Introduction aux Patrons Logiciels

Software Patterns

Gerson Sunyé
gerson.sunye@univ-nantes.fr

Master Alma – Université de Nantes

April 25, 2009

Bibliographie (1/2)

- *Software Patterns*. James Coplien. SIGS Books. New York, 1996.
- *Smalltalk Patterns: Best Practices*. Kent Beck. Prentice Hall, 1997, 256 pp., ISBN 0-13-476904-X.
- *Design Patterns: Elements of Reusable Object-Oriented Software*. Erich Gamma, Richard Helm,Ralph Johnson, and John Vlissides. Addison Wesley. October 1994.
- *Real-Time Design Patterns*. Bruce Powel Douglass. Addison Wesley. 2003.

Bibliographie (2/2)

- *Pattern Languages of Program Design*.
  1. Coplien & Schmidt
  2. Vlissides, Coplien, & Kerth
  3. Martin, Riehle, Buschmann
  4. Harrison, Foote, Rohnert

Liens

- Patterns Home Page: http://hillside.net/patterns/
- Portland Pattern Repository: http://c2.com/ppr/index.html
- Cetus Links: http://www.objenv.com/cetus/oo_patterns.html
- Patterns FAQ: http://g.oswego.edu/dl/pd-FAQ/pd-FAQ.html

Motivation

- Réutilisation.
- Passage de connaissances.
- Communication entre développeurs.
- Documentation.

Historique (1/5)

- Christopher Alexander:
  1. *Notes on the Synthesis of Form*, Harvard University Press, 1964.
  2. *The Oregon Experiment*, Oxford University Press, 1975.
  3. *A Pattern Language: Towns, Buildings*, Construction, Oxford University Press, 1977.
  4. *The Timeless Way of Building*, Oxford University Press, 1979.

Historique (2/5)

- Dans [TTWoB] Alexander propose un paradigme pour l'architecture, basé sur trois concepts:
  1. The Quality (a.k.a. "the Quality Without a Name").
  2. The Gate.
  3. The Way (a.k.a. "the Timeless Way").

Historique (3/5)

- Voir la page *History of Patterns* dans le WikiWikiWeb de Ward Cunningham: http://c2.com/cgi-bin/wiki?HistoryOfPatterns
- *Using Pattern Languages for Object-Oriented Programs*. Ward Cunningham and Kent Beck. OOPSLA'87.

Historique (4/5)

- *Advanced C++ Programming Styles and Idioms*. Jim Coplien, 1991.
- Workshops OOPSLA de 90 à 92.

Historique (5/5)

- Hillside Group (Conférences PLoP): Kent Beck, Grady Booch, Richard Gabriel et al. OOPSLA 1993, 94.
- Design Patterns [GoF]. 1995.

Définition

Définition générale: Un patron est une solution éprouvée d'un problème dans un contexte.

Alexander: Chaque patron est une règle en trois parties, qui exprime la relation entre un certain contexte, un problème et une solution.

Définition (TWoB 1/3)

Plus qu'une solution à un problème dans un contexte:
"Comme un élément du monde, chaque patron est une relation entre un certain contexte, un certain système de forces qui se produisent à plusieurs reprises dans ce contexte, et une certaine configuration spatiale qui permet à ces forces de se constituer. (. . .)

Définition (TWoB 2/3)

(...) Comme un élément du langage, un patron est une instruction qui montre comment cette configuration spatiale peut être utilisée, à répétition, pour constituer le système de forces, partout où le contexte le rend relevant. (...)

Définition (TWoB 3/3)

(. . .) En bref, un patron est en même temps une chose qui se produit dans le monde, et la règle qui nous dit comment la créer, et quand nous devons la créer. C'est donc les deux, un processus et une chose; la description d'une chose vivante et la description du processus qui générera cette chose (TWoB pp. 247)."

