Sciences & Technologie **B** – **N°26**, décembre (2007), 61-66.

VERS UN ENRICHISSEMENT DES COMPORTEMENTS DES DESIGN PATTERNS PAR LE LANGAGE DE SPECIFICATION FORMELLE LOTOS

Reçu le 15/06/2006 – Accepté le 24/06/2007

Résumé

La faiblesse sémantique des représentations actuelles des patterns entraîne des interprétations ambiguës et limite leur application. La spécification formelle s'avère être un mécanisme très utile permettant l'adaptation de solutions à un problème d'architecture ou de conception d'un système.

L'objectif de cet article est de définir la description de patterns par intégration des approches formelle et semi-formelle. Nous décrivons une méthode de spécification de patterns, intégrant deux paradigmes, la méthode UML (semi- formelle) et le langage de spécification formelle LOTOS.

***Mots clés**: design pattern, framework, lotos, spécification formelle, UML.*

Abstract

The semantic weakness of the current representations of the design patterns involves ambiguous interpretations and limits their application. The formal specification to be proved a very useful mechanism allowing the adaptation the solutions of a problem of architecture or a design of a system.

The objective of this article is to define the description of patterns by integration of the formal and semi-formal approaches. We describe a method of specification of patterns, integrating the two paradigms, the UML method (semi formal) and the language of formal specification which is LOTOS (formal).

***Keywords**: design pattern, framework, lotos, formal specification, UML.*

A. ZITOUNI
Laboratoire LIRE - Faculté des Sciences de l'Ingénieur
Université Mentouri
Constantine - Algérie

ملخص

قائص المدلولية للتمثيلات الحالية للنماذج التصميمية يؤدي إلى مشاكل في الترجمة ويحد من اسهالاتها. التخصصات القطعية هي أداة مفيدة تسمح بأقلمة الحلول لمشكلة المخطط العام أو لتصميم النظام.

الهدف من هذا المقال هو التعريف بموصفات النماذج عن طريق إدماج الطرق القطعية ونصف القطعية. نقدم في هذا المقال طريقة لتخصيص النماذج نضم المفهومين LOTOSوUML

***الكلمات المفتاحية:** النماذج التصميمية، إطار تصميم الأعمال، LOTOS التخصصات القطعية، UML*

Les patrons de conception (ou design patterns) [GHJV95] ont été proposés comme une technique d'aide à la conception de systèmes. Jusqu' à présent, ils ont été essentiellement définis pour modéliser des systèmes à objets. Les patterns proviennent de l'expérience des experts et décrivent un savoir-faire réutilisable par adaptation d'une solution générique à un problème courant. Ils sont un moyen de documenter les architectures et de disposer d'un vocabulaire commun. Ils sont essentiellement décrits par des notations graphiques tel que ceux fournis en UML (Unified Modeling Language) [BRJ98] et des exemples.

La faiblesse sémantique engendrée par les actuelles représentations des patterns entraîne des interprétations ambiguës et l'application automatisée de ceux-ci se voit limitée par leur manque de formalisation. Des approches telles que celles qui ont été proposées par B.Meyer [MEY92] avec la conception objet «par contrats» introduisent les notions de propriétés d'invariants de classes et de pré/post-conditions d'opérations dans la modélisation des objets.

L'approche fournie par Catalysis [DW98] décrit une démarche basée sur la notation UML et les contrats. Elle permet d'exprimer les collaborations entre objets, leur composition et le raffinement des modèles d'objets. Le langage OCL (the Object Constraint Language) a été proposé [WK99] comme faisant partie de UML, pour spécifier des invariants de classes, des contraintes, des pré/post-conditions d'opérations et des pré-conditions d'activation d'événements.

