Capture And Replay

soutenance TER, 17/06/2004

Giraud Rémy, Modhafar Tarek

Plan

1. **Notre projet le « Capture And Replay »**
2. **A quoi ça pourrait servir ?**
3. **Où sont passés les événements sous *Windows* ?**
4. **Et sous *Eiffel*, des changements ?**
5. **Découpage, explications des tâches**
6. **Conclusion**

Plan

1. **Notre projet le « Capture And Replay »**
2. **A quoi ça pourrait servir ?**
3. **Où sont passés les événements sous *Windows* ?**
4. **Et sous *Eiffel*, des changements ?**
5. **Découpage, explications des tâches**
6. **Conclusion**

Projet Capture And Replay

But

Concevoir et implémenter un ensemble de fonctionnalités qui permet à une application graphique, écrit en *Eiffel*, de mémoriser les événements générés par l'utilisateur lors d'une première utilisation et de les rejouer à sa prochaine exécution.

Projet Capture And Replay

Support utilisé

- **Dédié aux applications graphiques *Eiffel.***
- **Système d’exploitation requis : *Windows.***

Projet Capture And Replay

Définitions

Capture :

- récupérer les données affichées à l'écran et les sauvegarder (caractéristiques d'une fenêtre ou la position d'un clic de souris, . . .).

Replay :

- rejouer les actions qui se passent à l'écran et qui ont été mémorisées lors de la capture.

Projet Capture And Replay

Comment le mettre en œuvre ?

Capturer les événements :

- Modifier les librairies graphiques de *Eiffel* (Instrumentation).
  - Pour capturer les événements.
  - Sauvegarde des éléments nécessaires pour rejouer les événements.
- Stocker sur le disque.
  - Les événements sous forme de fichier binaire.
  - Les informations supplémentaires.

Plan

1. Notre projet le « Capture And Replay »
2. **A quoi ça pourrait servir ?**
3. Où sont passés les événements sous *Windows* ?
4. Et sous *Eiffel*, des changements ?
5. Découpage, explications des tâches
6. Conclusion

A quoi ça peut servir ?

Fonctionnement / déroulement

Phase de Capture

- L'utilisateur lance son application
  - en utilisant les classes graphiques standard *Eiffel* (que nous avons modifiées).
- Des événements sont générés par l'utilisateur.
- Ceux-ci sont capturés.

Phase de rejoue

- L'utilisateur relance la même application.
- Les événements de l'utilisateur sont inhibés.
- Les événements capturés sont reproduits.

A quoi ça peut servir ?

Fonctionnement / conséquences

Conséquences

- L'application lancée est toujours celle de l'utilisateur.
- Le contexte d'exécution est préservé.

Exemple

- Un événement signal le clic sur un bouton.
- Celui-ci est capturé et reproduit.
- Et la fonction (de l'utilisateur) associée au bouton sera elle aussi reproduite.

A quoi ça peut servir ?

Utilité

Comprendre comment on est arrivé dans un état particulier (analyse d'erreur, historique de l'interaction).

Démonstration.

Mise en place de scénarios.

Plan

1. Notre projet le « Capture And Replay »
2. A quoi ça pourrait servir ?
3. **Où sont passés les événements sous *Windows* ?**
4. Et sous *Eiffel*, des changements ?
5. Découpage, explications des tâches
6. Conclusion

Où sont passés les événements sous *Windows* ?

Pourquoi s'intéresser à *Windows*

***Eiffel* inclut dans son code l'interfaçage avec d'autre langage.**

***Eiffel* s'appuie largement sur l'API de *Windows* pour gérer les événements d'une application.**

***Eiffel* a reproduit le même modèle que sous *Windows*.**

Comprendre comment *Eiffel* gère les événements revient à comprendre comment les événements sont gérés sous *Windows*.

Mêmes noms d'événements, mêmes traitements.

Où sont passés les événements sous *Windows* ?

Principe / window_procedure

Une window procédure est une fonction de traitement d'un événement.

Chaque élément graphique (*Widget*) a sa propre window_procédure.

Elle spécifie pour chaque événement reçu ce que le *Widget* doit faire.

Chaque *Widget* enregistre l'adresse de leur window_procedure au niveau de l'application.

Où sont passés les événements sous *Windows* ?

Principe / boucle de lecture événements

Chaque événement posté est mis dans une file d'attente (*PostMessage*).

