REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE M'HAMED BOUGARA - BOUMERDES
FACULTE DES SCIENCES
DEPARTEMENT D'INFORMATIQUE

LABORATOIRE D'INFORMATIQUE FONDAMENTALE APPLIQUEE

Mémoire de magistère

THEME

Une approche de modélisation de procédés logiciels à base de composants

Présenté par
Mr BELKASMI Djamel

Soutenu publiquement le **08/11/2006** à **9h30'** au département d'informatique **Salle: 111**

Devant le jury composé de :

Mr. MEZGHICHE Mohamed	(Professeur U.M.B.B)	Président
Mr. AHMED NACER Mohamed	(Professeur U.S.T.H.B)	Rapporteur
Mme. ALIMAZIGHI Zaia	(Maître de conférence U.S.T.H.B)	Examinatrice
Mr. AHMED OUAMER Rachid	(Maître de conférence U.M.M.TO)	Examinateur
Mr. HARZELLAH Abdelkrim	(Docteur U.M.B.B)	Examinateur

ANNEE -2006-

Résumé

Le domaine des procédés logiciels est vaste et les procédés logiciels sont intrinsèquement complexes. De nombreux travaux de recherches poursuivent leurs efforts pour caractériser et pour mieux cerner les procédés. Des approches ont été proposées pour les modéliser et des environnements ont été développés pour les supporter.

Dans ce travail, Nous proposons une approche de modélisation de procédés logiciels à base de composants permettant à des équipes distribuées de coopérer dans la modélisation et l'exécution des procédés en utilisant des environnements hétérogènes.

Un FrameWork basé sur le langage XML a été développé pour supporter l'approche proposée.

Mots clés

Procédé logiciel, modèle de procédé logiciel, environnement centré procédé logiciel, composant procédé.

Abstract

The software process domain is vast and software processes are intrinsically complex. Many research works maintain their efforts to model and define these processes by proposing several approaches and environments to support them.

In our work, we propose an approach for modelling software process using components in order to allow distributed teams to cooperate when modelling and executing process using heterogeneous environments.

A Framework based on the XML language was developed to support the proposed approach.

Keywords

Software process, Software process model, software process centred environment, process component.

ملخص

إن مجال كيفية تصنيع البرامج واسع وتعتبر طرق البرمجة أكثر تعقيدا. عدة أبحاث لتزال تبذل جهود لإعداد نماذج و حصر هذه الطرق. لقد تم اقتراح عدة وجهات نظر من أجل صياغة هذه النماذج وأنجزت عدة أنظمة لدعمها.

في هذه الأطروحة, نقترح وجهة نظر جديدة من أجل تعيين طرق تصنيع البرامج مبنية على أساس المركبات. هذه النظرية توفر مجال عمل وتعاون من أجل تعيين وتنفيذ أساليب تصنيع البرامج لفرق موزعة جغرافيا على شبكة الانترنيت مستعملتا أنظمة غير متجانسة.

لقد تم تطوير نظام مشتق من لغة XML لدعم النظرية المقترحة.

مفاتيح

كيفية البرمجة, نماذج كيفية البرمجة, أنظمة مركزة على كيفية البرمجة, مركب إجراء.

Remerciements

Je tiens à remercier

Monsieur **MEZGHICHE Mohamed**, Directeur du laboratoire LIFAB, Professeur à l'université M'hamed Bouguara de Boumerdes, pour m'avoir fait l'honneur de présider le jury de ce mémoire.

Madame **ALIMAZIGHI Zaia**, Maître de conférence à l'université des sciences et de la technologie Houari Boumedienne de Bab-ezzouar, Monsieur **AHMED OUAMER Rachid**, Maître de conférence à l'université de Tizi-ouzou, et Monsieur **HARZELLAH Abdelkrim**, Docteur à l'université M'hamed Bouguara de Boumerdes, pour avoir accepté de lire et de juger ce mémoire.

Monsieur **AHMED-NACER Mohamed**, Professeur à l'université des sciences et de la technologie Houari Boumedienne de Bab-ezzouar, mon directeur de mémoire, pour m'avoir accueilli au sein de son équipe depuis ma 2ème année de magistère. Je tiens à lui témoigner ma reconnaissance pour m'avoir permis de travailler dans de bonnes conditions. Son expérience, ses commentaires, ses conseils et ses remarques, m'ont été très bénéfiques.

Tous les membres du laboratoire LIFAB : **IMACHE, BERRICHI, MERHOUM, MAOUCHE, SIACI, CHAABANI, BADAOUI, TOUIL (G), TOUIL (M) et NADIR** ... Merci pour les encouragements et pour la bonne ambiance.

L'équipe administrative du département d'informatique.

Finalement, ou plutôt avant tout, je tiens à remercier toute ma famille, qui a su me soutenir durant toutes ces longues années. Ce travail n'aurait pu aboutir sans son amour et ses encouragements.

A vous tous, un grand merci.

A mes très chers parents…
A celle qui occupe mes pensés…

Table des matières

Chapitre 1: Introduction générale.

Chapitre 2: Environnements centrés Procédés logiciels.

Chapitre 3: Une architecture de modélisation de procédés à base de composants.

Chapitre 4: Expérimentation de l'approche

Chapitre 5: Conclusion générale et perspective.

Liste des figures

Chapitre -1-

Introduction générale

1. Introduction.

L'objectif des organisations fabricants des logiciels est de produire de façon systématique et prévisible des logiciels de bonne qualité en réduisant les délais et les coûts. A cet effet, le meilleur atout de ce type d'organisation reste son *savoir-faire* ; c'est-à-dire les procédés de fabrication de logiciels (*software process)* qu'elle met en oeuvre.

En effet, la *maturité* de toute organisation est en relation directe avec la capacité de ses procédés à produire les résultats prévus. Pour les développeurs de logiciels, acquérir une telle *maturité* est une mission difficile qui peut prendre beaucoup de temps et de ressources. La difficulté réside dans le fait que la plupart des activités constituant les procédés de fabrication de logiciel ne sont pas automatisables (*human-intensive*) et, par conséquent, non déterministes car les interactions entre humains, ainsi qu'entre humains et outils, sont très variables et souvent imprévisibles [CONRADI et al. 92].

Ces dernières années, les procédés de fabrication de logiciels ont fait l'objet d'une attention grandissante dans le domaine du génie logiciel. Il est désormais admis que la qualité des logiciels peut être améliorée si les procédés de fabrication sont eux-mêmes améliorés. Les fabricants de logiciels commencent aussi à admettre que l'évaluation, et donc la *mesure,* est une tâche indispensable pour obtenir une amélioration systématique des procédés de fabrication**.**

Les recherches autour de procédés logiciels ont abouti, d'une part, à de nombreuses études et propositions de modélisation de procédés et, d'autre part, à la définition d'environnements guidés par les procédés [FINKELSTEIN et al. 94]**.** Les modèles de procédés déterminent et intègrent l'utilisation des services de l'environnement et guident les tâches des utilisateurs. Depuis la naissance du génie logiciel, il y a eu des progrès significatifs en ce qui concerne les méthodologies, les procédures et les outils permettant d'assister les procédés de fabrication de logiciels. Cependant, la maîtrise de ces procédés est encore loin d'être acquise. De nos jours, un projet de fabrication de logiciel est rarement réalisé dans les délais et avec l'affectation des ressources prévues. Une grande partie du problème est due à l'immaturité des organisations qui éprouvent de grandes difficultés à définir clairement leurs procédés de fabrication de logiciel et à les mettre en oeuvre de façon systématique avec les outils et le support adéquats**.**

Le coût total des ressources humaines dans la production mondiale de logiciels a été évalué à environ $250 milliards par année [FUGGETTA et al. 93]. Ce chiffre ainsi que l'importance des systèmes informatiques actuels, justifient largement tous les efforts visant à maîtriser les procédés de fabrication de logiciels.

2. Problématique.

Actuellement les procédés mis en œuvre dans notre société sont de plus en plus complexes et difficiles à gérer; la raison principale est l'évolution rapide des activités sociales et économiques durant les dernières décennies, en plus de la diffusion de la technologie de l'information dans beaucoup de domaines [AVRILIONIS 96]. En conséquence, la conception, l'exécution et la maintenance des modèles pour ces procédés sont de plus en plus complexes. Nous nous intéressons plus particulièrement aux procédés logiciels et à leur modélisation pour faire face à la complexité des procédés, la communauté du génie logiciel propose leur construction en faisant coopérer plusieurs équipes (peut-être géographiquement distribuées). En considérant que chaque équipe peut avoir ses propres environnements de développement (formalismes de représentation, moteur procédé, bases de données etc.), les modèles fournis par ces équipes seront hétérogènes. Et malheureusement, les formalismes de représentation actuels liés à l'infrastructure qui les supporte ne facilitent ni l'interopérabilité, ni la réutilisation de modèles procédés exécutables.

Comme nous le présentons dans l'état de l'art, les travaux cherchant à résoudre le problème de l'interopérabilité sont nombreux. Une approche intéressante consiste à appliquer la technologie des composants et objets au développement de procédés. Parmi ces travaux on trouve **APPEL, PIE** et **OZ**, qui proposent une architecture par composant et **PYNODE**, **ENDEAVORS** et **OPC**, qui proposent une approche par composant au niveau modélisation.

Les avantages attendus par l'approche composant pour le support de procédés sont :

- Obtention d'environnements de développement de logiciels coopératifs et ouverts,
- Interopérabilité entre des composants hétérogènes (fédération de composants),
- Dépendances entre procédés délimitées, mieux reconnues et gérées,
- Réutilisation des composants procédés.

3. Contexte.

Notre domaine de recherche s'inscrit dans une approche par composant pour le support automatisé aux procédés logiciels. Nous proposons une architecture orientée objet permettant de construire, d'exécuter et de faire évoluer des modèles de procédés logiciels. Le contexte de ce travail est caractérisé par l'étude des différents environnements de modélisation de procédés logiciels ainsi que la fédération d'environnements hétérogènes distribués, sans oublier d'approfondir le concept de composant procédé logiciel.

Les buts de notre travail sont :

- La présentation des concepts et des terminologies caractérisant la technologie des procédés logiciels.
- La proposition d'une architecture orientée objet pour le support de procédés qui permette :
    - De modéliser et d'augmenter le potentiel de réutilisation des modèles de procédés,
    - D'intégrer dynamiquement les composants.
    - D'assurer l'exécution distribuée des procédés entre composants hétérogènes.
- Le développement d'un prototype pour valider nos idées, en utilisant des technologies actuelles telles que Java, XML, protocoles de communication comme CORBA, etc.

4. Organisation du mémoire.

Le document est organisé en trois parties.

La première partie porte sur l'état de l'art :

- Nous focaliserons notre étude sur le domaine des procédés logiciels, leurs supports appelés environnements de génie logiciel centrés procédés que nous étudierons particulièrement sous l'angle des fédérations (*chapitre 2*),

La deuxième partie aura pour propos la présentation d'une architecture orientée objet pour la modélisation des procédés logiciels.

- Dans un premier lieu, nous exposerons les concepts et la terminologie de l'architecture proposée (*chapitre 3*),
- Par la suite, nous présenterons le prototype permettant la mise en oeuvre du modèle de composants procédé. Nous donnerons les interfaces de dialogue pour la création d'un modèle par composants, et pour la gestion de l'exécution de chacun des composants procédés (*chapitre 4*).

Enfin, le dernier chapitre présentera les conclusions ainsi que les perspectives de ce travail.

Chapitre -2-

Environnements centrés procédés logiciels

1. Introduction.

Il est connu et approuvé que la qualité des procédés logiciels détermine la qualité du produit à réaliser. On désigne, par *procédé logiciel*, l'ensemble des activités impliquant des équipes de personnes (souvent nombreuses), des techniques et des outils, dans le but d'assurer le développement et la maintenance des systèmes logiciels. Ces activités sont de natures différentes résumées par:

- Spécification du système,
- Analyse et gestion des risques,
- Conception,
- Prototypage,
- Implantation,
- Validation,
- Tests,
- Contrôle de qualité,
- Maintenance, …etc.

L'enjeu des sociétés productrices de logiciels est d'arriver à structurer et organiser les développements, de mettre en place les moyens et les outils les plus adéquats et d'engager les bonnes pratiques. La représentation du procédé sous forme de modèle est une première action d'amélioration, qui consiste à:

- Spécifier les étapes du procédé de développement,
- Préciser les intervenants, leurs missions et leurs activités.

Ce qui rend les procédés logiciels difficiles à gérer, c'est l'ampleur grandissante de leur complexité. Pour mieux comprendre cette complexité, nous allons présenter le procédé logiciel suivant deux dimensions (voir figure 2.1) :

1. *Une dimension horizontale* met l'accent sur la diversité des aspects mis en jeu. Il s'agit de décrire et gérer,
   - Les activités et les produits, Les ressources et les aspects organisationnels,
   - Le travail coopératif (la coordination, la communication et la collaboration),

- L'interaction des utilisateurs,
- L'évolution du procédé (afin de refléter les modifications non prévues au départ),
- Les aspect liés à la distribution et le pilotage.

2. *La dimension verticale* expose la diversité des niveaux d'abstraction. Par conséquent, les procédés logiciels ont une durée de vie longue (de quelques semaines à plusieurs mois). Ils englobent les phases suivantes:
   - L'analyse des besoins (qu'est ce qu'on veut ?),
   - La conception (Formaliser les besoins),
   - L'implantation (Implémentation des besoins).

   Ces phases impliquent l'intervention et la participation des différentes catégories d'intervenants qui sont:
   - Les responsables et gestionnaires administratifs,
   - Les ingénieurs,
   - Les programmeurs et les techniciens, …etc.

   Chaque catégorie d'intervenants expose des souhaits et des attentes différents. Un support fiable et efficace doit proposer des mécanismes et des moyens convenables à chacune des phases et à chaque catégorie d'intervenants.

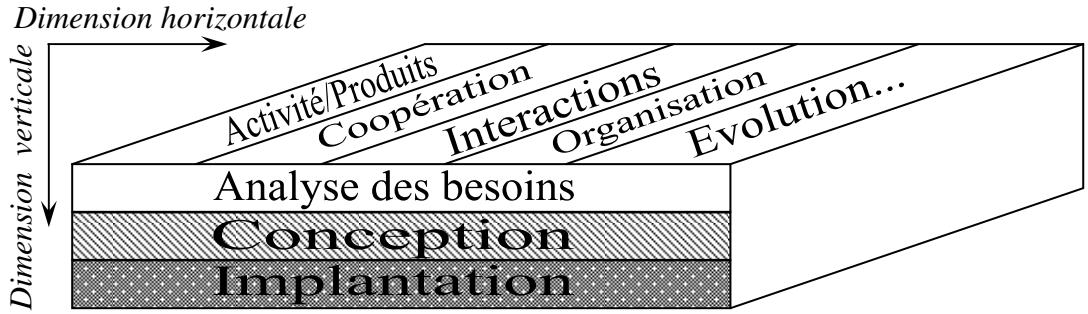


Figure 2.1 : *Les deux dimensions d'un procédé logiciel* [AMIOUR 99].

Après une introduction sur les procédés logiciels, nous proposons dans ce chapitre un état de l'art sur les environnements centrés procédés logiciels.

L'objectif est avant tout de voir leurs diversités, les problèmes auxquels ils sont confrontés, pour permettre de comprendre en quoi la direction prise au cours des dernières années n'a d'autres objectifs que de contourner les limitations des environnements de première génération.

2. Les procédés logiciels.

2.1. Introduction.

Bien que les fondements en matière d'automatisation remontent aux premiers langages de programmation, les technologies du procédé logiciel n'ont vraiment connu un essor que ces trente dernières années avec l'arrivée d'outils et d'environnements de génie logiciel.

C'est à la fin des années soixante qu'un phénomène précipita les choses : "*la crise du logiciel*".

C'est alors que de vastes programmes furent entrepris dont les objectifs étaient d'améliorer la fabrication des logiciels que ce soit en termes de coûts, de délais, de qualité, de technologies utilisées et de productivité, c'est-à-dire l'ensemble des paramètres entrant en jeu dans la chaîne de production des logiciels. C'est ainsi qu'on vit apparaître le concept environnement de génie logiciel.

Peu à peu, la programmation prit de l'ampleur; à la fois par la taille des systèmes informatiques à développer mais aussi des moyens humains mis en jeu, qui étaient, au départ, limités *"programming in the small"*, vers la réalisation de systèmes plus complexes *"programming in the large"*, nécessitant davantage de moyens humains et technologiques *"programming in the many"*. Ainsi, en réponse à la fois aux exigences de la complexification et de la quantité de logiciels à produire (besoin d'automatisation), mais également en réponse aux besoins de rationaliser la production de logiciels et d'accroître leur qualité, les premiers environnements de génie logiciel sont apparus.

[SOMMERVILLE 89] définit un environnement de génie logiciel comme étant un ensemble intégré d'outils et de mécanismes permettant de supporter toutes les phases de développement du logiciel (analyse, conception, écriture de code, test, etc.).

Des travaux se sont donc intéressés à définir, modéliser le logiciel et les étapes conduisant à sa fabrication, tant les facettes sont nombreuses (activités humaines, outils, automatisation, coopération, évolution, gestion des ressources et des produits, etc.). Parmi ces travaux sur les procédés logiciels, on trouve les modèles de cycle de vie du logiciel, les environnements intégrés de génie logiciel et les environnements centrés procédé logiciels.

2.2. Les procédés logiciels.

La manière dont le développement de logiciel est organisé, contrôlé, mesuré, supporté et amélioré (indépendamment du type de support technologique utilisé dans le développement) au sein d'une entreprise constitue le procédé logiciel de cette dernière [PROMOTER 99].

Même si elles exhibent des niveaux différents de sophistication dans la maîtrise de leurs procédés, toutes les organisations développant du logiciel suivent un procédé, qu'il soit implicite ou explicite, reproductible, adaptable ou autre.

Un procédé implique des personnes, des technologies, plus généralement un ensemble d'artefacts. Tout naturellement, un procédé logiciel définit un ensemble d'artefacts propres au domaine du génie logiciel. On parlera d'activités, d'étapes, de tâches, d'acteurs, de participants, d'outils, etc. dont la sémantique est propre au domaine. Dans le domaine du logiciel, [AVRILIONIS 96] a proposé un panel de définitions des termes couramment employés, tout comme [CONRADI et al. 92], [LONGCHAMP 93] qui ont proposé des définitions génériques pour les concepts abordés dans le domaine des procédé logiciels.