D'autres approches visent à définir des langages spécialisés pour la représentation de patterns. [BOS96] propose LayOM, un langage support pour les patterns de conception, basé sur la notion de couche permettant

d'encapsuler les objets. [EGY97] présente une méthode de spécification formelle couplée à une notation graphique. Dans [MMLS99] ses auteurs proposent de représenter des patterns par deux vues complémentaires, en UML [RGB98] et B [ABR96], de manière à bénéficier aussi bien des avantages des approches semi-formelles que de ceux des approches formelles. Les approches précédentes combinent plusieurs modèles dans le développement d'un système, permettant une meilleure compréhension d'un aspect particulier de celui-ci.

Cependant il n'existe pas aujourd'hui d'outils associés permettant d'analyser, de valider, de vérifier ou de simuler de telles spécifications. Il est nécessaire pour cela d'utiliser un langage de spécification formel outillé.

Dans [ZIT04] nous avons proposée une approche basée sur la spécialisation des modèles d'une architecture en appliquant de patterns. Nous avons donner une interprétation précise sur l'aspect structure et communication entre patterns. La démarche a consisté à enrichir par une spécification formelle en LOTOS (Langage Of Temporal Ordering Specification) les patterns décrits en UML, en vue de leur vérification.

Dans ce présent article nous allons proposé une démarche par utilisation des design patterns dans le but de spécifier, modéliser, analyser et réaliser des nouvelles générations de systèmes d'informations. Autrement dit: aider le développement des systèmes d'information basés web.

L'article est structuré comme suit. La section 2 décrit la spécification de patterns pour la modélisation de systèmes. La section 3 est une introduction au langage de spécification formelle Lotos. Nous définissons notre démarche globale, ou nous expliquons la transformation d'une description de pattern (du point de vue classe et comportement) en une spécification Lotos dans la section 4. Une proposition du pattern client serveur enrichi par la spécification Lotos est donnée dans la section 5, avec une démarche de l'application des patterns pour le développement d'une application. Enfin une proposition d'utilisation des relation inter-patrons est donnée dans la section 6.

1. SPECIFICATION DE PATTERNS POUR LA MODELISATION DE SYSTEMES

Les patterns ont été introduits par Christopher Alexander [AIS77]. Son idée de base est la suivante : "chaque pattern décrit un problème qui se manifeste constamment dans notre environnement et décrit le cœur de la solution à ce problème, de telle façon à ce qu' elle puisse être réutilisée des millions de fois et jamais deux fois de la même manière". Les experts écrivent des patterns à partir de leur expérience pratique. Ces patterns sont par la suite utilisés pour résoudre des problèmes de façon efficace.

1.1. Représentation de patterns

Un pattern décrit un problème devant être résolu, une solution particulière à celui-ci et le contexte considéré. Il n'existe pas aujourd'hui de consensus sur un formalisme de description de patterns. En général chaque auteur utilise sa propre notation. Les patterns peuvent être classés relativement à leur niveau d'abstraction: patterns d'analyse, patterns d'architecture, patterns de conception, et patterns d'implémentation. Par la suite, nous évoquons essentiellement des patterns d'architectures de systèmes distribués. Les représentations communément utilisées pour les patterns consistent à décrire ceux-ci en rassemblant des descriptions informelles, des diagrammes UML (ou d'autres notations graphiques) et des exemples d'implantation dans un langage orienté-objets particulier [GHJV95].

Bien que diverses informations sur un pattern soient fournies, leur manque de précision peut aboutir à des interprétations ambiguës et incorrectes. En conséquence, la sélection et la réutilisation se voient souvent limitées, par le manque de critères précis pour le choix du pattern le mieux adapté à un problème particulier.

2. PRESENTATION DU LANGAGE LOTOS.

LOTOS (Langage Of Temporal Ordering Specification) [ISO88] est un langage de description formelle normalisé par ISO, offrant des mécanismes pour la description des comportements parallèles et des mécanisme de description de communication. Lotos est basé sur deux méthodes formelles :

- Le calcul des processus, dérivé de CCS [MIL80] (Calculus of Communicating Systems), developpé à l'université d'Édimbourg, utilisé pour la représentation du comportement du système réparti.
- le langage Act One [EHR85], développé à l'université de Berlin, utilisé pour la représentation et la manipulation des données.

