Université Claude Bernard Lyon 1

Rapport de stage

effectué au Laboratoire de l'Informatique du Parallélisme

Sylvère KANAPA

Tuteurs de stage : Christian PÉREZ et Yves CANIOU

Étude du redéploiement du logiciel distribué DIET via l'environnement Concerto.

Du 17/05/2021 au 23/07/2021

29 janvier 2022

Remerciements

À **Christian PÉREZ** et **Yves CANIOU**, mes maîtres de stage, pour leurs conseils et leur patience au cours du stage.

À **Binjamyn MAIRESSE**, un ami et collègue de stage pour les remarques et discussions à propos du stage.

À **Clara PROTHET**, une amie qui a su me conseiller pendant mon stage et mes études.

À mes amis, pour m'avoir encouragé et soutenu au cours de mes études et de ce stage.

Table des matières

1 Introduction

Avec le Fog/Edge computing [1], le déploiement et la reconfiguration d'applications distribuées [2] sur des systèmes distribués [3] (tel que DIET [4]) on connu une phase de complexification. Les équipes projets INRIA, AVALON et STACK, situées respectivement à Lyon et à Rennes, conduisent des travaux afin de réaliser de telles opérations de manière sûre et très efficace. Ces travaux ont amené à la réalisation de la preuve de concept Concerto [5]. DIET est un système distribué qui permet distribuer des calculs sur des ressources géographiquement distantes. Concerto est un logiciel qui permet d'ordonnancer un logiciel ou un groupe de logiciel en fonction de leurs étapes de fonctionnement. Intégrer le logiciel distribué DIET dans l'environment Concerto permet de simplifier son déploiement et sa reconfiguration dynamique. La combinaison de ces deux logiciels a déjà été effectuée par le passé, mais ne permettait pas de reconfigurations dynamiques de DIET. C'est dans ce but que notre travail nous a conduit à proposer une API pour étendre les fonctionnalités de Concerto, OrchestrAPI, et à développer Symphony in DIET, une intégration de DIET dans Concerto en utilisant OrchestrAPI, qui permet la reconfiguration dynamique.

2 Présentation de l'entité d'accueil

2.1 Laboratoire de l'Informatique du Parallélisme

Le Laboratoire de l'Informatique du Parallélisme réunit une centaine d'enseignants-chercheurs et doctorants en thèse autour de différents sujets et problématiques de domaines divers de l'informatique tels que le calcul arithmétique, les architectures logicielles, les preuves de programmes.

Le Laboratoire de l'Informatique du Parallélisme est associé à diverses structures de recherche et d'enseignements. Ces structures sont : le CNRS, l'École Normale Supérieure de Lyon, l'INRIA et l'Université Claude Bernard Lyon 1. Le laboratoire fait partie du MILYON, un groupement de laboratoires situés à Lyon et Saint-Étienne, composé de 350 chercheurs en informatique et mathématiques et permettant de développer la recherche interdisciplinaire.

Le but du Laboratoire de l'Informatique du Parallélisme est d'étudier les fondements de l'informatique actuels puis d'inventer de nouveaux concepts dans les domaines ciblés et d'en mesurer les répercussions dans les autres branches scientifiques. Le laboratoire est dirigé par Nicolas Trotignon et subdivisé en 3 entités :

— L'équipe administrative

— Les équipes de recherches

— Les moyens informatiques

Les équipes de recherche sont au nombre de 7 et chacune réalise des études dans un ou plusieurs domaines informatiques. Celles-ci s'articulent généralement autour de leurs spécificités :

— AriC : *Calcul arithmétique*

— AVALON : *Algorithmes et architectures logicielles pour les plateformes distribuées et HPC*

— CASH : *Compilation et analyse, logiciel et matériel*

— DANTE : *Approche de capture temporelle et structurelle*

— MC2 : *Modèles de calcul, complexité, combinatoire*

— PLUME : *Programmes et Preuves*

— ROMA : *Optimisation de ressources*

1. Paradigme informatique où les serveurs de calculs et de stockage de données sont proches des sources qui les génèrent.
2. Logiciels et applications qui sont exécutés sur un système distribué.
3. Systèmes informatiques où les différents éléments sont situés sur différentes machines reliées entre elles par un réseau.
4. Logiciel distribué décrit dans la section 3.2.2.
5. Un logiciel permettant de l'orchestration automatique, description dans la section 3.2.3

2.2 L'équipe de recherche AVALON

Au cours de ce stage, j'ai intégré l'équipe AVALON dirigée par Christian Pérez. Les recherches d'AVALON s'orientent autour de 3 axes majeurs.

Premièrement, l'efficacité énergétique et la réduction de la consommation d'énergie sont devenues des enjeux importants dans divers secteurs industriels et économiques et les systèmes informatiques n'y échappent pas. Le but est de développer des mécanismes et d'améliorer les analyses pour développer des plateformes à grande échelle plus éco-responsables.

Deuxièmement, la complexité des machines et applications tend à augmenter et rend leurs éxécutions plus difficiles à gérer. Le but est de développer des modèles qui permettent de décrire les structures de ces applications permettant des éxécutions plus efficaces.

Troisièmement, l'équipe effectue des recherches autour de l'ordonnancement et la mise en correspondance de processus avec les machines à disposition dans le Cloud à grande échelle et de la recherche sur le traitement de flux données, la gestion autonome de ressources dans des Clouds multiples, ...