L'idée de procédé (ou processus) est liée au fait que l'humain résout les problèmes auxquels il est confronté en créant des descriptions de procédés appelés modèles de procédés. Il décline, pour chaque problème particulier un procédé particulier à partir d'une description qui se veut générique : il s'agit d'une instance. Puis, chaque instance peut être exécutée [OSTERWEIL 87].

Il est bien entendu que, pour la plupart des organisations attachées à leurs méthodes de travail et leurs visions de leurs procédés, la mise en oeuvre de ces derniers doivent être rationalisés le plus possible (réduction des coûts, possible rétroaction, etc.). Tout l'intérêt est donc de définir explicitement et aussi finement que possible les artefacts qui le composent afin de pouvoir reproduire, avec un maximum de généricité, le procédé ainsi défini.

2.3. Les modèles de cycle de vie.

Ce sont les premiers travaux visant à caractériser le procédé logiciel, c'est-à-dire à décrire l'enchaînement des étapes de la conception à la maintenance d'un produit logiciel : son cycle de vie. Les modèles de cycle de vie les plus connus sont ceux dits:

1. "en cascade" [ROYCE 70] et
2. "en spirale" [BOEHM 81].

Ces modèles ont permis de mieux comprendre le procédé logiciel par la description des activités abstraites et de leurs ordonnancements. Cependant ces modèles ne tiennent pas compte de l'activation, de l'échec ou au contraire du succès des activités et, en général, de la synchronisation des activités entre elles, pas plus qu'ils ne s'intéressent aux artefacts manipulés par ces activités (les ressources, l'organisation, etc.).

Toute autre information nécessaire à la compréhension du procédé qui ne peut être décrite par ces modèles fait généralement l'objet d'une description informelle à l'intérieur d'un document accompagnant le modèle.

2.4. La maîtrise des procédés, la qualité du logiciel.

Les procédés logiciels, bien que différents suivant l'organisation qui va les définir et les mettre en oeuvre, n'en possèdent pas moins un caractère générique. La qualité du logiciel obtenue n'est pas seulement liée au produit, mais également à l'organisation et au procédé de production qui est employé [DERNIAME et al. 94]. Des méthodes ont été proposées favorisant la compréhension générale du procédé et permettant aux organisations qui mettent en oeuvre des procédés logiciels de mieux les maîtriser, de les évaluer afin de pouvoir les améliorer (ce qui améliore le produit logiciel résultant).

Ainsi, plusieurs approches permettant de mesurer la maturité du procédé ont été proposées:

1. Le *Modèle d'Evaluation des Capacités du Logiciel* (en anglais *Capability Maturity Model* ou *CMM*) développé par **S.E.I** (*Software Engineering Institute*) au Carnagie-Mellon University [PAULK et al. 91] et,
2. Ses contreparties européennes telles que *Bootstrap* [KUVAJA 94] et *SPICE* [ROUT 95] supportés par **E.S.I** (*European Software Institute*), à Bilbao.

Ces efforts reflètent une évolution du concept de qualité du logiciel vers des environnements centrés procédés. Derrière cette évolution se trouve la conviction que l'amélioration de la qualité ainsi que la réduction des coûts sont mieux servies en certifiant des procédés et en faisant en sorte que ces procédés soient suivis.

Toutes ces méthodes n'ont d'autre but que d'évaluer les procédés mis en oeuvre au sein des organisations et de proposer des actions pour l'amélioration des procédés.

2.5. Les environnements intégrés de génie logiciel

Afin de remédier aux insuffisances des modèles de cycle de vie plusieurs travaux se sont succédés, **MAKE** [FELDMAN 79], **SMALTALK** [GOLDBERG 84], **INTERLIS** [TEITLEMAN et MASINTER 84], etc. visant à offrir un meilleur support au développement de logiciels. Ces travaux ont été dirigés vers le développement d'une nouvelle génération d'outils. Il s'agit des *environnements intégrés de génie logiciel***.** Cette notion se réfère à des collections intégrées destinées à assister la production de logiciel.

Cependant et malgré le succès de ce genre d'environnements, la dimension procédé n'est pas réellement abordée dans le sens où ni la définition, ni l'enchaînement des activités ne sont proposées (la façon dont le produit logiciel est fabriqué n'est pas explicité).

Il s'agissait en fait de proposer aux développeurs, sous une forme homogène et intégrée, un ensemble de fonctionnalités nouvelles reposant sur l'intégration d'outils souvent existants et

connus des développeurs (compilateurs, éditeurs de liens, gestionnaires de fichiers, etc.). L'aspect intégré permettait d'utiliser cette collection d'outils dans la perspective de produire des logiciels avec davantage de facilité. Pour remédier aux limitations des environnements intégrés, et dans le but de prendre en compte le procédé logiciel, la deuxième génération d'environnements sera présentée dans la section consacrée aux *Environnements de Génie Logiciel Centrés Procédé (EGLCP).*

3. Les environnements centrés procédés.

3.1. Introduction.

Les approches et travaux présentés dans la section précédente révèlent une limitation majeure résumée par:

1. La rigidité de la méthodologie proposée (modèles de cycle de vie),
2. L'intégration d'outils et des technologies logicielles sans méthodologie précise.

L'un des objectifs majeurs des *Environnements de Génie Logiciel Centrés Procédé (EGLCP)* est justement de fournir aux utilisateurs à la fois certaines des technologies de production du logiciel tout en intégrant une partie de la gestion de cette dernière dans un environnement relativement intuitif.

Au cours de ces dix dernières années, de nombreux travaux se sont focalisés sur les environnements centrés procédé logiciels :

MARVEL	[BEN-SHAUL et KAISER 93]
MERLIN	[JUNKERMANN et al. 94]
PROCESS WISE	[BRUYNOOGHE et al. 94]
OZ	[BEN-SHAUL et KAISER 95]
ARCADIA	[TAYLOR et al. 88]
PEACE	[ARBAOUI et al. 92]
ALF	[CANALS et al. 94]
SCALE	[OQUENDO 95]
PEACE+	[ALLOUI et al. 94]
OIKOS	[AMBRIOLA et al. 90]
EPOS	[CONRADI et al. 94]
SPADE	[BANDINELLI et al. 92]
JIL	[SUTTON et al. 97]
TEMPO	[BELKHATIR et MELO 92]
ADELE	[ESTUBLIER et CASALAS 94]
APEL	[ESTUBLIER et al. 98A]

3.2. Qu'est-ce qu'un EGLCP?

Par définition, un environnement centré procédé est un système dans lequel la manière dont le logiciel est fabriqué (doit être fabriqué) est définie de façon explicite et avec suffisamment de détails. Cette description, appelée le *modèle du procédé*, est exprimée dans un formalisme convenable appelé *langage de modélisation de procédés*.

Le langage offre des concepts pour décrire les rôles, les humains, les activités, les produits manipulés, les contraintes, etc. Le modèle du procédé étant au coeur de l'environnement, ce dernier se charge de son interprétation afin de guider les utilisateurs, les assister et automatiser les tâches (voir figure 2.2).

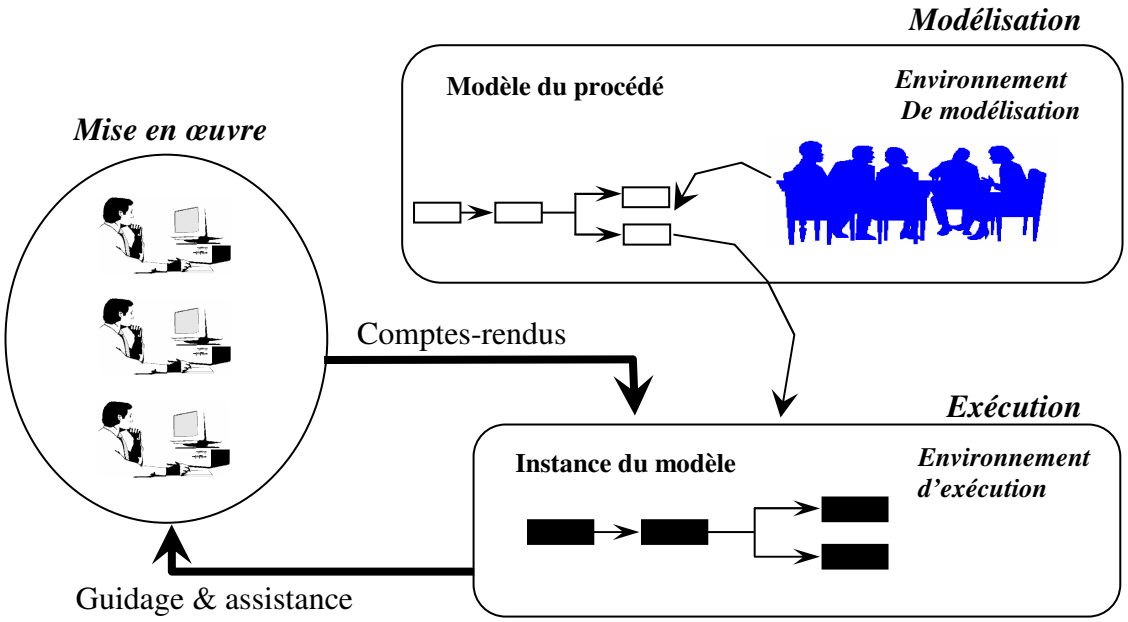


Figure 2.2 : *Principe des environnements centrés procédés logiciels* [AMIOUR 99].

Un modèle de procédé est une abstraction du procédé logiciel décrit de manière formelle ou semi-formelle. Un modèle de procédé comprend:

1. La description des ressources (outils, acteurs, etc.) que requiert le procédé ;
2. Les activités et les tâches pour lesquelles le procédé est défini et structuré ;
3. L'enchaînement (ou ordonnancement) de ces activités ou tâches ;
4. Les informations nécessaires à la définition du procédé.

Le modèle de procédé fixe les fondements nécessaires à la coopération et à la communication entre les différentes entités participant au procédé. Plusieurs sous-modèles ont été identifiés [PROMOTER 99], qui ne font pas tous l'objet de consensus :

1. Un modèle d'activité exprimant les activités simples et complexes (agrégation d'activités simples) décrit à l'aide d'un formalisme particulier ;
2. Un modèle de produit exprimant les données qui sont manipulées par les activités ;
3. Un modèle d'outils pour décrire les outils utilisés dans le cadre du procédé et leurs architectures. Cela peut être réalisé également par le modèle d'activités, en considérant l'outil comme l'enveloppe d'une activité ;
4. Un modèle organisationnel qui décrit la structure et contrôle les activités ainsi que leurs ressources ;
5. Un modèle utilisateur exprimant la manière dont les différents acteurs tirent partie de l'assistance fournie au travers du support technologique du procédé, leurs responsabilités dans le contexte du procédé, etc.

Cette description diffère légèrement de celle proposée dans [PROMOTER 99] voir figure 2.3. Nous attribuons cette différence au fait que ces modèles concernent différentes vues du procédé sous-jacent.

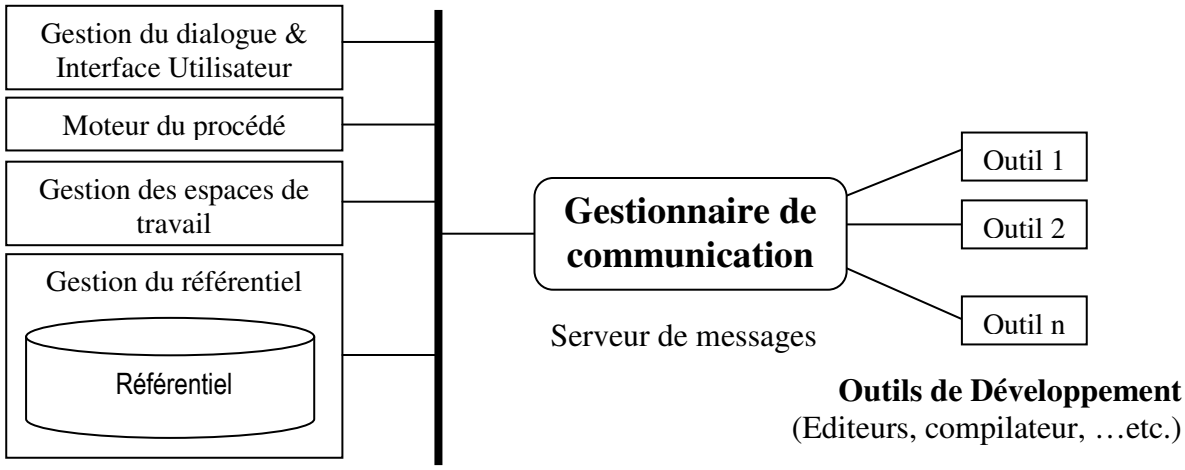


Figure 2.3 : *Architecture de référence de PROMOTER* [PROMOTER 99].

Les modèles peuvent se chevaucher dans les artefacts qu'ils utilisent/exploitent ou, au contraire, se compléter. La question cruciale qui demeure est de gérer la cohérence entre l'ensemble des vues d'un même procédé.

3.3. Les différents formalismes.

Plusieurs approches ont été à l'origine de formalismes différents. Entre autres nous notons les langages de programmation classiques, les formalismes de modélisation comme les réseaux de Pétri, etc. Parmi les approches et les langages de modélisation de procédé qui en découlent on peut noter [AMIOUR 99] :

1. L'approche procédurale illustrée dans les systèmes **ARCADIA, ProcessWise,** etc.
2. L'approche déclarative illustré par les systèmes **MARVEL**, **Merlin,** etc.
3. L'approche orientée but adoptée par l'environnement **PEACE** et **PEACE+**, **SCALE**.
4. L'approche basée sur les réseaux de Pétri illustré par les systèmes **Process Weaver**, **SPADE** et **LEU**.
5. L'approche fonctionnelle illustré par les systèmes **HFSP** [SUZUKI et KATAYAMA 91] et à **PDL** [INOUE et al. 89], etc.
6. L'approche mutli-paradigme prise en compte dans les systèmes comme **ALF**, **APEL**, etc.

3.4. Formalisation du procédé.

On distingue trois phases principales dans le cycle de vie d'un procédé logiciel :

1. La phase de spécification des besoins qui consiste à identifier et/ou documenter les besoins du procédé. On pourra, par exemple, définir la performance attendue, les objectifs, etc. ;
2. La phase de conception, de modélisation qui doit définir le procédé pour répondre aux besoins exprimés lors de la phase précédente ;
3. La phase d'implantation du procédé tel qu'il a été définit.

Afin de répondre aux objectifs du procédé et de chacune de ces trois phases et ainsi, de couvrir le cycle de vie, des langages ont été proposés selon trois catégories que l'on retrouve dans [AMBRIOLA et al. 97]:

1. Un langage de spécification répondant aux besoins de la phase de spécification, dont les concepts doivent avoir pour but de décrire l'organisation, les objectifs généraux, etc. ;
2. Un langage de modélisation pour modéliser le procédé : c'est ce langage qui devra couvrir, pour l'essentiel, les concepts de base du procédé logiciel ;
3. Un langage d'implantation dont l'objectif sera de décrire la façon dont doit être exécuté le procédé.

Il est acquis que les formalismes de haut niveau sont à privilégier afin de fournir des moyens adaptés (simples) pour la description du procédé.

En effet, le but du domaine n'est pas d'alourdir la compréhension du procédé ainsi décrit mais bien de la simplifier et de permettre aux différents acteurs (managers, développeurs, etc.) d'avoir une vision uniforme du procédé.

Idéalement, un formalisme de modélisation doit satisfaire les besoins suivants :

1. Il doit être exécutable,
2. Il doit permettre de décrire et de supporter l'ensemble du cycle de vie du procédé ainsi que tous ses niveaux d'abstraction,
3. Il doit prendre en compte la description dynamique de l'ordonnancement des activités du procédé,
4. Il doit permettre de supporter et de modéliser l'évolution des procédé et de leurs modèles,
5. Il doit permettre de modéliser et de gérer l'incertitude et l'incomplétude de certaines informations,
6. Il doit permettre d'exprimer et de supporter la communication, la coordination, la négociation et la coopération entre les divers intervenants dans le procédé.

Sachant que le but est donc d'assister le concepteur et le développeur dans la compréhension et dans la modélisation du procédé, des langages graphiques, semi-formels et de haut niveau ont été développés. C'est ainsi que les langages de modélisation tels que **OOA/OOD** [COAD et YOURDON 91A], [COAD et YOURDON 91B], **OMT** [RUMBAUGH et al. 91], **UML** [MULLER 96], etc. ont largement inspiré certains formalismes de description de procédé (**APPEL**, **Process Weaver**).

Ces langages viennent en complément de langages formels, indispensables pour décrire de façon rigoureuse les modèles de procédé.

4. Vers des EGLCP fédérés.

4.1. La réutilisation de composants dans les EGLCP.

La section précédente a introduit les principaux concepts du domaine des procédés logiciels et les principales caractéristiques des EGLCP. Certains travaux du domaine se sont intéressés à des aspects particuliers comme la distribution de l'environnement, la coopération, la coordination des composants, l'intégration de composants selon une approche non- intrusive, etc. D'autres ont défini des EGLCP comme des fédérations de composants qui doivent interopérer.

4.1.1. LEU.

LEU [GRAW et GRUHN 95] est un environnement basé sur les réseaux **FUNSOFT**. Le procédé peut être exécuté par plusieurs moteurs de procédé. Dans ce cas, chacun des moteurs va exécuter des fragments de modèles (ou bien sous-modèles). En ce sens, le procédé est donc lui-même distribué sur plusieurs moteurs de procédé formant ainsi un environnement global.

LEU est un environnement homogène dans le sens où l'ensemble des moteurs partage les concepts et le formalisme décrivant les procédés. Il est possible d'ajouter des moteurs de procédés à l'environnement qui va pouvoir exécuter des modèles (ou fragments de modèles).

4.1.2. PEACE, PEACE+.

PEACE [ARBAOUI et al. 92] est un EGLCP qui adopte une approche orientée but fondée sur une logique non monotone. L'approche orientée but se concentre sur la description (basée sur la logique des croyances de [MOORE 88]) des buts à atteindre dans un procédé plutôt que sur la description des activités.

