Une approche incrémentale d'évolution des modèles de procédés de développement de logiciels

Ali B. Kaba[1], Joachim Tankoano[2] and Jean- Claude Derniame[1]
e- mail:{kaba, derniame}@loria.fr, tanko@ouaga.orstom.bf

Résumé

Les environnements de production de logiciels reposent en général sur un réseau complexe de modèles (produits, activités, directions, outils, rôles, agents, ...) qui évoluent constamment.

Pour supporter l'évolution de ces modèles et de leur état, nous introduisons dans cet article une démarche incrémentale. Cette démarche est caractérisée par:

- un modèle d'anticipation des évolutions qui repose sur le concept de *handler*. Le handler est utilisé pour contrôler, propager et historier les évolutions du procédé de développement.
- un mécanisme de mutation qui permet de structurer convenablement les concepts d'un procédé qui sont issus d'une démarche exploratoire.

Mots Clés: Évolution d'environnement, Contrôle d' évolution anticipée, Modélisation dynamique, Modèle de procédé de développement.

1 Introduction

Les évolutions des procédés de développement de logiciels sont permanentes. Au cours du développement, toutes les composantes d'un produit logiciel sont sujettes à des changements: par exemple les spécifications, le code, les jeux de test évoluent constamment. Le procédé en cours d'enactement est également très évolutif. Ces évolutions se manifestent très souvent par l'adjonction de nouvelles phases dans le procédé courant. Enfin, l'environnement de support peut évoluer lorsque de nouveaux outils y sont introduits, ou lorsque l'on remplace le système de stockage d'informations sous- jacent.

Afin de supporter efficacement l'évolution de ces composantes, nous avons construit un modèle d'évolution qui utilise pleinement les propriétés réflexives des représentations à objet.

[1]CRIN- Bâtiment Loria BP 239 - 54506 Vandoeuvre- lès- Nancy - France
[2]Université de Ouagadougou (ESI) - 03 B.P. 7021 Ouagadougou 03 - Burkina Faso

Le premier élément du modèle d'évolution est le *handler*. Ce concept fournit au procédé un moyen flexible de s'auto- modifier. Le handler supporte les évolutions anticipées grâce à des mécanismes qui permettent:

- de représenter des relations sémantiques (relations d'attributs composites ou dérivées,...) entre les variables du procédé de développement qui sont maintenues automatiquement,
- la modélisation et la propagation des effets d'un changement aux autres composantes de l'environnement,
- le déclenchement de certains réflexes pour maintenir la consistance avant et après des modifications,
- l'observation des évolutions de l'environnement, afin de permettre le retour à des états antérieurs en cas de besoin.

Le second élément du modèle d'évolution est un mécanisme de structuration des objets du procédé de développement qui sont issus d'une démarche exploratoire. Ce mécanisme de structuration permet de faire évoluer les produits du développement vers une famille d'objets similaires sans que cela ne nécessite un effort important de restructuration des produits du procédé de développement.

La suite de cet article est organisée comme suit. Dans la section 2 nous dressons un panorama des techniques de support de l'évolutions dans les prototypes actuels. Ensuite, dans la section 3, nous donnons un aperçu des principes généraux de notre approche d'évolution. En section 4, nous exposons les principaux mécanismes d'adaptation et de raffinement des procédés de développements. La section 5 introduit le handler. Dans la section 6, nous proposons le mécanisme incrémental d'évolution exploratoire. Nous concluons en section 7 sur des problèmes qui sont voisins du problème résolu.

2 Support de l'évolution des procédés de développements dans les prototypes actuels

Le problème de l'évolution des procédés de développement est abordé dans la plupart des environnements centrés sur les procédés. Dans cette section, nous donnons un bref aperçu des mécanismes d'évolution dans les prototypes actuels. Mais auparavant, nous proposons une taxinomie des

principales catégories d'évolution des procédés de développements afin de classifier aisément les prototypes actuels.

