

---

# Une approche orientée règle pour la spécification formelle des architectures dynamiquement configurables

Riadh Ben Halima\*,\*\* — Mohamed Jmaiel\* — Khalil Drira\*\*

\* Unité de recherche ReDCAD

École Nationale d'Ingénieurs de Sfax, Tunisie

Mohamed.Jmaiel@enis.rnu.tn

\*\* LAAS-CNRS

7 avenue du Colonel Roche, 31007 Toulouse Cedex 4, France

{Riadh.ben.halima, Khalil}@laas.fr

---

**RÉSUMÉ.** Dans ce papier, nous traitons le problème de la conception formelle des systèmes logiciels réparties dynamiquement reconfigurables. Pour compléter les approches de reconfiguration comportementale souvent adoptées, nous introduisons la reconfiguration architecturale. Cette technique permet de construire dynamiquement l'architecture d'un système logiciel et de l'adapter à l'évolution de l'activité qu'il soutient. Pour formaliser les différentes instances d'architectures et styles, nous spécifions en Z les entités logicielles et leurs liens d'interaction par des schémas. Nous spécifions par des opérations Z, les règles qui régissent l'évolution dynamique des architectures de façon conforme à un style architectural spécifié aussi selon la même technique. Nous illustrons notre approche à travers les propriétés de dynamique des logiciels de support pour les activités coopératives réparties. Pour modéliser l'architecture, nous adoptons une approche orientée composant et nous distinguons différents modes d'interaction (pull/push) entre les différents composants. Notre approche est soutenue par un environnement logiciel de simulation visuelle qui interprète les spécifications Z. Nous dispensons ainsi les concepteurs de la maîtrise de ce formalisme et nous leur offrons une interface "boîte et traits" qui fait référence dans la conception informelle des architectures. Notre environnement de simulation est couplé avec l'outil Z/EVES. Il gère les spécifications Z au format de Z/EVES tout en vérifiant leur complétude et leur exactitude. Il interprète aussi visuellement les spécifications architecturales codées au format Z/EVES.

**ABSTRACT.** In this paper, we deal with the problem of the formal design of the distributed software systems which are dynamically reconfigurable. The architectural reconfiguration is used to complete the often adopted behavioral reconfiguration approaches. This technique allows to dynamically build the architecture of a software system and to adapt it to the supported activity evolution. To formalize the different architecture instances and styles, we specify in Z the software entities and their interactions by schemas. We specify with Z

*operations, the rules which govern the architecture dynamic evolution in conformity with an architectural style also specified with the same technique. We illustrate our approach with a distributed cooperative system. We adopt a component oriented approach and we distinguish different interaction modes (pull/push) between components. Our approach is supported by a visual simulation environment which interprets Z specifications. We offer to designers an interface based on "box and arc" which refers to the architecture informal design. Our simulation environment is coupled with the Z/EVES tool. It manages Z specifications with the Z/EVES format while checking its completeness and its exactitude. It permits also to visually interpret a Z/EVES specification.*

*MOTS-CLÉS: Composant, Architecture Dynamique, Méthode Formelle, Vérification, Simulation, Reconfiguration.*

*KEY WORDS: Component, Dynamic Architecture, Formal method, Verification, Simulation, Reconfiguration.*

---

## **1. Introduction**

La construction des logiciels adaptables, reposant sur des architectures dynamiquement reconfigurables, requiert des règles pour la modification dynamique de la configuration et son déploiement par l'activation et la désactivation de nouveaux composants et services ainsi que la modification de leurs interconnexions conformément aux besoins de l'activité soutenue par ces logiciels. La conception de ce type de logiciels est une tâche complexe, non soutenue par les approches de conception traditionnelles. Elle nécessite une spécification à la fois rigoureuse et maîtrisable par des non spécialistes en techniques de description formelle.