Définition (Riehle et Zullighoven)

Dirk Riehle et Heinz Zullighoven dans *Understanding and Using Patterns in Software Development*:

> *"Un patron est l'abstraction d'une forme concrète qui se reproduit dans un contexte spécifique et non arbitraire".*

Définition (Coplien 1/2)

Jim Coplien, dans *Software Patterns*:

> *"J'aime bien faire la relation entre cette définition et les patrons de couture.*
> *Je pourrais vous expliquer comment faire un habit en spécifiant la route que les ciseaux doivent suivre à travers le tissu en terme d'angles et longueur de coupe. (. . .)*

Définition (Coplien 2/2)

(. . .) Ou alors, je peux vous donner un patron. En lisant la spécification, vous n'auriez aucune idée de ce qui était en train d'être fait, ou si vous aviez fait ce qu'il fallait, avant la fin.
Le patron annonce le produit: il est la règle pour réaliser une chose, et aussi, dans différents aspects, la chose elle-même."

Définition (wiki 1/3)

Wiki wiki web:

> *Les patrons de conception sont une technique utile pour la transmission de connaissances sur des problèmes récurrents dans le développement logiciel.*

Définition (wiki 2/3)

Wiki wiki web (bis):

> *Un patron de conception nomme, motive et explique systématiquement une conception générique qui s'adresse à un problème de conception récurrent dans les systèmes à objets.*
> *Il décrit le problème, la solution, le moment d'appliquer cette solution et ses conséquences. (. . .)*

Définition (wiki 3/3)

(. . .) Il donne aussi des conseils et des exemples de mise en oeuvre.
La solution est un arrangement d'objets et classes qui résolvent le problème.
Elle est personnalisée et mise en oeuvre pour résoudre le problème dans un contexte particulier.

Contre-définition (1/2)

Qu'est-ce que n'est pas un patron?

- Une règle.
- Une recette consacrée.
- Une structure de données.
- Une solution isolée à un problème dans un contexte.

Contre-définition (2/2)

Contre-exemple:

Problème : comment allouer des objets dans la mémoire?

Contexte : un système à objets dans la mémoire virtuelle.

Solution : exécuter quelques problèmes typiques et découvrir quels objets communiquent fréquemment et les placer dans une même page.

Langages de patrons (1/2)

- Chaque patron est une phrase dans un langage de patrons.
- Un langage de patrons est une collection de patrons.
- Les patrons forment un graphe.
- Chaque patron produit un contexte pour ceux qui le suivent.

Langages de patrons (2/2)

- Chaque patron se construit dans le contexte de ceux qui l'ont précédé.
- Un langage de patrons définit un *style architectural*.
- Les patrons isolés ont une utilité limitée.

Exemple I

Nom, problème, contexte

Nom : Cinq couleurs dans l'assiette.

Problème : Comment rendre appétissant, à peu de frais, un plat simple tel qu'une assiette de crudités?

Contexte : La cuisine au quotidien, notamment la cuisine familiale où les parents et les enfants partagent la même table, le même repas.

Forces

Cinq couleurs dans l'assiette

- Au jour le jour, la plupart d'entre nous avons peu de temps à consacrer à la préparation des repas.
- Les techniques culinaires avancées sont peu familières du grand public.
- La préparation des plats contribue de façon significative au plaisir des repas.
- Une certaine diversité des aliments est nécessaire pour une nourriture équilibrée.
- Les enfants mangent plus volontiers quand l'assiette est appétissante.

Solution

Cinq couleurs dans l'assiette

- Pour une même assiette, choisir les ingrédients de façon à ce que les cinq couleurs suivantes soient présentes: blanc, vert, rouge, jaune, noir.
- Les quantités des ingrédients pour chaque couleur n'ont pas besoin d'être identiques: une seule touche de couleur peut suffire.
- Les couleurs peuvent être approximatives: par exemple, un ingrédient orange peut donner la couleur rouge ou jaune en fonction du reste de l'assiette.

