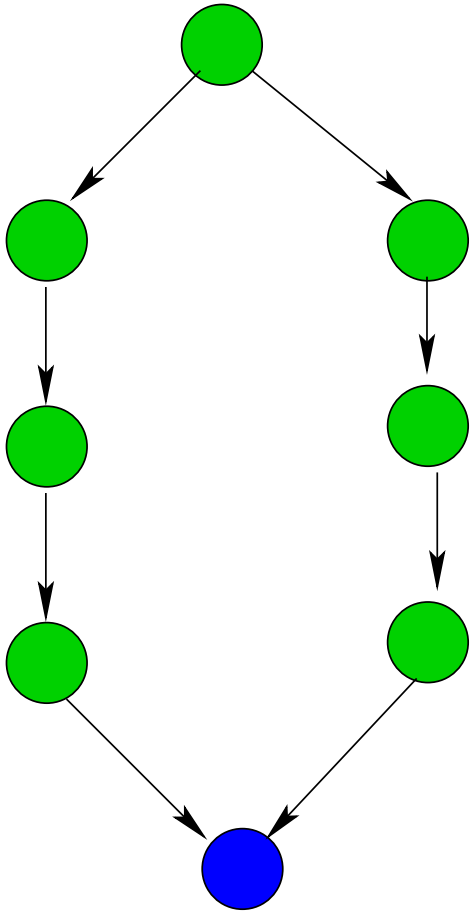
Exécution des workflows



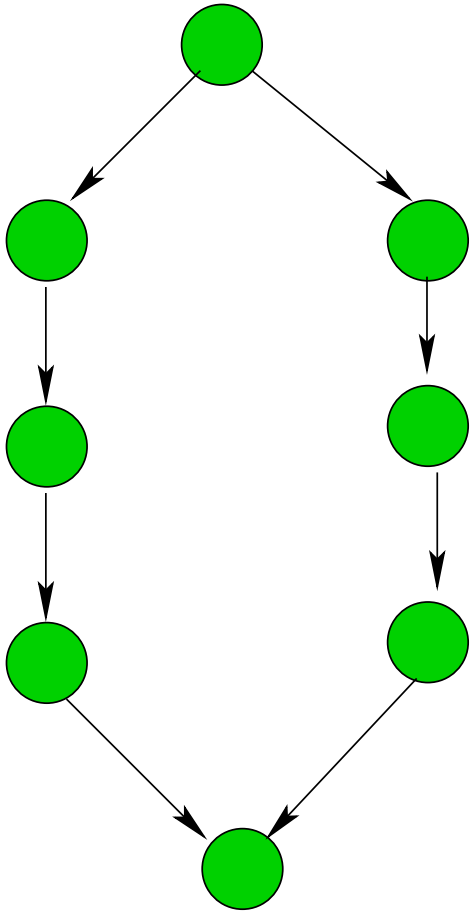
Exécution des workflows



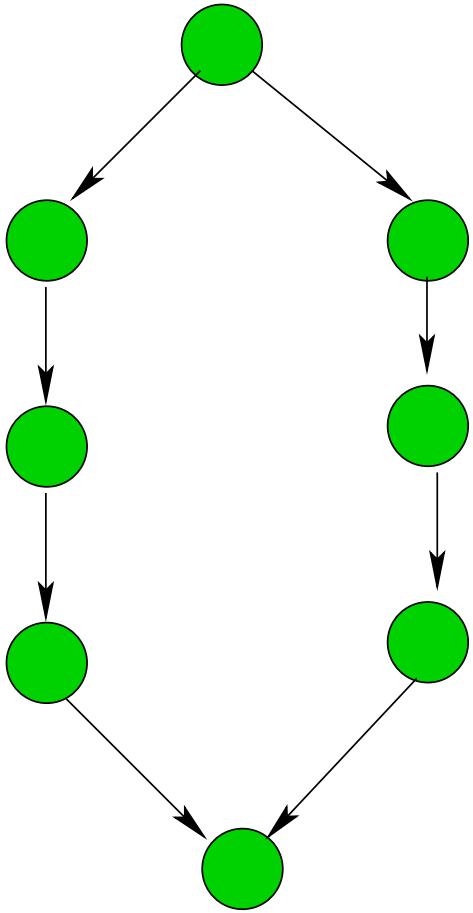
Exécution des workflows



Exécution des workflows



Exécution des workflows



- Deux questions
  - Quel ordre d'exécution entre tâches prêtes?
  - Mapping des tâches sur les ressources?

Objectifs

- Avoir un environnement de gestion de workflows
  - construction et exécution
  - utilisation d'algorithmes d'ordonnancement avancé
  - prise en charge des variations du système
  - gestion de multi-workflows

Langages de workflows

- Pas de standard (XML, scripts)
  - BPEL pas adapté
- Exemples:
  - Condor DAGMan: script
  - Pegasus: DAX (xml)
  - Taverna: XScuffl (xml)
  -
- La plupart des moteurs de workflows utilisent des DAG ou des réseaux de pétri.
- Le workflow est décrit à deux niveaux:
  - abstrait: description uniquement de l'application
  - concret: description de l'application et des informations nécessaires à l'exécution

Description du workflow dans DIET

- Format XML
- Profile diet: problème (id), paramètres (in, out et inout)
- Description des nœuds et des dépendances de donnés

```
<dag>
  <node id="node_id" path="path_name" [mul="value"]>
    <arg name = "arg_id" type="data_type" value="arg value"
      [base_type="base type"] [nb_rows="nb_rows"]
      [nb_cols="nb_cols"] [matrix_order="matrix_order"]  />
    <in name  = "port_id" type="data_type" [sink="source id"]
      [base_type="base type"] [nb_rows="nb_rows"]
      [nb_cols="nb_cols"] [matrix_order="matrix_order"]
      [mul="value"]   />
    <out name = "port_id" type="data_type" [sink="sink id"]
      [base_type="base type"] [nb_rows="nb_rows"]
      [nb_cols="nb_cols"] [matrix_order="matrix_order"]
      [mul="value"]   />
  </node>
... </dag>
```