L'approche proposée dans ce papier consolide cette direction et permet d'élaborer des spécifications qui décrivent de façon précise, différents styles architecturaux et différentes architectures dynamiquement adaptables. Nous offrons pour les concepteurs novices en techniques formelles un environnement de conception et de simulation avec une interface de conception visuelle selon les notations standards courantes. L'environnement de simulation permet la création et la modification de différentes instances architecturales en ajoutant/supprimant des composants et/ou des connexions. Nous adoptons la notation Z [SPI 92] pour spécifier les styles architecturaux, les opérations de reconfiguration ainsi que les propriétés architecturales. L'environnement conçu permet de simuler visuellement le comportement d'une application à base de composant conformément aux notations visuelles de UML 2.0 [OMG 03] tout en respectant le style architectural. Via son interface graphique, l'environnement permet aux concepteurs, qui ne maîtrisent pas les techniques de description formelle, d'appréhender eux-mêmes la validation des règles qui régissent la reconfiguration des systèmes qu'ils conçoivent. Pour corriger d'éventuelles erreurs de conception ou de spécification Z, l'outil supporte un cycle de révision et de raffinement jusqu'à l'obtention d'une spécification correcte et complète. Nous avons intégré notre simulateur avec l'outil Z/EVES [ORA ] permettant ainsi d'éditer, d'analyser et de prouver les propriétés architecturales des

styles et architectures décrits en Z. Dans ce papier, nous mettons l'accent sur la conception et l'implémentation de cet environnement de simulation aussi bien que son intégration avec l'outil Z/EVES. Nous illustrons notre approche avec la spécification d'une architecture prouvée correcte d'un système logiciel soutenant l'activité de la Revue Coopérative de documents. Notre travail a servi en préalable à une phase d'implantation de ce logiciel comme un ensemble de services Web dans le cadre du projet européen WS-DIAMOND.

Ce papier est organisé comme suit. La Section 2 offre un survol sur l'état de l'art. La Section 3 présente l'approche adoptée et son application au système de la Revue Coopérative. La Section 4 présente la conception et les fonctionnalités de notre simulateur. La Section 5 décrit les étapes du processus de simulation. Finalement, la Section 6 décrit le bilan et les perspectives de ce travail.

## 2. État de l'art

Dans le contexte de l'ingénierie des besoins, la simulation est définie comme la visualisation du comportement d'une instance de l'architecture du système. Les bénéfices des modèles logiciels sont fixés par les environnements de simulation correspondants. En conséquence, plusieurs recherches sont concentrées dans cet axe. Parmi les solutions industrielles, nous citons en premier lieu ObjectGeode [Objja] qui est un ensemble d'outils commerciaux consacré à l'analyse, la conception, la vérification et la validation par simulation et test d'applications temps-réel. Il supporte une intégration cohérente et complémentaires des approches orientées objet et temps-réel basées sur des standards tel que UML. En second lieu, nous évoquons ObjecTime Développeur [objb] qui est un environnement de modélisation graphique pour la conception orientée objet (avec UML) et la simulation de systèmes temps-réel. En effet, tous ces simulateurs ne supportent pas la simulation d'un modèle à base de composant (UML2.0). En plus, la plupart d'entre eux n'utilisent pas les spécifications formelles comme entrée du processus de simulation. Cependant, l'avantage de notre approche apparaît dans sa capacité de produire un modèle formel présenté à l'utilisateur selon la notation visuelle UML durant le processus de simulation. Ainsi, d'une part, contrairement à une approche utilisant seulement le formel, nous supposons que l'utilisateur de notre environnement ne doit pas être un expert des techniques formelles. D'autre part, nous ne réduisons pas la capacité rigoureuse de notre modèle en utilisant seulement la notation UML.

