Définition d'un modèle d'execution fluxionnel *

[Extended Abstract] †

Etienne Brodu‡
addr line 1
addr line 2
addr line 3
etienne.brodu@insa-lyon.fr

Stéphane Frénot§
addr line 1
addr line 2
addr line 3
stephane.frenot@insa-lyon.fr

Fabien Cellier¶
addr line 1
addr line 2
addr line 3
fabien.cellier@worldline.com

Frédéric Oblé
addr line 1
addr line 2
addr line 3
frederic.oble@worldline.com

ABSTRACT

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

ACM proceedings, LaTeX, text tagging

1. INTRODUCTION

La croissance des plateformes du Web est dû partiellement à la capacité d'Internet à favoriser le développement de services avec une mise en production minimale très rapide. En quelques heures, il est possible de mettre en ligne un produit fonctionnel afin d'accueillir une première audience. "Release early, release often" est souvent entendu parmi les communautés open source pour capter rapidement une communauté d'utilisateurs.

Si le service répond correctement aux attentes de l'audience, celle-ci va très probablement grossir au fur et à mesure que le service gagne en popularité. Afin de pouvoir faire face à cette croissance, la quantité de ressources utilisé par le service augmente en conséquence, et il arrive un moment dans le développement du produit où la taille des données à traiter et la quantité de ressources nécessaires, imposent l'utilisation d'un modèle de traitement plus efficace. Ces modèles plus efficaces passent par une segmentation des échanges entre fonctions, en utilisant différents paradigmes de communication comme les approches *three-tiers*, les événements, les messages ou les flux. [bilbio] Une fois segmenté, les différentes parties communiquent entre elles par un principe de messagerie le plus souvent asynchrone. De très nombreux outils ont été définis qui permettent d'exprimer ces différentes parties, leurs interactions, et de prendre en charge l'acheminement des messages [Storm, MillWheel, Spark, TimeStream ...]. Cependant, ces outils utilisent des interfaces ou des langages particuliers. Il est nécessaire de former les équipes de développement à l'utilisation de ces nouveaux outils, d'engager des experts familiers avec ces outils et de réécrire le service initial en utilisant ces nouveaux outils. Cette nouvelle architecture est globalement moins souple et moins propice aux changements rapides. // TODO à vérifier et documenter [biblio] Ce changements de paradigmes de développement représente une prise de risque dans la poursuite du projet car ces outils sortent du cadre grand public suffisamment accessible pour favoriser l'émergence spontanée de nouveaux services.

Nous proposons un outil permettant d'éviter de forcer ce changement de paradigme en proposant une vision segmentée de programme 'standard'. Nous visons des applications Web dont les sollicitations proviennent des flux de requêtes utilisateurs et dont le développement initial est réalisé selon une approche web 'classique' (serveur web / traitement applicatif / data). Nous pensons qu'il est possible d'analyser cette classe d'applications dès les premières étapes d'exploi-

*(Does NOT produce the permission block, copyright information nor page numbering). For use with ACM_PROC_ARTICLE-SP.CLS. Supported by ACM.

†A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2$_\epsilon$ and BibTeX* at `www.acm.org/eaddress.htm`

‡title notes

§title notes

¶title notes

tations afin de les re-exprimer plus ou moins concrètement sous la forme de flux d'échange.

Nous supposons que les applications serveur sont développés dans un langage dynamique comme Javascript, et nous proposons un outil capable d'identifier les flux internes, de définir des unités de traitement de ces flux, et de pouvoir gérer de manière dynamique ces unités. L'outil identifie ces unités sans être intrusif dans le code existant mais en proposant une sur-expression du programme initial en utilisant le paradigme de fluxion que nous allons définir et qui servira au cœur de notre proposition.

TODO La section 2 présente le principe de fluxion en le positionnant par rapport à l'existant. La section 3 ...

2. MODÈLE D'EXÉCUTION FLUXIONNEL

2.1 Fluxions

Le principe de modèle d'exécution fluxionnel est d'identifier des unités d'exécution autonomes fondés sur des flux. Une unité est autonome quand elle peut être déplacé dynamiquement d'environnement d'exécution pendant sont activité. Déplacer une unité d'exécution nécessite de déplacer avec le code d'exécution son contexte courant. C'est à dire l'ensemble des variables d'état et de mémoire provenant des précédentes exécutions de l'unité. Dans notre approche, ce contexte est encapsulé sous forme de flux pour être manipulé par l'unité d'exécution. (Pour notre approche nous "transférons" ce contexte dans un flux propre à l'unité déplacé.) Ainsi, déplacer une telle unité consiste à déplacer le code fonctionnel vers une nouvelle destination, puis à rediriger les flux d'entrées et de sorties en conséquence. Une telle unité peut alors être déplacé de nœud en nœud sans y être supprimée (attachée ?). Seul les flux ont besoin d'être redirigés en conséquence.