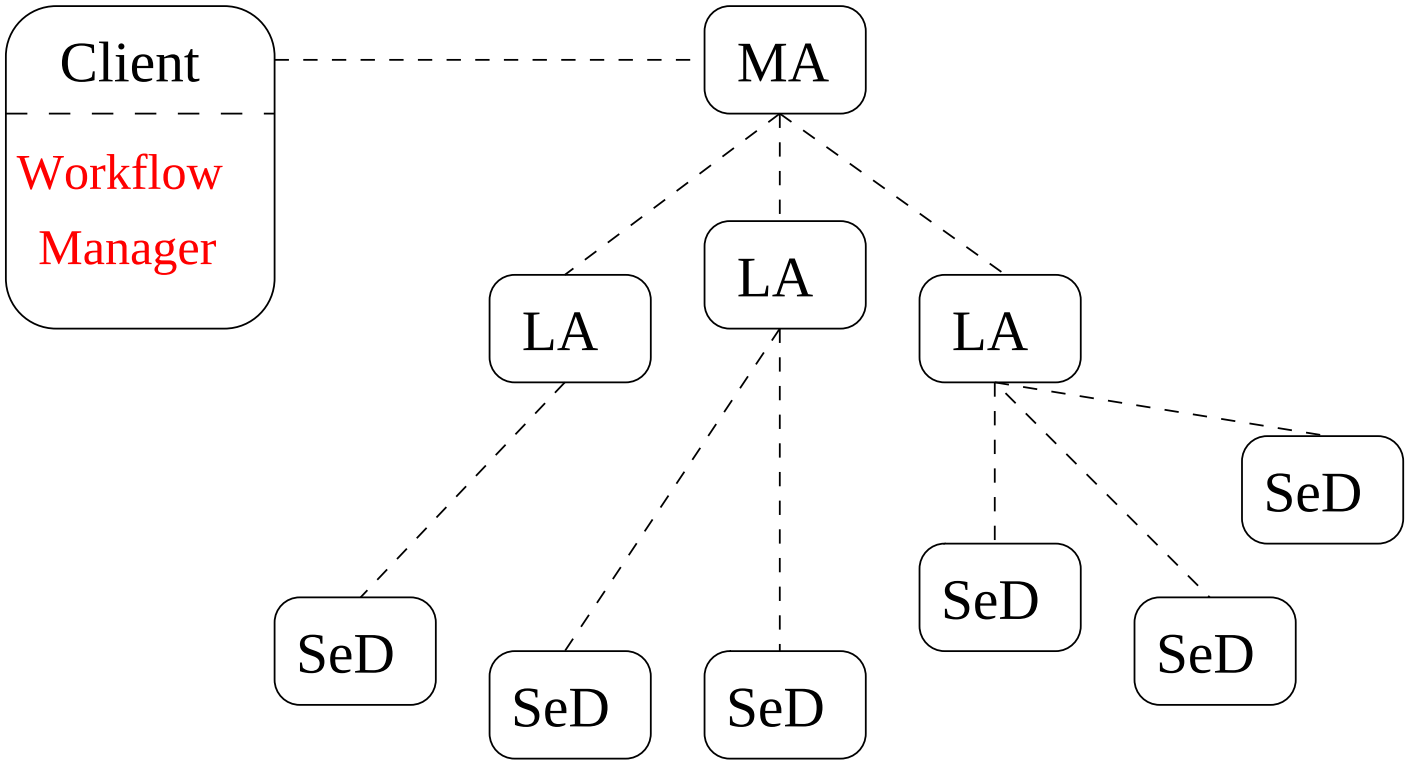
Quelle architecture?

Quelle architecture?

- Généralement le méta-ordonnanceur de workflows se trouve côté client

Quelle architecture?

- Généralement le méta-ordonnanceur de workflows se trouve côté client



Architecture logicielle

Workflow manager dans le client

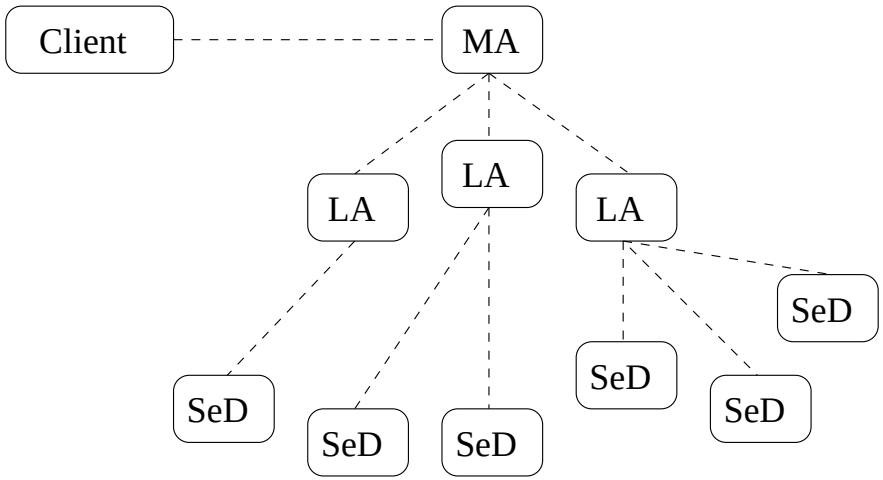
- Inconvénients:
  - aucune coopération entre les différents workflow managers.

Architecture logicielle

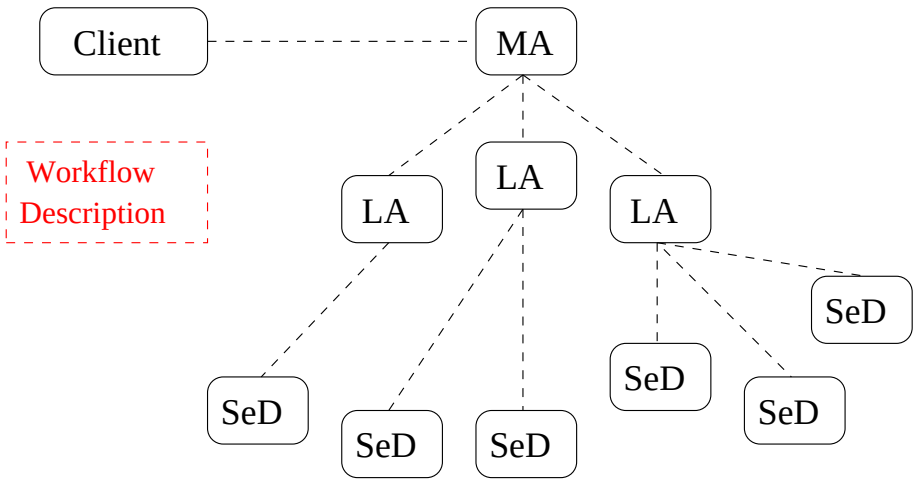
Workflow manager dans le client

- Inconvénients:
  - aucune coopération entre les différents workflow managers.
- Avantages:
  - plus de souplesse pour tester de nouveaux algorithmes d'ordonnancement.