2.1 Taxinomie des principales catégories d'évolution

Nous distinguons deux catégories d'évolutions:

Les micro- évolutions Ce sont des évolutions induites par l'attachement de nouvelles valeurs (customisation et tailoring) aux variables du procédé grâce à des mécanismes qui sont implémentés dans le noyau du système. Le mécanisme de *liaison tardive* est le plus puissant des mécanismes de *micro-évolution.*

Les macro- évolutions Ce sont des évolutions qui occasionnent des changements profonds sur la structure des variables du procédé courant. Les macro- évolutions comprennent:

- *Les évolutions des composantes statiques du procédé.* Ces évolutions n'ont pas d'effet sur l'exécution du procédé en cours, car ils portent sur le texte de définition du procédé. Par exemple, une meilleure compréhension des procédés futurs peut amener à les décrire en attendant leur utilisation.

- *Les évolutions des composantes dynamiques.* Ces évolutions altèrent la définition du procédé en cours. Elles deviennent nécessaires lorsque par exemple on découvre des fautes dans la définition du procédé, ou lorsqu'on dispose de nouvelles informations sur les performances du procédé.

Il est également important de catégoriser les évolutions en fonction du moment où elles sont mises en oeuvre. En effet, selon l'état du procédé courant, certaines évolutions peuvent être anticipées, tandis que d'autres sont pris en compte au fur et à mesure qu'ils apparaissent.

- Les *évolution anticipées.* Ce sont les évolutions qui peuvent être prévues, par conséquent on peut définir à priori des structures de contrôle de ces évolutions. Ces structures de contrôle peuvent être mises à contribution pour propager les effets d'une modification, pour réexécuter des parties polluées du procédé, ou servir à restaurer et/ou restructurer le contexte de travail d'un procédé de développement.

- Les *évolutions non- anticipés.* Ce sont les évolutions imprévisibles, qui, dans les cas extrêmes peuvent interrompre le déroulement du procédé en cours. Cette catégorie d'évolution est courante dans les procédés de développement qui sont de nature exploratoire. Par exemple, lorsque nous développons un nouveau type de produit, nous n'avons pas de connaissances sur son processus de conception. Par conséquent, nous ne pouvons pas donner une caractérisation complète des produits fabriqués dès le départ. On ne peut pas donc anticiper l'évolution de ces objets.

- Les *évolution post- mortem* des procédés. Ce sont les conservations de états successifs du procédé, et les retours à des états antérieurs. Ces évolutions sont basées sur les mécanismes de gestion de versions et d'historique. Ces mécanismes sont nécessaires pour reconstituer un état précédent.

2.2 Les principales approches d'évolution des procédés de développement dans les prototypes actuels

Prototype	Syst.	Support				Mécanismes d'évolution
		Stat	Dyn	Dyn Ant	Post Mort	
Adele [Belkhatir 92]	Couplé (Tempo)			√	√	- *React Rules*: objets d'anticipation et de propagation des évolutions, - *Versioning*, - *Liaison dynamique.*
ALF [Derniame 92]	Intégré	√				*IMASPs*: Instances successives des MASPs avant leur exécution.
Articulator [Mi 91]	Dédié			√		Anticipation des évolutions avec des *activités alternatives.*
EPOS [Jaccheri 93]	Intégré		√		√	- Re- planification, - Versioning.
GRAPPLE [Huf 91]	Intégré			√		*Méta- plans* permettant de - modifier les instances du processus de production - remplacer une partie du plan par un autre en ajoutant un sous- but.