PEACE décompose le procédé en fragments de procédé représentant une étape du procédé ainsi qu'un but à atteindre. **PEACE** connut deux extensions, chacune d'entre elles reposant sur une approche multi-agents :

1. **PEACE+** [ALLOUI et al. 94] dont l'objectif est d'améliorer les aspects liés aux travaux dans les environnements coopératifs;
2. **PEACE*** [LATROUS et OQUENDO 95] dont l'objectif est d'améliorer la gestion de l'évolution des procédés.

PEACE est un environnement homogène, le procédé global est distribué au niveau des différents agents de l'environnement. Il permet également à d'autres PSS[1] d'interopérer avec lui en utilisant le format d'échange **WPDL** [WFMC 96B].

4.1.3. PROVENCE.

PROVENCE [BARGHOUTI et KRISHNAMURTHY 93A], [BARGHOUTI et KRISHNAMURTHY 93B] est un EGLCP reposant sur une approche client/serveur selon laquelle les clients peuvent s'abonner aux évènements qui sont produits durant l'exécution du procédé. Le but de cet environnement est de reposer sur des outils existants. Il peut être vu, dans une certaine mesure, comme une généralisation de **MARVEL** en terme d'interopérabilité et d'intégration d'outils. Cette dernière est assurée par un composant de l'architecture: le gestionnaire de mise en oeuvre et son rôle consiste à faire interopérer les outils logiciels participant à l'environnement.

Provence a été implémenté avec les composants suivants :

1. **MARVEL** [BEN-SHAUL et KAISER 93] qui gère la cohérence, la disponibilité des objets manipulés au sein de l'environnement, gère la coordination et la coopération entre les

[1] On trouve dans la littérature le terme anglophone "Process Sensitive-Software" pour "systèmes sensibles aux procédés".

différents acteurs (personnes au sein du même procédé logiciel) et permet la définition du modèle de procédé en termes de règles ;

2. **YEAST** assure la liaison avec les clients et déclenche les actions correspondantes aux requêtes des utilisateurs ;
3. **3D FILE SYSTEM** fournit un mécanisme permettant aux utilisateurs de créer des vues dynamiques du logiciel et d'effectuer des changements relatifs à cette vue sans affecter la base du logiciel ;
4. **DOTY**, éditeur graphique, qui permet de créer et de manipuler les graphes de façon interactive en utilisant le multifenêtrage. L'interprétation des graphes se fait par **DOTY** et le langage qui lui est associé.

Provence pourrait être vu comme une fédération d'outils dans le sens où il repose sur une volonté de proposer un environnement de "composants inter-opérables". En ce sens, **PROVENCE** offre une architecture relativement indépendante des outils qui le composent, contrairement à la plupart des environnements existants où les outils sont fortement imbriqués, ne pouvant être remplacés. Pourtant si les outils sont bien identifiés et ne sont pas fortement intégrés dans l'environnement, le phénomène d'ajout ou de retrait d'outils implique une forte modification de l'interface entre les composants et doit tenir compte des spécificités liées aux composants d'origine. En effet, certaines hypothèses ont été faites selon les possibilités de ces outils. Par exemple, **MARVEL** doit connaître les chemins des différents fichiers composant le projet et doit savoir quels outils utilisés sur quels objets.

D'autre part, l'architecture proposée s'appuie sur les possibilités intrinsèques de **MARVEL, YEAST, DOTY, 3D FILE SYSTEM**. Le remplacement de ces outils par d'autres ne devrait se faire qu'à la condition que les nouveaux outils disposent des mêmes fonctionnalités et répondent aux mêmes caractéristiques d'implémentation que ceux utilisés. Ces contraintes n'étant pas décrites dans l'environnement (aucune description des contraintes et des caractéristiques des composants), le respect de ces dernières, reste au bon vouloir du concepteur de l'environnement. On peut également noter que l'environnement, tel qu'il a été conçu, ne peut pas fonctionner si l'un ou l'autre des outils est absent.

4.1.4. ENDEAVORS.

Le système Endeavors [BOLCER et TAYLOR 96] est un système distribué dont le but est de supporter les procédés logiciels et les procédés de workflow. L'originalité de Endeavors repose

sur une approche homogène pour la modélisation des procédés et l'approche hétérogène pour leurs exécutions.

Cela signifie qu'un modèle de procédé est défini selon plusieurs modèles basés sur des concepts identiques et décrits à l'aide d'un même formalisme mais qu'ensuite, chacun de ces modèles peut être interprété sur des plates-formes différentes: en effet, il existe des "handlers", écrits en langage Java, qui sont l'implémentation des modèles.

Les "handlers" selon **ENDEAVORS** constituent les composants de l'environnement à l'exécution. En ce sens, ils ressemblent aux fragments de procédé décrits dans l'environnement **PEACE+**.

4.2. Vers des fédérations de composants inter-opérables pour les EGLCP.

Les environnements précédents reposent tous sur des architectures à base de composants qui doivent interopérer en vue de réaliser les objectifs décrits par les modèles de procédé sous-jacents.

Certaines de ces approches ont davantage porté sur la distribution du procédé au niveau de plusieurs moteurs d'exécution (**ALF**, **SCALE**, **LEU**, **PEACE+**), d'autres ont mis l'accent sur les problèmes d'hétérogénéité (**ENDEAVOUR**), d'autres encore considèrent un environnement à partir de composants existants et hétérogènes (**PROVENCE**).

En revanche, aucun de ces systèmes n'aborde réellement la définition des fédérations en considérant l'existence de "règles communes de fonctionnement" [VERJUS et OQUENDO 98], [ESTUBLIER et VERJUS 99] et ne fournissent des moyens pour la modélisation des fédérations.

Les systèmes qui vont suivre mentionnent explicitement la définition de "fédération".

4.2.1. Oz.

Le projet **Oz** a étudié l'interopérabilité entre environnement de génie logiciel centrés procédé; en particulier [BEN-SHAUL et KAISER 95] porte sur l'interopérabilité d'EGLCP et les différentes possibilités notamment en ce qui concerne l'hétérogénéité de modèles de procédé. Les mêmes auteurs ont poursuivi leurs travaux sur des propositions d'architectures [BEN-SHAUL et KAISER 98] et, en particulier, ont proposé une architecture pour des fédérations d'EGLCP selon une perspective homogène :

1. Les composants de la fédération sont tous des environnements **MARVEL** pouvant être potentiellement géographiquement distribués ;
2. Le comportement de la fédération est régit selon la métaphore de *l'Alliance Internationale* selon laquelle les composants peuvent constituer des "sommets" à partir

desquels il en découle des "traités" qui vont régir le fonctionnement de l'environnement **Oz**. Chaque environnement garde son autonomie dans la mise en oeuvre de ses procédés. Cependant, il leur est possible d'interagir pour la réalisation d'une activité commune (le traité). Les sommets n'expriment donc que les stratégies de mise en oeuvre des traités; dans un sommet, un environnement coordonne l'activité commune:

a. Il reçoit les données nécessaires à l'activité et provenant des autres environnements participant au sommet,
b. Il met en oeuvre l'activité et,
c. Il transmet les résultats aux autres environnements.

Ce comportement est obtenu par l'implémentation des procédures et protocoles du sommet comme mécanismes de base de chacun des environnements de la fédération.

L'approche consiste donc à distribuer l'exécution des modèles de procédés sur des sites différents et en fonction de la disponibilité des ressources nécessaires à leurs exécutions.

L'architecture proposée est relativement flexible dans le sens où elle permet l'ajout et le retrait de composants **MARVEL** de façon dynamique.

L'une des principales limitations est le caractère homogène de l'approche (modèles de composants et formalisme utilisé) et le mode de fonctionnement (la métaphore) qui ne peut être changé.

4.2.2. Modéliser des fédérations d'EGLCP.

L'approche prise dans [TIAKO 99] porte sur des fédérations de fragments de procédé dérivés de LCPS [DERNIAME et al. 94]. Des modèles de contrats entre environnements peuvent alors être constitués permettant la délégation de modèles, etc. Il s'agit donc d'une approche permettant de modéliser les fédérations sous l'angle de l'organisation des environnements la composant et supportant la mobilité de ces derniers. Chaque contrat définit également les compétences respectives de chaque EGLCP impliqué dans ce dernier. En fait, l'approche prise considère les compétences d'un EGLCP sous la forme des services qu'il peut offrir et les possibles négociations permettant à un EGLCP de déléguer l'exécution d'un modèle ou fragment (appelé aussi composant de procédé) de modèle de procédé à un autre EGLCP.

[TIAKO 98] propose alors une architecture à cinq niveaux comprenant (voir figure 2.4) :

1. un méta-modèle qui permet à la fois de décrire des modèles de procédé mais également des modèles de fédération (d'EGLCP);
2. un niveau modèle regroupant un ensemble de modèles de procédé logiciels et de procédé de fédération, basés sur les concepts du méta-modèle;

3. des procédé logiciels d'une part et des procédé de fédération d'autre part sont obtenus comme instances des modèles (respectivement de procédé et de fédération);
4. une couche produit qui contient l'ensemble des produits locaux et importés par un EGLCP. Ces produits peuvent être échangés entre EGLCP grâce au niveau infrastructure;
5. une couche intermédiaire (infrastructure) entre les constituants permet alors les échanges et la communication.

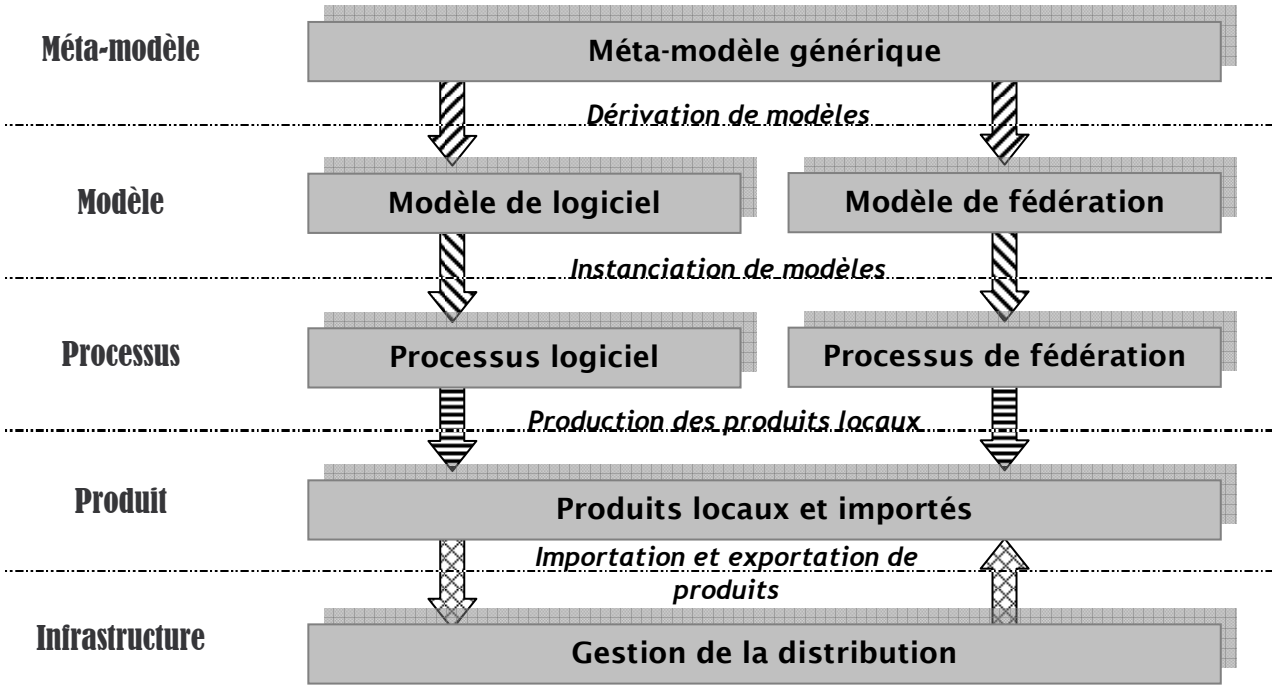


Figure 2.4 : *Architecture pour des fédérations EGLCP* [TIAKO 98].

L'originalité de cette proposition tient au couplage des procédé logiciels et des procédé de fédérations rendu possible par le partage de nombreux concepts (comportement, similarités des concepts de rôle et de compétences, des concepts d'agents et de participants,…) [TIAKO 98], [TIAKO 99]. D'où l'idée de recourir à un méta-modèle générique permettant de définir les procédé de fédérations au même niveau d'abstraction que les procédé logiciels. L'approche constitue une alternative à **Oz** par les protocoles de négociations, de délégations et la mobilité des composants de procédé. En revanche, il faut définir les types de base (en concepts dérivés de LCPS) qui seront utilisés pour modéliser les procédés partagés par les différents EGLCP.

4.2.3. APELv4.

L'approche prise par [AMIOUR 99], [ESTUBLIER et al. 99] consiste à proposer un "environnement coopératif et ouvert dans lequel le support se fait à travers un ensemble de composants. Chaque composant prend un charge la gestion d'un aspect particulier du procédé". Un tel environnement est qualifié de *fédération de composants inter-opérables*.

L'environnement **APEL v4** [AMIOUR et al. 97] est la continuité des travaux effectués sur l'environnement **APEL** [ESTUBLIER et al. 97A], [ESTUBLIER et al. 97B], [ESTUBLIER et al. 98A].

La justification de l'adjectif coopératif se trouve dans les mécanismes d'interaction entre les composants qui collaborent à la réalisation du procédé. Quant à l'adjectif ouvert il est justifié si on considère qu'il est toujours possible d'ajouter un composant de l'environnement, sous réserve du respect de certaines contraintes (voir plus loin dans cette section).

Les composants qui sont pris en compte ici respectent un certain schéma : ils sont de véritables logiciels sensibles aux procédé (LSP) offrant :

1. Un méta-modèle qui contient les aspects pertinents liés à l'aspect du procédé auquel le composant est dédié ;
2. Des formalismes permettant de décrire l'aspect en question ;
3. Une plate-forme d'exécution permettant d'exécuter les modèles description des descriptions. Cette plate-forme comprend, entre autre, un moteur d'exécution, une base de données pour le stockage des informations, etc.

Cette description des composants qui sont pris en compte est restreinte à une catégorie de composants (LSP) et ne peut être généralisée. Par contre, tout logiciel sensible aux procédé, qu'il soit un composant issu du marché ou un système patrimoine peut faire partie des fédérations décrites selon cette approche, sous réserve d'adhérer au méta-modèle **d'APEL**.

L'architecture conceptuelle de la fédération comprend cinq parties (voir figure 2.5) [AMIOUR 99]:

1. Un méta-modèle commun et un langage commun ;
2. Un modèle commun ;
3. Une fondation ;
4. Un ensemble de composants ;
5. Une infrastructure de communication.

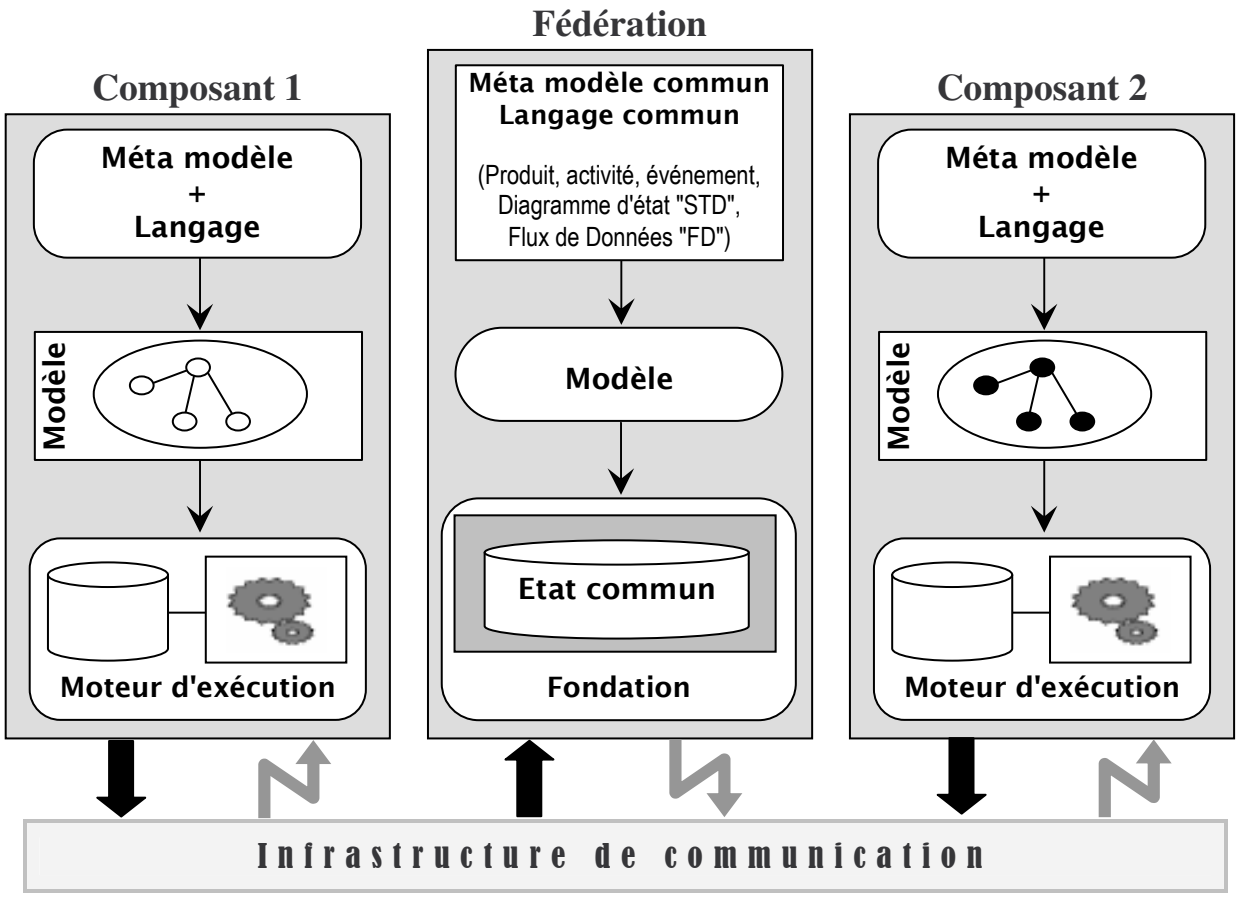


Figure 2.5 : *Architecture conceptuelle de la fédération* [AMIOUR 99].