Architecture 1: fonctionnement

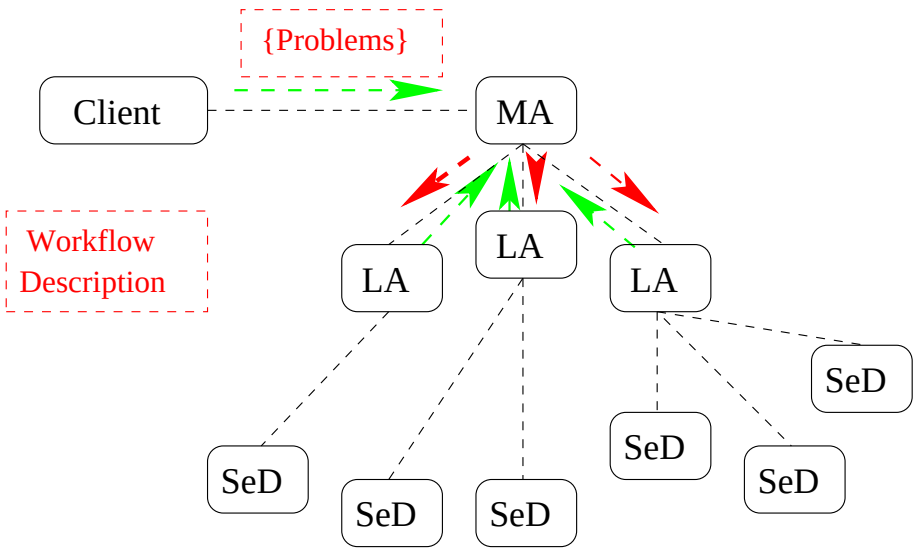


Architecture 1: fonctionnement



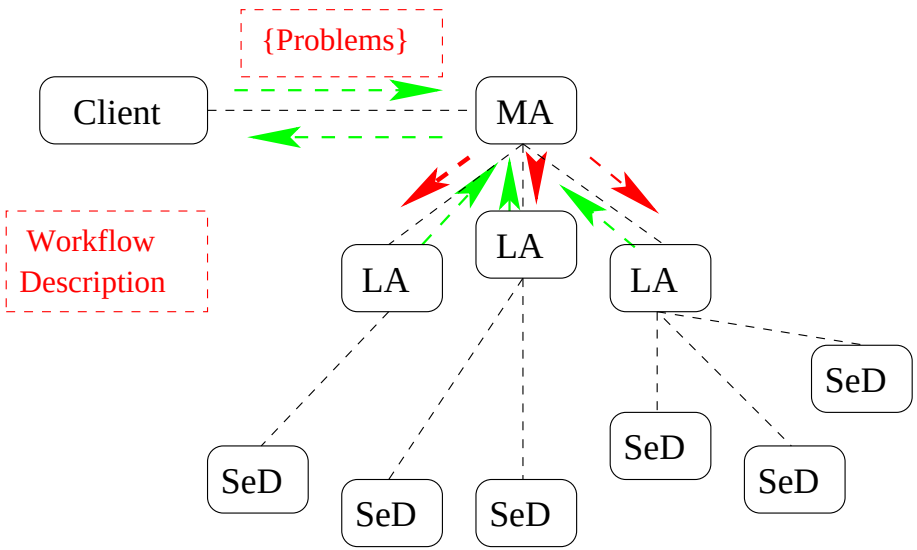
1. Le client traite la description du workflow

Architecture 1: fonctionnement



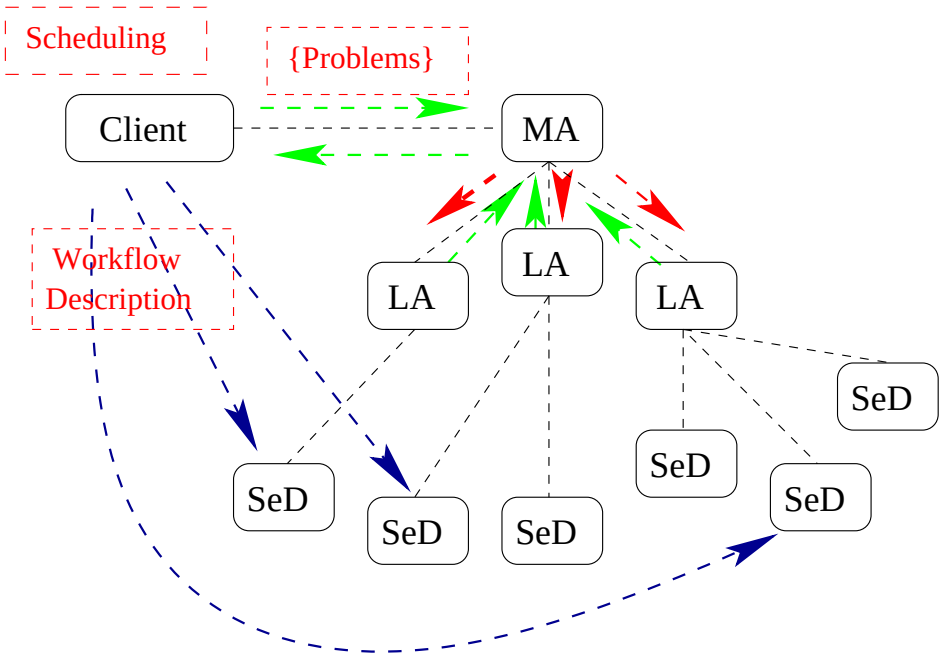
1. Le client traite la description du workflow
2. Le client envoie la liste des problèmes composants le workflow au Master Agent.

Architecture 1: fonctionnement



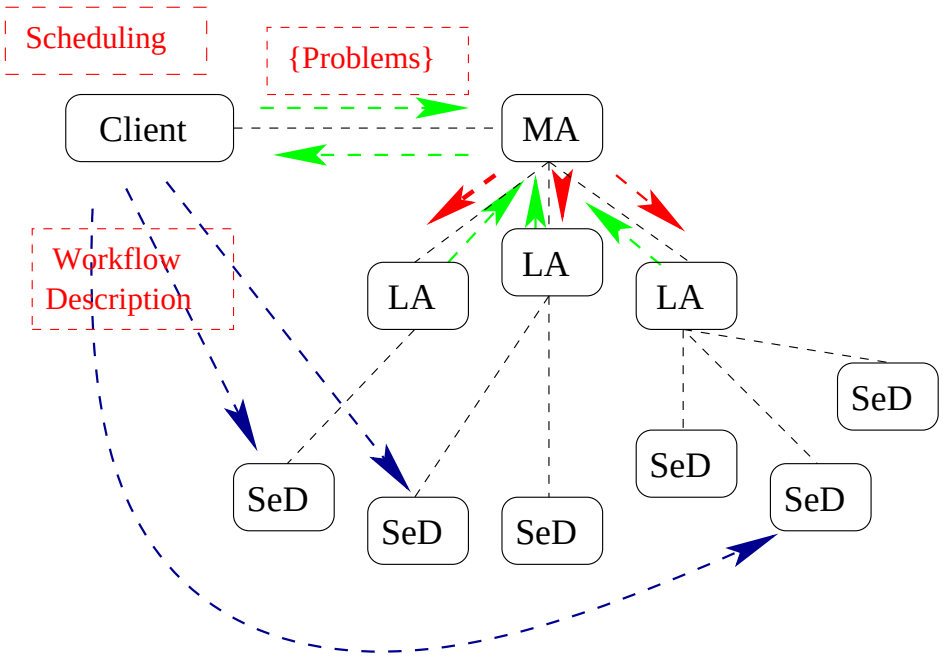
1. Le client traite la description du workflow
2. Le client envoie la liste des problèmes composants le workflow au Master Agent.
3. En cas de succès,le client reçoit la liste des serveurs nécessaires à l'exécution du workflow.