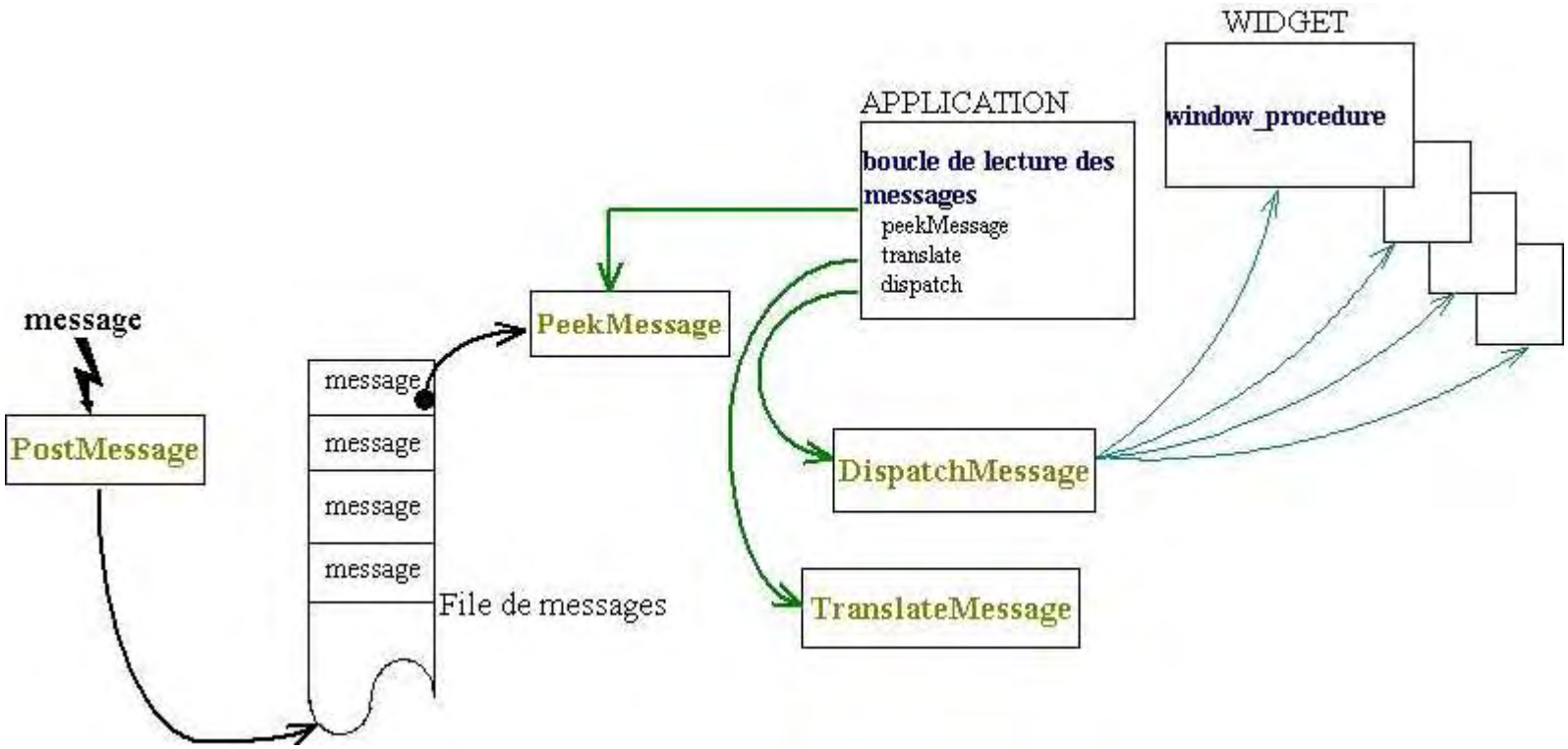
Au niveau de l'implémentation de l'application :

- Une boucle de lecture
  - lit l'événement (***PeekMessage***).
  - Convertit si besoin l'événement (***translateMessage***).
  - l'envoie à la bonne procédure qui traitera l'événement (***dispatchMessage***).

Chaque événement contient un pointeur sur le *Widget* concerné.

Où sont passés les événements sous *Windows* ?

Principe / Résumé



Plan

1. Notre projet le « Capture And Replay »
2. A quoi ça pourrait servir ?
3. Où sont passés les événements sous *Windows* ?
4. **Et sous *Eiffel*, des changements ?**
5. Découpage, explications des tâches
6. Conclusion

Et sous *Eiffel*, des changements ?