Exemples

Cinq couleurs dans l'assiette

- Ce patron convient particulièrement bien pour la composition d'assiettes de crudités.
- Les couleurs courantes dans les crudités sont le vert (salade, germes), le blanc (navet, céleri), le rouge (radis, tomate).
- Le noir et le jaune sont moins répandus: une simple olive noir au milieu de l'assiette, un quartier ou une rondelle de citron sur le bord transforment une assiette banale en une assiette appétissante.
- La salade grecque, par exemple, respecte ce patron: tomates, concombre, feta, olives noires, huile d'olive.

Implémentation

Cinq couleurs dans l'assiette

- La couleur de la porcelaine ou de la faïence, les décorations peintes sur l'assiette peuvent être prises en compte pour obtenir le compte des couleurs.
- On aura dans sa cuisine en permanence des ingrédients qui apportent les couleurs:
  - olives noires et vertes.
  - citrons.
  - tomates, tomates cerise.
  - salade, germes de luzerne.

Auteur

Cinq couleurs dans l'assiette

Origine : Cuisine chinoise traditionnelle, notamment la cuisine macrobiotique.

Auteur : Bruno Borghi. Mont St-Michel, 7 juin 2000.

Canevas de description

- Les patrons sont décrits sous la forme littéraire.
- Ils ne sont que “documentation”.
- Les composants essentiels sont: Nom, Problème, Contexte, Forces et Solution.
- Il existe plusieurs canevas de description.
- Exemples: GoF, Alexander, Coplien, Cockburn, Portland.

Canevas essentiel

- Contexte.
- Nom.
- Problème.
- Forces.
- Solution.
- Auteur et date.

Contexte

- La place du patron dans le langage de patrons.
- Habituellement négligé ou pas mis en valeur dans les patrons logiciels.

Nom (1/2)

- Un label significatif qui traduit le principe du patron.
- Le nom a tendance à être basé sur la solution.
- Il doit être court, concis, riche sémantiquement, révélateur ou suggestif.
- Les noms ingénieux (mais pas trop) sont mémorables.

Nom (2/2)

- Le nom devient une partie du vocabulaire du domaine.
- Les patrons du GoF sont des analogies (Bridge, Façade, etc.).

Problème

- Habituellement, le problème est posé comme une question, ou comme un énoncé.
- C'est la première chose que l'on regarde.
- La compréhension du problème vient avec l'analyse des forces.

Forces (1/3)

- Le coeur d'un patron.
- La totalité est un champ de forces ou compromis.
- Les patrons individuels encapsulent des forces en relation.
- Permettent de traiter un patron à la fois.