Prototype	Syst.	Support				Mécanismes d'évolution
		Stat	Dyn	Dyn Ant	Post Mort	
HFSP [Katayama 89]	Intégré			√		*Méta- Opérations* pour changer l'état des procédés (ex.: Redoing) .
IPSE2.5 [Snowdon 92]	Intégré		√			Modification dynamique de la topologie du réseau d'instances de rôles de PML.
MARVEL [Kaiser 93]	Couplé (Evolver)	√				- Génération de *commandes batch* pour effectuer des réparations. - Anticipation des évolutions par introduction de *règles alternatives.*
MELMAC [Deiters 90]	Intégré			√		*Modifications points* attachés aux modèles susceptibles de changer en cours de développement avec les informations nécessaires pour la modification.
PEACE [Arbaoui 92]	Dédié		√			- Modélisation de l'information incomplète avec la *logique non monotone,* - *Ordonnancement dynamique.*
PRISM [Madhavji 92]	Intégré		√			Modification du procédé par *rétro-action* entre les phases d'opération et de simulation.
Process Weaver [Fernstrom 93]	Dédié		√			*Liaison dynamique*: les réseaux de tâches sont des Réseaux de Pétri(RdP) inter- connectés qui sont liés dynamiquement à des programmes C++.
PSEA [Heimbigner 93]	Dédié		√		√	- *Re- exécution rapide,* - *Versioning,* - Création d'*états hypothétiques.*
SPADE [Bandinelli 93]	Intégré			√		*Méta- RdP* invoqué incrémentalement pour générer un RdP exécutable qui prend en compte les besoins d'évolution.

3 Principes généraux de l'approche d'évolution

Notre proposition sur l'évolution des procédés de développement s'appuie sur des représentations à base de FRAMES. Un aperçu de nos structures de représentations ainsi que de leur environnement de support dans le cadre de la modélisation des procédés est présentée dans [Tankoano 94].

Le modèle que nous proposons permet de supporter de manière incrémentale les principales catégories d'évolutions de la section 2.1.

Nous supportons les micro- évolutions, en tirant profit des facettes multiples et des attachements procéduraux offerts par les frames. Ces mécanismes sont utilisés pour *enrichir* (ou appauvrir) et *modifier* la structure des produits du développement par ajout (ou suppression) de facettes et de slots. Les fragments de procédé peuvent également évoluer par *clonage*. Le mécanisme de clonage réalise une reproduction d'un produit dont les valeurs sont celles du slot du modèle. Le produit ainsi cloné est physiquement différent de son modèle.

Ces mécanismes de micro- évolution permettent essentiellement au process designer de définir à l'aide du modèle du produit embryon la structure de ses instances à un très haut niveau d'abstraction, leur comportement et le procédé de fabrication (i.e. les connaissances génétiques) de leurs sous-produits.

Nous proposons deux moyens pour réaliser les macro- évolutions:

- Le premier moyen est l'infrastructure de contrôle et de suivi des évolutions anticipées que nous appelons `Change- Handler`. Ce concept utilise la propriété de réflexivité pour permettre au processus de développement de s'auto- modifier dynamiquement. D'un point de vue structurel, ce concept possède quatre composantes:

  - les structures de dépendances,
  - l'observateur des historiques et des versions,
  - une référence au handler des infrastructures d'évolutions,
  - les actions à déclencher avant, pendant et après les évolutions.

  Dans l'environnement, un handler est associé au méta- modèle du procédé. Lorsqu'une variable du procédé est créée, une spécialisation du handler est définie afin de prendre en compte les anticipations d'évolution de cette variable(cf fig. 1). Cette nouvelle structure ainsi créée aura pour rôle d'intercepter les besoins d'évolution des objets

afin de déclencher les actions nécessaires pour faire évoluer le modèle de procédé concerné ainsi que ses composantes.