2.1 Syntaxe de basic Lotos

Nous définissons la syntaxe de LOTOS de base qui permet de décrire le comportement des processus parallèles en terme de séquence de portes observées durant leur exécution sans que l'échange de valeurs via ces portes ne soit pris en compte.

Soit PN l'ensemble des variables de processus parcouru par X et soit G l'ensemble des noms de portes définis par l'utilisateur (ensemble des actions observables). L dénote tout sous-ensemble de G, l'action interne est désignée par i, B parcouru par E, F,…. dénote l'ensemble des expressions de comportement dont la syntaxe est:

B:=stop / exit / X[L] / i;E / g;E / E[ ]E / E||E / E>>E / E[>E

2.2. Sémantique opérationnelle

Lotos est une approche dite par transformation, qui consiste à un ensemble de comportements Bi et de transitions.

Une transition d'action va être écrite sous la forme: B…..g…..>B', qui signifie que le comportement B exécute une action "g", et se transforme en un comportement B' (chaque action est la conséquences de l'exécution d'un opérateur (; , [ ] , || , >> , [> …)).

3. APPROCHE GLOBALE "SPECIFICATION DE PATTERNS EN UML ET LOTOS".

Nous proposons une approche consistant à définir deux niveaux de représentation complémentaires pour les patterns: d'une part, une description semi-formelle visant à en donner une définition intuitive; d'autre part, une spécification formelle visant à donner une description précise du comportement des objets (figure 1). Les deux descriptions permettent d'appliquer des patterns de façon méthodique et systématique lors de la spécification d'un système.

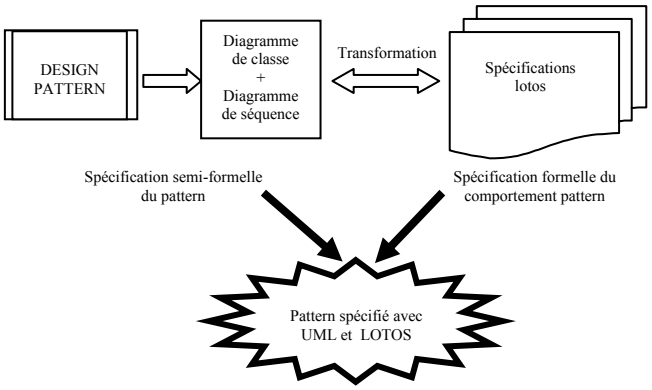

Figure 1 : Modèle objet du pattern Client-Serveur

3.1. Motivation de l'approche proposée.

La description semi-formelle d'un pattern est construite en utilisant le langage de modélisation orienté-objets UML [BRJ98]. Elle est composée de diagrammes d'objets décrivant les aspects statiques relatifs à la structure du pattern : les objets participants, leurs attributs, leurs opérations et leurs associations. Des diagrammes d'interactions permettent de modéliser certains aspects dynamiques du pattern. Cependant ces diagrammes peuvent être construits en désaccord avec le modèle objet et ne permettent ni de valider ni de vérifier les comportements des patterns. Les erreurs sont en conséquence rarement détectées.

C'est en spécifiant formellement un pattern que les contraintes ainsi que les propriétés associées aux objets et leurs relations peuvent être exprimées et vérifiées.

Nous proposons d'utiliser le langage de spécification formel Lotos [ISO88] pour les raisons suivantes :

- La spécification Lotos permet de modéliser de manière plus détaillée le comportement et l'interaction entre les différentes entités d'un modèle.
- Lotos est une méthode est basé sur la logique, qui permet de formaliser la sémantique comportementale des systèmes concurrents
- Lotos permet de modéliser de manière plus détaillée le comportement et l'interaction entre les différentes entités d'un modèle,
- La spécification Lotos permet une compréhension approfondie de l'application à développer. Elle met en évidence la plupart des ambiguïtés laissées par les spécifications informelles
- Les spécifications Lotos sont basées sur les représentations mathématiques, ce qui permet la validation de leurs propriétés.