L'équipe est composée d'une dizaine d'enseignants-chercheurs et de 4 membres du personnel parmis lesquels des assistants adminitratifs et des chercheurs. À cela se joint une dizaine d'étudiants effectuant leur thèse au sein de l'équipe.

3 Présentation du stage

3.1 Objectif du stage

L'objectif du stage est de réaliser une étude de cas sur la pertinence du logiciel Concerto [6] ainsi que d'implémenter des fonctionnalités manquantes éventuelles. Ainsi, nous utiliserons le déploiement et la reconfiguration dynamique de DIET [7] comme étude de cas.

Le stage s'est déroulé en plusieurs sous-objectifs :

- La prise en main de DIET – compilation et lancement à la main d'un exemple démo (fourni par DIET)
- La mise en œuvre d'un déploiement de DIET via un script dédié
- La définition et la mise en œuvre de scénario de reconfiguration de DIET
- La prise en main des concepts liés au déploiement et à la reconfiguration, et en particulier de l'environnement Concerto
- L'implémentation de fonctionnalités manquantes éventuelles à Concerto
- Le déploiement sur des conteneurs Docker
- La réalisation d'expériences sur Grid'5000 [8]

3.2 Cadre du travail

3.2.1 Grid'5000

Grid'5000 est un parc de serveurs dédié à la recherche scientifique dans l'informatique. Voici une citation provenant de la page de présentation du projet qui résume comment Grid'5000 est utilisé :

> « Grid'5000 est une plateforme d'essais à grande échelle orientée dans l'expérimentation scientifique dans tous les domaines de l'informatique, avec un focus sur les calculs parallèles et distribués incluant le Cloud (computing), HPC [9], Big-Data [10] et l'IA [11]. »

6. `https://gitlab.inria.fr/VeRDi-project/concerto`
7. `https://graal.ens-lyon.fr/DIET`
8. `https://www.grid5000.fr`
9. High-Performance Computer
10. « Méga-Données »
11. Intelligence Artificielle

3.2.2 DIET

DIET [1] a été developpé par plusieurs membres de l'équipe AVALON. Il permet de distribuer dynamiquement des problèmes sur des serveurs capables de les résoudre.

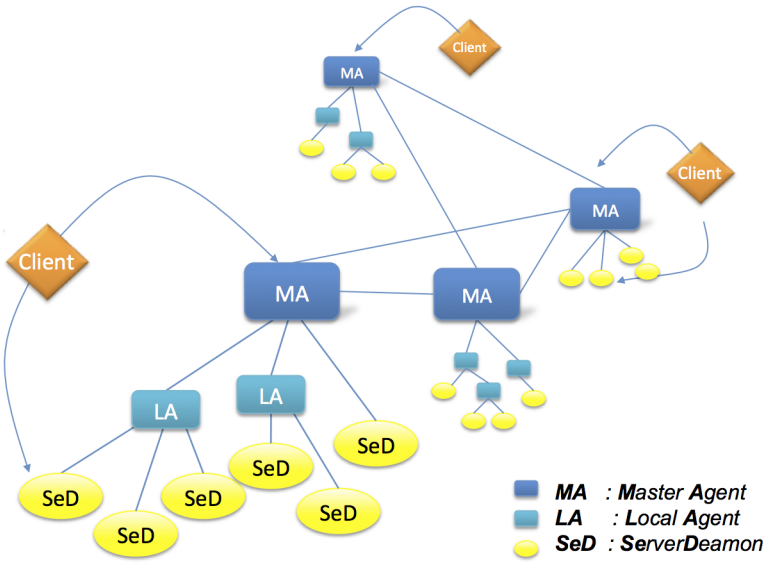

FIGURE 1 – Une hierarchie d'Agents DIET

Dans la figure 1, on peut y voir une hierarchie DIET déployée. Les différents éléments de cette hierarchie sont :

— Client : Un client est une application DIET développée capable de résoudre des problèmes à distance. Une grande partie des différents types de clients sont capables de se connecter à un réseau DIET depuis une page web, un PSE [12] tel que Matlab [13] ou Scilab [14], ou depuis un programme compilé.

— Server Daemon : Un Server Daemon encapsule un server de calculs. Par exemple il peut être situé au point d'entrée d'un ordinateur parallèle. Un Server Daemon peut sauvegarder différentes informations, par exemple une liste de tous les problèmes qui peuvent être résolus dessus et/ou des informations en relation des performances telles que la quantité de mémoire disponible ou le nombre de ressources disponibles. Quand il s'enregistre, un Server Daemon déclare les problèmes qu'il peut résoudre à son parent Local Agent ou Master Agent.

— Local Agent : Un Local Agent transmet les requêtes au sein de l'arborescence/hierarchie DIET. Un Local Agent sauvegarde la liste des services disponibles dans la sous-arborescence : pour chaque service, le Local Agent sauvegarde/enregistre une liste de ses enfants (Agents et Server Daemon) qui peuvent être contactés pour trouver le service. En fonction de la topologie du réseau sous-jacent, une hierarchie de Local Agents peut être déployée entre le Master Agent et les Server Daemon. La fonction d'un Local Agent est aussi de faire de l'ordonnancement partiel sur sa sous-arborescence, ce qui reduit la quantité de travail pour le Master Agent.

— Master Agent : Un Master Agent est un Local Agent particulier : celui-ci fait le lien entre la hierarchie DIET (particulièrement le Server Daemon nécessaire) et le Client. La référence du Server Daemon choisi est renvoyé au Client qui pourra se connecter directement au Server Daemon. Un Client peut obtenir une référence au Master Agent par un nom de serveur spécifique ou une page web qui contient les différentes localisations des Master Agents.