Il existe plusieurs travaux qui ont essayé d'animer et de simuler des spécifications avec la notation Z, à savoir Jaza (Just Another Z Animator), PiZA (Prolog Implemented Z Animator) et ZANS (Z ANimator System). L'outil Jaza [UTT 05] est utilisé pour évaluer les schémas d'une spécification Z. Cependant, il présente des limites, à savoir : l'absence du traitement de toutes les composantes du langage Z (par exemple les définitions génériques et les déclarations axiomatiques), et l'impossibilité de déclarer les fonctions ou relations in-fixées, pré-fixées et post-fixées. L'outil PiZA [HEW 97] permet la conversion de parties restreintes d'une

spécification Z en Prolog et leur exécution. Toutefois, il est lourd à utiliser, car il nécessite l'installation d'autres logiciels et comporte, au moins, deux étapes de traduction. L'outil ZANS [JIA 95] permet l'évaluation d'expressions et de prédicats et l'exécution de schémas d'opération. Il permet d'animer des sous-ensembles finis ou infinis de spécifications Z.

### 3. Spécification des architectures dynamiques

Notre approche repose sur le même fondement théorique que [LOU 04]. Nous interprétons l'architecture comme un ensemble d'éléments et de relations assimilés respectivement à des noeuds et à des arcs d'un graphe. Selon cette approche, le style architectural d'un système logiciel est décrit par un schéma Z comme suit :

<i>Style</i>
<i>Type de composants</i> <i>Type de relations</i>
<i>Propriétés architecturales</i>

#### Schéma 1: Style architectural

Pour mieux illustrer l'approche adoptée, nous traitons une architecture coopérative dynamique : il s'agit de la *Revue Coopérative* [LOU 06]. La spécification Z associée est décrite par le schéma suivant :

<i>Rev coop</i>
<i>Chairman</i> : F CHAIRMAN <i>Author</i> : F AUTHOR <i>Reviewer</i> : F REVIEWER <i>Notif_Serv</i> : F NOTIF_SERV <i>push_Chairman</i> : CHAIRMAN ↔ NOTIF_SERV <i>pul_Chairman</i> : NOTIF_SERV ↔ CHAIRMAN <i>push_Author</i> : AUTHOR ↔ NOTIF_SERV <i>pull_Author</i> : NOTIF_SERV ↔ AUTHOR <i>push_Reviewer</i> : REVIEWER ↔ NOTIF_SERV <i>pull_Reviewer</i> : NOTIF_SERV ↔ REVIEWER
#Chairman < 3 #Notif_Serv < 1 $\forall x : Author \cdot \#\{(\text{ran}(\{x\} \triangleleft \text{push\_Author}))\} < 3$ $\forall x : Author \cdot \forall y : \text{Notif\_Serv} \cdot (x, y) \in \text{push\_Author} \cdot \#\{(\text{ran}(\{y\} \triangleleft \text{pull\_Reviewer}))\} \geq 2$

#### Schéma 2: Schéma de la Revue Coopérative

Dans la première partie du schéma 2, nous développons une partie déclarative qui précise, dans ce cas, les ensembles de types de composants ainsi que les types de connexions pouvant exister entre eux. Dans cette spécification, *Chairman* représente l'ensemble des présidents de la conférence, *Author* l'ensemble des auteurs ayant des papiers à publier, *Reviewer* l'ensemble des relecteurs qui valident les papiers soumis et *Notif\_Serv* l'ensemble des services de notification. Cette partie intègre aussi un

ensemble de relations représentant les liens de communication entre les composants. C'est ainsi que les auteurs soumettent des papiers au service de notification via la connexion *push\_Author*. À son tour, le service de notification informe le président de la conférence via la connexion *pull\_Chairman*. Chaque membre de la comité de lecture est alors informé, par e-mail de la part du président, de la liste des papiers qu'il doit évaluer (la connexion *push\_Chairman* du président vers le service de notification, et puis vers les relecteurs à travers la connexion *pull\_Reviewer*). Après, les relecteurs doivent télécharger les papiers du service de notification et soumettre, ensuite, les rapports contenant les revues au service via la connexion *push\_Reviewer*. Le président de la conférence sera informé par la soumission des rapports et il doit les télécharger du service de notification. Par la suite, les auteurs seront notifiés de la décision via la connexion *pull\_Author*.