Architecture 1: fonctionnement



1. Le client traite la description du workflow
2. Le client envoie la liste des problèmes composants le workflow au Master Agent.
3. En cas de succès,le client reçoit la liste des serveurs nécessaires à l'exécution du workflow.
4. Construction de l'ordonnancement puis exécution du workflow

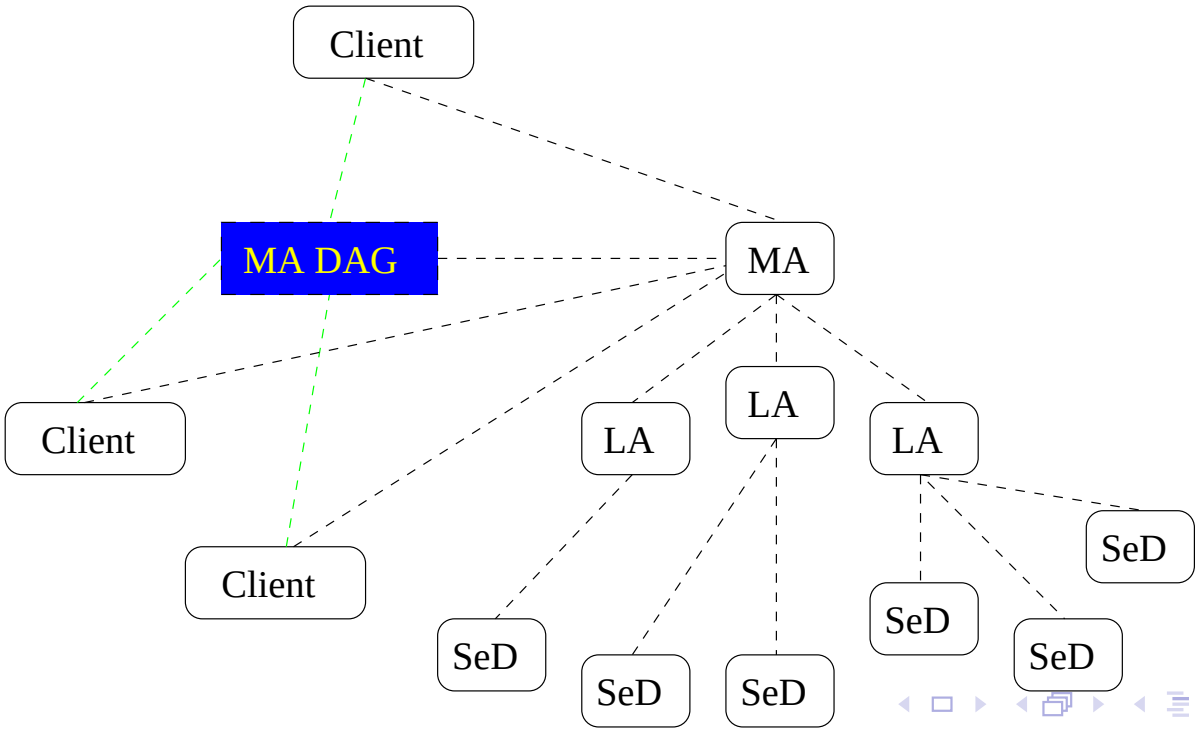
Architecture 1: fonctionnement



1. Le client traite la description du workflow
2. Le client envoie la liste des problèmes composants le workflow au Master Agent.
3. En cas de succès,le client reçoit la liste des serveurs nécessaires à l'exécution du workflow.
4. Construction de l'ordonnancement puis exécution du workflow
5. Récupération des résultats.

Architecture 2 : Workflow manager externe centralisé : le MA DAG

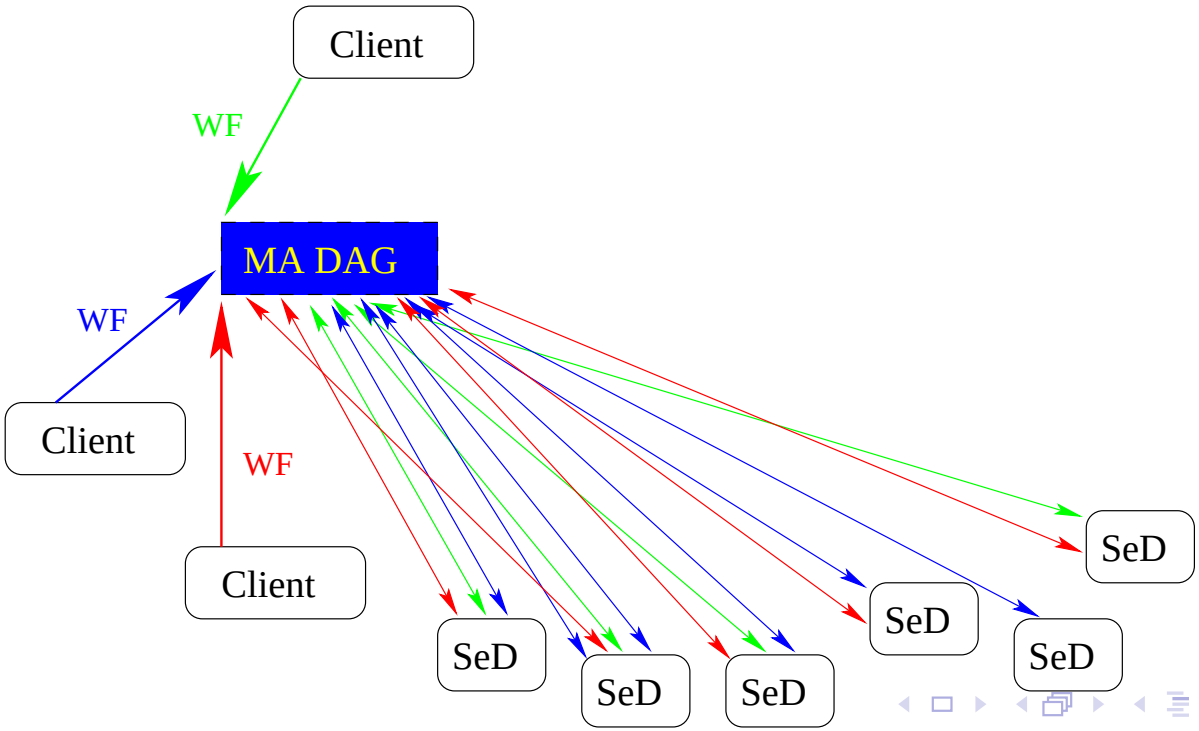
- Motivation: gérer le multi-workflows/multi-clients



Architecture 2 : Workflow manager externe centralisé : le MA DAG

- Motivation: gérer le multi-workflows/multi-clients

$\Rightarrow$ problème : gestion des données!