12. Power System Engineering
13. `https://www.mathworks.com/products/matlab.html`
14. `https://www.scilab.org/`

De plus, pour interconnecter les Agents par un réseau, il faut gérer leurs noms. Pour cela, nous utilisons un « Naming Service ». Dans le cas de DIET, CORBA [15] est utilisé.

3.2.3 Concerto

Concerto a été developpé en Python par Maverick Chardet dans le cadre de sa thèse [2], et est une extension du modèle de Madeus [3]. Il permet de coordonner un logiciel (ou un groupe de logiciels) et de permettre l'automatisation de son lancement.

Un « Composant » représente en général un module d'un logiciel distribué, mais peut représenter n'importe quoi tant que son interface (les services et données nécessaires à son fonctionnement et les services et données fournis) peut être exprimée de manière explicite. Un Composant définit un cycle de vie de la même manière qu'un automate [16] (ou un réseau de Petri [17]).

Ces Composants sont définis par différents éléments qui doivent être instanciés par l'utilisateur de Concerto :

- `places` : statut du logiciel, varie entre différentes étapes (exemple : *Installé, Configuré, Éteint, Démarré*)
- `groups` : l'union de plusieurs étapes (exemple : *Démarrable : Configuré, Éteint*)
- `transitions` : représentent les transitions entre les étapes (exemple : *Installation, Configuration, Démarrage, Extinction*)
- `dependencies` : représentent des « ports » qui peuvent être connectés avec d'autres. Les `dependencies` bloquent l'exécution d'une transition tant que la dépendance n'est pas satisfaite (connectée, ou dans l'attente qu'une information soit transmise à travers ces ports)(exemple : *Machine-Mère (Démarré : Est_Démarrée), Machine-Fille (Démarrage : Machine_Mère_Démarrée)*)
- `behaviors` : représentent une suite de transitions à effectuer les unes à la suite des autres (exemple : *Déploiement : Installation, Configuration, Démarrage*)

Pour un exemple d'un Composant instancié, voir les Figures 3 à 6. Pour la légende, se réferrer à la Figure 2.

Les dépendances peuvent être liées via l'« Assemblage » avec d'autres dépendances venant d'autres composants (il y a des dépendances d'utilisation/d'usage et de provision, les premières doivent être connectées aux secondes pour être satisfaites, les secondes ne sont pas obligatoirement connectées pour être satisfaites). L'« Assemblage » est la mise en relation de différents Composants permettant un ordonnancement automatique des transitions. Une fois assemblés et un « Comportement » demandé, les composants se lancent dès que possible en respectant les dépendances. Les dépendances non satisfaites provoquent une attente passive [18].

4 Une extension à l'API de Concerto : OrchestrAPI

4.1 Introduction

Concerto est un outil qui permet de faire de l'orchestration, mais il ne propose pas de solution pour déployer les fichiers nécessaires au fonctionnement du (ou des) logiciel importé, ni pour permettre le déploiement à distance d'un logiciel.

OrchestrAPI [19] permet d'étendre les fonctionnalitées de Concerto, le tout en laissant une interface plus simple à programmer et des interactions intégrées avec le système, notament la gestion/déploiement de fichiers.

15. `https://corba.org/`
16. `https://en.wikipedia.org/wiki/Transition_system`
17. `https://en.wikipedia.org/wiki/Petri_net`
18. Une forme d'attente où le Système d'Exploitation signale lorsque l'attente prend fin, celle-ci ne prend pas de temps processeur.
19. `https://gitlab.inria.fr/skanapa/orchestrapi`

4.2 Description d'OrchestrAPI

Voici une liste des méthodes disponibles dans l'API, celles-ci étendent les fonctionnalités de l'API de Concerto :

- `get_path()` : retourne le répertoire de travail de l'API
- `get_hostname()` : retourne le nom de l'hôte sur laquelle l'API est appelée
- `write_script(var_export, cmdline, filename, gen_pid_file)` : permet d'écrire la commande passée en paramètre dans un script pour une exécution ultérieure
- `async_files_transfer(source, destination, rsync_queue)` : permet d'ajouter la source et la destination (locale ou distante) d'un fichier à une queue
- `flush_files_transfer()` : permet de vider la queue en transférant tous les fichiers qu'elle contient
- `transfer_files(source, destination)` : permet de transférer directement des fichiers
- `run_cmd(cmdline)` : permet de lancer une simple commande en local ou sur une machine distante
- `kill_scripted_cmd(scriptname)` : est un « wrapper » de la commande Unix « kill [20] »

En plus de ces méthodes, OrchestrAPI étant une Classe, elle posséde des données membres permettant d'articuler ses méthodes autour de l'usage voulu par l'utilisateur. Des exemples sont fournis dans la section 5.4.

5 Symphony in Diet : une intégration de Diet dans Concerto

5.1 Introduction

Symphony in Diet [21] est l'intégration de Diet dans Concerto que j'ai développée en reprenant les travaux déjà effectués avec « Concerto-Diet », « Mad-Remote » et « Mad-Diet », qui sont des anciennes intégrations de Diet dans Concerto ou une extension de l'API de Concerto. Ces projets datent de plusieurs années, et ne permettaient respectivement pas de faire une reconfiguration dynamique de Diet, ni d'étendre et de simplifier l'API de Concerto. Ceux-ci ayant été développés en Python, Symphony in Diet fut développé de même.