La deuxième partie du schéma *Rev\_coop* (cf. schéma 2) décrit les propriétés architecturales du système qui doivent être toujours respectées. Par exemple, le premier prédicat stipule que le système doit contenir au maximum 3 présidents. Le quatrième précise que le nombre de relecteurs pour un papier est au minimum 2.

Selon l'approche présentée dans [LOU 04], la dynamique de l'architecture est décrite à l'aide de schémas  $\Delta$  de Z (cf. Schéma 3). En effet, chaque schéma  $\Delta$  représente une opération de reconfiguration.

<p><i>Nom_opération</i></p> <p><math>\Delta</math>Style</p> <p><math>Par_1 ? ; Par_2 ? ; \dots ; Par_n ?</math></p>
<p><i>Pré-conditions</i></p> <p><i>Post-conditions</i></p>

**Schéma 3:** Schéma  $\Delta$

La spécification de la Revue Coopérative contient différentes opérations de reconfiguration permettant de faire évoluer le système tout en tenant compte des règles architecturales décrites dans le schéma *Rev\_coop*. Par exemple, le schéma *insert\_CHAIRMAN* (cf. Schéma 4) exprime une opération d'insertion d'une instance d'un composant de type *CHAIRMAN*, à condition que le système ne contient pas déjà trois présidents.

<p><i>insert_CHAIRMAN</i></p> <p><math>\Delta</math>Rev_coop</p> <p><math>x? : CHAIRMAN</math></p>
<p><math>\#Chairman \leq 3</math></p> <p><math>Chairman' = Chairman \cup \{x?\}</math></p>

**Schéma 4 :** Insertion d'un nouveau président

Nous avons utilisé l'outil Z/EVES [ORA ] pour éditer et vérifier la syntaxe et les types de la spécification de la Revue Coopérative ainsi que prouver que l'opération de reconfiguration indiquée préserve le style architectural.