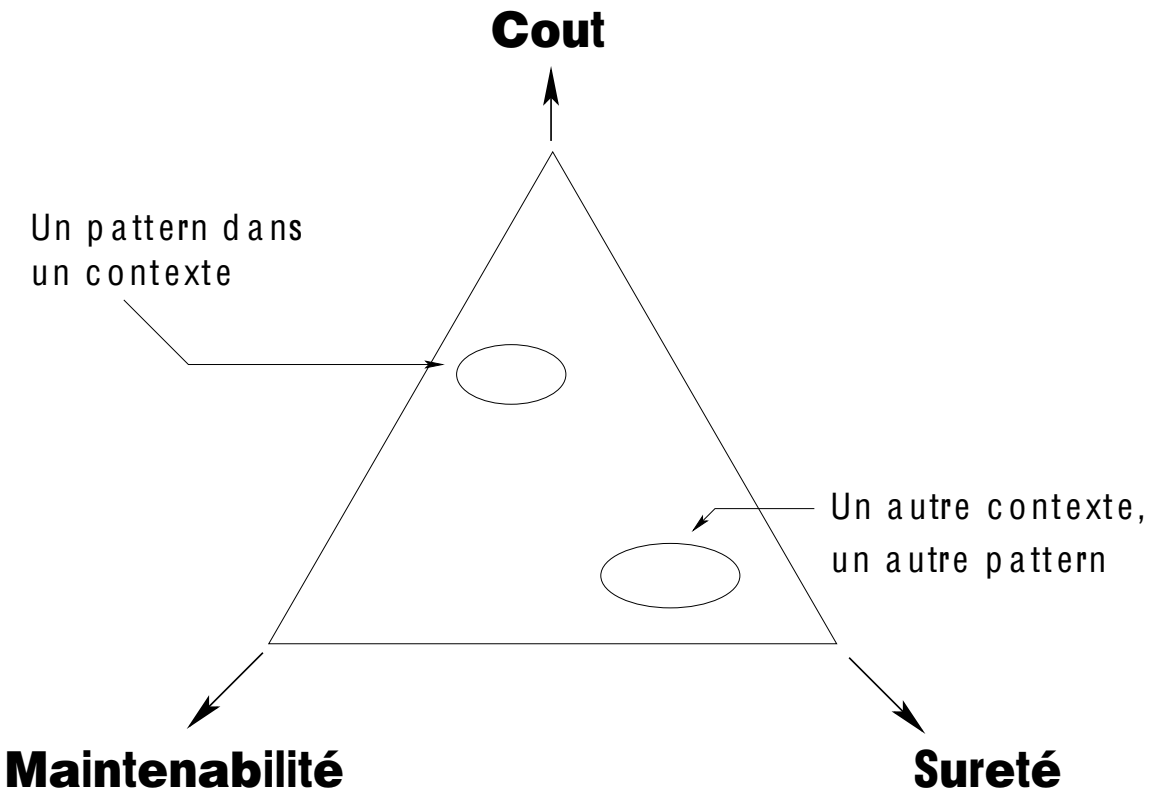
Forces (2/3)

Qu'est-ce qui fait une force?

- Ce qui rend le problème complexe.
- Pourquoi les solutions évidentes ne sont pas adéquates.
- Les facteurs qui contrecarrent d'autres approches d'ou autres solutions.
- Les contradictions et les conflits.

Forces (3/3)

Les patrons équilibrent les forces dans un contexte:



Forces (Alexander 1/2)

"Personne ne deviendra un meilleur concepteur en suivant aveuglément une méthode (. . .)

Forces (Alexander 2/2)

(. . .) Si on essaye de comprendre l'idée que l'on peut créer des patrons abstraits, en étudiant l'implication d'un système limité de forces, et que l'on peut créer des nouvelles formes par la combinaison de ces patrons – et réaliser que cela ne marchera qu'avec des patrons qui traitent avec un système de forces dont l'interaction interne est très dense et dont l'interaction avec d'autres forces du monde est très faible – alors, dans le processus d'essai pour créer ces diagrammes ou patrons pour soi, on peut trouver l'idée centrale à laquelle ce livre se consacre".
Préface de Notes on Synthesis of Form.

Solution (1/2)

- Les solutions équilibrent les forces.
- Les solutions s'appliquent au système comme un tout: elles sont transformables et non fixes.

Solution (2/2)

- Le centre d'intérêt de la plupart des gens.
- Souvent, un patron décrit une solution directe et simpliste: il faut donc essayer de résoudre le problème à un niveau plus profond.
- Plutôt que recenser les solutions, les patrons doivent encourager le lecteur à manoeuvrer les forces et remanier les solutions.

Solution (Alexander)

"Encore, le patron opère sur la totalité: ils ne sont pas des parties, qui peuvent être ajoutées – mais des relations, qui s'imposent aux précédentes pour mieux détailler, mieux structurer et donner plus de substance – de manière à faire émerger graduellement, mais toujours dans sa totalité, la substance de la construction, dans chaque phase de sa croissance.
Alexander, The Timeless Way of Building, p. 459.

Canevas GoF (1/3)

Nom : Un label significatif qui traduit le principe du patron.

Intention : Le problème de conception concerné.

Alias : Autres noms connus, s'il en existe.