La première étape de conception a été de revoir les étapes et transitions des composants de Diet de manière à prendre en compte une reconfiguration dynamique, *i.e.,* modifier la configuration de l'arborescence de Diet sans avoir à redémarrer toute une infrastructure Diet. Pour ce faire, chaque composant a eu sa propre implémentation dans Symphony in Diet.

Symphony in Diet est une intégration des différents agents de Diet ainsi que d'un Naming Service dans Concerto, il ne permet pas un assemblage automatique des composants.

5.2 Description des composants

Chaque Agent de Diet ainsi que le Naming Service a son composant dans Symphony in Diet. Chaque composant différe des autres car les Agents Diet n'ont pas les mêmes étapes de fonctionnements, par exemple (*cf* Figure 3 à Figure 6) :

20. Commande qui permet d'envoyer un signal à un processus, notament pour l'arrêter.
21. `https://gitlab.inria.fr/skanapa/symphony-in-diet`

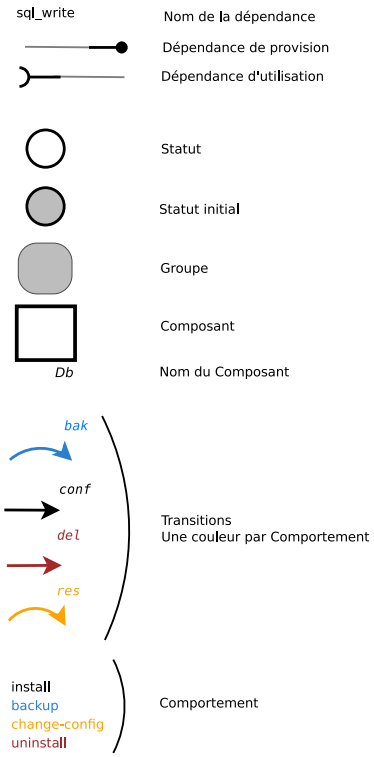

FIGURE 2 – Légende des schémas

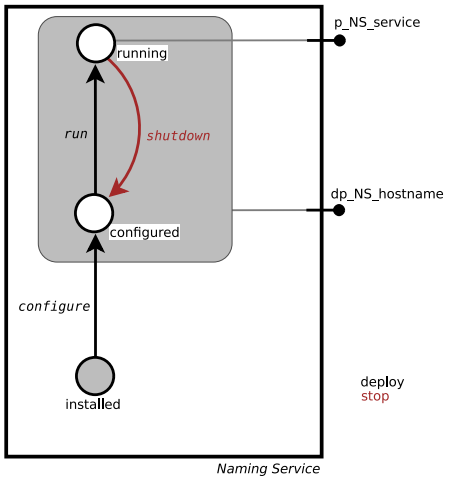

FIGURE 3 – Intégration du Naming Service en un composant Concerto

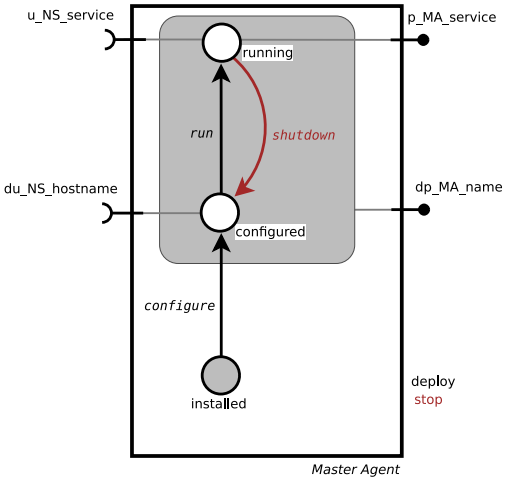

FIGURE 4 – Intégration du Master Agent en un composant Concerto

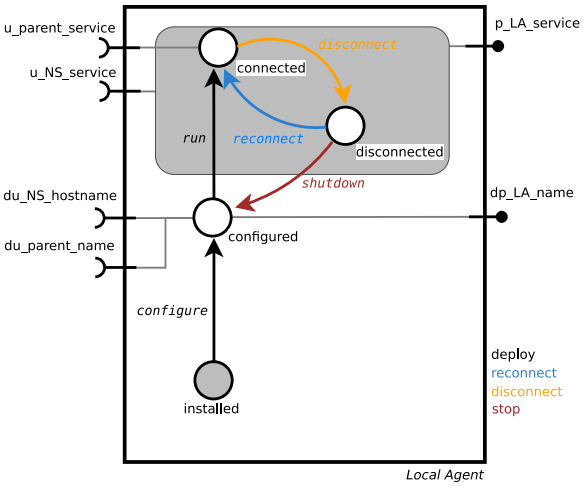

FIGURE 5 – Intégration du Local Agent en un composant Concerto

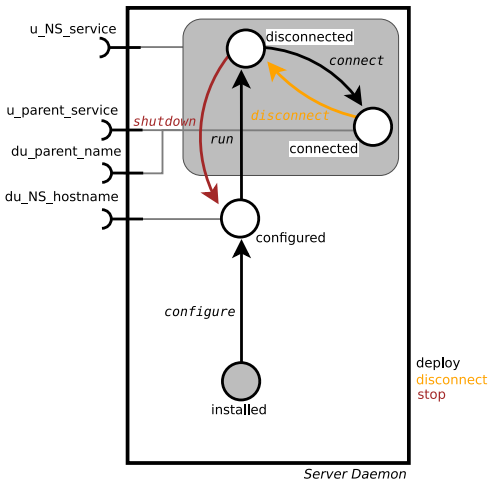

FIGURE 6 – Intégration du Server Daemon en un composant Concerto

— Dans le schéma du composant Naming Service que l'on peut voir dans la Figure 3, figurent 3 `places`, 3 `transitions`, 2 `dependencies` de provisions, ainsi que l'union des `places` « running » et « configured » qui forment un `group`.