Changement / Similitude

Similitude

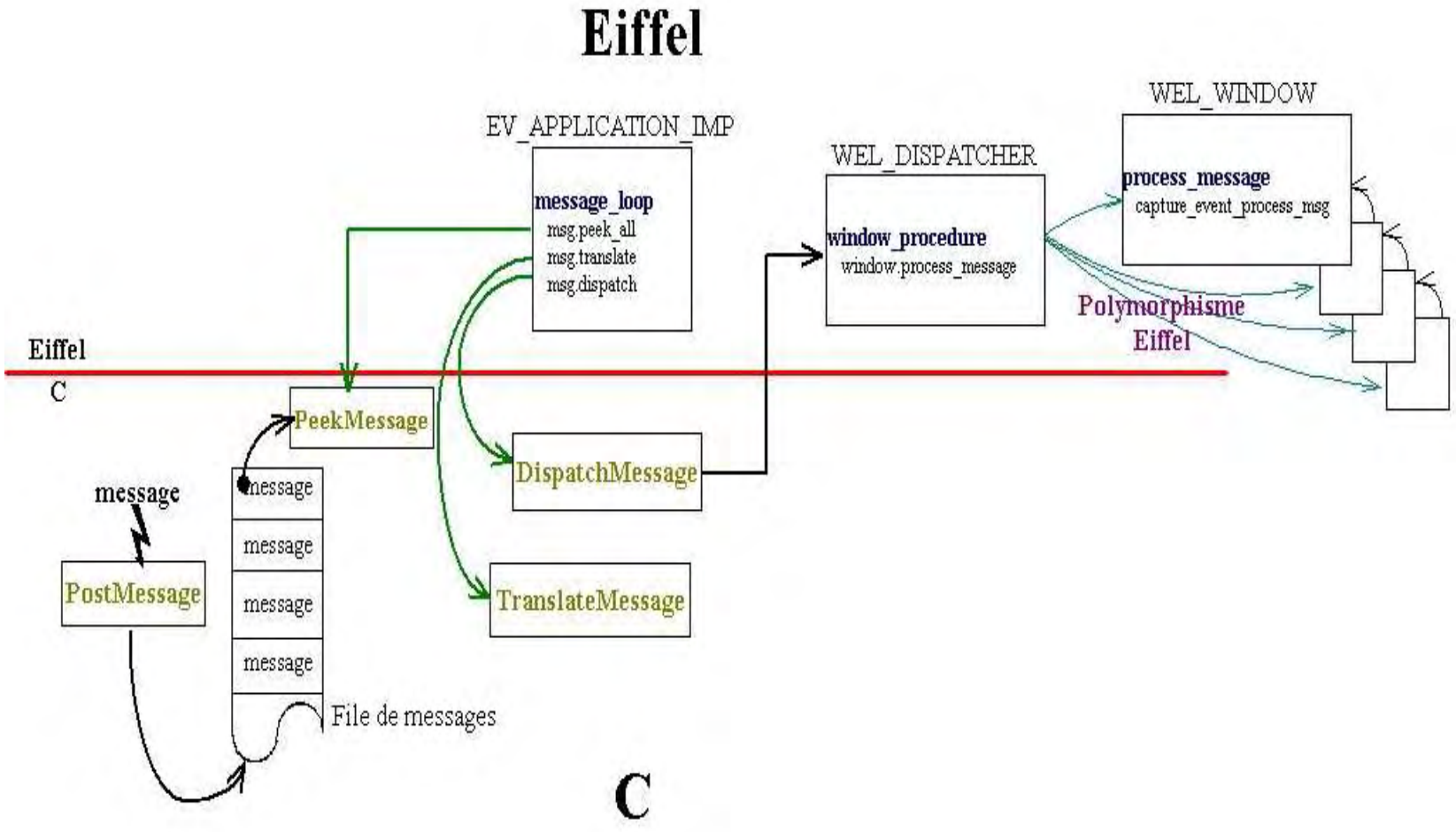
- *Eiffel* utilise le même procédé en appelant les mêmes fonctions C pour lire, transformer et dispatcher l'événement.

Changements

- Chaque *Widget* a sa propre *window_procedure* soit héritée soit redéfinie (*process_message*).
- L'événement est dispatché toujours à la même fonction.
- Par le biais du polymorphisme, la bonne w*indow_procedure* est appelée.