3.2. Transformation de la représentation du Pattern en UML vers la spécification Lotos.

La transcription du pattern architectural en une spécification Lotos est décrite par la relation de correspondance que nous allons définir.

Définition: Soient $\mathfrak{Rc}$ une relation, **Ci** l'ensemble des classes d'un pattern, et Π**i** l'ensemble des opérations d'une classe donnée alors :

- **Ci $\mathfrak{Rc}$ Pi $\Rightarrow$ (Toute classe se transforme en un processus Lotos)**
- **Πi $\mathfrak{Rc}$ Ei $\Rightarrow$ (l'ensemble des opérations (méthodes) d'une classe se transforme en une suite d'actions décrivant le comportement de cette classe).**

3.3. Etude de cas.

3.3.1. Spécification UML du pattern Client-Serveur.

Nous illustrons notre approche à l'aide du pattern de conception Client-Serveur fournit dans [BER92]. Il est définit comme un style d'architecture, permettant la construction de systèmes distribués. Il apparaît comme un pattern générique, définissant deux types de correspondants : les clients, demandeurs de services, et les serveurs, fournisseurs de services. La relation entre clients et serveurs consiste en la demande d'un service par un client à un serveur. Dans une première définition du pattern, aucun détail sur la communication entre les clients et les serveurs n'est considéré. Le modèle Client-Serveur (figure 2) est un style permettant de construire une première version d'un système, dont seul le choix d'architecture est déterminé. De ce fait, il peut être considéré comme la donnée d'un problème consistant à décrire des clients et des serveurs qui communiquent.

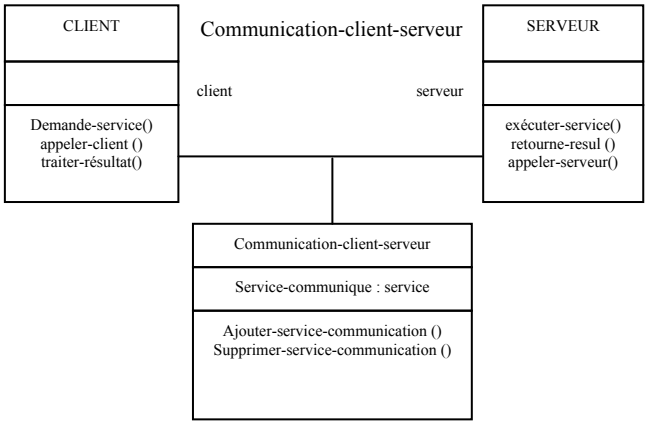

Figure 2 : Modèle objet du pattern Client-Serveur

Le diagramme d'interaction de la figure 3 illustre les échanges de messages entre les objets du pattern Client-Serveur.

Par la suite nous complétons la description du pattern en lui associant une spécification formelle Lotos.

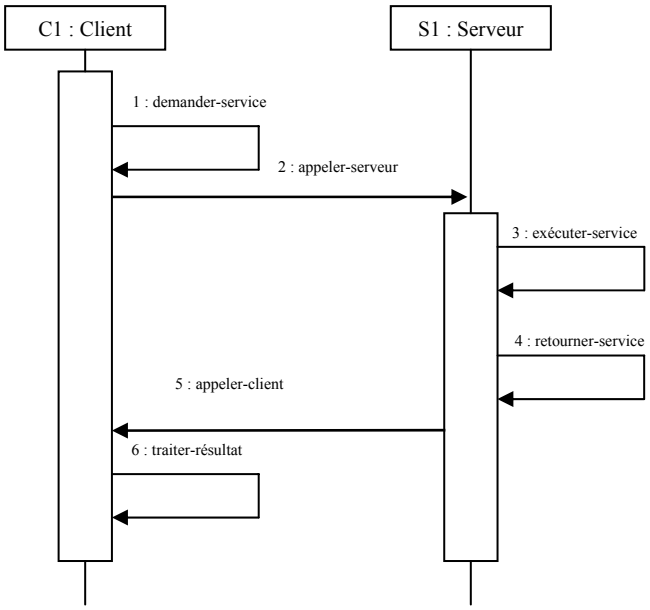

Figure 3 : Diagrammes d'interactions du pattern Client-Serveur]