— Le composant Master Agent, Figure 4 est sensiblement le même que le composant Naming Service, les différences étant les binaires exécutés et 2 `dependencies` d'usage supplémentaires pour le composant Master Agent.

— Le composant Local Agent, Figure 5, comporte 4 `places`, 5 `transitions`, 6 `dependencies` dont 4 d'usage et 2 de provision, ainsi que l'union des `places` « connected » et « disconnected » qui forment un `group`.

— Le composant Server Daemon, Figure 6, est sensiblement le même que le composant Local Agent, les différences étant les binaires exécutés et 2 `dependencies` de provisions en moins pour le composant Server Daemon.

Les ports « `dp_NS_hostname` » et « `p_NS_service` » appartenant au Naming Service permettent dans le cas du premier de fournir le nom de la machine sur laquelle le Naming Service se trouve et dans le cas du second si celui-ci est bien en cours d'éxécution.

Différents ports communs peuvent être identifiés : les ports « `du_NS_hostname` » et « `u_NS_service` » communs à tout les agents DIET se connectent respectivement aux ports « `dp_NS_hostname` » et « `p_NS_service` » du Naming Service. Le premier port permet aux agents de récupérer le nom de la machine sur laquelle se connecter, et le second recupère le statut du Naming Service.

De manière similaire, les ports « `du_parent_name` » et « `u_parent_service` » permettent à un agent de récuperer le nom et le statut de son parent dans la hierarchie DIET, pour permettre de s'y connecter lorsque c'est possible.

Les ports « `dp_MA_name` », « `p_MA_service` », « `dp_LA_name` » et « `p_LA_service` », qui appartiennent respectivement aux agents Master Agent et Local Agent, ont un rôle similaire à ceux du Naming Service, la différence étant que ceux-ci sont spécifiques à ces agents.

Chaque composant est une classe qui hérite de la classe Composant de Concerto (via une classe intermédiaire qui contient les champs de données membres communes à tous les Agents DIET).

Ensuite, OrchestrAPI est utilisée pour permettre la synchronisation des fichiers nécessaires pour le fonctionnement de DIET : le fichier de configuration, puis les scripts nécessaires pour l'éxécution à distance des commandes.

5.3 Démonstration de Symphony in DIET

L'exemple fourni par Symphony in DIET permet de faire un déploiement et une reconfiguration dynamique. On peut voir dans les Figures 7 et 8 ce que l'exemple déloie :

— Un Naming Service
— Un Master Agent
— Deux Local Agent
— Deux Server Daemon

Chaque Server Daemon est connecté à un Local Agent, le Local Agent 2 est connecté au Local Agent 1, et celui-ci est connecté au Master Agent (Figures 7 et 8).

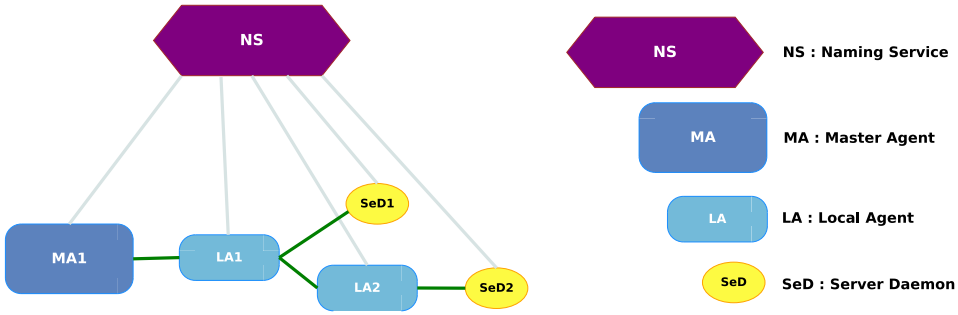

FIGURE 7 – Configuration initiale

Lors de ce déploiement, chaque composant de Symphony in DIET est déployé de manière concurrente, leurs ordres de déploiement peuvent donc changer à chaque nouvelle itération du déploiment. Les composants sont déployés dès que possible en fonction de leurs dépendances.

Voici en exemple un déploiement possible par étape où les sous étapes peuvent être dans un ordre arbitraire :

1. Le composant NS transitionne dans le statut `configured`.
2. Les composants MA1, SeD1 et SeD2 transitionnent dans le statut `configured`, le composant NS transitionne dans le statut `running`.
3. Le LA1 transitionne dans le statut `configured`, le composant MA1 transitionne dans le statut `running`, les composants SeD1 et SeD2 transitionnent dans le statut `disconnected`.
4. Le composant LA1 transitionne dans le statut `connected`, le composant LA2 transtionne dans le statut `configured`.
5. Les composant SeD1 et LA2 transitionnent dans le statut `connected`.
6. Le composant SeD2 transtionne dans le statut `connected`.