Et sous *Eiffel*, des changements ?

Résumé



Et sous *Eiffel*, des changements ?

Bilan

Phase d'étude des événements sous *Eiffel* indispensable à la compréhension de notre projet.

Difficile à comprendre car

- Informations éparpillées sur le site de msdn.
- Les appels aux fonctions C nous perdent dans les classes.
- Pas le code C associé mais juste la documentation sommaire sur le site.
- Obligation d'aller plus loin dans la compréhension des événements.
- Classes *Eiffel* pas très documentées.
- Ignorance totale de notre part de la gestion des événements.

Plan

1. Notre projet le « Capture And Replay »
2. A quoi ça pourrait servir ?
3. Où sont passés les événements sous *Windows* ?
4. Et sous *Eiffel*, des changements ?
5. **Découpage, explications des tâches**
6. Conclusion

Découpage et explication des tâches

Découpage

Un seul objectif :

- obtenir un Capture And Replay qui fonctionne bien
  - sans erreur.
  - ni bug.
  - qui rejoue les principaux événements possibles.

Liberté totale pour y arriver.

Découpage et explication des tâches

Découpage

Rémy	Tarek
Intégration des fichiers	Recherches msdn
Recherches msdn	Recherches *Eiffel*
Recherches *Eiffel*	Tests et débugages
Temporisation	Process_message
Script	
Thread	
Tests et débugages	
Process_message	

Découpage et explication des tâches

Tâches Rémy / Script d'installation

Réalisation d'un script écrit en C pour

- Permettre d'installer les fichiers modifiés dans l'environnement.
- utiliser un fichier de propriétés pour trouver les fichiers à installer.
- Permettre aussi de sauvegarder les fichiers.

Difficultés

- Beaucoup de manipulations de chaînes de caractères.

Découpage et explication des tâches

Tâches Rémy / Temporisation

Les événements rejoués doivent être rejoués au même moment que lors de la capture.

- Prise en compte du délai passé depuis le début du lancement de l'application.
- Attendre ce délai avant de rejouer l'événement.

Difficultés

- 1er appel à une fonction C (écrite par moi-même) qui récupère le temps système.
  - Difficulté conversion valeur retour.
  - Difficulté d'intégration de la compilation de mon fichier C.