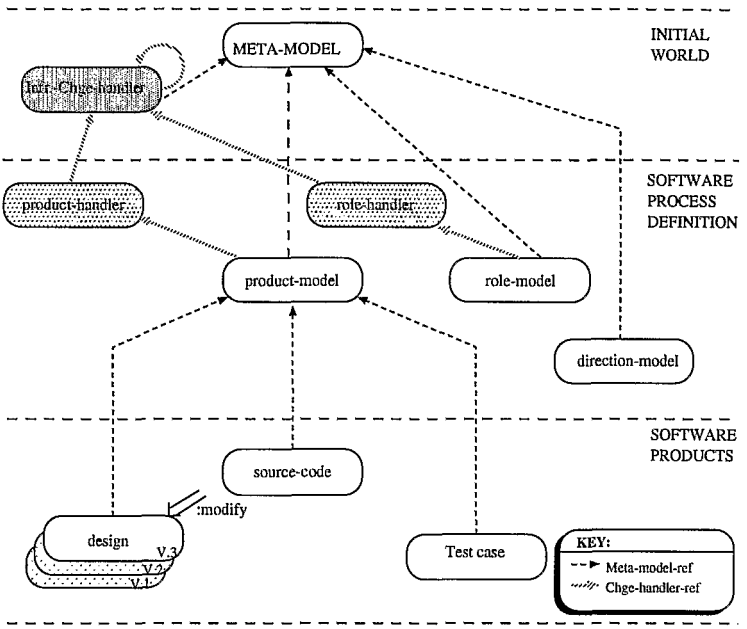

Figure 1 : Anticipation des évolutions d'un procédé avec un handler

- Le second moyen que nous proposons pour supporter les macro- évolutions est le mécanisme de *mutation*. Ce mécanisme permet de structurer convenablement les produits du développement que nous élaborons en utilisant une démarche exploratoire. Cette démarche consiste à partir d'un modèle initial du procédé et à le réviser continuellement jusqu'à l'obtention d'un modèle définitif. Ces révisions génèrent une famille d'objets que nous allons organiser de sorte à ne pas avoir à restructurer les objets, ou à ne pas être confronté au problème de récupération des instances d'objets générés en cours de développement.

  Le mécanisme de mutation des prototypes se base sur un lien nommé *has-drift* qui permet de lier de manière incrémentale un frames avec les éléments qui introduisent une différence dans le modèle.

Les concepts et la démarche seront approfondis dans les sections suivantes.

4 Le support des micro- évolutions

Dans cette partie, nous introduisons les mécanismes de l'environnement de support qui sont nécessaires pour créer, structurer, raffiner et adapter les procédés de développement. Nous avons volontairement réduit le jeu des mécanismes d'évolution pour mieux illustrer l'approche incrémentale. Nous ne considérerons donc que les mécanismes de clonage, d'enrichissement, et de modification des procédés de développement.

L'enrichissement incrémental des modèles

Ces mécanismes permettent de décrire progressivement des modèles de procédés initialement incomplets. L'enrichissement se fait de deux manières:

- Par décomposition des produits en sous- produits. Dans ce cas, nous ajoutons de nouvelles facettes correspondant aux descriptifs des sous-produits du produit courant. Le mécanisme `add-facet` permet d'ajouter une facette dans un modèle. `add-facet` prends comme argument le nom du produit et le contenu du sous-produit.
  Usage: `[ aProduct add-facet aSubProduct ]`.

- Par création d'instances d'un produit. Dans ce cas, nous attachons un nouveau Frames au slot value du produit déjà défini. L'ajout d'un slot à un objet se fait à travers le mécanisme `-addSlots`. En écrivant `[ aProduct -addSlots anotherProduct ]`, nous ajoutons tous les slots de `anotherProduct` à ceux de `aProduct`.

La modification des modèles

Si un modèle de procédé existant est inadapté, nous pouvons supprimer ou modifier les valeurs qui sont attachées au modèle. La modification peut consister à remplacer le contenu des slots du modèle. Dans ce cas, nous utilisons le mécanisme `-addSlot` en prenant soin de donner le nom de l'ancien modèle au nouveau modèle. Si nous voulons supprimer un slot, nous utilisons le mécanisme `-removeSlot` de la manière suivante:
`[aModel -removeSlot ModelName]`.

Le clonage des modèles

Le mécanisme de clonage réalise la copie physique d'un objet. Au moment de sa création, le clone généré devient une instance du méta- modèle de l'objet initial. En outre, le clone partage les valeurs de son originateur, mais n'en partage pas les slots.

Après le clonage, l'objet et son clone évoluent séparément sans interférence. Toutes les évolutions du clone qui respectent le méta- modèle sont acceptées. Les mécanismes qui sont utilisés pour faire évoluer un clone sont les mêmes que ceux qui sont utilisés pour faire évoluer tout objet de l'environnement (y compris le clonage).