Se lance ensuite une reconfiguration dynamique pour déconnecter et éteindre le Local Agent 1, connectant directement le Local Agent 2 et le Server Daemon 1 au Master Agent (*cf* Figures 9 et 10).

Lors de la reconfiguration dynamique, les agents LA1, LA2, SeD1 et SeD2, transitionnent tous dans le statut `disconnected` et les ports « `du_parent_name` » et « `u_parent_service` » des composants SeD1, et LA2 sont déconnectés du composant LA1, puis connectés aux ports « `dp_MA_name` », « `p_MA_service` » du composant MA1. Ensuite, le composant LA1 s'éteint, il transitionne dans le statut `configured`, les composants LA2, SeD1 et SeD2 transitionnent dans le statut `connected`.

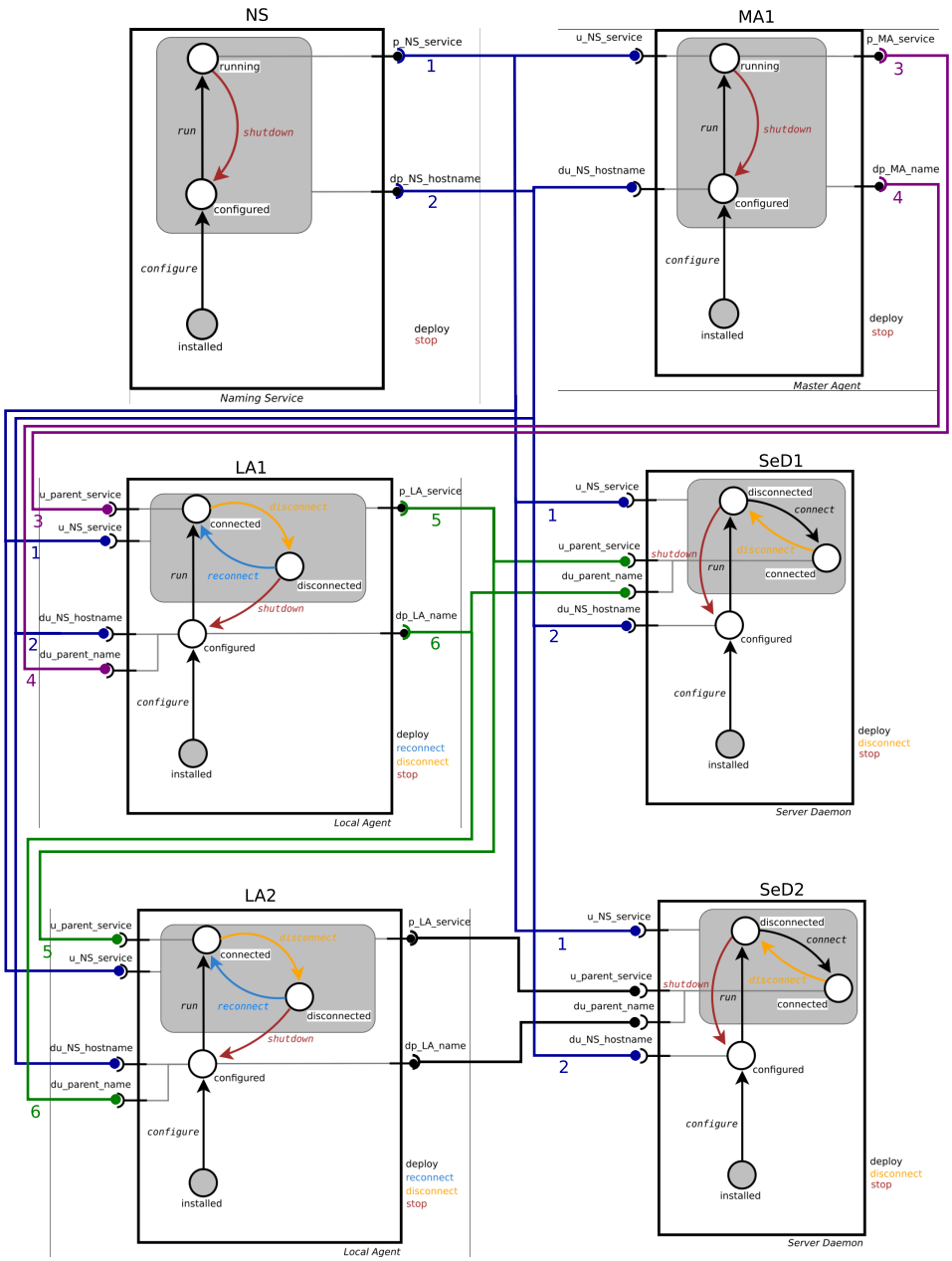

FIGURE 8 – Schéma configuration assemblage initial

5.3.1 Déploiement local

Dans le cadre du developpement de Symphony in DIET et de OrchestrAPI, j'ai utilisé une image docker Debian 10 Buster [22] adaptée avec les outils nécessaires, et avec un volume qui permet d'experimenter et de modifier le code rapidement. L'exemple fourni dans l'archive de Symphony in DIET n'a pas besoin d'être modifié pour du déploiement local.