Architecture 2 : Workflow manager dans le client et le MA DAG

- Diviser le méta-ordonnanceur entre le client et le MA DAG

Architecture 2 : Workflow manager dans le client et le MA DAG

- Diviser le méta-ordonnanceur entre le client et le MA DAG
- La partie client du méta-ordonnanceur :

Architecture 2 : Workflow manager dans le client et le MA DAG

- Diviser le méta-ordonnanceur entre le client et le MA DAG
- La partie client du méta-ordonnanceur :
  - gestion des données

Architecture 2 : Workflow manager dans le client et le MA DAG

- Diviser le méta-ordonnanceur entre le client et le MA DAG
- La partie client du méta-ordonnanceur :
  - gestion des données
- La partie MA DAG du méta-ordonnanceur:

Architecture 2 : Workflow manager dans le client et le MA DAG

- Diviser le méta-ordonnanceur entre le client et le MA DAG
- La partie client du méta-ordonnanceur :
  - gestion des données
- La partie MA DAG du méta-ordonnanceur:
  - création d'un ordonnancement initial des workflows soumis

Architecture 2 : Fonctionnement

Mode 1: ordering et mapping

- le client envoie la description du workflow au MA DAG

Architecture 2 : Fonctionnement

Mode 1: ordering et mapping

- le client envoie la description du workflow au MA DAG
- le MA DAG (vérifie auprès du MA) la disponibilité des services

Architecture 2 : Fonctionnement

Mode 1: ordering et mapping

- le client envoie la description du workflow au MA DAG
- le MA DAG (vérifie auprès du MA) la disponibilité des services
- en cas de succès le MA DAG construit son ordonnancement (ordering et mapping) et renvoie la réponse au client.

Architecture 2 : Fonctionnement

Mode 1: ordering et mapping

- le client envoie la description du workflow au MA DAG
- le MA DAG (vérifie auprès du MA) la disponibilité des services
- en cas de succès le MA DAG construit son ordonnancement (ordering et mapping) et renvoie la réponse au client.
- le client exécute le workflow

Architecture 2 : Fonctionnement

Mode 1: ordering et mapping

- le client envoie la description du workflow au MA DAG
- le MA DAG (vérifie auprès du MA) la disponibilité des services
- en cas de succès le MA DAG construit son ordonnancement (ordering et mapping) et renvoie la réponse au client.
- le client exécute le workflow

Mode 2: ordering

Architecture 2 : Fonctionnement

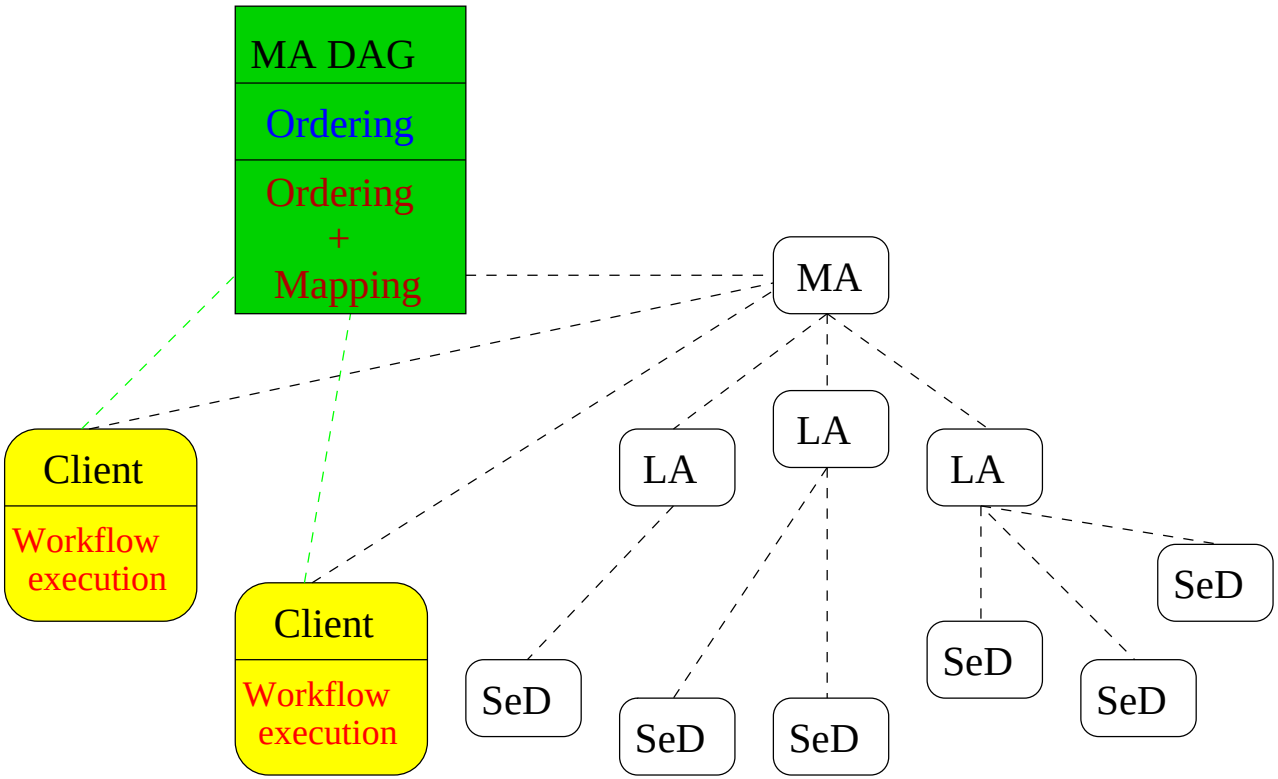
Mode 1: ordering et mapping

- le client envoie la description du workflow au MA DAG
- le MA DAG (vérifie auprès du MA) la disponibilité des services
- en cas de succès le MA DAG construit son ordonnancement (ordering et mapping) et renvoie la réponse au client.
- le client exécute le workflow

Mode 2: ordering

- la réponse du MA DAG comprend uniquement un ordering, le client repasse par le Master Agent pour trouver le bon SeD.