Motivation : Un exemple d'application du problème de conception en cause et la solution proposée.

Indications d'utilisation : Le contexte dans lequel le patron peut être appliqué.

Canevas GoF (2/3)

Structure : Le diagramme de classes (OMT) représentant la solution.

Participants : Le rôle des différentes classes qui composent le diagramme.

Collaborations : La collaboration entre les participants.

Conséquences : Compromis et bénéfices de son utilisation.

Canevas GoF (3/3)

Implémentation : Astuces et techniques pour l'implémenter.

Exemples de code : Extraits de code exemple en Smalltalk et C++.

Utilisations connues : Références à des systèmes réels (au moins deux) où le patron est utilisé.

Patrons analogues : Les patrons en relation étroite avec celui-ci, leurs différences, la manière de les associer.

Canevas de Coplien

- Nom.
- Contexte.
- Problème.
- Forces.
- Solution.
- Croquis.
- Contexte résultant.
- Raisonnement.

Canevas de Cockburn (1/2)

- Onglet.
- Forces en équilibre.
- Facteurs affectant l'équilibre.
- Action recommandée.
- Contexte résultant.
- Effet de surdosage.

Canevas de Cockburn (2/2)

- Patrons en relation.
- Spécialisations.
- Principes concernés.
- Situations (exemples).
- Lecture.

HOPP - Applicabilité

. . . une fois que vous avez décidé de vos objets et de leurs associations, c'est l'heure de déterminer comment vos objets sont placés par rapport à l'espace d'adressage dans lequel ils vont s'exécuter. Ce patron vous aide à déterminer où ils sont placés et leur position relative aux frontières de l'espace d'adressage.

HOPP - Problème

Parfois, un même objet doit apparaître sur différents contextes informatiques (i.e. espaces d'adressage).
Comment pouvons nous rendre transparent la différence entre un et plusieurs (e.g. local vs. distribué) espaces d'adressage?

HOPP - Forces (1/8)

- Complexité.
- Distribution.
- Disponibilité d'information.
- Coût.
- Performance.

HOPP - Forces (2/8)

Souvent, les systèmes informatiques doivent être mis en œuvre sur des multiples espaces d'adressage, pour différentes raisons:

- Coût.
- Taille,
- Répartition physique.
- Diversité des environnements de programmation.
- Normalisation.
- etc.

HOPP - Forces (3/8)

Parfois, ces systèmes peuvent être facilement décomposés en objets, vivant chacun dans un espace d'adressage.

Quelques objets seront contraints à exister dans certains espaces d'adressage par le couplage avec le matériel (senseurs, lecteurs de disque, etc.).

D'utres objets peuvent être placés en fonction de ces couplages.

HOPP - Forces (4/8)

Parfois, cependant, un concept existe en deux espaces différents, et il est très difficile de le décomposer.

Ou alors, un objet doit collaborer avec d'autres objets, dans plus d'un espace adressage, parce qu'il a besoin d'informations de plus d'un espace d'adressage pour accomplir son comportement.

De même, des demandes (asynchrones) arrivant d'objets des deux espaces doivent être traitées.

HOPP - Forces (5/8)

Dans ce cas, aucun des espaces ne peut réaliser la tâche seul.

Par conséquent, un objet doit exister dans plus d'un espace d'adressage.

L'objet en question doit être divisé en un ou plusieurs objets par espace d'adressage.

HOPP - Forces (6/8)

Diviser un objet à travers plusieurs espaces d'adressage introduit une complexité supplémentaire, puisqu'un seul objet est plus simple que deux demi-objets, plus un protocole entre les deux.

Un demi objet pourrait implémenter tout le comportement et consulter les autres objets quand il a besoin de retrouver des informations ou réaliser une action.

HOPP - Forces (7/8)

Si ces interactions/collaborations sont fréquentes, le coût (d'exécution ou d'attente) de construction des objets-message et de les envoyer à un objet d'un autre espace d'adressage peut être inacceptable.