#### 4. Le simulateur : présentation et fonctionnalités

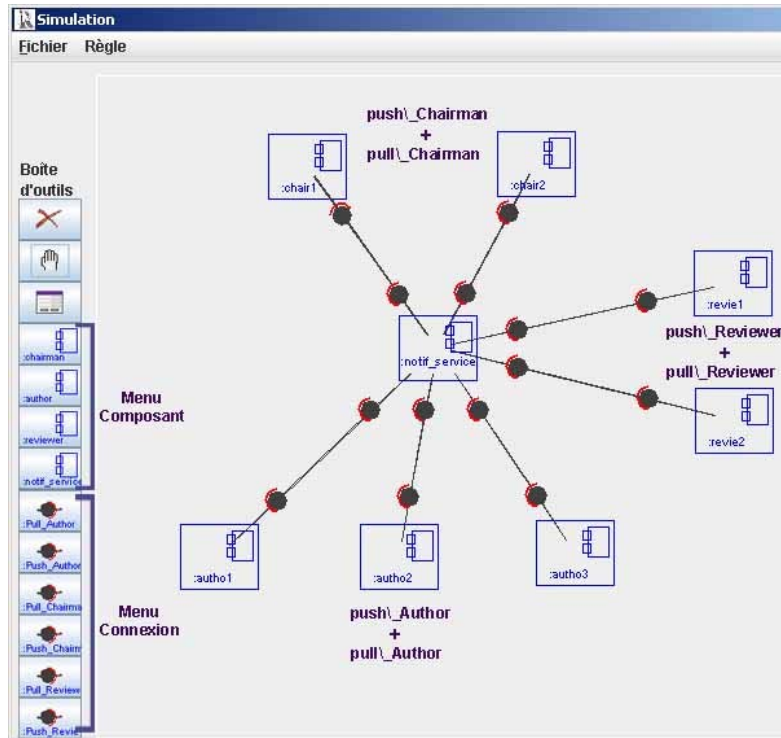


Figure 1. L'interface graphique du simulateur

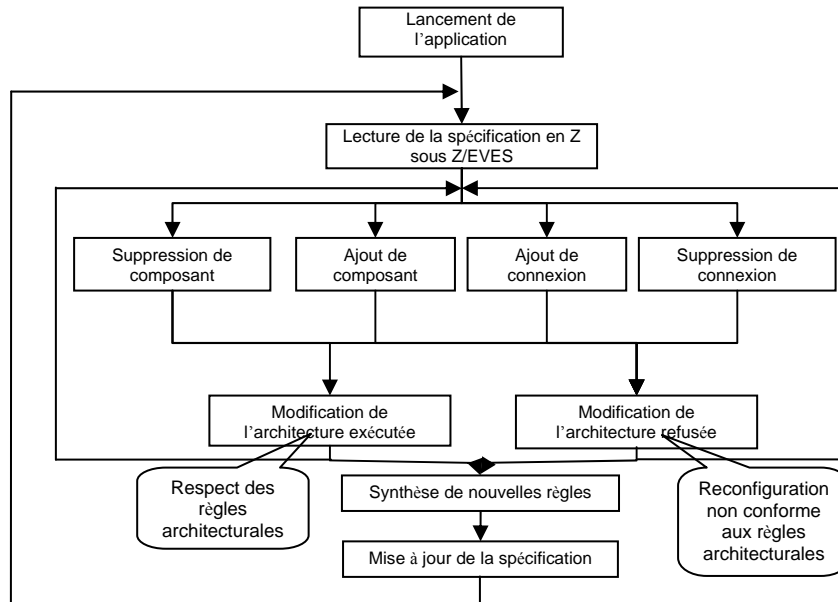
Le simulateur est une application générique de simulation du mode de fonctionnement des applications à base de composants. Il représente les composants et les connexions en cohérence avec les conventions de la notation visuelle UML 2.0. Il fournit de plus la possibilité d'ajouter et/ou de supprimer des composants et des connexions. Chaque action de reconfiguration (ajout/suppression) se déroule en respectant sa description en Z, comme décrit par l'approche présentée dans la section 3. L'interface principale (cf. figure 1) contient des menus, une boîte d'outils et un panneau de dessin.

Nous illustrons dans la figure 2 les différents scénarios possibles de fonctionnement de notre environnement de simulation.

#### 5. Couplage du simulateur avec Z/EVES

La première étape de notre processus de simulation est la spécification de l'architecture comme décrit dans la section 3. La deuxième étape est la simulation visuelle avec la notation UML2.0. Le processus de simulation consiste à valider des

Une approche orientée règle pour la spécification formelle  
des architectures dynamiquement configurables



**Figure 2.** Le scénario de fonctionnement du simulateur

spécifications formelles des styles architecturaux. Cette validation englobe trois propriétés, à savoir : la complétude de la spécification, sa conformité aux besoins de l'utilisateur ainsi que sa consistance. La complétude nous garantit que toutes les propriétés données dans les besoins ont été formulées dans la spécification. Tandis que la conformité assure que chaque propriété a été formulée exactement comme le besoin de l'utilisateur. Nous utilisons Z/EVES afin d'aider les utilisateurs à éditer et vérifier la spécification. Avec le "theorem prover" de Z/EVES, nous prouvons les systèmes spécifiés en Z. Donc, nous vérifions la consistance du style architectural. Pendant la deuxième étape, les utilisateurs peuvent concevoir l'architecture avec la notation UML 2.0. Ainsi, nous pouvons vérifier la conformité du diagramme UML conçu avec la spécification des besoins. De plus, nous fournissons la possibilité de rectifier la spécification Z à tout moment sans suspendre le processus de simulation. Nous terminons la modification quand nous obtenons une spécification complète.