Découpage et explication des tâches

Tâches Rémy / Thread

Tentative d'ajout d'un nouveau thread

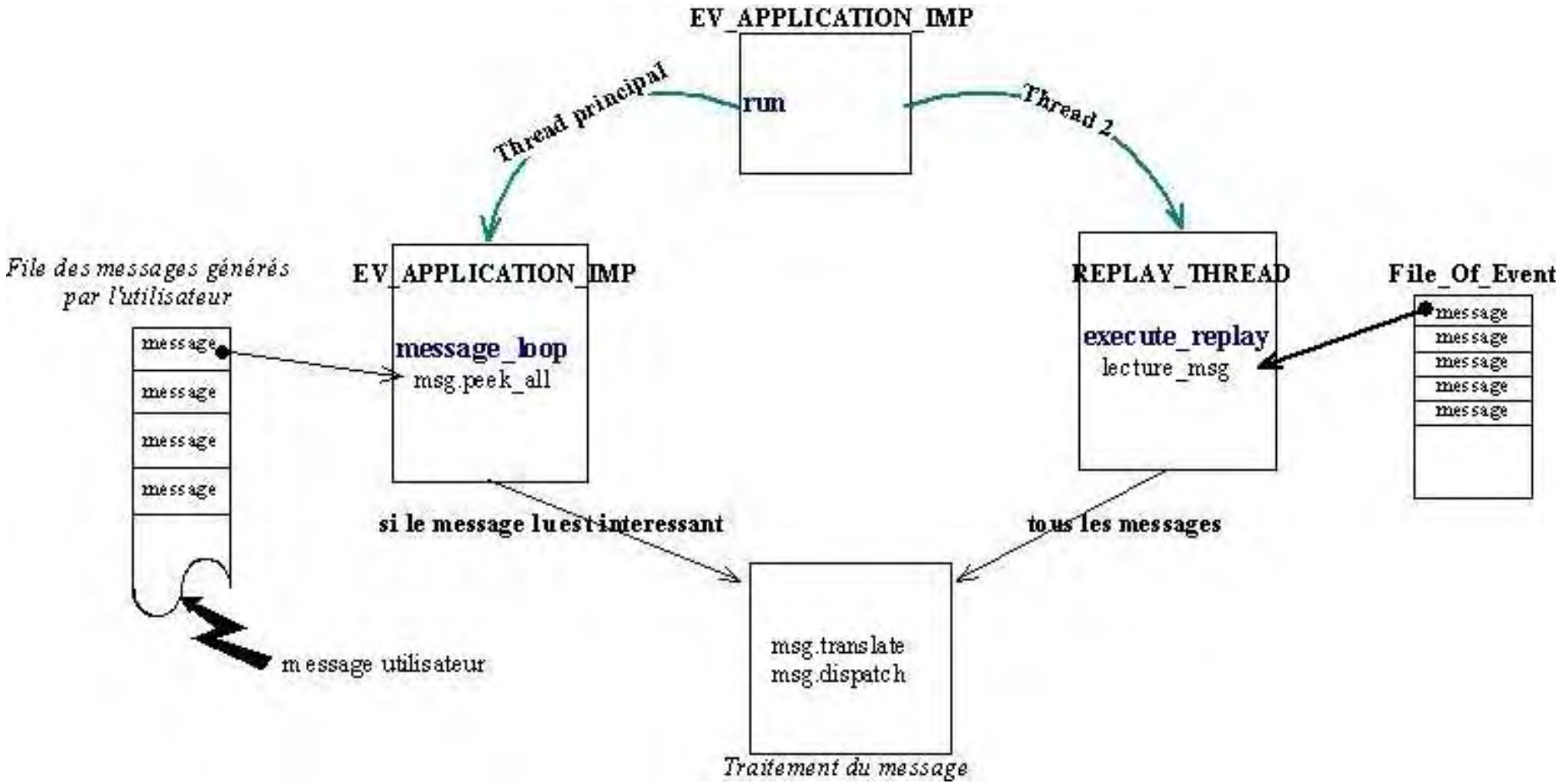
- Pour traiter tout de même certains événements de l'utilisateur (rafraîchissement de l'application par exemple).
- Pour éviter des exceptions lors du rejoue.

Difficultés

- Le partage de variables statiques (once routine) n'est pas aisé.
- Petit programme de test qui fonctionne bien mais pas dans le vrai.
- Encore plus d'exceptions.

Découpage et explication des tâches

Tâches Rémy / Thread schéma



Découpage et explication des tâches

Tâches Tarek / msdn

Recherche d'informations sur les boîtes de dialogues

- *Windows* fourni une interface utilisateur commune pour la création des boîtes de dialogues.
- Boîtes de dialogues formées à partir de modèles en mémoires.

***Eiffel* doit fournir les paramètres utiles à la création des boîtes de dialogues.**

Découpage et explication des tâches

Tâches Tarek / *Eiffel*

Classes définissant le contenu des boîtes

- *ABOUT_DIALOG*
- *EV_CONFIRMATION_DIALOG*
- *WEL_MODAL_DIALOG*

Les actions à réaliser sont définies dans d'autres classes.

Découpage et explication des tâches

Tâches / Process_message

Fonction définissant les actions à réaliser lorsque l'on reçoit des *window_message*.

Possibilité de récupérer les événements à ce niveau

- **On peut capturer chaque événement.**
- **A l'heure actuelle on ne capture que les événements n'ayant pas de pointeur dans leurs données membres.**

Découpage et explication des tâches

Tâches / Tests et débugage

Diversité des événements

- **Liaison entre les événements.**
- **A-t-on capturé tous les événements pour qu'une action puisse se rejouer ?**

Déterminer quel événement provoque un bug.

Difficultés pour suivre les exécutions lors des appels de fonctionnalités externe en C.

Plan

1. Notre projet le « Capture And Replay »
2. A quoi ça pourrait servir ?
3. Où sont passés les événements sous *Windows* ?
4. Et sous *Eiffel*, des changements ?
5. Découpage, explications des tâches
6. **Conclusion**

Conclusion

Projet intéressant car

- Étude de l'API de *Windows*.
- Découverte du langage *Eiffel*.

Projet difficile

- A rentrer dedans car il met en œuvre beaucoup de notions nouvelles.
- La phase de débugage est trop flou.
- Jamais la certitude que les modifications introduites vont apporter des changements (beaucoup de déchets).

Bilan

- Phase de capture marche bien.
- Phase de rejoue à revoir.