Applications collaboratives

“Reusing single-user applications to create multi-user Internet applications”
2001

auteurs: Stephan Lukosch & Joerg Roth

mercredi 2 février 2004,
Christophe GRAVIER

Plan de la présentation

1- Introduction

2- l’être ou le paraître
3- proposition de l’article
4- transformons !
5- scénarii d’utilisation

6- conclusion

Cadre de la proposition

Proposer des outils et une méthodologie pour réutiliser les applications mono-utilisateur et pour les transformer en applications collaboratives.

Enjeux double:

- exploiter l'existant (99.99% des applications)
- ne pas faire de développement spécifique pour se concentrer sur le métier

2 approches possibles

l'être ou le paraître

- Etre *(collaboration-aware)*
  - développement d'une application spécifique pour garantir une véritable réponse aux besoins collaboratifs
- Paraître: *(collaboration-transparent)*
  - exécuter une application mono-utilisateur dans un environnement partagé

« être » *collaboration-aware*

- Forces
  - répond précisément aux besoins fonctionnels
  - degré de conscience de groupe approprié (*« group-awareness »*)
  - gestion appropriées des données distribuées (intégrité)
- Faiblesses
  - coût (financier et/ou humain)
  - maintenance
  - difficulté de réutiliser les briques logicielles

« paraître » *collaboration-transparent*

- Forces
  - idée séduisante et techniquement triviale
  - coût
  - maintenance
  - facilité de migration (même application)
- Faiblesses
  - est-ce une collaboration ou une compétition ?
    - baisse de la conscience de groupe (ex: scroll-wars)
  - gestion des données distribuées enfouie dans l'application mono-utilisateur
    - Intégrité ? Disponibilité ? Non répudiation ? Propriété ?

alors, être ou paraître ?

- proposition des auteurs:
  - transformer les applications mono-utilisateur en multi-utilisateurs pour:
    - prendre le meilleur des 2 mondes
    - en évitant les écueils de chacune des approches
- moyens proposés:
  - un contexte d'exécution (« *runtime system* »)
  - une API pour automatiser la transformation
  - une méthodologie pour effectuer la transformation
- efficacité d'une surcouche applicative propriétaire dans un monde où le temps réel est facteur critique ?

Panorama de l'existant

(autres auteurs)

- DistEdit [1]
  - outil de transformation d'éditeurs de texte
  - Légitimité de la notion de « collaboration »: verrou sur document
- DistView [2]
  - distribution de la fenêtre via un serveur central
    - L'unité devient la fenêtre
    - Faiblesse de l'architecture et grain toujours insuffisant
- Suite [3]
  - ≈ proposition d'implémentation de DistView
    - Avec les mêmes contraintes

[1] M.J. Knister & A. Prakash. « *DistEdit: a distributed toolkit for supporting multiple group editors* » (90)

[2] A. Prakash & H.S. Shim. « *DistView: Support for Building Efficient Collaborative Applications using Replicated Objects* ». (94)

[3] P. Dewan & R. Choudhary. « *A High-Level and Flexible Framework for Implementing Multiuser Interfaces* » (92)

Méthodologie 3 étapes pour transformer

- Intégrer l'application dans le « runtime system » de la proposition
- Organiser la gestion des données partagées distribuées
- Ajouter les services pour la conscience de groupe

(« group awareness »)

Intégration à la plateforme

- Lancer l'application via le panneau des applications du « runtime system »
- Via l'interface de l'application:
  - déclarer les sessions de travail collaboratives
    - problématique de gestion de versions
    - découverte des acteurs potentiels
  - Rejoindre, via l'interface, les sessions ouvertes
- note: Très (trop?) semblable à JWS de Sun ?
  - hormis la notion de session qui est gérée dans l'applicatif et non par le « runtime system»

Gérer les données partagées distribuées

- But:
  - ne pas enfouir la gestion des données dans l'application mono-utilisateur
  - Gérer l'intégrité et l'accès concurrentiel aux données
- Étapes:
  - Identifier les données susceptibles d'être partagées
    - Travail trivial si modèle Arch, MVC, PAC …
  - Utiliser un substitut offrant un point d'entré sur la donnée
    - Masquer le mécanisme de distribution de données dans le framework
  - Configurer le substitut
    - autoriser ou non l'accès concurrentiel aux méthodes d'un objet.

Et la conscience de groupe !

- Ajouter cette conscience de groupe via les widgets
  - Widgets de l'API de l'application mono-utilisateur
    - Ex: JButton en Java (non orienté collaboration)
  - Widgets dérivés des widgets usuels
    - Ex: « distributed mouse pointers » fournit dans la proposition.
  - Widgets créés par le développeur « transformateur »
    - objet graphique non standard développé pour l'application qui endosse l'aspect collaboratif.
    - Ex: une barre d'outils spécifique à l'application
- « Tracking windows »: voir ce que fait un autre acteur.

Cas d'utilisation de transformation

- Editeur de diagramme entités-relation
  - rendre l'applicatif collaboratif via la plateforme
  - 190 lignes de code sur les 1786 du programme
- Outil de feuille de calcul
  - programme de démonstration du JDK Sun
  - 121 lignes sur 997.

- Et si la complexité du code augmente ?
- Et si la « vétusté » de l'application augmente ?

Conclusion 1/2

- L'apport est double:
  - Idée de réutilisation des IHM
    - original/séduisant … et ambitieux
  - Couple outil + méthodologie
- Une plateforme opérationnelle
- Une couche d'abstraction du travail collaboratif pour le développeur
- illustration de l'implémentation par des cas d'utilisation.

Conclusion 2/2

Cependant:

- Une surcouche applicative (coût à l'exécution)
- originalité technique ? (assemblage de briques)
- Et si la complexité augmente ?
- Et si l'application n'est pas un cas d'école (comme la transformation sur la démo du JDK Sun) ?
- A partir de quel point est-il finalement plus intéressant d'investir dans un développement spécifique ?

Questions !