22. `https://www.debian.org/`

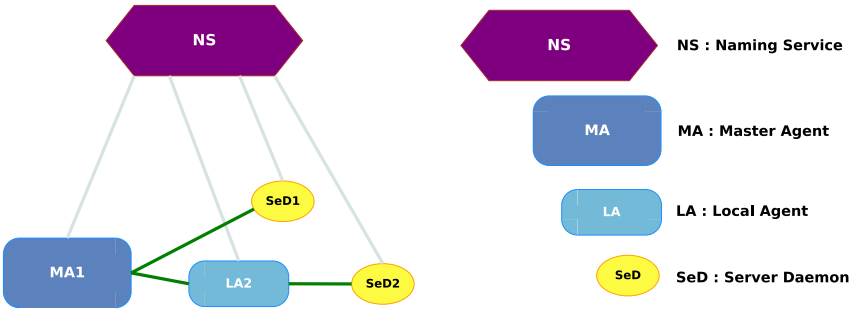

FIGURE 9 – Résultat de la reconfiguration dynamique

5.3.2 Déploiement distant

Pour des essais de déploiement sur des machines distantes, il faut modifier l'exemple fourni et ajouter le bon nom de machine hôte pour chaque composant de DIET. Le reste est géré par OrchestrAPI.

5.4 Apports d'OrchestrAPI à Concerto

Dans cette partie, nous allons analyser les apports d'OrchestrAPI à l'API déjà existante de Concerto.

5.4.1 Uniformisation du déploiement local et distant

L'utilisation d'OrchestrAPI permet d'uniformiser la gestion des composants. Celle-ci n'utilise qu'une variable (donnée membre de la classe d'OrchestrAPI) pour s'adapter entre un déploiement local ou distant. Cette variable contient le nom d'hôte de la machine sur laquelle le Composant est déployé par OrchestrAPI. Par défault, le Composant se déploie sur la machine locale.

5.4.2 Déploiement des fichiers nécessaires à un composant

Pour permettre la gestion de fichiers de configuration ainsi que les scripts d'exécution, OrchestrAPI fournit des méthodes de transfert de fichiers. Parmis ces méthodes, il y a la constitution d'une liste de fichiers à transférer qui peuvent être transférés au même moment dans les transitions du Composant. Une fois les transferts effectués, la liste est vidée. Cela permet de limiter les différentes interruptions et de ne faire qu'un seul transfert permettant d'économiser du temps sur la latence car une seule connexion est utilisée (et nécessaire).

5.4.3 Gestion de l'exécution de commandes liées aux transitions d'un composant

OrchestrAPI permet d'exécuter des commandes à l'aide d'un script qui permet de conserver les journaux de sorties standard et d'erreur, ainsi que de conserver le PID du processus. Ce script permet aussi d'exécuter la commande en tâche de fond et de la détacher du processus qui l'a invoquée, ce qui lors d'une exécution au travers d'une connexion SSH, permet de garder la commande active même si la connexion est interrompue.

Pour cela le PID du processus est conservé dans un fichier dans le répertoire de travail du programme : ce qui autorise l'arrêt d'un processus sur une machine distante, même après déconnexion du service SSH, ou sur une machine locale sans avoir à connaître le PID du processus.

6 Conclusion

J'ai effectué ce stage au Laboratoire de l'Informatique du Parallélisme pendant 2 mois au cours duquel mon travail a consisté à développer Symphony in DIET et OrchestrAPI. Étant donné que le stage s'est déroulé 100% en distanciel, les réunions hebdomadaires pour suivre mes avancées se sont faites via le service de