Nous avons appelé cette unité d'exécution autonome une fluxion. C'est à dire une fonction, au sens de la programmation fonctionnelle ne dépendant pour ses entrées et ne produisant sur ses sorties que des flux.

2.1.1 Persistance des états

Les fluxions n'ont pas d'état afin de pouvoir être transporté d'un nœud à l'autre en cours d'exécution. Comme de nombreuses applications reposent sur l'existence d'un tel état, nous l'avons transféré dans les flux d'entrées et de sortie des fluxions. Ainsi l'état suit la mobilité d'une fluxion lors de la redirection de ses flux. // TODO est-ce qu'on laisse cette explication sur l'état volontairement plus simpliste, ou est-ce qu'on explique le détail technique de 'apply' ? L'état d'une fluxion est ainsi modélisé par un flux d'entré supplémentaire appelé scope. Lorsqu'une fluxion a fini de s'exécuter elle envoie dans son flux scope les données qu'elle souhaite persister, et qui seront re-agregé à son prochain appel.

2.1.2 Adressage et mobilité des fluxions

Le système de messagerie distribue les messages aux fluxions en se basant sur l'adresse de destination du message. Chaque fluxion est associé à une adresse permettant au système de messagerie de l'identifier de manière unique.

Lorsqu'une fluxion se déplace d'un nœud à l'autre, son adresse se déplace aussi, et ce avec les flux associé à cette adresse. Les deux nœuds communiquent ce déplacement entre eux pour le valider, et auprès des autres nœuds pour les en informer. De cette manière chacun des nœuds connaît l'emplacement de chacune des fluxions sur les autres nœuds. Ainsi le système de messagerie composé de l'ensemble de ces nœuds peut acheminer n'importe quel message sur l'ensemble du système.

2.1.3 entrées sorties

2.2 Cycle de vie

Une application fluxionnelle est composée d'un enchaînement de fluxions. Chaque fluxion présente le même comportement :

- elle est invoqué par un système de messagerie à la réception d'un message,
- effectue des opérations à partir du message reçu,
- modifie son état interne déporté dans le système de messagerie sous forme de scope,
- puis renvoie un message.

Dans notre approche, un message est une structure de couples clé / valeur contenant deux couples : le nom de la fluxion à invoquer 'addr' et le corps du message 'body'. Le système de messagerie impose deux fonctions :

- une fonction d'enregistrement register (<nom>, <fn>, <conte
  et
- une fonction de déclenchement de la chaîne de traitement start(<nom>,<param>).

Les données et la logique d'une application étant cloisonné de manière distincte pendant l'exécution, il est possible de mettre à jour une fluxion en la remplaçant dans le système, sans impacter l'exécution de l'application.

(Comme les données propres au fonctionnement d'une application sont stockées dans le système de messagerie et que les fluxions ne possèdent pas de données propres, l'installation d'une nouvelle version se fait en enregistrant en cours d'exécution les nouvelles fluxions.)

De plus la relocalisation d'une fluxion se fait de manière transparente par l'application, par le système de messagerie qui connaît la localisation exacte des fluxions. Nous y reviendrons plus tard, mais la relocalisation d'une fluxion consiste à déplacer les fluxions sur un nouveau nœud et de rediriger les messages en conséquences. Comme les types de messages, leur débit et le contexte propre d'une fluxion sont connus, on connaît a priori le coût de migration à chaud d'une fluxion.

2.3 Architecture Web

Le système fluxionnel ne manipule que des fluxion par l'intermédiaire d'un système de messagerie. Afin de pouvoir interagir avec le monde extérieur, il faut définir des interfaces de bordure. Notre approche repose sur une espérance de gain technologique principalement sur les architectures Web. Le premier point d'entré visé est ...(?) les interfaces REST. Le schema suivant présente la séparation du système en fonctions indépendantes.

TODO schema

Le système Web est donc le déclencheur d'une chaîne de traitement de requêtes à chaque nouvelle requête d'un utili-

sateur un appel à la fonction start ('/', <param>) est réalisé dans le système de messagerie. Au démarrage du système Web, deux demi-fluxions sont lancées. La demi-fluxion 'in' n'est pas enregistré dans le système de messagerie. Elle prend les paramètres de la requête Web, place l'identifiant de la connexion client dans le contexte de la demi-fluxion de sortie, puis lance le traitement de la requête en invoquant la fonction 'start' du système de messagerie.

APPENDIX