L'objectif du modèle commun est de décrire les parties communes entre les différents composants de façon à assurer que le procédé est exécuté de manière cohérente. Concrètement, il s'agit de décrire :

- Les activités et les produits qui font partie du procédé que l'on veut gérer. Cette description inclut aussi bien l'aspect statique que dynamique.
- L'ordre d'exécution des activités (flot de contrôle)
- L'échange de produits entre les activités (flot de données).

Le modèle commun constitue donc, dans une certaine mesure, un accord et un objectif commun de la fédération. Une fois décrit, la fédération doit donner les moyens de garantir son exécution de manière cohérente; l'entité en charge de la cohérence de l'exécution est appelée la fondation. Une instance du modèle commun en exécution est appelée l'état commun constituée d'objets et de relations représentant les activités, les produits, etc. Cet état est sous le contrôle de la fondation.

Le modèle commun doit être décrit de façon à pouvoir être interprété (compris) par l'ensemble des composants (hétérogènes). Ces derniers utilisent les éléments du modèle pour définir leurs propres modèles (locaux). Dans cette perspective, le méta-modèle commun et le langage commun ont pour objectif de permettre la description du modèle commun :

- Le méta-modèle commun fournit une ontologie[2] commune permettant aux divers composants de se comprendre tout en préservant leur autonomie. Il définit une sémantique pour l'ensemble des concepts ;
- Le langage commun est un formalisme compréhensible par les composants et permet de matérialiser le méta-modèle commun et de décrire le modèle commun de la fédération.

Le rôle principal de la fondation est de coordonner les différents composants. Elle peut être considérée comme un serveur d'état enrichie du modèle commun. Elle remplit les fonctions suivantes:

1. *Interprétation du modèle commun:* Garantit l'exécution cohérente d'une instance du modèle commun par l'interprétation du modèle. Elle comprend donc un moteur d'exécution capable d'interpréter le modèle ce qui constitue une différence par rapport à un serveur d'état "simple" ;
2. *La gestion de l'état commun:* Elle reçoit et traite les requêtes manipulant l'état commun qui émanent des composants. A cet égard, elle gère les problèmes de persistance, reprise après panne, etc. ;
3. *La notification des changements:* Elle notifie les composants des changements intervenants dans l'état commun au travers d'un mécanisme de souscription / publication.

Les composants doivent quant à eux respecter certaines spécifications aux différents niveaux (méta-modèle, modèle et exécution). Un composant n'est pas obligé d'accepter l'ensemble des concepts du méta-modèle mais uniquement ceux qu'il est amené à utiliser. Il est doté d'une interface lui permettant de communiquer avec l'infrastructure de communication.

L'infrastructure de communication est en charge :

- De l'acheminement des requêtes (des composants vers la fondation) ;
- De l'acheminement des notifications (de la fondation aux composants).

---

[2] Une ontologie est définie comme une spécification explicite d'une conceptualisation [GRUBER 93] ou comme un "accord sur une conceptualisation partagée par une communauté" [DIENG et al. 00]. Une ontologie fournit un cadre unificateur pour réduire et éliminer les ambiguïtés et les confusions conceptuelles et terminologiques et assurer une compréhension partagée par la communauté visée [USCHOLD et GRUNINGER 96]. Une ontologie comporte des définitions fournissant le vocabulaire conceptuel permettant de communiquer au sujet d'un domaine ; cela permet de définir (a) les concepts utilisables pour décrire la connaissance, (b) les relations entre de tels concepts et (c) leurs contraintes d'utilisation [DIENG et al. 00].

4.2.4. PIE.

Le projet européen **PIE** (*Process Instance Evolution*) a pour objectif d'étudier l'évolution des instances de procédé, les problèmes de déviation et de concordance [CIMPAN et OQUENDO 98], [CIMPAN 00], de réconciliation, d'implémentation des changements selon une ou des stratégies d'évolution et de fournir un environnement, support aux procédés et aux procédés d'évolution. Le serveur de procédés de l'environnement **APEL** a servi de composant pivot (voir figure 2.6) dans l'environnement de **PIE** dans le sens où:

1. Il est la fondation de l'environnement **PIE**, l'univers commun, l'état global de la fédération ;
2. Il permet aux autres composants (composant de monitoring, composant de décision, composant d'implémentation des changements, composant gérant la stratégie d'évolution, etc.) de participer à l'environnement grâce à une couche de communication.

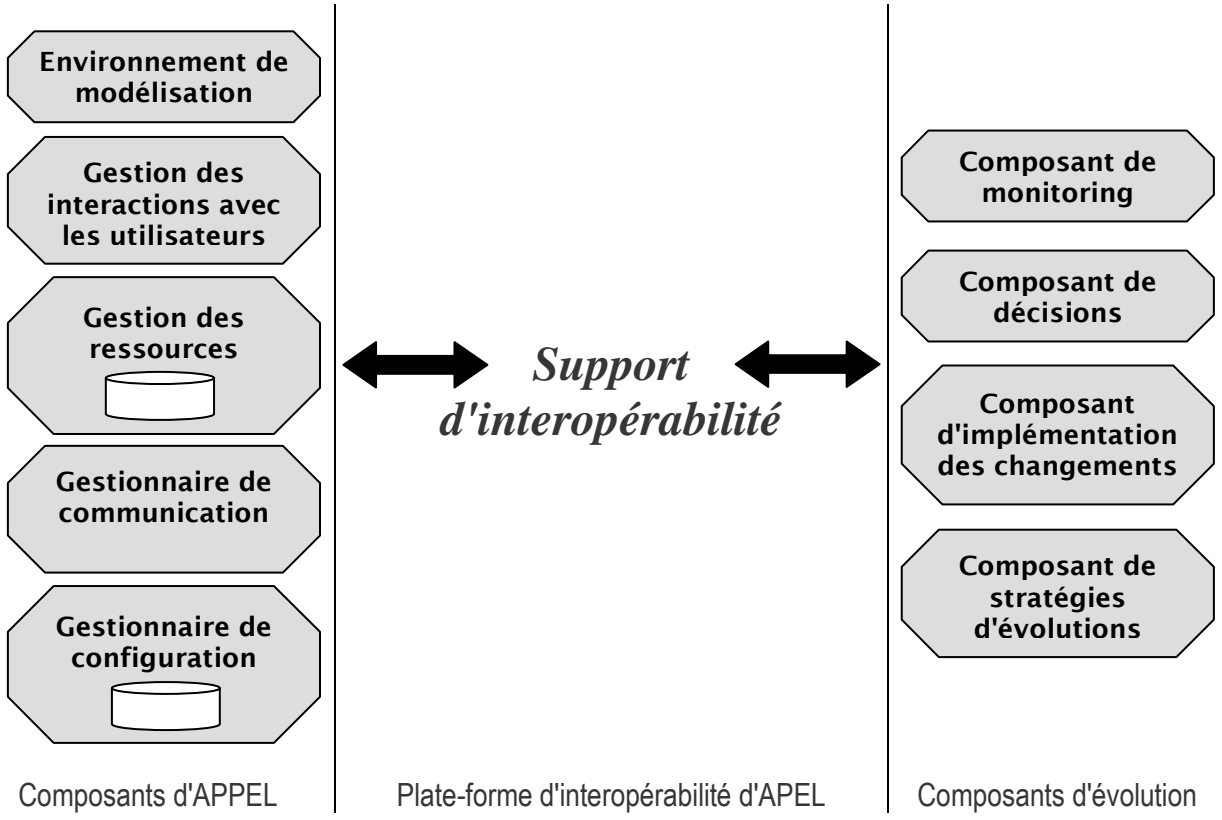


Figure 2.6 : *Architecture de l'environnement **PIE*** [AMIOUR et ESTUBLIER 98B].

Les composants étaient "libres" d'observer le procédé en s'abonnant aux sujets qui les concernaient et en recevant les notifications associées (notifications envoyées par la fondation). D'autre part, tout composant pouvait opérer certains changements sur l'état courant des instances du procédé, en invoquant les services de l'interface (API) de la fondation.

5. Bilan et conclusion.

Les environnements dans le domaine des procédés logiciels ont eu pour principal objectif de répondre aux besoins immédiats dans le domaine du développement de logiciel, en proposant une vision "unifiée" des outils utilisés.

Cependant, rares sont les environnements ouverts et flexibles permettant à des outils de type COTS[3] de participer à un environnement tout en restant disponibles par ailleurs (préservation de l'autonomie).

5.1. Les apports du domaine des EGLCP.

On peut noter quelques apports du domaine des procédés logiciels :

- Le guidage du développement de logiciels grâce aux procédés logiciels : les technologies des procédés logiciels permettent de guider le comportement d'un environnement et des acteurs de ce dernier en vue de réaliser une application logicielle parfois complexe ;
- La formalisation des procédés : les approches et les technologies développées permettent de structurer des procédés de développement par l'identification de concepts, la définition de modèles basés sur ces derniers et la définition d'instances ;
- La mise en synergie de fonctionnalités hétérogènes pour satisfaire les besoins des développeurs : les derniers travaux portant sur les EGLCP ont mis en évidence une demande grandissante en fonctionnalités diverses, de mettre à disposition des développeurs une "boîte d'outils adaptable" dont il est possible de définir un certain mode de fonctionnement et, entre autre, permettre l'automatisation de certaines tâches.

5.2. Les limitations.

Les principales limitations des EGLCP sont les suivantes :

- Complexité, redondance et insuffisance des supports : les EGLCP sont des environnements de plus en plus complexes car ils intègrent de nombreuses fonctionnalités. Cependant, certaines sont manquantes alors que d'autres sont redondantes avec des fonctionnalités proposées par des composants existants sur le marché ;
- Rigidité, monolithisme et faible évolution des supports : les EGLCP sont des environnements qui sont générés à partir d'une description. L'évolution de tels supports impose l'évolution de la description. Toute modification de la description induit une re-génération de l'environnement qui ne peut se faire en cours d'exécution ;

[3] Abbreviation Anglo-Saxon de "commercial off the shelf".

- Faible hétérogénéité des constituants : les EGLCP "modulaires" intègrent rarement des composants hétérogènes ;
- Faible autonomie des constituants : l'intégration de composants est forte et ne permet pas à ces derniers de préserver leur autonomie et de pouvoir être utilisés par ailleurs ;
- Peu (pas) d'EGLCP intègrent des composants quelconques issus du marché (COTS) : les outils intégrés par les EGLCP sont bien souvent de type "load and go".

5.3. Conclusion.

Comme nous l'avons vu dans la première partie de ce chapitre, le domaine des procédés logiciels est vaste et les procédés logiciels sont intrinsèquement complexes. De nombreuses recherches poursuivent leurs efforts pour caractériser et pour mieux cerner les procédés : des approches ont été proposées pour les modéliser et des environnements ont été développés pour les supporter.

Dans le domaine du développement des logiciels, les approches et les technologies évoluent rapidement, de nouveaux outils apparaissent fréquemment, intégrant eux mêmes de nouvelles fonctionnalités. Les utilisateurs (acteurs des procédés logiciels) sont de plus en plus exigeants, les besoins continuent de croître et font de plus en plus appel à des outils logiciels spécifiques qu'il faut pouvoir faire interopérer tout en préservant la flexibilité et l'évolution des environnements. Plusieurs travaux ont porté sur l'intégration, la coopération, l'interopérabilité de composants logiciels au sein des EGLCP en vue de fournir un environnement complet.

Nous constatons cependant qu'aucune proposition ne satisfait simultanément aux critères :

- D'hétérogénéité,
- De distribution,
- De flexibilité/ouverture,
- D'autonomie,
- De contrôle.

De plus, la plupart des fonctionnalités requises au niveau d'un EGLCP peuvent être fournies par des outils logiciels issus du marché. Cette tendance vient s'opposer aux environnements générés et homogènes existants. Ainsi, les environnements doivent :

- Pouvoir intégrer des outils hétérogènes et distribués de façon non intrusive ;
- Garantir l'autonomie des outils qu'ils intègrent ;
- Garantir le fonctionnement global et la cohérence de l'environnement composé de différents outils, comme s'il s'agissait d'une application développée spécifiquement pour des besoins propres.

Il s'agira donc de proposer des environnements flexibles, adaptables, ouverts, permettant de faire du "sur mesure".

En outre, les supports existants reposent essentiellement sur une approche qui considère l'outil logiciel comme simple ressource ou alors comme acteur d'une activité : il n'est pas identifié comme un concept majeur et sa dimension est réduite alors que paradoxalement, les besoins sont nombreux et la participation des outils logiciels ne cesse de croître (objectifs d'automatisation et de productivité). Les problèmes bien connus qui se posent comme le recouvrement de concepts entre outils, la cohérence des modèles opérationnels des différents outils, la gestion de la synchronisation sont totalement occultés.

Chapitre -3-

Une architecture de modélisation de procédés à base de composants

1. Introduction.

Dans le domaine de la modélisation des procédés, l'approche par composants vise à construire de nouveaux procédés de façon coopérative, dans un environnement distribué et hétérogène. Pour qu'un système puisse supporter de tels besoins, il doit compter sur une architecture physique appropriée.

La figure 3.1 illustre le scénario pour la construction et l'exécution du procédé "conception d'un logiciel".

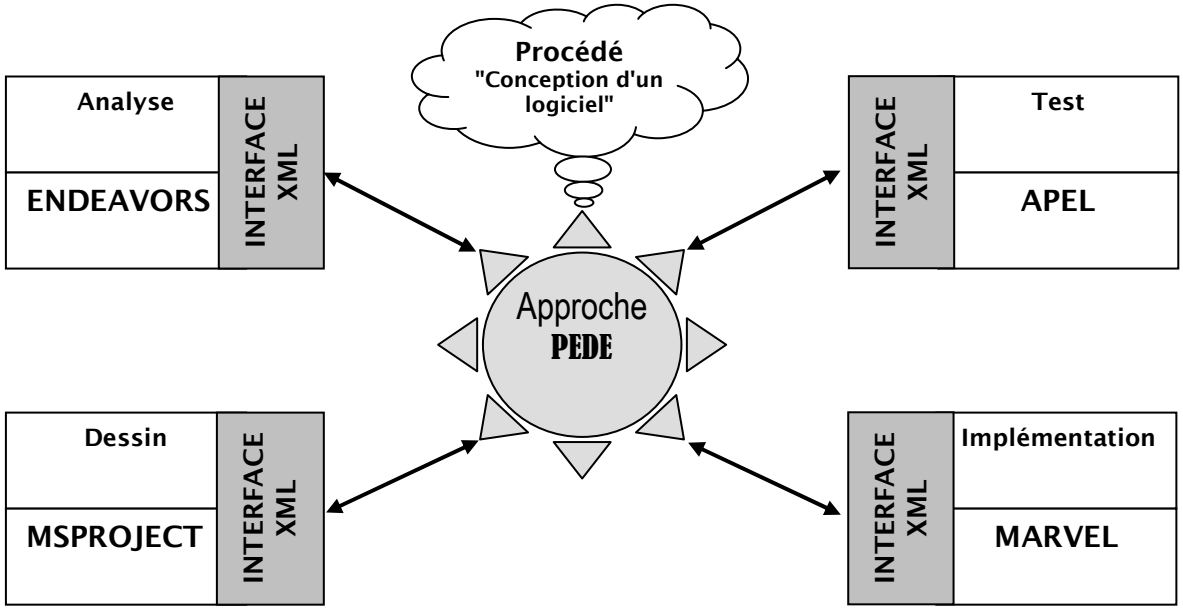


Figure 3.1 : *Conception d'un procédé par composants.*

Nous allons appliquer la philosophie des objets distribués et des composants pour le développement du procédé. Au cours de ce chapitre, nous présenterons l'architecture globale de notre approche ainsi que ses concepts de base.

2. Architecture de l'approche PEDE.

L'architecture de l'environnement centré procédé basé sur l'approche de modélisation à base de composant est composée de plusieurs modules qui sont en interaction permanente (voir figure 3.2).