Architecture 2



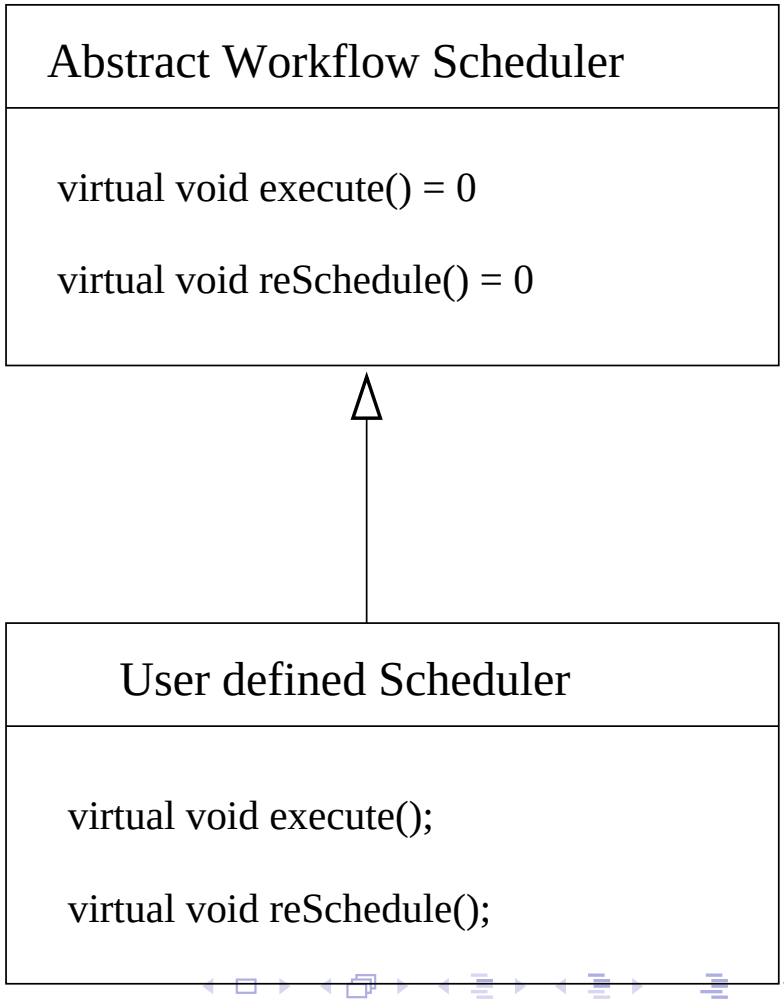
Implémentation

Au lieu d'en choisir une, les architectures retenues peuvent être utilisées conjointement dans la même plateforme.

- Architecture 1: le workflow manager est dans le client
- Architecture 2: utilisation du MA DAG
  - deux modes de fonctionnement: le MA DAG fournit le scheduling complet (ordering et mapping) ou seulement l'ordering

Schedulers dans DIET

- Scheduler de base (par défaut): Scheduler aléatoire
- Flexibilité pour ajouter de nouveaux ordonnanceurs
  - Architecture 1: le client peut fournir son propre scheduler
    - ne nécessite pas la recompilation de DIET
  - Architecture 2: choix du scheduler à la compilation
    - ajouter un nouveau scheduler $\Rightarrow$ la recompilation de DIET



Implémentation dans DIET

- Schedulers avancés
  - Problème de prédiction de performances/temps de communication
  - Prédiction de performances
    - FAST ou CORI (Plugin Scheduler)?

Implémentation dans DIET

- Schedulers avancés
  - Problème de prédiction de performances/temps de communication
  - Prédiction de performances
    - FAST ou CORI (Plugin Scheduler)? $\Rightarrow$ Plugin Scheduler
  - Temps de communication
    - NWS?

    $\Rightarrow$ nécessité de mise en œuvre simple pour l'utilisateur

Réordonnancement

- $\forall$ l'ordonnanceur utilisé:
  - prédiction $\Rightarrow$ risques d'erreur
  - grille de calcul $\Rightarrow$ variations dynamiques

Réordonnancement

- $\forall$ l'ordonnanceur utilisé:
  - prédiction $\Rightarrow$ risques d'erreur
  - grille de calcul $\Rightarrow$ variations dynamiques
  - il y a un risque que l'ordonnancement construit ne soit pas «optimal»

Réordonnancement

- $\forall$ l'ordonnanceur utilisé:
  - prédiction $\Rightarrow$ risques d'erreur
  - grille de calcul $\Rightarrow$ variations dynamiques
  - il y a un risque que l'ordonnancement construit ne soit pas «optimal»
- nécessité de réordonnancer le workflow en cas de problèmes

Quand réordonnancer?

- périodiquement par le MA DAG

Quand réordonnancer?

- périodiquement par le MA DAG
- lorsqu'un nouveau client demande à exécuter un workflow

Quand réordonnancer?

- périodiquement par le MA DAG
- lorsqu'un nouveau client demande à exécuter un workflow
- par le MA lorsque de nouveaux serveurs apparaissent/disparaissent de la plateforme

Quand réordonnancer?

- périodiquement par le MA DAG
- lorsqu'un nouveau client demande à exécuter un workflow
- par le MA lorsque de nouveaux serveurs apparaissent/disparaissent de la plateforme
- quand l'ordonnancement prévu par le client ne correspond pas aux temps réels

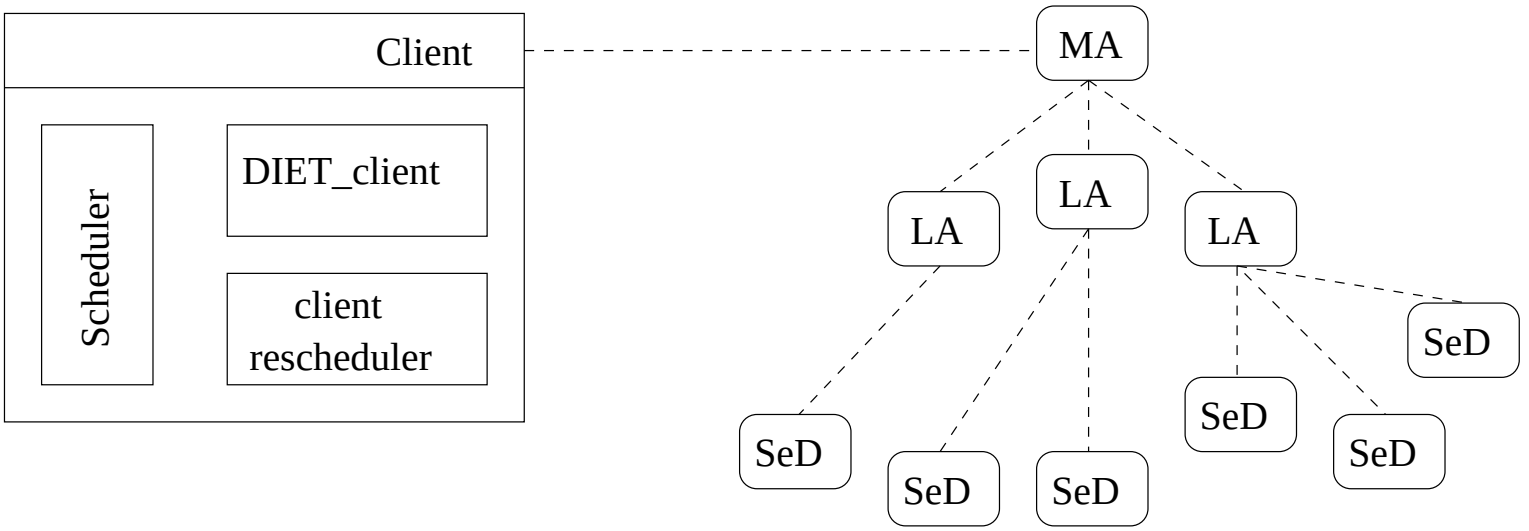
Quand réordonnancer?