4.3.2. Spécification Lotos du pattern Client-Serveur

Dans le but de formaliser et valider la spécification, nous allons par la suite montrer une partie de la description Lotos associée au pattern décrit précédemment en UML.

Ceci est faisable par l'application de la relation de correspondance que nous avons définit précédemment, ainsi que la syntaxe de basic Lotos, nous amène à la spécification du pattern architectural Client- Serveur .

Pi:= {Pattern-client-serveur} et {Communication-client-serveur ; Client ; Serveur}

Ei : = {ajouter-service-communication; supprimer-service-communication; demander-service ; appeler-serveur ; traiter-résultat ; exécuter-service; retourner-service;appeler-client}

Process Pattern-Client-Serveur **: =**

Client |CCS|
Communication-client-serveur |CCS|
Serveur

End process

Exécution en parallèle des trois processus

Process Communication-Client-Serveur [CCS] : =

Ajouter-service-communication ; Client
|CCS|
Serveur ;supprimer-service-Communication

End process

Exécution inéterministe des processus client et serveur

Process Client [CCS] : =

Demander-service ;
Appeler-serveur ; **exit** ;
>> Serveur ;s
Traiter-résultat

End process

Spécification de l'exécution du processus client

Process Serveur [CCS] : =

Exécuter-service ;
retourner-service
appeler-client ; **exit** ;
>> Client

End process

Spécification de l'exécution du processus serveur

Nous remarquons que dans cette spécification, le comportement du processus Pattern-Client-Serveur est donné par "l'exécution en parallèle" des trois processus (classes) du pattern. Ces processus se synchronisent à travers le port de synchronisation (CCS).

Le comportement des processus est donné par la liste des interactions des classes du pattern. Enfin le comportement du processus communication-client-serveur est donné par le comportement du processus Client, ou le comportement du processus serveur (ce choix est indéterministe).

5. PROPOSITION D'UN META PATTERN CLIENT-SERVEUR ENRICHI PAR UNE SPECIFICATION LOTOS (PATTERN CLISEREL)

Nous parlerons de notre représentation d'un pattern de conception comme d'un **méta-pattern**. Le terme de méta-pattern a été emprunté à [PRE94], et qui propose une réification partielle de l'aspect structurel des patterns dits de compositions, et a été étendu afin de prendre en compte toutes les informations d'un pattern, mais également d'y effectuer des manipulations (instanciation, spécification).

Chaque entité d'un méta-pattern est un pattern, Chaque pattern porte à la fois des informations structurelles et comportementales qui lui sont propres.

UML ne fournit pas de notation spécifique pour modéliser les designs patterns ou même les méta patterns. Nous avons dû étendre le méta modèle UML afin d'y intégrer notre notation. Cette extension consiste en l'introduction d'un nouveau stéréotype: <<pattern>> [ZIT05].

Le pattern CliSerEL a pour but d'expliciter et de formaliser les interactions et les relations entre les différentes classes du pattern (client, serveur et communication-client-serveur) (figure 4).