Notre environnement procède à identifier (dans la spécification Z) les types de composant et de connexion ainsi que les règles architecturales. En effet, pour chaque composant correspond un objet Java, et pour chaque connexion correspond une méthode. Par exemple, la méthode "M(obj1, obj2,...)", dessine un trait entre deux composants. Ce lien, de obj1 (composant1) vers obj2 (composant2) est enregistré dans un vecteur dans les objets déjà cités, comme suit : le vecteur "out" dans obj1, et le vecteur "in" dans obj2. Ces vecteurs seront utiles pour l'application des règles et la vérification durant la procédure de simulation. Les règles sont de trois types:

- Règles appliquées à un composant, qui seront représentées par des attributs de l'objet Java représentant ce composant. Exemple de règles : limitation de nombre de connexions avec les autres composants (Règle 3, schéma 2).
- Règles appliquées à l'architecture globale, qui seront placées dans le canevas qui contient les composants et les méthodes représentant les connexions. Exemple de règles : limitation du nombre d'instances d'un composant (Règle 1, schéma 2).
- Règles appliquées aux connexions, qui seront exprimées par des variables locales aux méthodes représentant les connexions. Exemple de règles : interdiction des connexions non définies dans la spécification Z.

Si le l'utilisateur n'obéit pas aux règles, le simulateur l'interdit tout en affichant un message d'erreur. Par exemple, quand nous partons d'une configuration contenant trois présidents et nous voulons ajouter un quatrième, alors cette reconfiguration est ignorée (# Chairman < 3, dans le schéma 2).

## 6. Conclusion

Dans ce papier, nous présentons un environnement logiciel pour la simulation, selon les notations UML, de l'évolution d'une architecture spécifiée en Z. Notre approche vise à fournir aux concepteurs un environnement visuel permettant la spécification formelle d'architectures dynamiques correctes et complètes. La représentation graphique, que notre simulateur produit, établit une traduction automatique d'une spécification formelle en Z vers la notation visuelle UML 2.0.

## 7. Bibliographie

- [BEN 05] BEN HALIMA R., JMAIEL M., DRIRA K., « Graphical simulation of the dynamic evolution of the software architectures specified in Z. », *8th International Workshop on Principles of Software Evolution (IWPSE 2005)*, 5-6 September 2005, Lisbon, Portugal, IEEE Computer Society, 2005, p. 45–48.
- [HEW 97] HEWITT M. A., O'HALLORAN C., SENNETT C. T., « Experiences with PiZA, an Animator for Z », *ZUM '97 : Proceedings of the 10th International Conference of Z Users on The Z Formal Specification Notation*, Springer-Verlag, 1997, p. 37–51.
- [JIA 95] JIA X., « An Approach to Animating Z Specifications », *COMPSAC '95 : Proceedings of the 19th International Computer Software and Applications Conference*, IEEE Computer Society, 1995, page 108.
- [LOU 04] LOULOU I., KACEM A. H., JMAIEL M., DRIRA K., « Toward a unified graph-based framework for dynamic component-based architectures description in Z », *ACS/IEEE International Conference on Pervasive Services ICPS'04*, 2004.
- [LOU 06] LOULOU I., KACEM A. H., JMAIEL M., « Compositional specification of event-based software architectural styles », *The 4th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-06)*, 2006.
- [Obja] « ObjectGeode », available at [www.telelogic.com/products/additional/objectgeode](http://www.telelogic.com/products/additional/objectgeode)
- [objb] « Objecttime », available at <http://www.objecttime.com/>.
- [OMG 03] OMG, « UML Superstructure 2.0 -Draft Adopted Specification », , 2003.
- [ORA ] ORA, « Z/EVES », <http://www.ora.on.ca/Z-eves>.
- [SPI 92] SPIVEY J., *The Z Notation: a Reference Manual*, Prentice-Hall, 1992
- [UTT 05] UTTING M., « Jaza User Manual and Tutorial », <http://www.cs.waikato.ac.nz/>