L'évolution d'un clone n'a pas d'effet sur son originateur et vice- versa. Donc, lorsqu'on modifie les valeurs initiales d'un clone, les valeurs correspondantes de l'originateur ne sont pas affectées. De même, lorsqu'on ajoute de nouvelles composantes à l'objet initial, le clone n'est pas affecté. Le clonage se fait comme suit: `aProductVariant = [aProduct -clone]`

Exemple 1:

Soit un produit logiciel `P` auquel nous avons attaché le slot `system`. Ce slot indique le système d'exploitation sous lequel le produit est utilisé.

`P = [P add-facet (system Unix-BSD)]`

L'envoi du message `system` à `P` retourne "Unix-BSD".

Soit `C1` un objet créé en clonant `P`: `C1 = [P -clone]`

`C1` est automatiquement lié à *Product* qui devient son méta- modèle. L'envoi du message `system` à `C1` retourne la valeur "Unix-BSD", partagée au moment de la création du clone.

Nous pouvons faire évoluer maintenant `C1` vers un produit fonctionnant sous le système d'exploitation SOLARIS. Toutes les valeurs du produit sont modifiables, y compris la valeur de `system` qui devient "SOLARIS". Après cette modification, l'envoi du message `system` à `C1` retourne "SOLARIS" alors que `P` reste inchangé.

5 Les évolutions anticipées: concept de handler

Le modèle d'évolution anticipé est basé sur le concept de **handler**. Ce concept permet de définir une politique locale d'évolution d'un modèle.

5.1 Les principales composantes du handler

Le handler est composé de quatre éléments fondamentaux(cf. fig. 2):

Les structures de dépendances

Ces structures décrivent les relations de dépendance qui existent entre l'objet contrôlé par le handler, et, les éléments qui sont affectés par l'évolution de cet objet. Le concept `DEPENDANCY-STRUCTURE` a deux composantes:

- La `CHANGE-ITEMS-list`: cette composante répertorie les objets qui seront affectés par l'évolution de l'objet contrôlé.