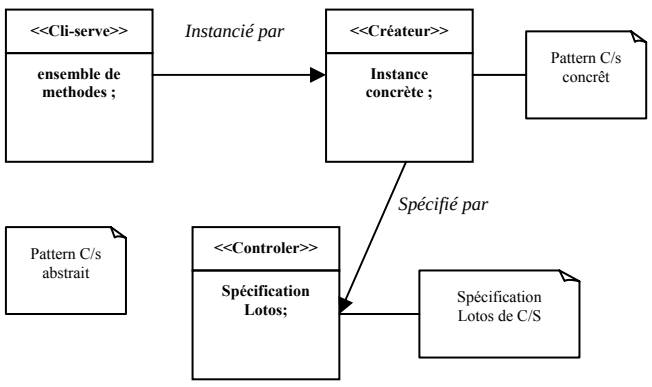

Figure 4 : Pattern CliSerEL

Notre modèle est constitué, en plus du pattern client-serveur de deux patterns

- **Créateur:** ce pattern est lui aussi un modèle de conception relativement intuitif. Il consiste en la détermination de la responsabilité de la création d'une instance d'un pattern.
- **Contrôleur:** ce pattern représente le scénario issu d'un cas d'utilisation. Et est chargé de traiter tous les événements systèmes contenus dans un scénario de cas d'utilisation.

Tout ceci va nous amener a en proposer un framework à nos besoins spécifiques, et qui consiste a une proposition d'une démarche vers le développement des systèmes d'informations basées web.

5.1. Vers une proposition d'une démarche de développement des systèmes d'informations basées web

Une application immédiate de ce travail consiste pour nous à regarder l'impact d'une telle méta-modélisation sur les schémas d'applications (framework). En effet, un des avantages de cette modélisation est la représentation des dépendances entre les éléments du pattern afin d'expliciter leur instanciation commune. Or, c'est justement à notre sens un des problèmes lié à l'instanciation d'un framework que de comprendre comment les différents éléments qui le composent s'organisent.

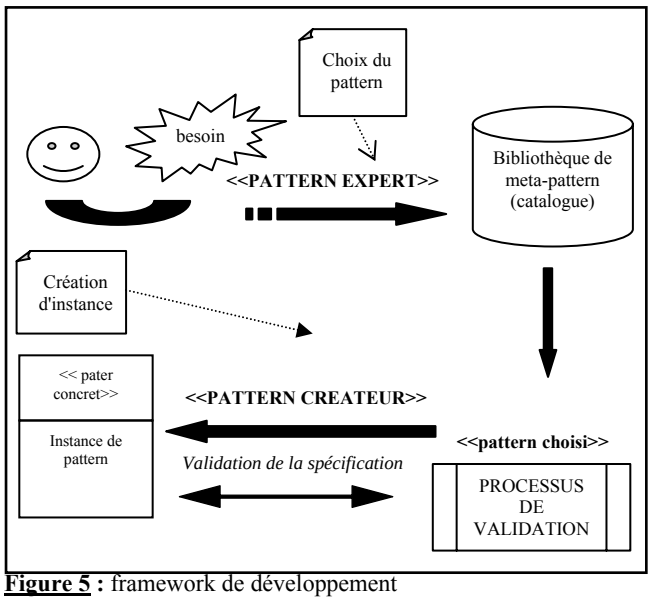

Figure 5 : framework de développement

Notre framework propose à un utilisateur une bibliothèque de design patterns représentés en UML. Dans les applications distribuées et en particulier les SI webs, les patterns les plus utilisés et qui sont nécessaires. Citons les patterns: Decorator, Factory method, Proxy, Abstact factory, interceptor, Observer [GHJV95].

Il s'agit pour un utilisateur de choisir un design pattern pour son application et de l'introduire dans son modèle de conception. (figure 5).

Le pattern "Expert": l'objectif de ce pattern est d'affecter au mieux une responsabilité à un pattern selon les cas d'utilisation

Une fois que tous les patterns concernés par notre application seront choisis et validés, l'étape suivante serait de les composer afin de modéliser l'application complète.

6. VERS UNE PROPOSITION DE COMPOSITION DES PATTERNS DE CONCEPTION

Une fois que tous les patterns concernés par notre application seront choisis et validés, l'étape suivante serait de les composer afin de modéliser l'application complète.