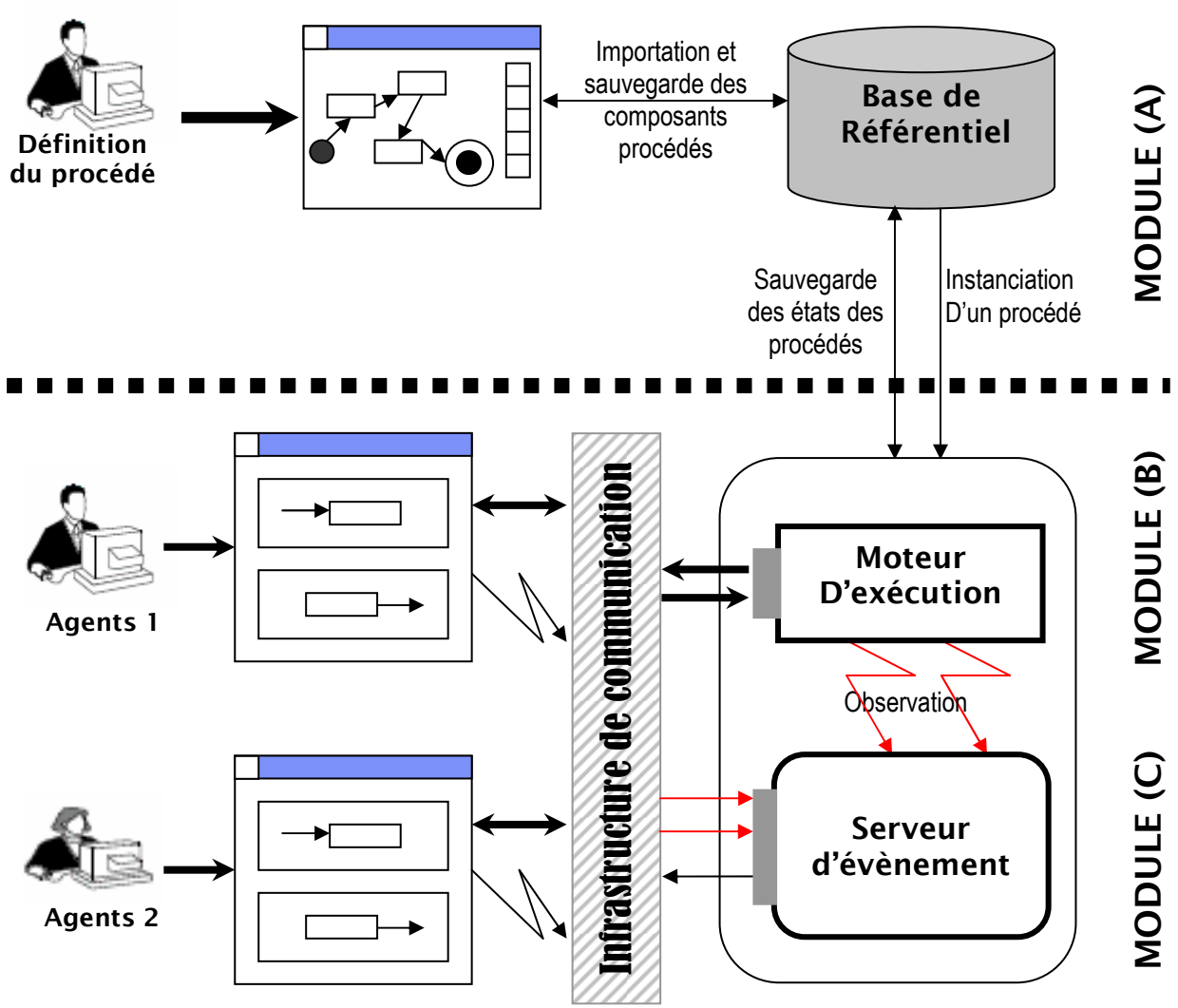


Figure 3.2 : *Architecture de l'approche centrée procédée PEDE.*

a) Le premier module constituant cet environnement et le module de définition des procédés, ce module doit être doté d'une interface de modélisation des procédés en utilisant les composants procédés, ces derniers peuvent être directement crées en spécifiant les propriétés de chaque composant (le rôles, l'agent, les artefacts en entrés, les artefacts en sortie…), ou bien les importés d'une bibliothèque de composants déjà créé et qui sont stockés dans la base de donné de l'environnement.

b) Le deuxième module représente le moteur d'exécution du procédé. Il a comme tâche l'exécution des procédés déjà crées dans le module de définition. Le moteur d'exécution est considéré comme étant une plate-forme d'exécution pour les procédés basé composants, il fournit les mécanismes nécessaires à l'exécution des composants procédés, le moteur de procédés est en permanente communication avec le serveur d'événements, ce dernier est chargé de suivre l'évolution de l'exécution des composant procédé ainsi il offre certains services : le service d'abonnement et le service de notification. (voir la section suivante).

c) Le troisième module concerne module de visualisation il permet aux agents impliqués dans un procédé donné de visualisé les tâches qui leurs sont confiées. Le module de visualisation fourni aux agents toutes les inputs nécessaires à la mission des agents et déclenche des événements relatifs au composant.

3. Concepts du module de définition.

3.1. Modèle.

Un modèle est la représentation abstraite d'un procédé dans un formalisme donné, qui permet de le comprendre et de le simuler. Ce modèle est ensuite spécialisé pour obtenir une instance pouvant être exécutée par un environnement centré procédé.

3.2. Composant procédé.

Un composant procédé est l'instance d'un procédé, dérivée d'un modèle stocké dans une base de données. Le composant encapsule sa représentation et fournit une interface simple pour établir la communication avec l'extérieur. Si on approfondit le concept du procédé, un composant correspond à une activité ou à un sous procédé qui contribue à la réalisation d'un but spécifique dans le procédé. Il vise à générer ou à modifier un ensemble de produits.

3.3. Modélisation basée composants.

Le but de la modélisation basée composant est la construction de modèles plus complexes, à travers la réutilisation de modèles consistants et fonctionnels. Chacun d'eux constitue un composant procédé. Voici les points à considérer pour la modélisation par composants :

1. Construire une base d'information sur les composants réutilisables,
2. Déterminer la fonction de chaque composant dans le nouveau modèle procédé,
3. Composition dynamique des composants hétérogènes.

3.4. L'exécution basée composants procédés.

Notre vision concernant le support de procédés est un environnement où deux ou plusieurs composants hétérogènes exécutent et coopèrent de façon distribuée. Cet environnement doit être capable de rendre l'hétérogénéité et la distribution des composants transparentes. Ainsi, par exemple, le rôle chargé de la modélisation, voit les composants comme des boîtes noires, qu'il utilise pour créer un nouveau modèle.

3.5. Le vocabulaire commun.

La terminologie utilisée dans notre approche est inspirée de [CONRADI et al. 92]. Ce vocabulaire commun sera exprimé à travers un schéma conceptuel dans le prochain paragraphe.

- **ComposantProcédé :** Il correspond à une activité ou à un sous-procédé qui contribue à la réalisation d'un but dans le procédé.
- **Produit (le résultat d'un procédé) :** Il correspond à des objets tels que des documents, codes ou services.
- **Agent :** C'est un performeur d'un procédé, cela peut être un humain ou un outil logiciel. Un agent est caractérisé par les propriétés de son rôle.
- **Rôle :** Décrit l'ensemble des responsabilités, droits et compétences d'un agent dans le contexte des activités où il intervient.

3.6. Le schéma conceptuel procédé.

Un schéma conceptuel procédé rassemble toute l'information des entités qui appartiennent au procédé. Le schéma conceptuel pour notre approche est illustré par la figure 3.3.

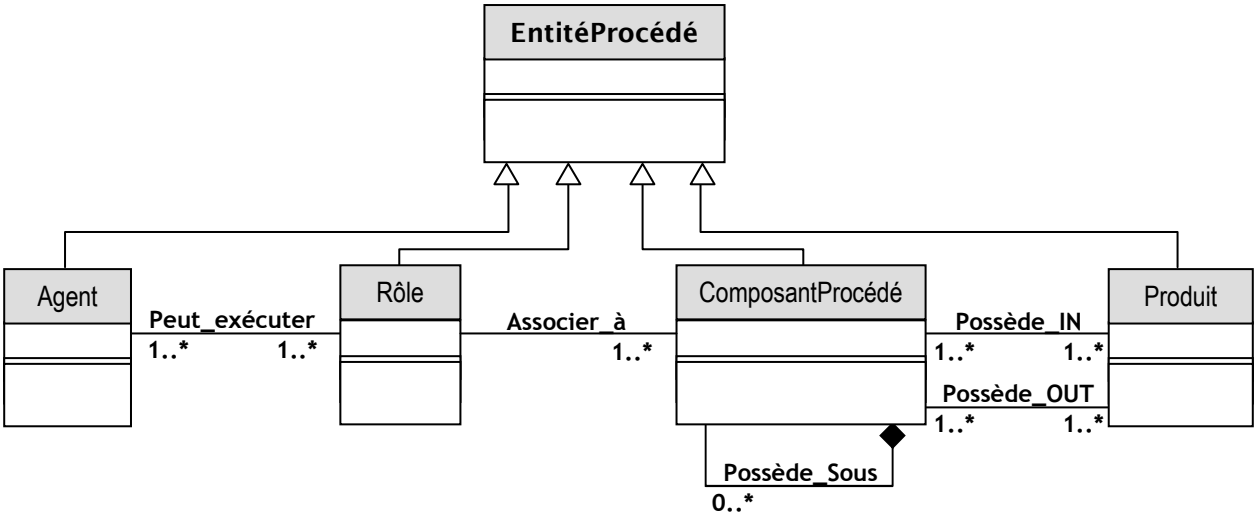


Figure 3.3 : *Schéma conceptuel procédé.*

3.7. FrameWork orienté objet.

Un FrameWork est un ensemble de classes qui coopèrent et permettent des conceptions réutilisables. Il apporte de nombreuses fonctionnalités et des relations entre classes d'objets, ainsi qu'une infrastructure au développeur. On spécialise un FrameWork afin qu'il répond aux caractéristiques d'une application, par création de sous classes spécifiques à partir des classes abstraites du FrameWork [GAMMA et al. 99].

3.8. L'état d'un procédé.

L'état d'un procédé correspond au comportement des entités d'un procédé. Le changement d'état est déclenché par des événements produits quand des méthodes sont exécutées.

Exemple :

L'exécution de la méthode *ChangeEtatProcédé(Suspendre)* produit l'événement : (:ComposantProcédé, suspendu) et en suite, il produit un changement d'état de l'entité ComposantProcédé.

3.9. Le classement hiérarchique.

Nous proposons un classement hiérarchique. Il consiste à structurer la base de composants réutilisables par domaine d'application.

Exemple :

- Analyse
- Dessin
- Implémentation
- Test

Chaque domaine d'application peut être raffiné en sous domaines.

Exemple : on peut raffiner Implémentation en :

- Implémentation
    - Les structures de données,
    - Les composant graphiques,
    - ...etc.

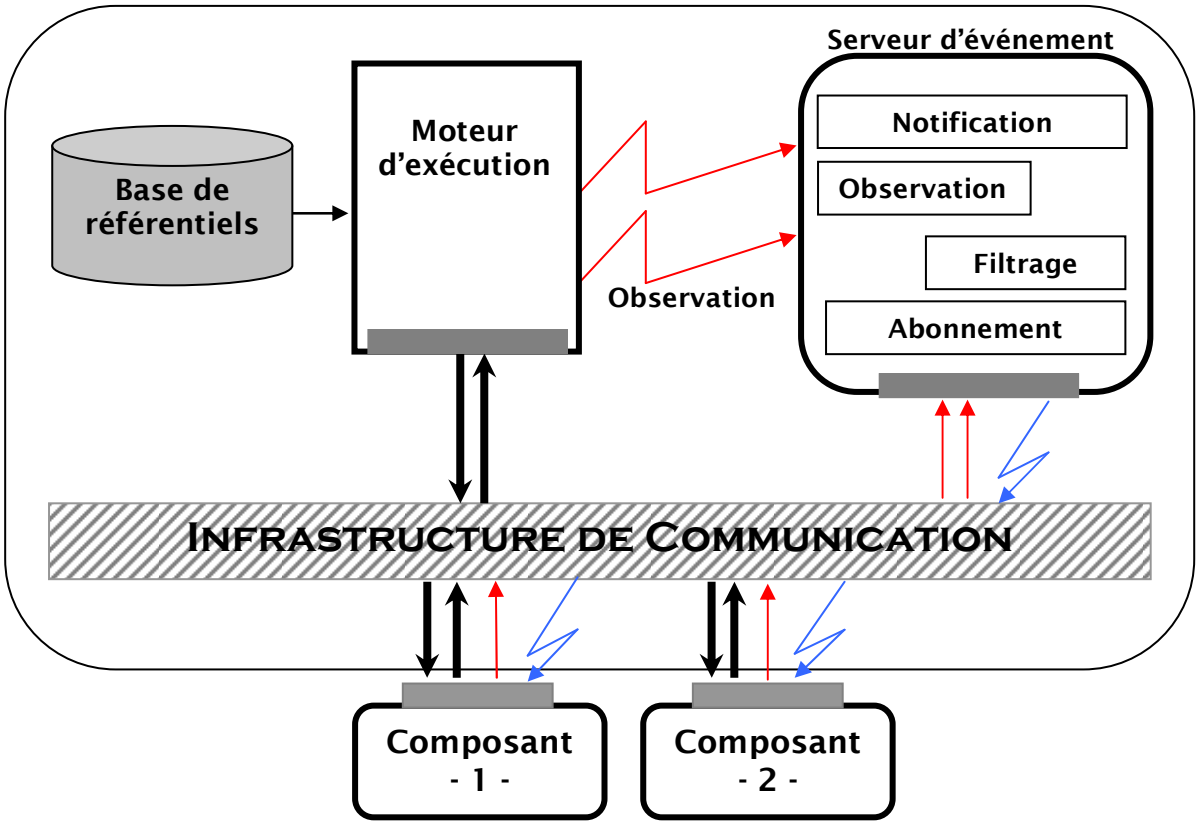
Pour entrer un composant dans la base, on détermine à quel domaine il correspond, puis à quel sous domaine, et ainsi de suite jusqu'à arriver à l'emplacement qui lui convient.

4. Module d'exécution et d'observation.

Nous rappelons que le procédé globale est géré de manière répartie sur différents composant, chaque composant est autonome et ne dépend que du moteur d'exécution Le moteur d'exécution doit intégrer et inclure les moyens qui permettent l'exécution des composants procédés et de géré les interactions entres ces derniers. Plusieurs techniques et mécanismes existent pour implémenter de tel moteur d'exécution, parmi ces mécanisme on trouve : *l'approche à base d'évènement*

Evénement : Un événement est une action instantanée et non persistante, qui apparaît pendant l'exécution d'une instance d'un certain procédé. Un événement est levé pour indiquer l'état actuel de l'instance du procédé.

Service d'événements : Au temps d'exécution, le serveur d'événements capture des événements provenant de l'exécution des composants ; et en suite, il fait la notification à des composants abonnés à tels événements.



Envoi d'événement	
Abonnement	

Interface de communication	
Communication	

Figure 3.4 : *l'Architecture du Moteur d'exécution.*

4.1. Fonctionnement du serveur d'événement.

L'approche à base d'événements est fondée sur le paradigme *d'abonnement/Notification.* Les composants s'abonnent aux événements qui les intéressent (1). Le système se charge ensuite d'observer les événements qui se produisent (2, 3), les "filtrer" en fonction des abonnements (4), et de les notifier aux abonnés (5). *(Voir la figure Suivante).*

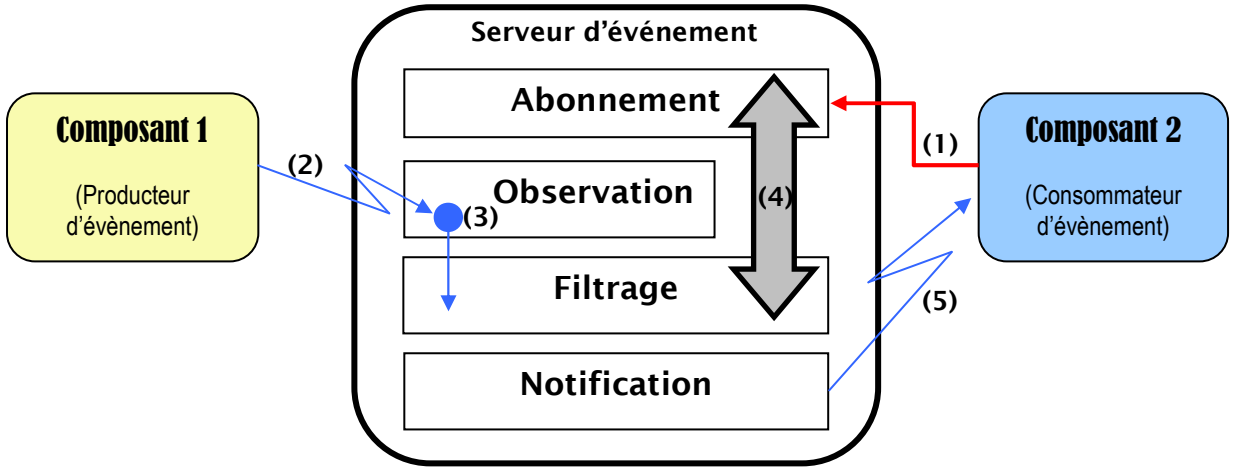
Le développement d'un serveur d'événement consiste à développer les services suivants :

a) Un service d'observation pour détecter les événements,

b) Un service d'abonnement afin de permettre aux composants de s'abonner aux événements (définition et gestion d'abonnement, les informations associées aux événements, etc.),

c) Un service de filtrage pour filtrer les événements par rapport à chaque composant, de façon à ne notifier à un composant donné que les événements auxquels il s'est abonné,

d) Un service de notification qui transmet les événements filtrés aux composants.



(1) Abonnement	le consommateur déclare son intérêt à recevoir l'événement
(2) Génération d'évènement	le producteur génère l'événement
(3) Observation et Détection d'évènement	le serveur d'événement détecte l'occurrence de l'événement
(4) Filtrage des évènements	le serveur d'événements vérifie si l'abonnement correspond à l'événement
(5) Notification	le serveur d'événement notifie le consommateur d'évènement

Figure 3.5 : *Serveur d'évènement.*

5. Modèle UML pour des Composants procédés.

Le modèle que nous proposons pour des composants procédés, est représenté par la notation **UML** [MULLER 97]. La démarche adoptée consiste à considérer un modèle de procédé, représenté dans un formalisme quelconque, comme un composant réutilisable, qui fournit une interface simple à des concepteur de procédés.

Dans ce qui suit, nous allons décrire les diagrammes de modélisation, en commençant par les cas d'utilisation, puis les diagrammes de séquence, le diagramme d'état et finalement le diagramme de classes.

5.1. Les cas d'utilisation.

A travers la présentation des cas d'utilisation nous mettons en évidence les rôles qu'implique l'introduction d'un composant procédé dans la modélisation d'un procédé. Nous identifions trois rôles principaux :

1. Celui chargé de *la définition* d'un composant procédé : Il va définir chaque composant procédé, dont les fonctionnalités et les propriétés sont similaires à un autre, en appliquant les changements à une définition existante (dans une base de données) de composant. Le chargé de la définition va spécifier la composition des composants (comment les composants doivent interagir avec d'autres composant procédés).
2. Celui chargé de *l'exécution* : Une fois que le procédé est déclenché, chaque composant sera exécuté dans l'ordre et sous les conditions définies par celui chargé de la définition.
3. Celui chargé de *l'observation* : Les besoins de ce type d'acteur sont de visualiser l'information concernant les composants qui l'intéressent, en cours d'exécution (de combien de temps il dispose pour le mener à bien ? si le procédé a fini ? dans quel état se trouve-t-il ? etc.). Pour eux (les observateurs) la distribution des composants est transparente. Ils les observent et les contrôlent comme si les composants étaient dans leurs propres sites.