HOPP - Forces (8/8)

Répartir un objet en deux parties égales permet à chaque partie de répondre à plusieurs demandes sans consulter un autre espace d'adressage, mais peut résulter en une duplication de fonctionnalité et en un besoin de maintenir les deux objets synchronisés.

Cette répartition isole l'interface de l'objet du protocole utilisé entre les demi-objets.

HOPP - En conséquence (1/3)

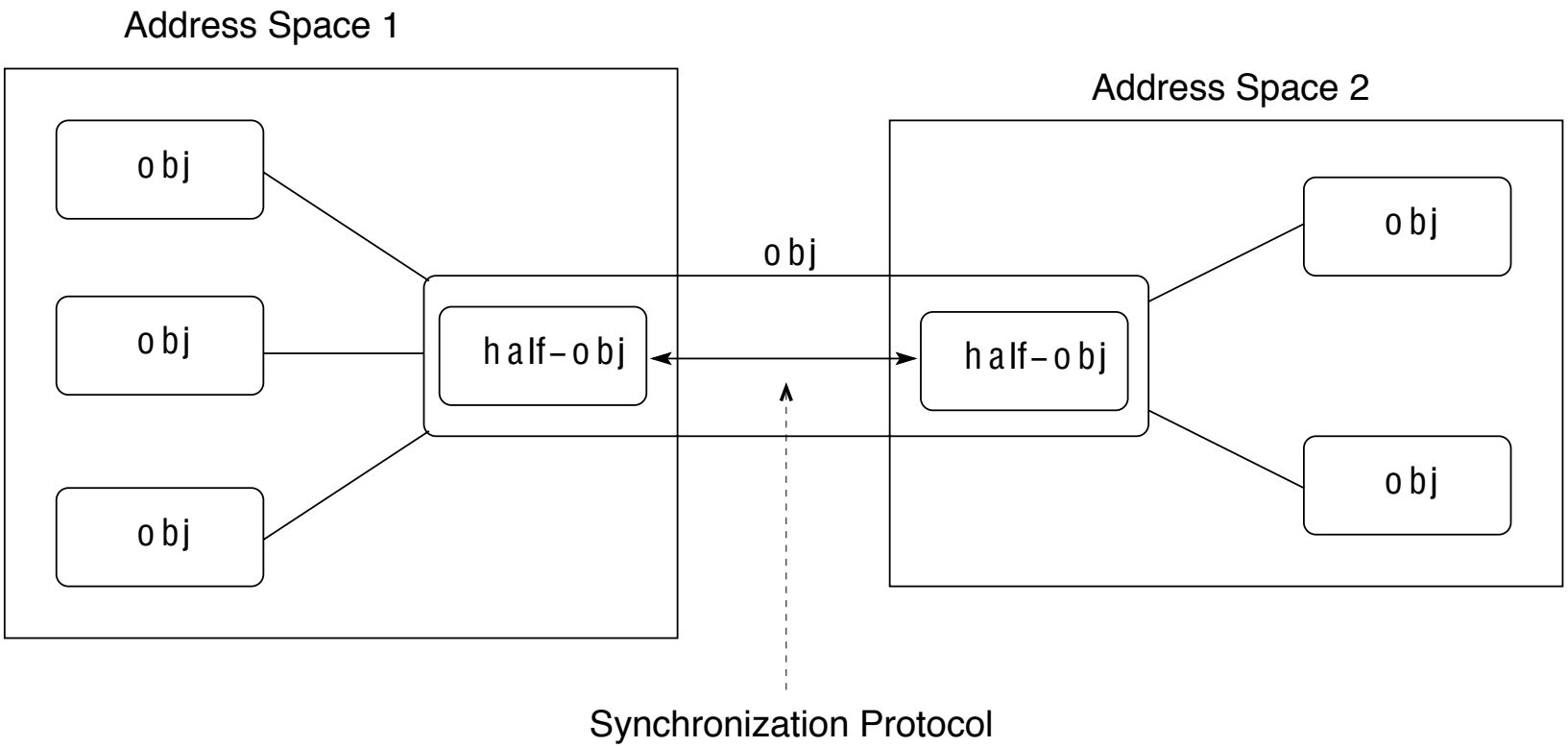
Diviser l'objet en deux demi-objets indépendants (un dans chaque espace d'adressage) avec un protocole entre eux.