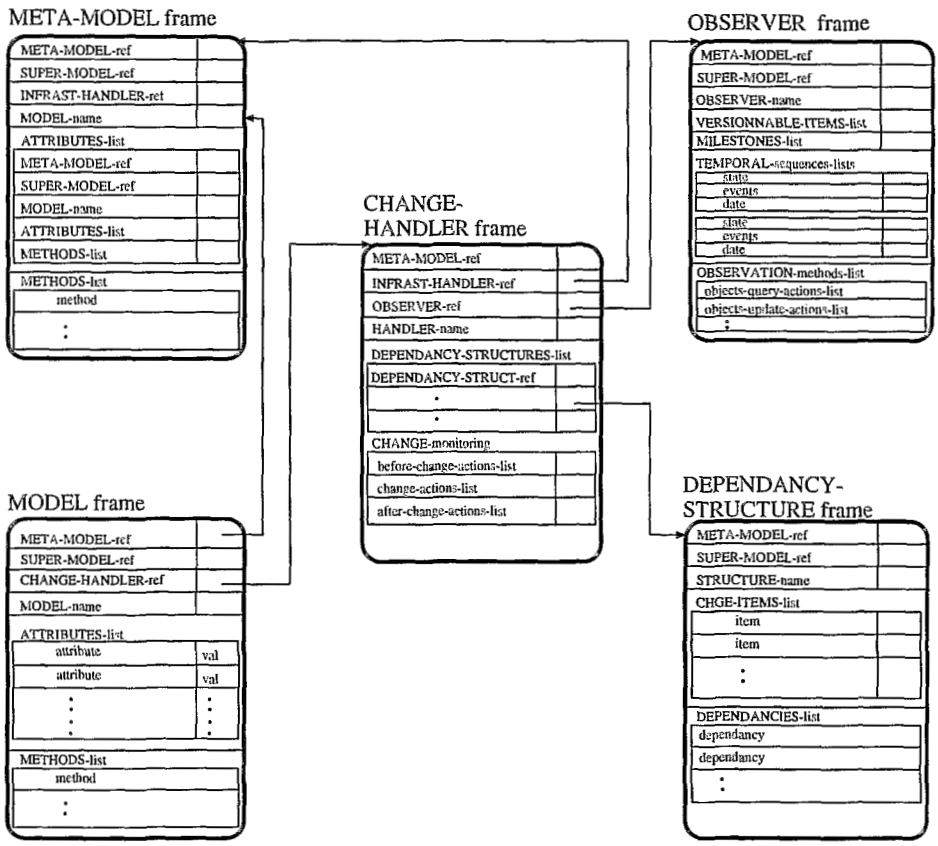

Figure 2 : Modèle générique de support des évolutions anticipées

- La `DEPENDANCIES-list`: elle décrit les relations de dépendances (dérivation et composition) entre les `change-item`. La spécification des relations de dépendance contiennent souvent une `dependancy-specification` qui indique comment les attributs dérivés sont calculés. Les détails d'implémentation d'une relation de dépendance sont fournis par le process designer.

Un exemple de `DEPENDANCY-STRUCTURE` est décrit par l'exemple 2

L'observateur d'évolutions

L'observateur permet d'historier de manière plus ou moins automatique l'évolution de l'état des principaux objets de l'environnement. C'est à travers cette structure que les objets accèdent à d'anciens modèles afin

d'effectuer des activités tels que des *retrying*, des tests, des comparaisons. Les propriétés statiques de cette structure permettent de *backtracker* vers d'anciennes versions, grâce à des rétro- propagations à travers les structures de dépendances. L'observateur sur un produit est décrit par:

Les propriétés externes du produit. Elles sont composées de:

- la `MILESTONES-list`. C'est la liste des dates qui jalonnent les principaux événements qui vont rendre les versions des modèles qui sont sous le contrôle du handler persistants.
- la `VERSIONABLES-ITEMS-list`. Cette liste répertorie les objets qui seront versionnés dès que le produit courant est modifié.
- la `TEMPORAL-sequences-list`. C'est la liste des séquences temporelles d'événements qui se sont produits sur les objets qui sont sous le contrôle du handler. Un élément de la séquence contient: - l'état du système au moment de l'historisation, - l'action qui a causé la transition d'état, - la date d'exécution l'action. Ces valeurs permettent de revenir à différents états consistants du procédé.

Les actions d'interrogation et d'enrichissement de l'historique Ces actions travaillent par calcul d'événements. Parmi elles, on distingue les opérations:

- `Version-Create` qui permet de sauvegarder l'état courant d'un objet comme une nouvelle version stable de l'objet.
- `transform` qui lorsqu'il est appliqué sur une méthode des `change-actions-list` permet d'effectuer la transformation appropriée, et, l'état courant de la transformation est sauvegardée comme une version stable.
- `PopUp` qui permet d'obtenir d'anciennes versions d'un objet.

Notons que tout message vers un objet versionné est détecté par l'OBSERVER qui effectue la tâche de gestion de version appropriée. Un exemple d'OBSERVER est décrit par l'exemple 2.

Le handler d'évolutions l'infrastructurelles

Cette composante fait référence à la structure qui fait évoluer le handler. Les évolutions de l'infrastructure sont moins fréquentes que l'évolution du procédé de développement. Cette faible fréquence provient de la stabilité du handler avant son utilisation croissante. Toutefois, l'évolution de

l'infrastructure est supportée par un modèle général prédéfini pour tous les handlers du procédé.

Les actions de mise en oeuvre de l'évolution

L'évolution désirée est prise en charge par des opérations de type `before-change-actions`, `change-action`, `after-change-actions`. La sémantique de ces opérations varie en fonction du modèle qu'on veut faire évoluer.