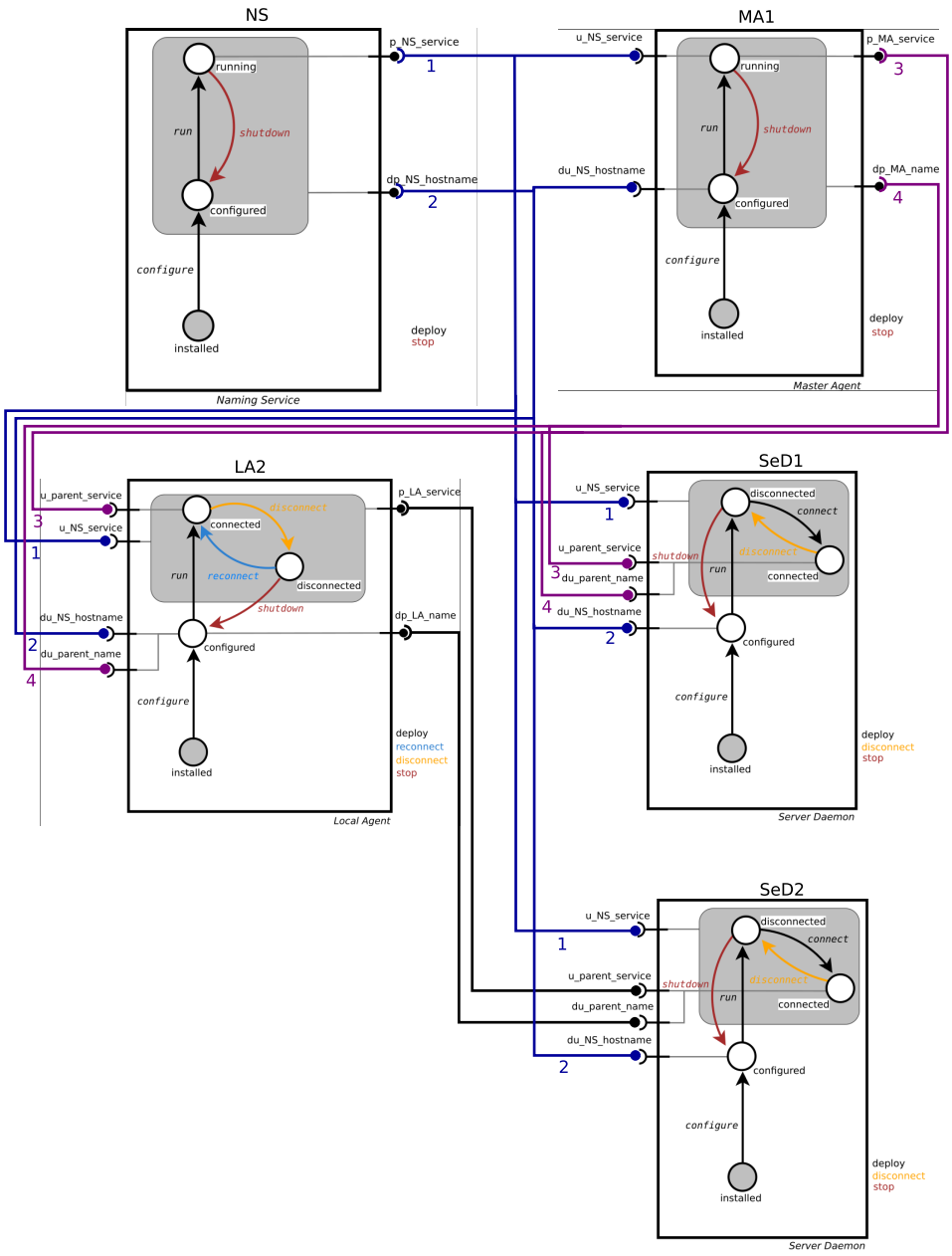

FIGURE 10 – Schéma reconfiguration assemblage dynamique

visio-conférence « Jitsi [23] », et pour toute demande d'aide en dehors de ces réunions, nous communiquions par e-mail et via la plateforme « Mattermost [24] ».

Le développement de Symphony in DIET et d'OrchestrAPI a permis la simplification du déploiement et de la reconfiguration de DIET en simplifiant le déploiement des composants, et le transfert de fichiers, et en rendant possible l'automatisation d'un tel procédé.

23. `https://jitsi.org/`
24. `https://mattermost.com/`

Bien que le développement d'OrchestrAPI et de Symphony in DIET fut couronné de succès, certaines fonctionnalités supplémentaires pourraient être ajoutées. L'assemblage automatique des composants en est une, mais on pourrait aussi imaginer utiliser l'API pour surveiller le fonctionnement des processus lancés, ou créer un service de transfert centralisé avec tous les composants sur la même machine pour permettre une synchronisation des transferts encore plus large, *etc.*

7 Annexes

7.1 Agenda et compétences acquises et/ou renforcées

Lors du premier mois du stage, je me suis familiarisé avec DIET et Concerto en lisant la documentation et en experimentant avec des cas croissants en complexité. Pendant le second mois, j'ai développé Symphony in DIET et OrchestrAPI, puis rédigé ce rapport.

Durant ce stage, j'ai pu mettre en pratique mes connaissances dans des langages de script tels que Bash pour expérimenter avec le déploiment de DIET, et Python lors de mon travail d'intégration de DIET dans Concerto. J'ai pu renforcer ces mêmes connaissances en partant du travail déjà effectué par d'anciens stagiaires ou par Christian Pérez, lors de l'utilisation de Concerto pour DIET. De plus, j'ai appris à utiliser les différents outils mis à ma disposition que sont LaTeX pour la rédaction du rapport et MarkDown pour la documentation. J'ai aussi utilisé GIT [25] afin de partager et sauvegarder les avancées de mon travail et les résultats obtenus [26]. Afin d'expérimenter rapidement sur ma machine, j'ai utilisé Docker [27]. Pour la réalisation de tests sur plusieurs machines ou des essais de déploiement à distance, j'ai eu accès à Grid'5000, ce qui m'a permi de renforcer mes connaissances et compétences avec SSH et en Administration Système. De plus, j'ai pu renforcer mes compétences liées à la création et développement de projet, ainsi qu'à la communication autour d'un projet.

7.2 Impressions personnelles

En dépit du fait que le stage se soit passé entièrement en distanciel, j'ai beaucoup apprécié et appris énormément. C'est une expérience positive et enrichissante qui me permettra, je l'espère, de mieux réussir mes études à venir. Cela m'a aussi permi d'affiner mon projet professionnel : ce stage m'a permis de confirmer mon envie de poursuivre mes études ou une carrière professionnelle dans l'Administration Système ou le développement de logiciels.

25. `https://git-scm.com/`
26. Le code source OrchestrAPI est disponible dans la Section 4 et celui de Symphony in DIET dans la Section 5.
27. `https://www.docker.com`

Références

[1] E. Caron, F. Desprez.
DIET : A Scalable Toolbox to Build Network Enabled Servers on the Grid. *HAL Id : inria-00070406,* https://hal.inria.fr/inria-00070406, Article https://hal.inria.fr/hal-01429867v1

[2] M. Chardet.
Reconciling Parallelism Expressivity and Separation of Concerns in Reconfiguration of Distributed Systems. *Thèse N° : 2020IMTA0194,* https://graal.ens-lyon.fr/~cperez/2020IMTA0194_Chardet-Maverick.pdf, Article https://hal.inria.fr/tel-03230476v1

[3] M. Chardet, H. Coullon, D. Pertin, C. Pérez.
Madeus : A formal deployment model. *HAL Id : hal-01858150,* https://hal.inria.fr/hal-01858150