Les manipulations de base définies pour les composants procédés sont résumées par :

1. Définir un composant procédé selon le méta-modèle.
2. Définir un composant procédé, dont les fonctionnalités et les propriétés sont similaires à un autre, en appliquant les changements à une définition existante de composant procédé.
3. Spécifier la composition. Comment les composants procédés interagissent avec d'autres composants procédés.

4. Instancier un composant procédé en spécifiant les liaisons nécessaires pour lui permettre d'être exécuté.
5. Exécuter un composant procédé.

La figure 3.6 illustre les différents cas d'utilisation décrits auparavant.

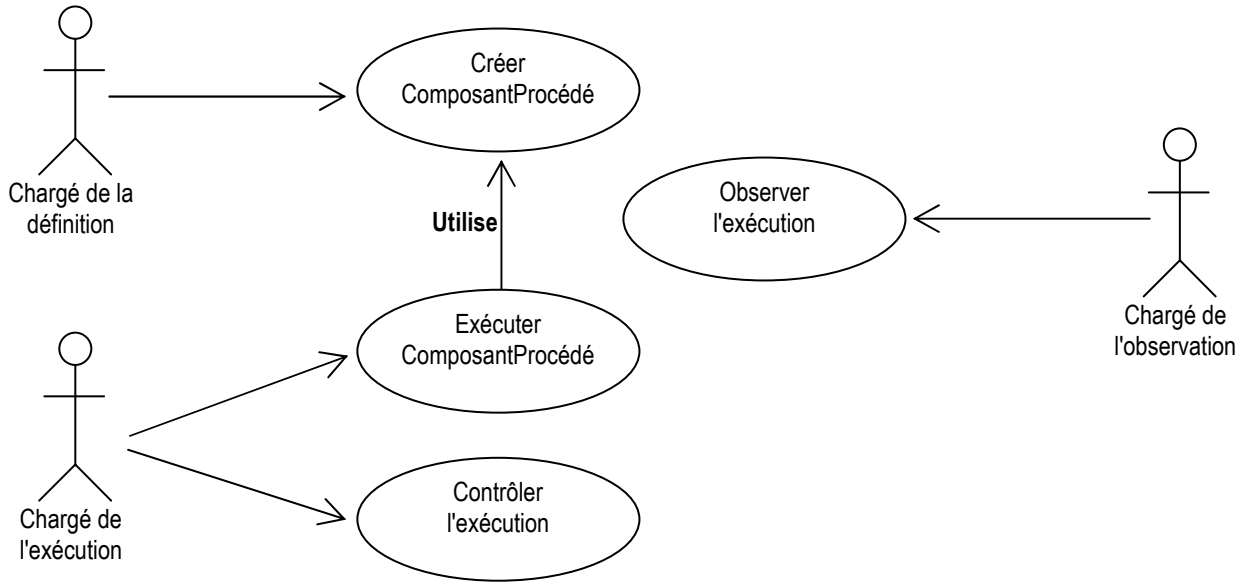


Figure 3.6 : *Les différents cas d'utilisation.*

5.2. Scénarios principaux identifiés au travers des cas d'utilisation.

5.2.1. Créer un composant procédé.

Le scénario suivi, par le rôle "*chargé de la définition"*, pour la création d'un composant procédé est résumé par :

1. Le rôle "*chargé de la définition"* demande la liste de composants procédés à la base de données.
2. La base de données affiche la liste des composants (à travers une interface de dialogue) disponibles dans la bibliothèque de composants procédés.
3. Le rôle "*chargé de la définition"* d'un procédé sélectionne les composants procédés à partir de fabriques (*factory's*) composants concrets.
4. L'objet FactoryComposant crée un composant procédé concret.
5. La FactoryComposant est responsable de la création du diagramme de transition d'état du composant en utilisant le FactoryDiagrammeEtat.
6. Le FactoryDiagrammeEtat crée le DiagrammeEtatComposant.

Le diagramme de séquence de la figure 3.7 illustre bien ce scénario.

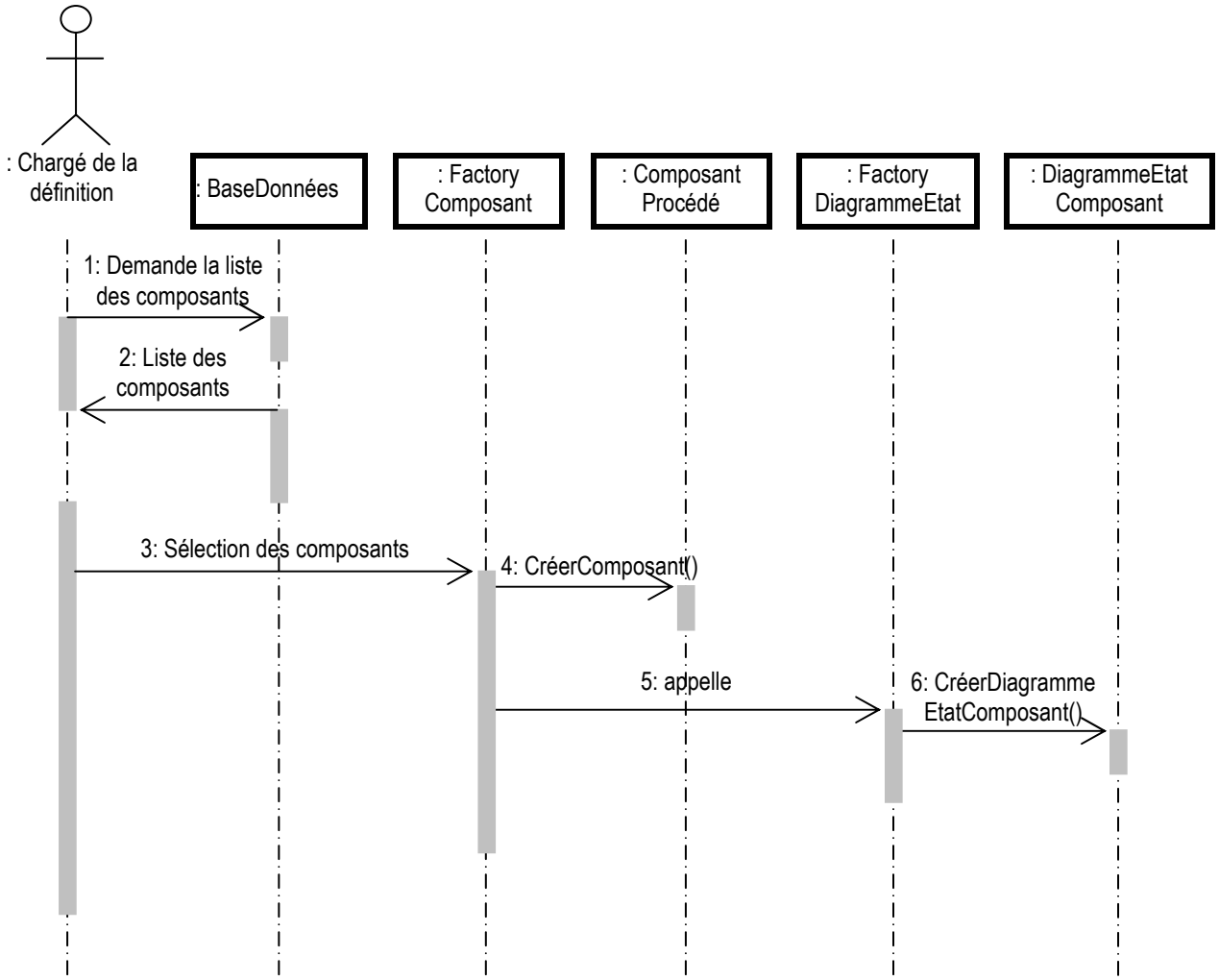


Figure 3.7 : *Création d'un composant procédé.*

Les objets FactoryComposant deviennent les fabriques de composants procédés. Ils correspondent à des modèles des composants procédés enregistrés dans une base de données. Chaque modèle a un formalisme de représentation dépendant de l'environnement centré procédé sur lequel il a été créé.

5.2.2. Exécuter un composant procédé.

1. Le rôle chargé de l'exécution demande l'instanciation et l'exécution du composant procédé
2. Avec l'instanciation commence proprement l'exécution du composant procédé.
3. Un observateur global est déclenché pour observer le développement de l'exécution pour ce composant.

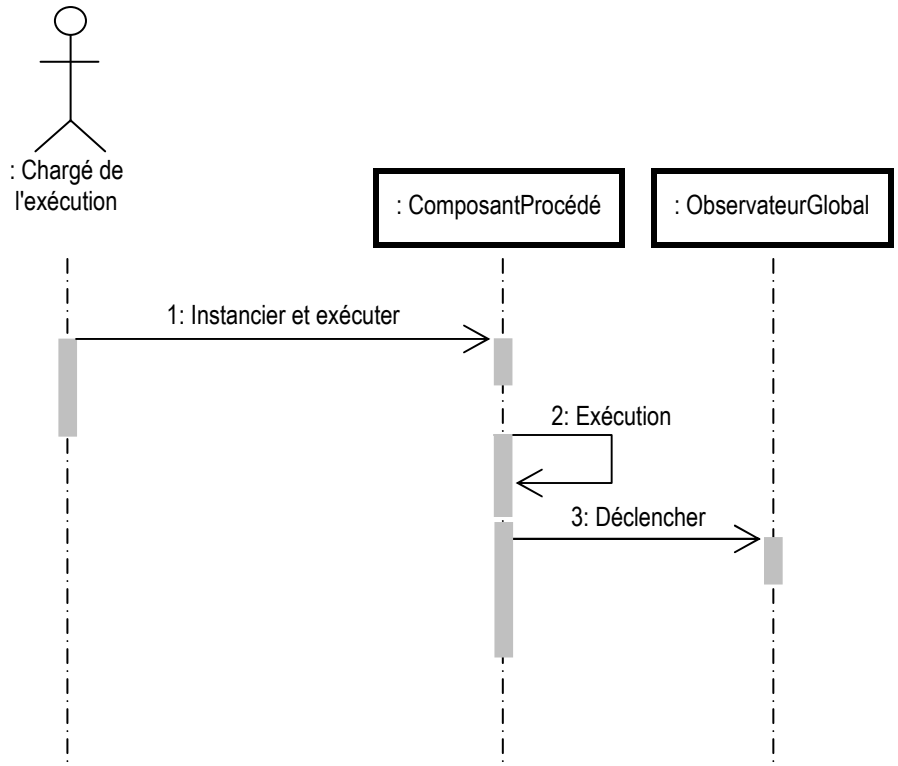


Figure 3.8 : *Exécution d'un composant procédé.*

5.2.3. Contrôler l'exécution d'un composant procédé.

1. Le rôle chargé de l'exécution active l'observateur global correspondant, avec la méthode P*ropriétésObservateur()*.
2. L'observateur global affiche les propriétés du composant et des sous composants fils, enregistrés jusqu'à présent.
3. Le rôle chargé de l'exécution demande le changement d'état du composant procédé au travers de l'observateur global.
4. L'observateur global exécute par exemple, la méthode *Suspendre()*.
5. Quand la méthode s'exécute, il se produit un changement d'état.
6. Le composant procédé notifie le changement d'état à son propre observateurs global et à ceux observateurs externes intéresses par tel événement.
7. La méthode *MiseAJour()* actualise les propriétés enregistrées par l'observateur global.
8. L'observateur global affiche les propriétés plus récentes du composant procédé.

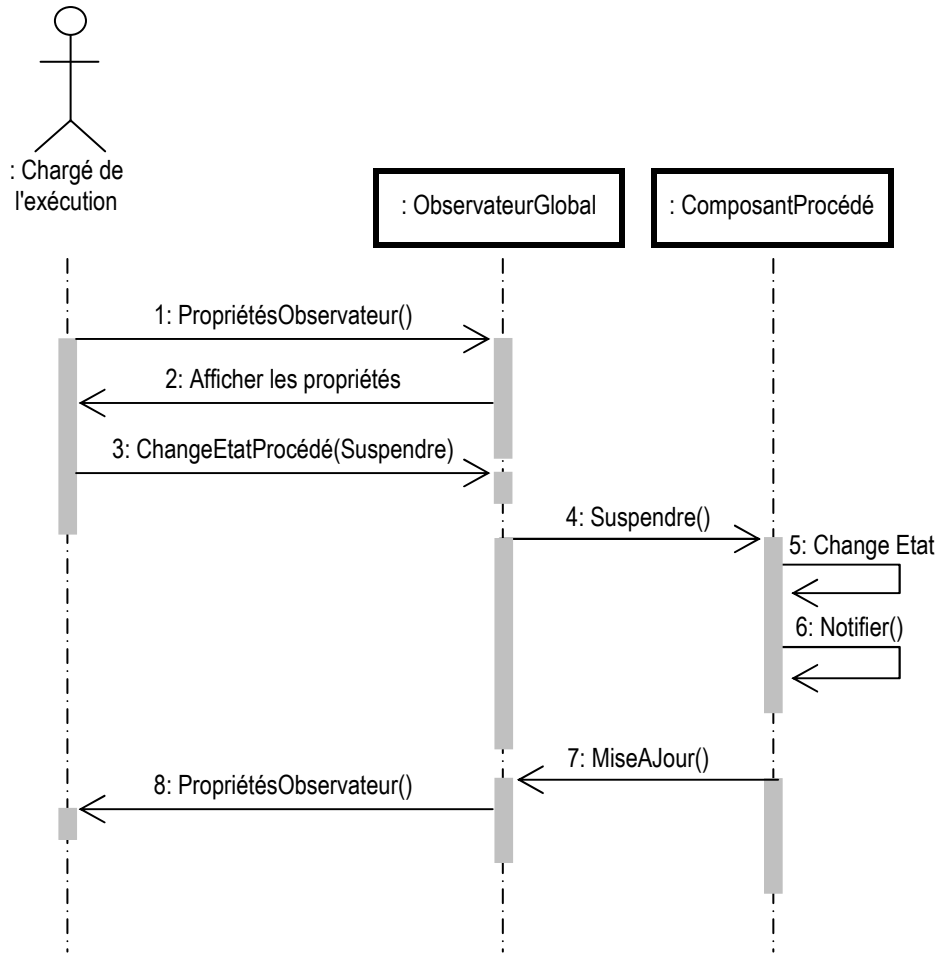


Figure 3.9 : *Contrôle l'exécution du ComposantProcédé.*

5.2.4. Abonner un observateur global externe.

1. L'observateur global externe demande à être abonné à des événements via la méthode *AttacheObservateur().*
2. Le composant procédé retourne la liste d'événements.
3. L'observateur sélectionne le ou les événements auxquels il veut s'abonner.
4. Validation de l'attachement pour éviter par exemple d'abonner plusieurs fois un observateur au même événement ou bien d'abonner un observateur sans permission.
5. Si l'attachement est validé, le message de confirmation est envoyé.
6. En même temps le composant procédé fait l'attachement de l'observateur.
7. Dans l'autre cas le composant envoie un message d'erreur.

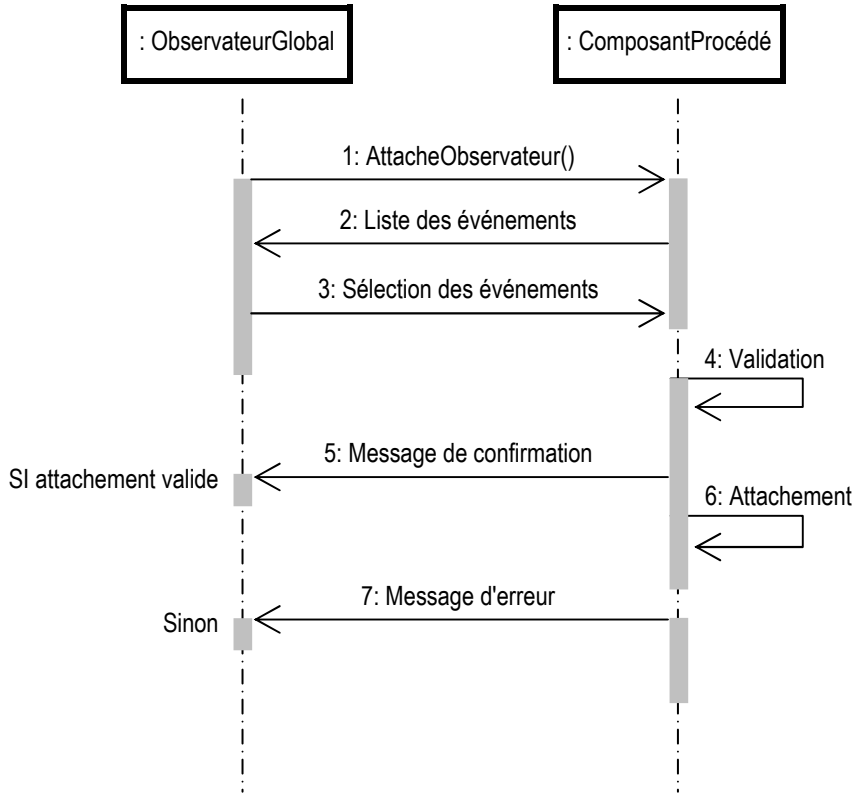


Figure 3.10 : *Abonnement d'un observateur.*

5.3. Diagramme de transition d'états.

Les états qui peuvent arriver au composant procédé sont : **Prêt**, **Exécuté**, **Abandonné**, **Suspendu** et **Terminé**.

Les transitions entre ces différents états peuvent se résumées par :

1. Le composant procédé passe de l'état **prêt** à l'état **exécution** avec l'arrivée de l'événement "DébutProcédé"
2. Le composant procédé sera terminé normalement si l'exécution est menée sans erreurs.
3. Le composant procédé est suspendu si durant l'exécution, des erreurs arrivent.
4. Après correction des erreurs, le composant procédé suspendu reprend l'exécution.
5. Le composant procédé sera abandonné si les erreurs ne peuvent pas être corrigé.

Le diagramme d'état de la figure 3.11 illustre ces transitions.