Exemple 2: handler du produit "design" (fig. 1)

DEPENDANCY-STRUCT de "design"

```
(DEPENDANCY-STRUCT MODIFY-design
  (CHANGE-ITEMS
     (design   des)
     (source   src)
     (TestUnit tst)
  )
  (DEPENDANCIES-list
     ((des.modified TRUE)
      ($edit     src)
      ;After the design modification
      ;determine source by editing
     )
     ((src.modified TRUE)
      ($generate src tst)
      ;After the source modification
      ;generate test cases from
      ;source code.
     )
  )
)
```

OBSERVER de "design"

```
(OBSERVER
    (MILESTONES 1 2 3)
    (VERSIONNABLE-ITEMS
       (design   des)
       (source   src)
    )
    (TEMPORAL-sequences-list
       ((Version 01)
        ($modify des)
        (date 1))
       ((Version 04)
        ($VersionCreate des)
        (date 1))
       ((Version 04)
        ($VersionCreate src)
        (date 1))
       ((Version 04)
        ($generate  src  tu)
        (date 1))
    )
)
```

6 Les évolutions exploratoires des procédés

En raison de la complexité du produit à délivrer, ou de la nouveauté de ce produit, le process designer n'a pas une vue globale sur son procédé de fabrication. Par conséquent, il ne peut pas lui donner une caractérisation à priori complète qui restera valide. La démarche que nous préconisons dans ce cas consiste à partir d'un modèle initial qui sera révisé continuel-

lement jusqu'à obtenir un modèle définitif. Cet individu évolue ainsi vers une famille d'objets qui lui sont similaires. Le support de cette catégorie d'évolution nécessite un mécanisme incrémental pour représenter le résultat de l'évolution.

Le mécanisme d'évolution exploratoire: le lien *has-drift*

La mutation de prototypes s'effectue à l'aide d'un nouveau type de lien prédéfini *has-drift*. Ce lien est multi- valué permettant ainsi au modèle de procédé courant de référencer les nouvelles descriptions qui introduisent une différence entre l'individu précédemment développé, et le produit courant. Le lien `has-drift` n'a pas strictement la même sémantique que les liens type/sous- type. Car le nouvel objet ainsi dérivé de l'objet précédent conserve non seulement la structure du schéma du modèle précédent, mais il conserve également les propriétés du schéma. Ce lien permet de représenter la similarité entre les objets sans qu'on ait à changer la structure de représentation des objets dans l'environnement.

Avantages du mécanisme

- L'avantage fondamental de ce mécanisme est qu'il permet de faire évoluer progressivement les modèles de procédés vers une famille de modèles sans qu'on ait besoin des restructurer l'ensemble des instances. Le problème de récupération d'instances relatifs à un modèle qui vient d'être altéré est moins complexe.

- Le mécanisme est incrémental, car il permet d'ajouter progressivement des informations au modèle sans qu'on ait besoin de modifier la structure des dits produits.

- Le versioning est implicite dans les descriptions. Car, pour extraire les versions, il suffit de prendre l'union de tous les liens précédant un objet. On peut également retrouver toutes les variantes successives qui ont été développées pour le produit en parcourant les branches alternatives `has-drift`.

7 Conclusion

Dans ce travail, nous avons proposé une démarche d'évolution incrémentale des procédés de développement de logiciels adapté aux environnements de développement de logiciels basés sur les produits et le paradigme

orienté objet. La démarche s'appuie sur deux concepts fondamentaux: (a) Un handler d'anticipation des évolutions, et, (b) un mécanisme de mutation, qui structure les évolutions exploratoires.

Ces deux concepts offrent un support flexible qui permet de contrôler les évolutions non- monotones des informations incomplètes et incertaines des environnements de développement de logiciel. Cette qualité, pourtant nécessaire n'est pas disponible dans les environnements de développement de logiciels existants qui sont basés sur les produits [Belkhatir 92, Snowdon 92, Jaccheri 93].

Dans l'état actuel, les mécanismes proposés n'offrent qu'un cadre générique pour la mise en oeuvre de procédés développement évolutionnaires. Il apparaît donc nécessaire d'approfondir l'étude de ces concepts, afin de les adapter aux particularités de chaque artéfact de l'environnement de développement de logiciel.

Par ailleurs cette étude a des liens étroits avec d'autres problèmes de recherches tels que l'évolution des schémas dans les bases de données [Banerjee 87], les problèmes de gestion de version[Tichy 89], et les problèmes de re-configuration automatique des systèmes. Le rapprochement de ce travail avec ces domaines contribuera à sa maturation.