Pour ce faire nous nous proposons l'utilisation les notions de relations inter-patrons qui peuvent exister et nous allons définir.

6.1.Relations inter-patrons

Dans cette section nous nous intéressons à l'étude des relations qui relient les patrons qui se trouvent au sien d'un même système de patrons.

Ces relations sont exprimées au moyen de trois rubriques correspondant aux relations entre patrons les plus utilisées**: <<utilise>> ,<<raffine>> et <<requiert>>.**

Utilise: c'est la relation la plus utilisée dans les système de patrons. Elle permet la décomposition d'un problème complexe en sous-problèmes de complexité moindre; Autrement dit: un patron P1 utilise un patron P2 si et seulement si les problèmes posés par P1 peuvent être résolus par P2 . Formalisée par:

Utilise: Si un patron P1 utilise un patron P2 (figure 6), alors :
- la solution de P1 doit être exprimée en utilisant le patron P2
- le contexte de P2 peut être enrichi par rapport à celui de P1

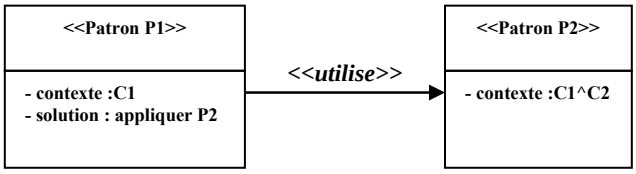

Figure 6 : Relation <<utilise>>

Raffine: Si un patron P1 raffine un patron P2 (figure 7), alors :
- le problème de P1 doit être une spécialisation de celui de P2
- le contexte de P2 peut être enrichi par rapport à celui de P1

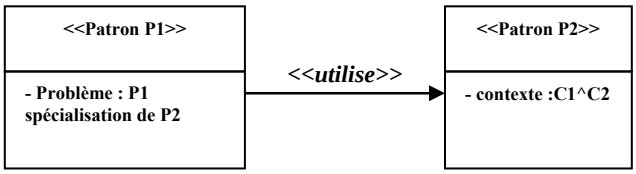

Figure 7 : Relation <<Raffine>>

Altenative: Si un patron P1 est une alternative du patron P2 (figure 8), alors :
- la solution de P1 doit être exprimée en utilisant le patron P2
- le contexte de P2 peut être enrichi par rapport à celui de P1

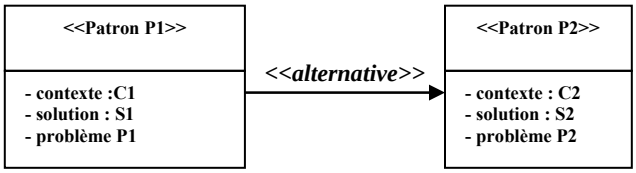

Figure 8 : Relation <<Alternative>>

CONCLUSION

Dans ce papier, nous avons présenté une approche pour spécifier des patterns pour des architectures de systèmes en UML et en LOTOS, permettant ainsi de bénéficier aussi bien des avantages des approches semi-formelles que de ceux des approches formelles. Nous avons choisi le pattern Client-Serveur pour expliquer notre approche, car c'est un pattern simple, académique et relativement bien connu

Nous n'avons pas abordé dans cet article, les différentes discussions sur l'intérêt de tel ou tel pattern [AC98], ni sur la multiple représentation d'un pattern en fonction du langage visé [Sun99]. Néanmoins, à l'instar de plusieurs autres travaux [EGY97, ACL96], nous sommes convaincus que c'est en apportant une approche plus formelle à la représentation des patterns que ces problèmes seront résolus.

Enfin, il faut toujours avoir à l'esprit que ce n'est parce qu'une spécification est formelle qu'elle ne contient pas des fautes. Ces fautes peuvent provenir d'une mauvaise compréhension des besoins de l'utilisateur., d'ou la nécessité de l'introduction de techniques de vérification formelle. C'est dans cette optique, que se situe la suite de notre travail, et qui consistera à utiliser des techniques formelles pour la vérification de nos spécifications.