- périodiquement par le MA DAG
- lorsqu'un nouveau client demande à exécuter un workflow
- par le MA lorsque de nouveaux serveurs apparaissent/disparaissent de la plateforme
- quand l'ordonnancement prévu par le client ne correspond pas aux temps réels

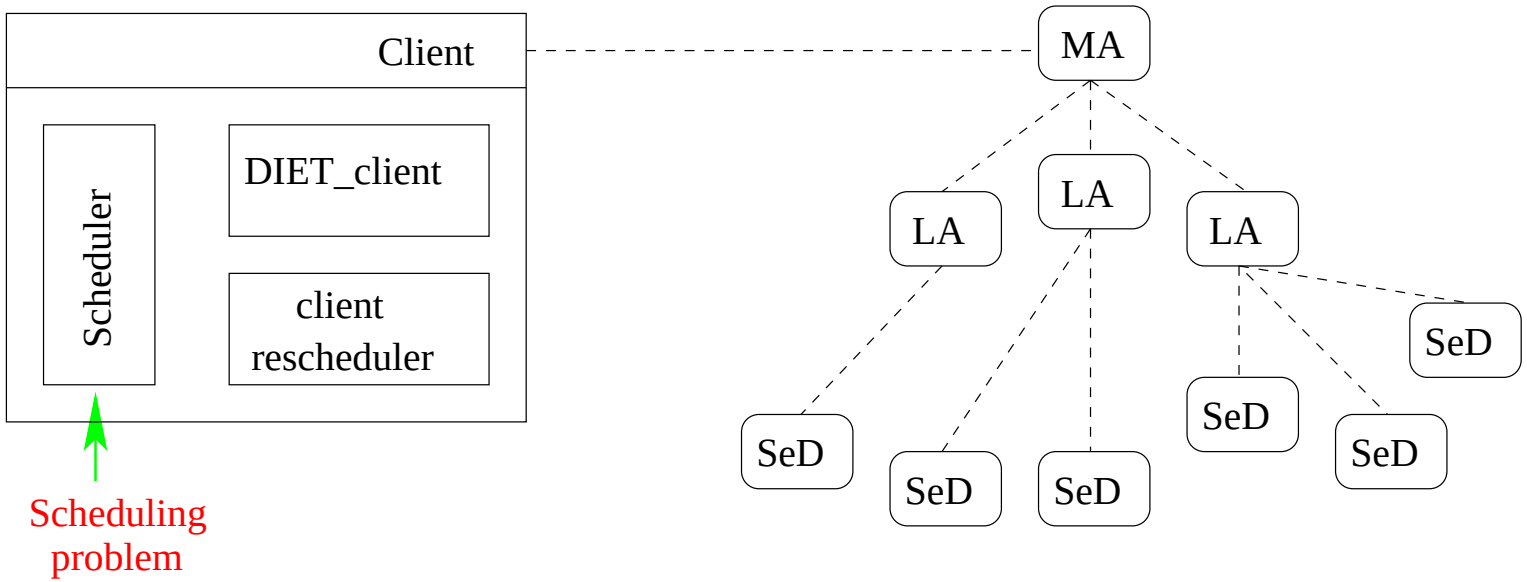
Quand réordonnancer?

- périodiquement par le MA DAG
- lorsqu'un nouveau client demande à exécuter un workflow
- par le MA lorsque de nouveaux serveurs apparaissent/disparaissent de la plateforme
- quand l'ordonnancement prévu par le client ne correspond pas aux temps réels
  - delta et le nombre de nœuds