Nous prévoyons d'implanter le modèle proposé sur la plate- forme PCIS[3] qui est en cours de développement. Sur cette implantation nous envisageons de valider l'approche avec un scénario évolutionnaire de procédé de développement basé sur le DoD-2167A.

Références

[Arbaoui 92] **Arbaoui S., Mouline S., Oquendo F., Tassart G.** *PEACE: Describing and Managing Evolving Knowledge in the Software Process* in Derniame J. C Software Process Technology. LNCS 635. Springer-Verlag. Sept 92.

[Bandinelli 93] **Bandinelli S., Fuggetta A., Ghezzi C.** *Process Model Evolution in the SPADE Environment* IEEE Transactions on Software Engineering. Special Issue on Process Evolution December 1993.

[Banerjee 87] **Banerjee J., Kim W.** *Semantics and Implementation of Schema Evolution in Object-Oriented Databases.* In ACM SIGMOD Annual Conference on the Management of Data, p 311-322. San-Francisco C.A, May, 1987. Special Issue of SIGMOD Record, 16(3), December 1987.

[3] PCIS est un projet de l'OTAN auquel participe l'équipe Génie- Logiciel du CRIN

[Belkhatir 92] **N. Belkhatir & al** *Supporting software maintenance evolution processes in the Adele system* Proc. of the 30th Annual ACM Southeast Conf. pp 165–172 Apr. 92.

[Deiters 90] **Deiters W., Gruhn V.** *Managing Software Processes in the Environment MELMAC* Procedings of the 4th ACM SIGSOFT Symposium on Software Development Environment Irvine 1990.

[Derniame 92] **Derniame J. C. & al** *Process Centered IPSE's in ALF.* Proc. 5th CASE Workshop. Montréal July 92.

[Fernstrom 93] **Fernstrom C.** *PROCESS WEAVER: Adding Support to UNIX* in Proceed. of the Second International Conference On Software Process (ICSP2), Berlin, Feb. 1993.

[Heimbigner 93] **Heimbigner D.** *A Reviosinist Approach to Process Change* Proceed. of Eighth International Software Process Workshop (ISPW8),Dagstuhl, Germany, March 1993.

[Huf 91] **Huff K. E.** *Supporting Change in Plan-Based Processes* Proceed. of 7th Intern. Software Process Workshop (ISPW7),Yountville, California, Oct. 91.

[Jaccheri 93] **M.L. Jaccheri and R. Conradi.** *Techniques for Process Model Evolution in EPOS.* IEEE Trans. on Software Engineering , December 1993.

[Kaiser 93] **Kaiser G., Ben-Shaul I. Z.** *Process Evolution in the MARVEL Environment* Proceed. of Eighth International Software Process Workshop (ISPW8),Dagstuhl, Germany, March 1993.

[Katayama 89] **Katayama T., Suzuki M.** *Mechanisms for Software Process Dynamics* Proceed. of Fifth International Software Process Workshop (ISPW5),Kennebunkport, Maine, USA, 9/1989.

[Madhavji 92] **Madhavji N.** *Environment Evolution: The PRISM Model of Changes* IEEE Transactions on Software Engineering, Vol. 18, No 5, May 1992.

[Mi 91] **Mi S., Scacchi W.** *Articulation: Supporting Dynamic Evolution of Software Engineering Processes* Proceed. of Seventh International Software Process Workshop (ISPW7),Yountville, California, October 1991.

[Snowdon 92] **Snowdon R.** *An Example of Process change* in Derniame J. C Software Process Technology. LNCS No 635. Springer-Verlag. Sept. 92.

[Tankoano 94] **Joachim Tankoano , Jean-Claude Derniame, Ali B. Kaba** *Software Process Design Based on Products and the Object Oriented Paradigm* in Warboys B. C. Soft. Process Techn. LNCS No 772. Springer-Verlag Feb 94.

[Tichy 89] **Tichy W. F** *Change Oriented Versionning in an Software Engineering Database* Procedings of the 2nd Int. Workshop on Software Engineering Database Princeton 1989.