Dans chaque espace d'adressage, implémenter toute fonctionnalité nécessaire pour l'interaction avec les autres objets dans cet espace d'adressage. (Ceci peut résulter en une fonctionnalité dupliquée, i.e., implémentée dans les deux espaces d'adressage). (. . .)

HOPP - En conséquence (2/3)

(. . .) Définir le protocole entre les deux demi-objets qui coordonne les activités des deux et porte l'information essentielle ayant besoin d'être passée entre les espaces d'adressage.

HOPP - En conséquence (3/3)



HOPP - Patrons connexes

Une fois que la fonctionnalité de chaque demi-objet est déterminée, le protocole entre eux peut être défini.

Ceci implique l'utilisation d'autres patrons pour concevoir le protocole (Message as Object, Message Parameter as Object), plus les mécanismes pour réaliser la création (information collection, formatting) et la réception (parsing, handling) du message.

HOPP - Exemples (1/3)

- Distribution physique:
  - Débogueurs non natifs.
  - Services téléphoniques distribués à travers un réseau national.
  - Systèmes accédant à des dispositifs matériels, comme des senseurs.
  - Interfaces utilisateurs distantes de l'information (bases de données centralisées).
  - Bases de données réparties (dupliquées).

HOPP - Exemples (2/3)

- Taille:
  - Systèmes où les fonctions doivent être accomplies dans un microprocesseur esclave, en raison du coût en temps réel de la fonction.
- Environnement de programmation hétérogènes:
  - L'interface utilisateur écrite en Smalltalk d'un débogeur C++.

HOPP - Exemples (3/3)

- Normalisation:
  - Les points de contrôle de service (SCP) contiennent les logiciels créés par des opérateurs téléphoniques pour modifier les appels qui passent par les switches du réseau téléphonique.
  - Ces logiciels peuvent ne pas avoir la permission de s'exécuter dans ces switches.

  → *(C'est aussi un exemple d'environnement de programmation hétérogène, car les switches ont des environnement propriétaires, tandis que les SCP utilisent des environnement basés sur UNIX).*

Auteur

- Gerard Meszaros
- Affiliation: BNR, Canada
- PO Box 3511, Station C,
- Ottawa, Ontario Canada K1Y 4H7
- E-mail: gerard@bnr.ca
- Phone: (613) 763-7860
- Fax: (613) 765-4855

Current contact information: gerard.meszaros@acm.org, 87 Connaught Dr. NW, Calgary, Alberta, Canada T2K 1V9, 403-264-5840, FAX 403-233-2021.

Caractéristiques d'un bon patron

- Il doit être une solution à un problème dans un contexte.
- Il est l'élément nécessaire à l'intégrité d'un ensemble, compris dans le contexte d'autres patrons dans cet ensemble.
- Il doit être capable d'expliquer à celui qui doit résoudre le problème, comment le faire.
- La solution doit être mature et éprouvée.

Caractéristiques d'un bon patron

- Il doit contribuer au confort humain ou à la qualité de vie.
- La solution doit être construite à l'aide de la perspicacité de celui qui doit résoudre le problème, et peut être implémentée un million de fois sans être deux fois la même.
- Il doit avoir un ensemble dense de forces en interaction, qui sont indépendantes des forces d'autres patrons.
- Il tend à avoir une harmonie géométrique.

Un patron bien écrit

- Il est court.
- Il raconte une histoire.
- Il est une phrase dans un langage de patrons.
- Il est attentif à l'esthétique.
- Il a un élément humain explicite.