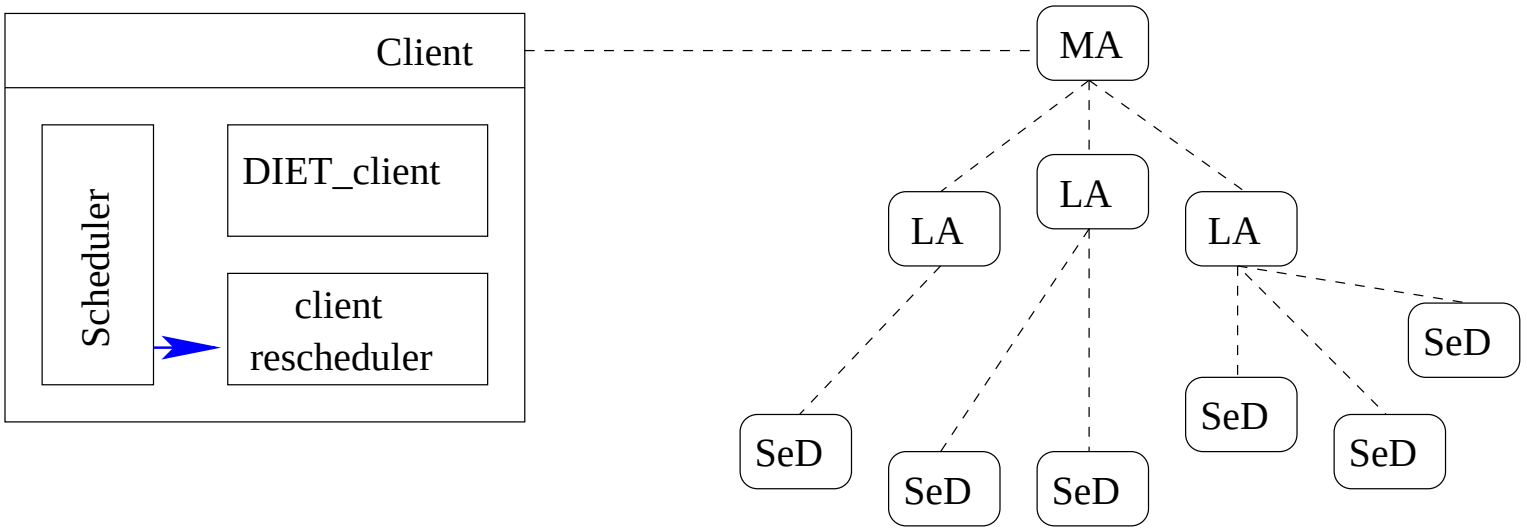
Implémentation



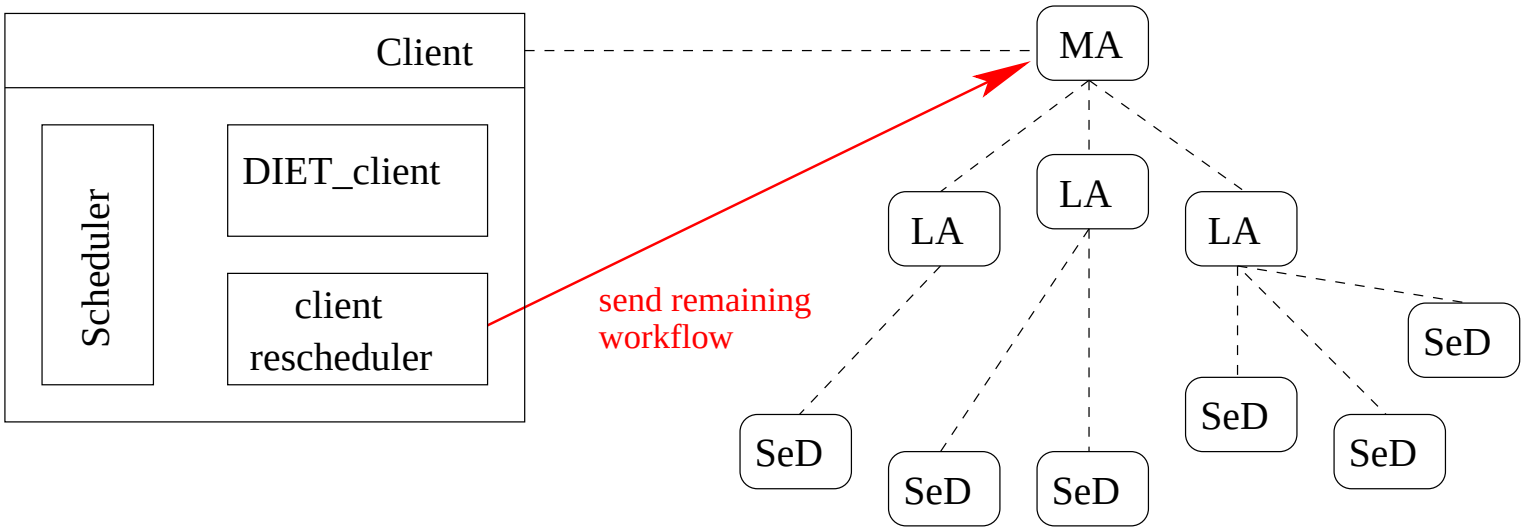
Implémentation



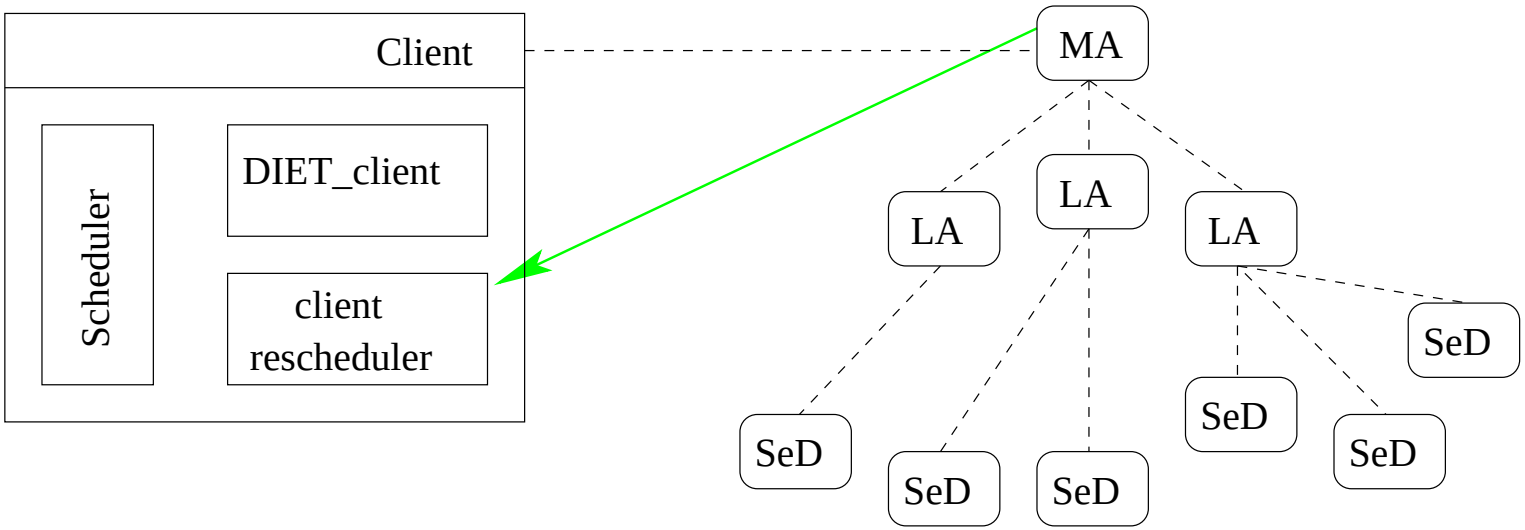
Implémentation



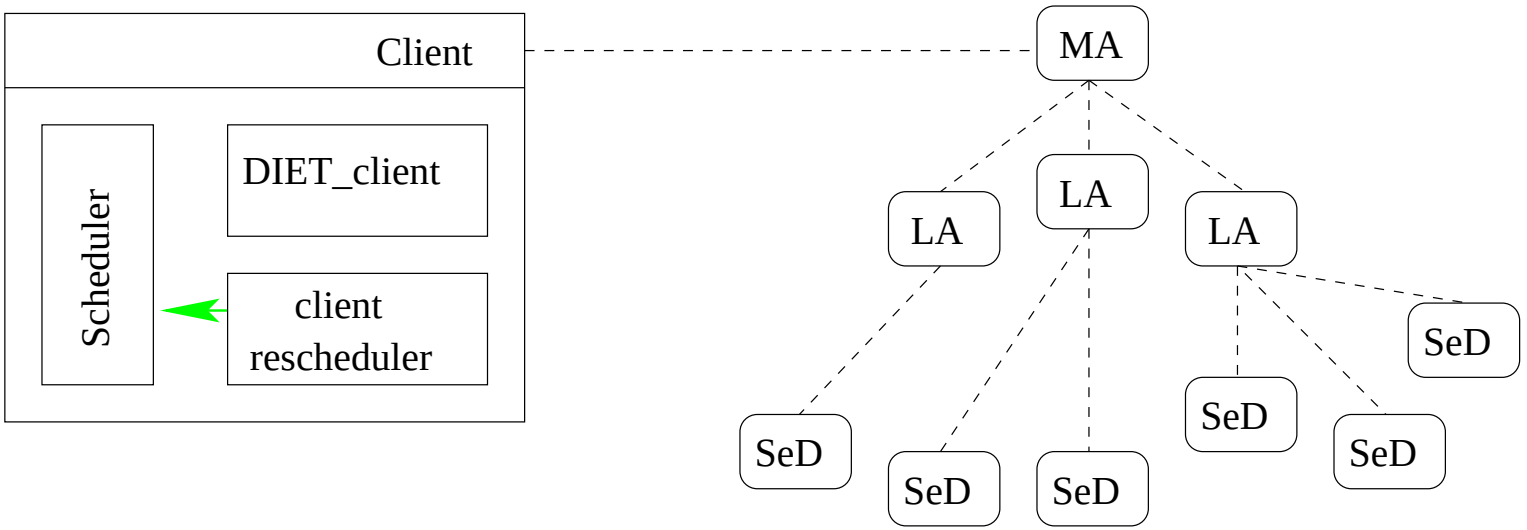
Implémentation



Implémentation



Implémentation



Conclusions

- Design et implémentation d'une architecture pour la gestion des workflows dans DIET
- Flexibilité dans la mise en place d'algorithmes de gestion de workflow (niveau client et MA Dag)
- Ordonnancement aléatoire et HEFT
- Mise en place de mécanismes de réordonnancement
- Gestion multi-workflows