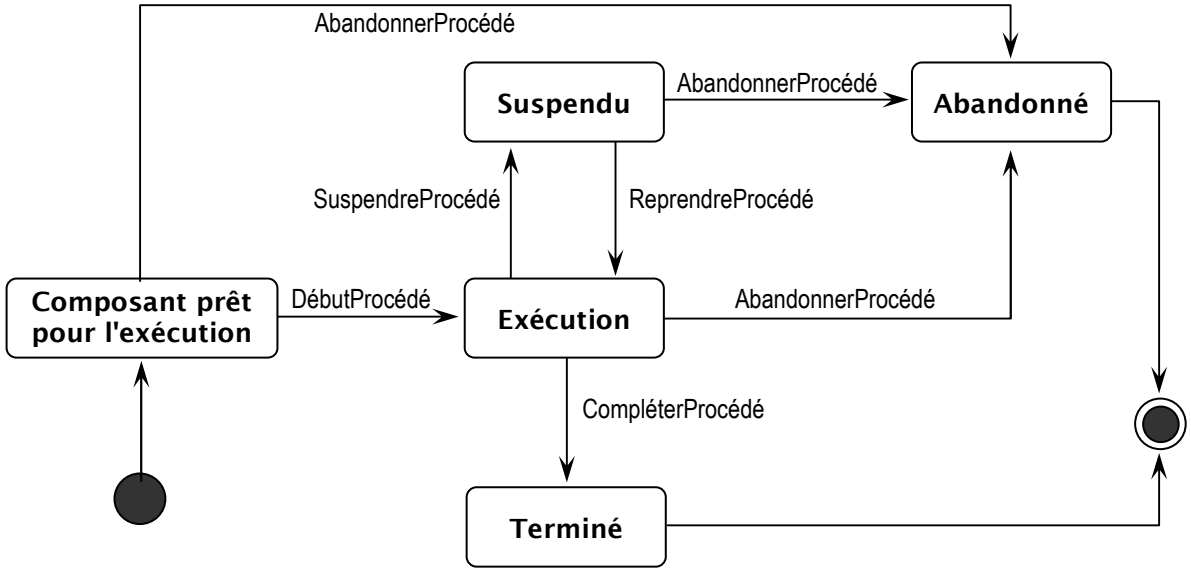


Figure 3.11 : *Diagramme de transition d'états pour un composant procédé.*

5.4. Diagramme de classes.

Dans la suite, nous décrivons le FrameWork orienté objet (voir figure 3.15) permettant :

1. La définition des composants procédés selon le méta-modèle;
2. La création des composants procédés dont les fonctionnalités et propriétés sont similaires à un autre, en appliquant les changements à la définition d'un composant procédé existant; et finalement
3. L'instanciation d'un composant procédé en spécifiant les liaisons que lui permet d'être exécuté.
4. le contrôle de l'exécution du composant procédé en abonnant des observateurs permettant de suivre l'évolution de son exécution.

On peut identifier trois patrons basiques dans ce modèle **:** *fabrication (factory), observateur (observer) et façade (front).*

Dans ce qui suit nous allons analyser ces patrons et les classes principales qui constituent notre FrameWork.

5.4.1. Le patron Fabrication.

Nous proposons un FrameWork conçu pour la modélisation susceptible d'intégrer différents composants procédés. Les classes basiques dans notre modèle de fabrication sont :

FactoryComposant et *ComposantProcédé*. Ce sont deux classes abstraites, par conséquent, nous devons en dériver des sous classes pour réaliser l'implémentation spécifique d'un composant procédé.

Exemple :

Considérons le composant procédé représenté dans le formalisme APEL. Alors *ModèleAPEL* devient la classe concrète de *FactoryComposant* et *Test* la classe concrète de *ComposantProcédé*.

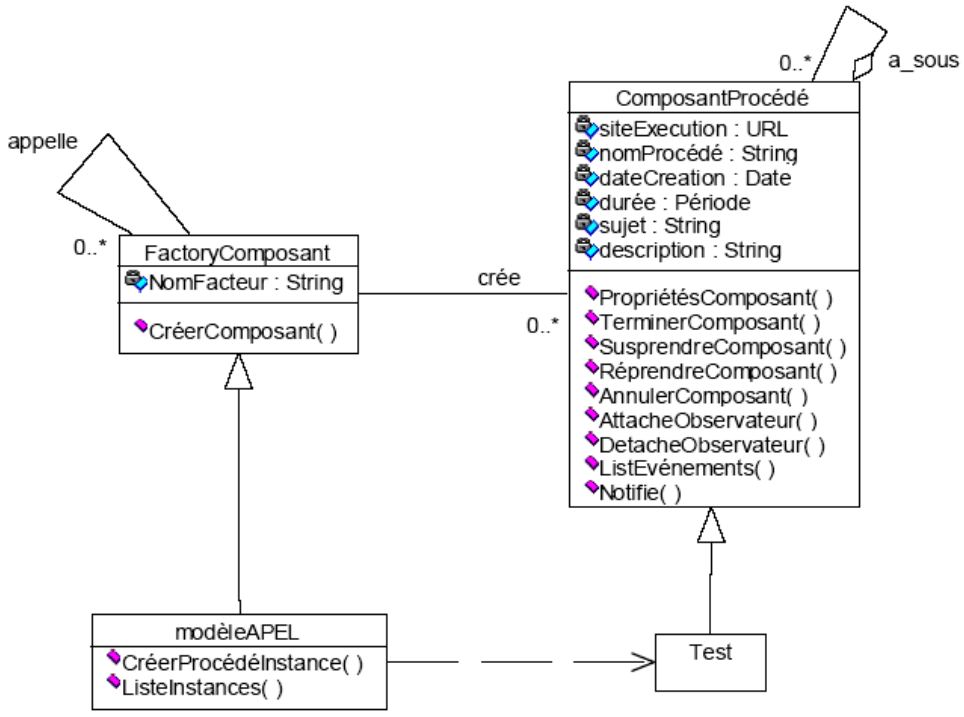


Figure 3.12 : *Le patron Fabrication.*

L'interprétation pour le diagramme de classes de la figure 3.12 :

- *ComposantProcédé* : définit l'interface des objets créés par le *FactoryComposant*.
- *FactoryComposant* : déclare la fabrication, celle-ci renvoie un objet de type *ComposantProcédé*.
- Le *Test* implémente l'interface *ComposantProcédé*.
- *ModelApel* surcharge la fabrication pour renvoyer un objet de type *Test*.

5.4.2. L'Observateur global.

La classe *observateurglobal*, illustrée par la figure 3.13, a deux types de relations :

1. La relation *lié_à* définit une relation d'un observateur global avec un composant procédé. Ici la tâche de l'observateur est d'enregistrer l'information la plus importante sur le composant procédé auquel il appartient et l'information sur les éventuelles sous composants fils, tel que : nom du composant, son état, la durée, le rôle responsable de l'exécution du composant, ... etc.

   L'observateur a le contrôle sur le composant et sur ses sous composants fils ; il peut par exemple, modifier l'état du ou des composants procédés.
2. La relation *abonner* définit une interdépendance de façon telle que, quand le composant procédé change d'état, tous les observateurs globaux externes qui en dépendent soient notifiés, et automatiquement mis à jour. L'observateur global externe n'aura pas la permission de modifier l'état du composant procédé en question.

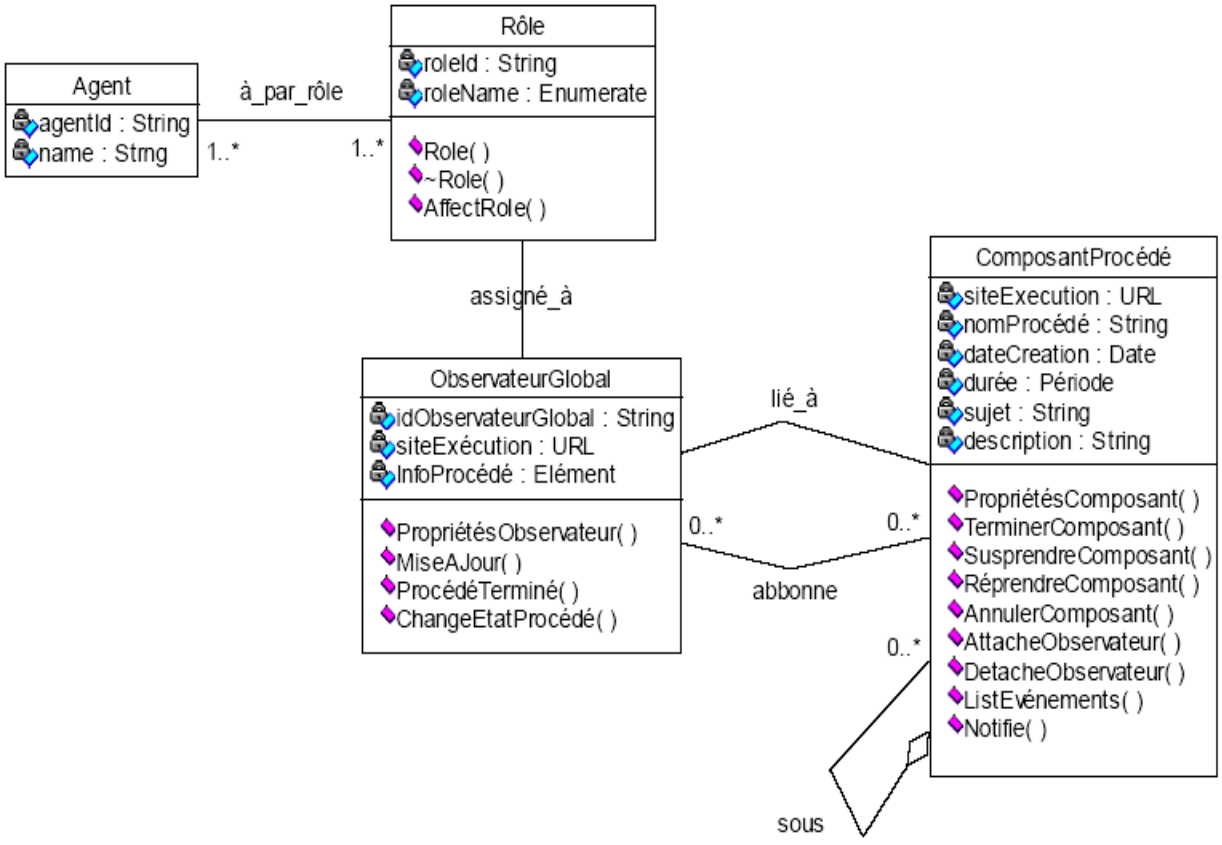


Figure 3.13 : *L'observateur global.*

5.4.3. Le patron de façade.

La classe *ComposantProcédé* joue le rôle de façade : elle doit offrir à l'utilisateur une interface unique et simple vers les services du *ComposantProcédé*. Elle connaît les classes pour une requête. Mais les classes du sous-système ne connaissent pas la façade, c'est à dire qu'elles n'ont pas de référence à celle-ci. Les observateurs communiquent avec le composant procédé en envoyant des requêtes à la façade, qui répercute celles-ci aux objets appropriés du sous-système.

5.4.4. Le Composant MétaData.

Le composant procédé produit des méta données constituées par les classes : attribut, méthode et événements. On peut ainsi manipuler l'information et les opérations des composants procédés. Il suit un schéma conceptuel bien établi. La figure 3.14 illustre le diagramme de classes correspondant.

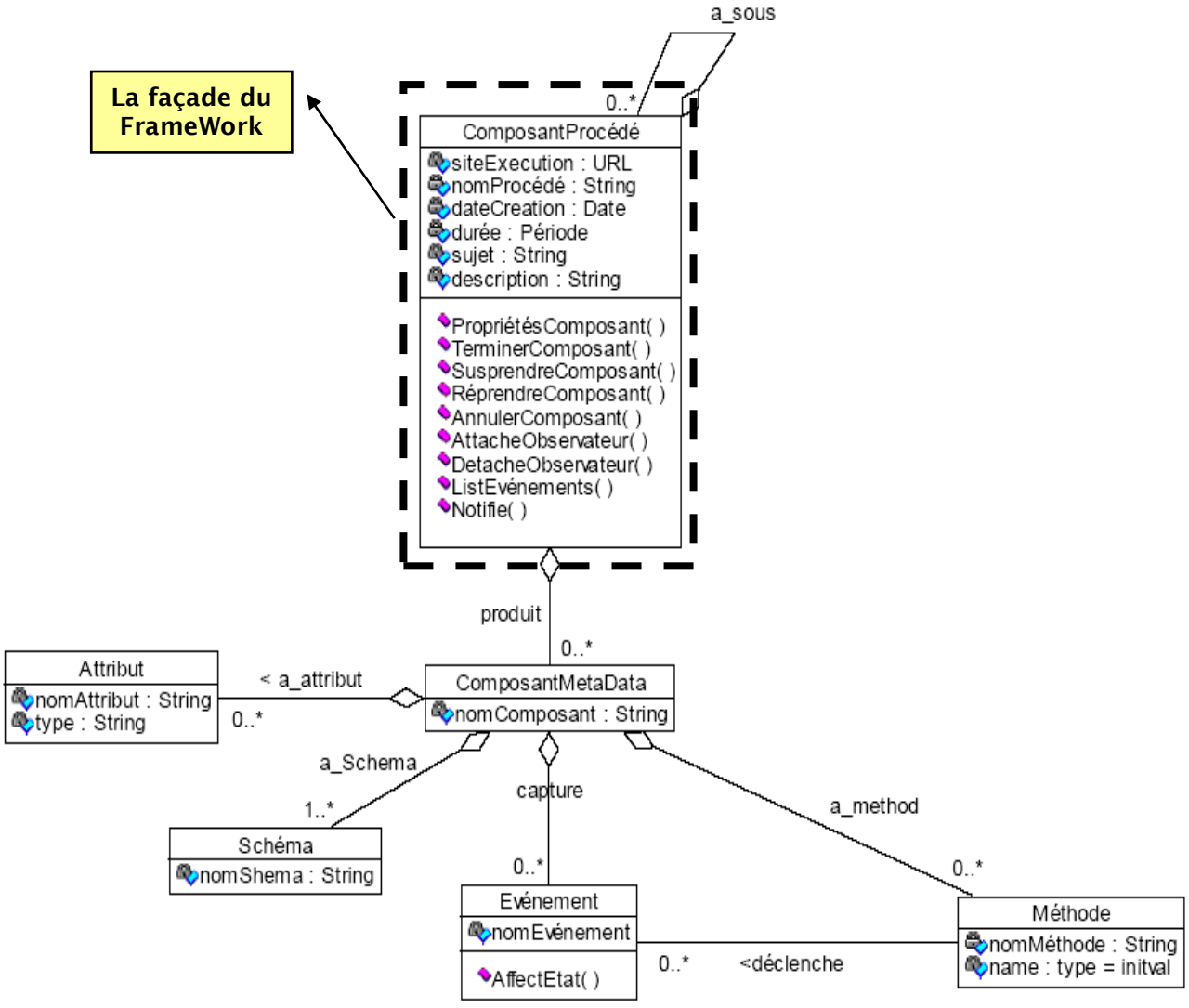


Figure 3.14 : *MétaData produits par un composant procédé.*

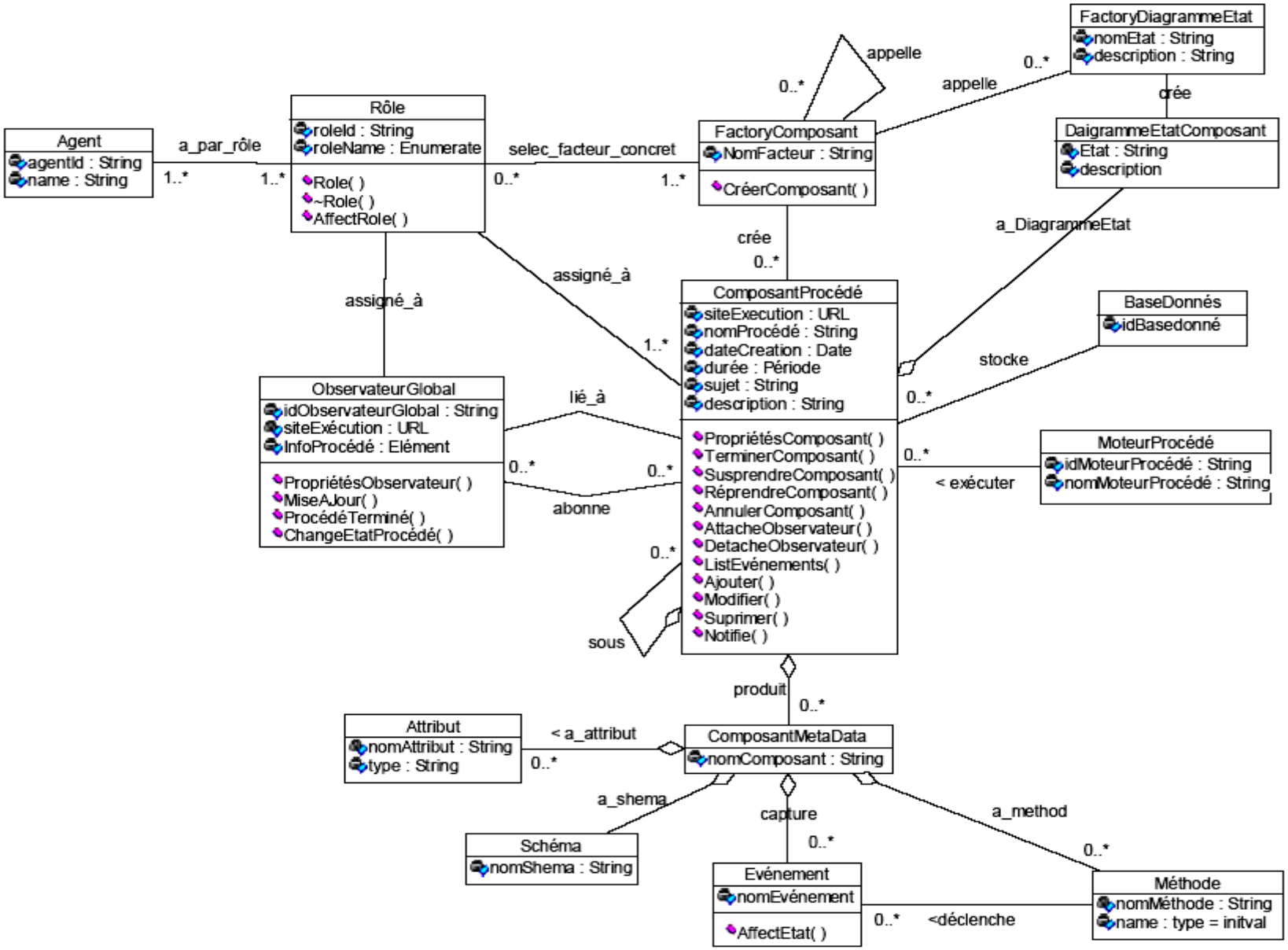


Figure 3.15 : *FrameWork pour des composants procédés.*

6. Exemple de fonctionnement.

Remarquons, que la création et l'exécution d'un modèle par composants est assez dynamique, surtout car elle peut impliquer l'exécution parallèle de plusieurs composants, chaque composant à son tour peut déclencher l'exécution de d'autres sous composants parallèles, enchaînés ou imbriqués (modèles étudiés dans l'état de l'art).