Une autre perspective de ce travail, est d'exploiter l'enrichissement des designs pattern par le langage de spécification Lotos en tant qu'ADL (Architecture Description Language), pour spécifier et formaliser les comportements des Middlewares,

RÉFÉRENCES

[ABR96] Abrial (J.R.). – *The B Book - Assigning Programs to Meanings.* – Cambridge University Press, 1996. ISBN 0-521-4961-5.

[ACL96] Alencar (P.S.), Cowan (D), Lucena (J.P.). « A Formal Approach to Architectural Design Patterns », Proceedings of the 3rd International Symposium of Formal Methods Europe, pp. 576-594, 1996.

[AC98] E. Agerbo(E.), Cornils (A.). « How to preserve the benefits of Design patterns », OOPSLA'98, page 134-143.

[AIS77] Alexander (C.), Ishikawa (S.), Silverstein (M.), Jacobson (M.), FiskdahlKing (I.) et Angel (S.). – *A Pattern Language.* – Oxford University Press, 1977. ISBN 0-195-01919-9.

[BER92] Berson (A.). – *Client Server Architecture.* – McGraw Hill, 1992.

[BRJ98] Booch (G.), Rumbaugh (J.) et Jacobson (I.).– *The Unified Modeling Language User Guide.*– Addison-Wesley, 1998. ISBN 0-201-57168-4.

[BOS96] Bosch (J.). – Language Support for Design Patterns. *In : Proceedings TOOLSEurope'96.* – Paris (F), February 1996.

[CON00] J. Conallen, concevoir des applications WEB avec UML, Eyrolles edition, 2000

[DW98] D'Souza (D.) et Wills (A.C.). – *Object Components and Frameworks with UML : the Catalysis method.* – Addison-Wesley, Object Technology Series,1998. ISBN 0-201-31012-0.

[EGY97] Eden (A.), Gil (J.) et Yehudai (A.). – Precise Specification and Automatic Application of Design Patterns. *JOOP*, may 1997.

[EHR85] Ehrig (S) -Algebraic concept for software development in Act-one, act-two and Lotos Berlin, march1985

[GHJV95] Gamma (E.), Helm (R.), Johnson (R.) et Vlissides (J.). – *Design Pattern -Elements of Reusable Object-Oriented Software.* – Addison Wesley, 1995. ISBN 0-201-63361-2.

[PRE94] Pree, (W.) « Meta Patterns - A means for capturing the essential of reusable object oriented design », ECOOP'94, p150 – 162. Springer-Verlag, 1994.

[ISO88] Lotos, a formal description technique of the OSI connection-oriented network service ISO IS 8807, November 1988

[MEY92] Meyer (B.) – *Applying design by contracts.* IEEE Computer, oct. 1992.

[MIL80] Milner (R) -Operation and algebraic semantic of concurent processes, LFCS, report series, february 1988

[MMLS99] Marcano (R.), Meyer (E.), Lévy (N.) et Souquières (J). – Utilisation de patterns dans la construction de spécifications en UML et B. *Dans : actes AFADL'2000.* – Grenoble, janvier 2000.

[SUN99] G. Sunyé (G.), « Génération de code à l'aide de patrons de conception »,LMO'99 Hermes science.

[WK99] Warmer (J.) et Kleppe (A.). – *The Object Constraint Language, precise modeling with UML.* – Addison-Wesley, Object Technology Series, 1999.

[ZIT04] Zitouni (A), Un framework pour l'utilisation des design patterns par intégration du langage de spécification Lotos, 4éme séminaire national en Informatique Biskra mai 2004

[ZIT05] Zitouni (A), Un framework pour l'utilisation des design patterns dans le developpement des systemes d'information bases web (Siweb) , Congrès International en Informatique Appliquée CIIA05, Bordj Bou Arréridj , Algérie ISBN: 9947-0-1042-2