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

Comme pour les autres systèmes scalables, le principe de modèle d'exécution fluxionnel est d'identifier des unités d'exécution autonomes fondées exclusivement sur des flux comme paramètre d'entrée et de sortie des unités. Une unité est autonome quand elle peut être déplacée dynamiquement d'environnement d'exécution au cours de son activité. Déplacer une unité nécessite de déplacer avec son code d'exécution son contexte d'exécution courant, c'est à dire l'ensemble des variables de mémoire dont dépend l'unité. Dans notre approche, le contexte est capturé dans flux spécifique réinjecté dans l'unité à chaque d'exécution. 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. L'état, les paramètres d'appel et les paramètres de retour étant capturé dans des flux, on a une estimation instantané du coup de déplacement de la fonction, proportionnel au débit des flux concernés.

Nous avons appelé cette unité d'exécution autonome une fluxion. C'est à dire une fonction, au sens de la programmation fonctionnelle, dépendant exclusivement de flux de données.

2.2 Plateforme de fluxions

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

- elle est invoquée par un système de messagerie,
- elle effectue des opérations à partir du message reçu,
- elle modifie son état interne rendu persistant dans le système de messagerie
- puis elle retourne un ensemble de messages à destination d'autres fluxions.

Dans notre approche, un message est une structure clé/valeur contenant deux couples : le nom de la fluxion à invoquer *name* et le corps du message *body*, contenant l'ensemble clé/-valeur des attributs nécessaires à l'exécution de la fonction.

Le système de messagerie repose sur uniquement deux fonctions publiques :

- l'enregistrement de fluxion `register(<nom>, <fn>, <contexte>)`
- le démarrage d'une chaîne de traitements `start(<nom>,<param>)`

Les données et la logique d'une application sont cloisonnés à l'exécution, il est possible de mettre à jour une fluxion en la remplaçant dans le système, sans impacter l'exécution de l'application. De plus déplacer une fluxion ne nécessite pas de la supprimer de son nœud initial, car seul la réception de message déclenche une exécution.

La relocalisation d'une fluxion se fait de manière transparente par l'application, par le système de messagerie qui connaît la localisation des fluxions.

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 l'intégration des interfaces REST. Le schéma **??** présente les éléments d'un système Web fluxionnel.

TODO schema

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