Analysons un exemple pratique qui représente "Le procédé test dans le développement d'un système logiciel".

Les sous composants du procédé *test* sont :

- *Test d'intégration et,*
- *Test unitaire*.

Chacun sous composant reçoit comme entrée un plan de test et comme sortie ils délivrent un document contenant le rapport du test. Pour chaque composant il existe un rôle *gestionnaire* responsable de l'exécution de chaque composant procédé. La figure 3.16 illustre le schéma pour le test avec des éléments décrits au dessus :

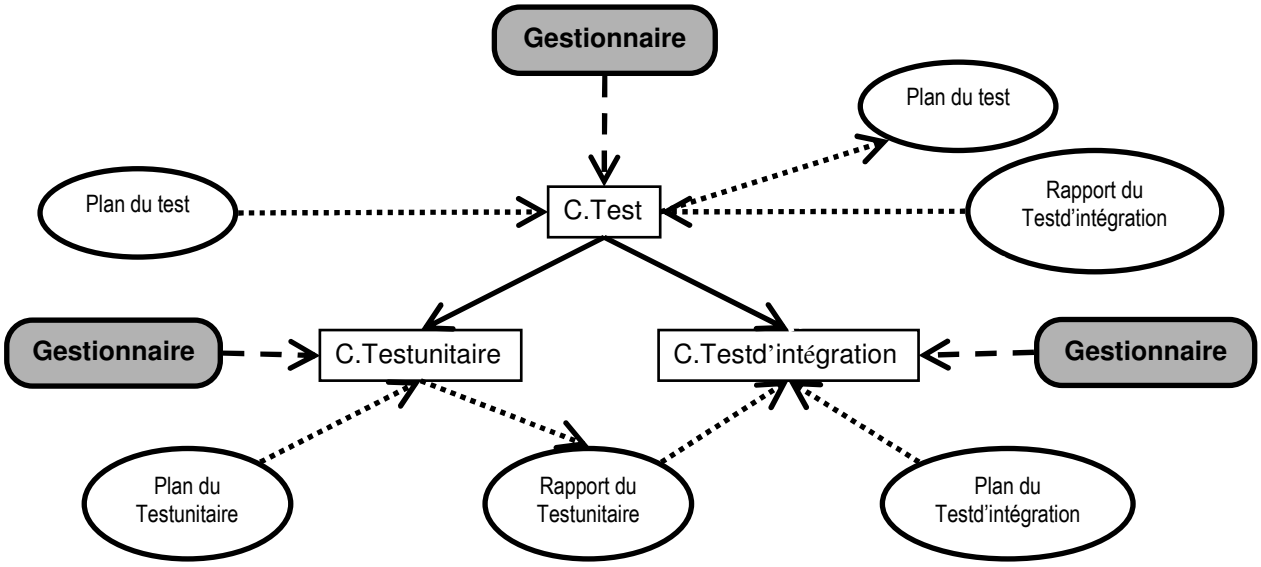
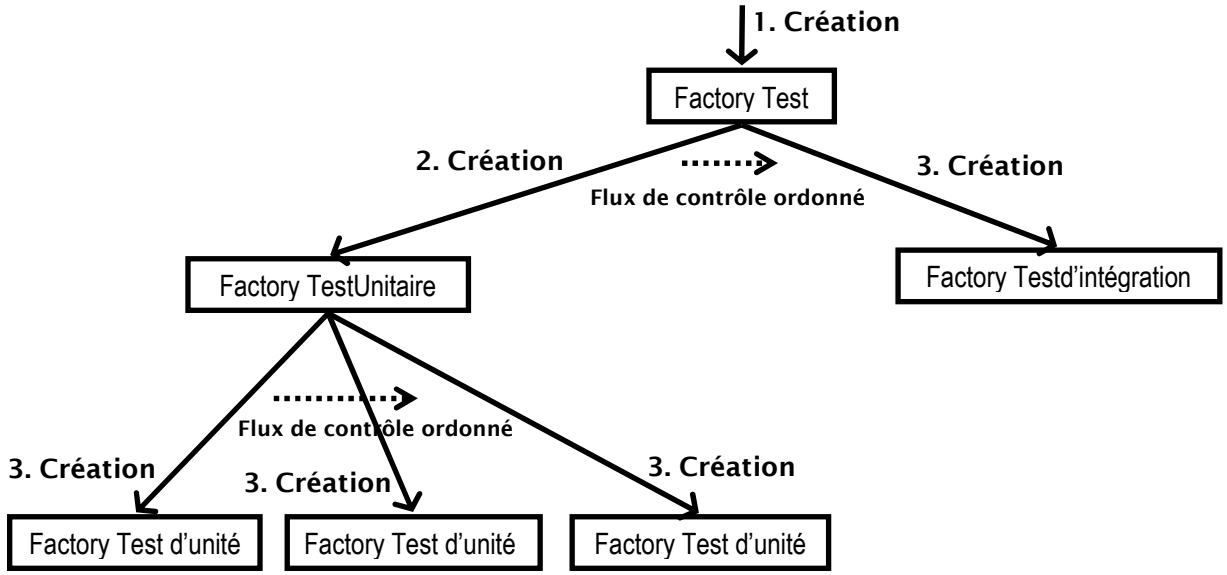


Figure 3.16 : *Schéma pour le procédé Test.*

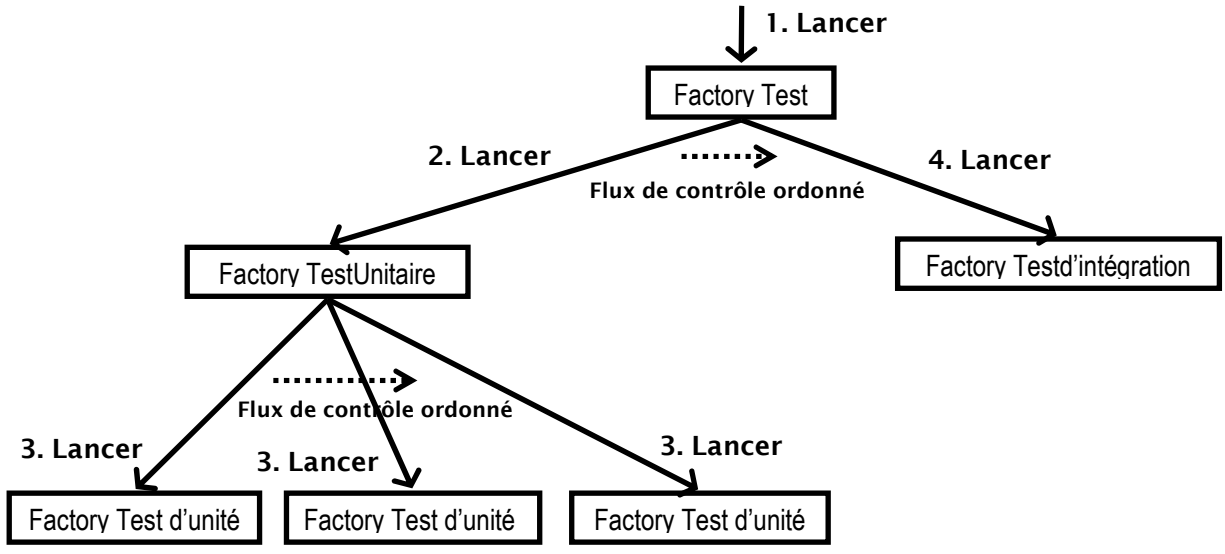
- Le composant "test unitaire" est le niveau de test le plus bas réalisé pour les classes, les blocs et les paquets de service.
- Le but composant "test d'intégration" est de tester si différentes unités développées fonctionnent correctement ensemble.

Chaque composant décris ci-dessus est composé, à son tour, de sous composants Test dans son domaine d'application. Pour simplifier notre exemple, nous considérons le cas où l'exécution du test se trouve dans le niveau le plus bas, i.e. dans les composants d'unité.

La création de composants procédés est menée à bien par des fabriques *« factory »* illustrés par l'arbre (a) de la figure 3.17. En cours d'exécution, une *factory* en appelle une autre pour demander la création d'un composant procédé. La phase d'exécution est illustré par l'arbre (b), ici le composant test lance l'exécution des sous composants test unitaire, qui à leurs tour lancent l'exécution en parallèle de sous composants test d'unité. Le composant *test d'intégration* est mis en exécution, une fois que les *tests d'unité* ont fini leur exécution.



(a)



(b)

Figure 3.17 : *Arbres illustrant la création et exécution des sous composants test.*

Regardons le comportement de la machine d'exécution pour le composant test au travers des diagrammes de séquence (voir figure 3.18 et 3.19) illustrer par les scénarios suivants :

Exécution de test.

1. Le composant test demande la liste de composants procédés à la base de données.
2. Après il appelle l'outil qui va créer le composant sélectionné.
3. Cet outil va créer l'environnement nécessaire pour mener à bien l'exécution du composant procédé *TestUnitaire*, à savoir : le diagramme d'états, les variables, etc.
4. En suite un événement est diffusé signalant l'existence du nouveau composant *TestUnitair*e.
5. Test lance l'exécution de ce composant,
6. Un autre événement est diffusé signalant le début de l'exécution de *TestUnitair*e.
7. Quand il termine son exécution, l'événement terminé est diffusé via le gestionnaire d'événements.

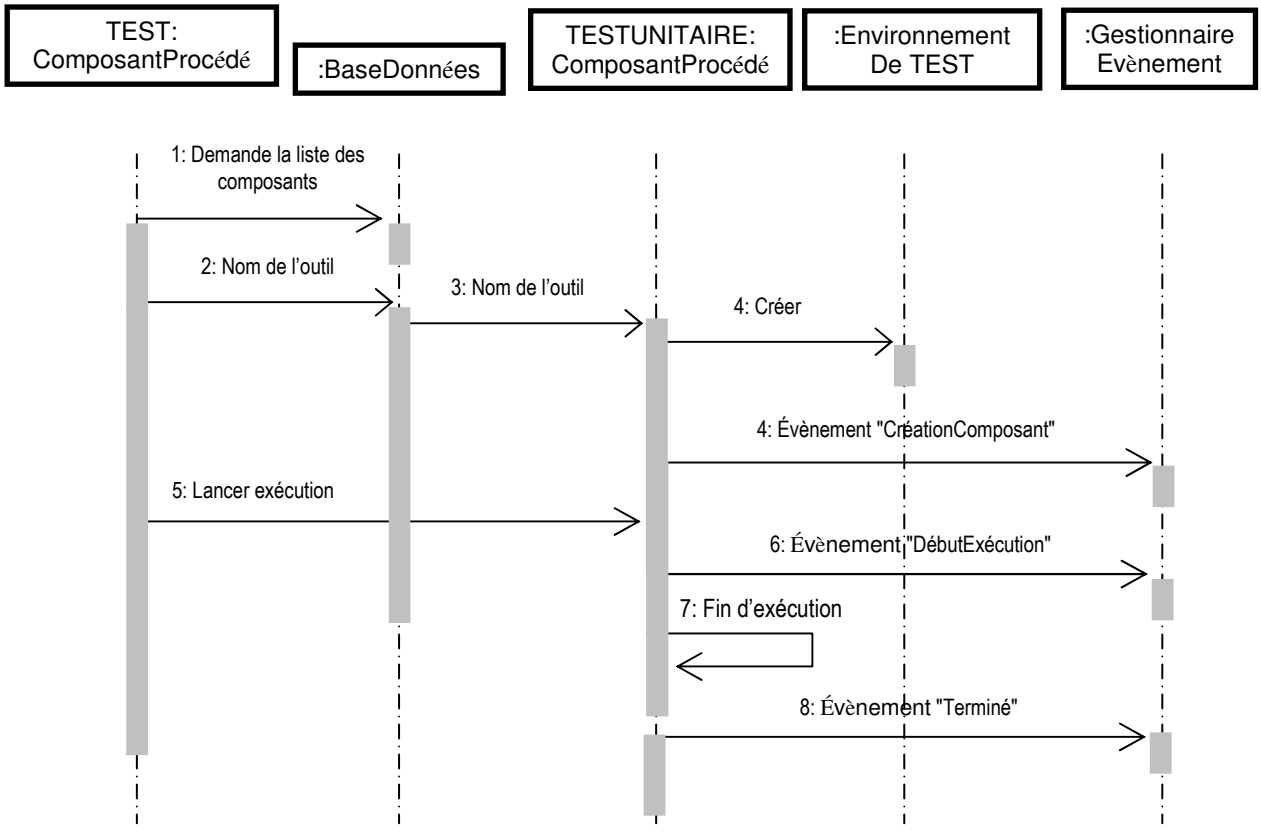


Figure 3.18 : *Exécution du procédé Test.*

Interopérabilité, entre des composants, basé sur des événements.

1. Le composant Test s'abonne à des événements fin de l'exécution du *TestUnitaire* et *TestIntegration.*
2. Le composant *testUnitaire* est en train de s'exécuter.
3. Quand le *TestUnitaire* termine son exécution, un événement "terminé" est diffusé par le gestionnaire d'événements, et en suit il renseigne le composant test sur le changement d'état.
4. Après, le composant procédé lance l'exécution du composant *TestIntégration.* Cette action déclenche l'événement "débutProcédé".
5. La fin de l'exécution du composant *TestIntégration* finalisera le composant *Test.*
6. Ce composant notifiera aux observateurs abonnés à l'événement "terminé Test" afin de que ceux ci déclenchent des actions propres à son exécution.

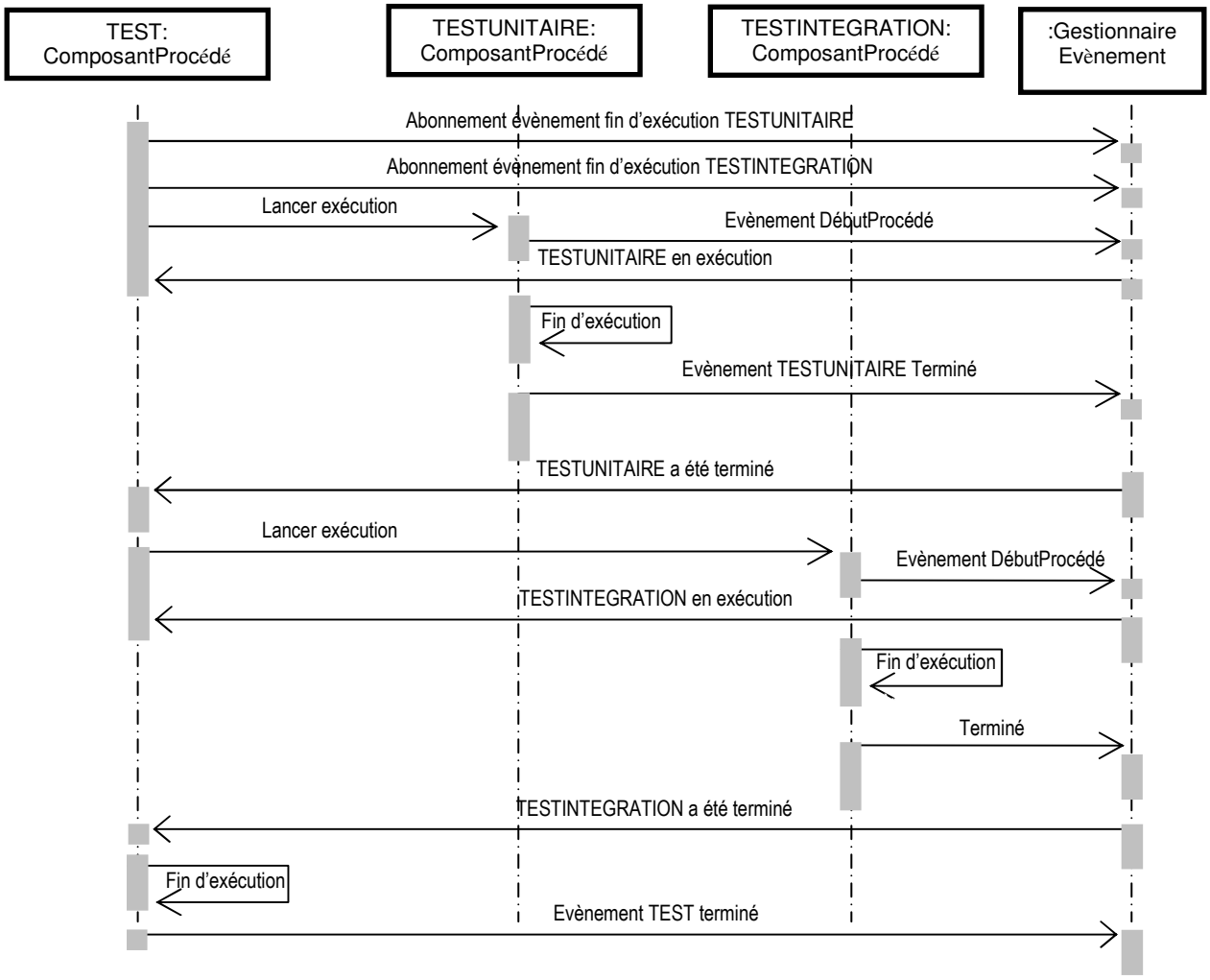


Figure 3.19 : *Interopérabilité entre des composants procédés.*

Le diagramme d'état de notre exemple est illustré par la figure 3.20 ci dessous. Remarquons que tous ces sous composant auront le même comportement, la récursivité est indiquée par la flèche "créer sous composant test".

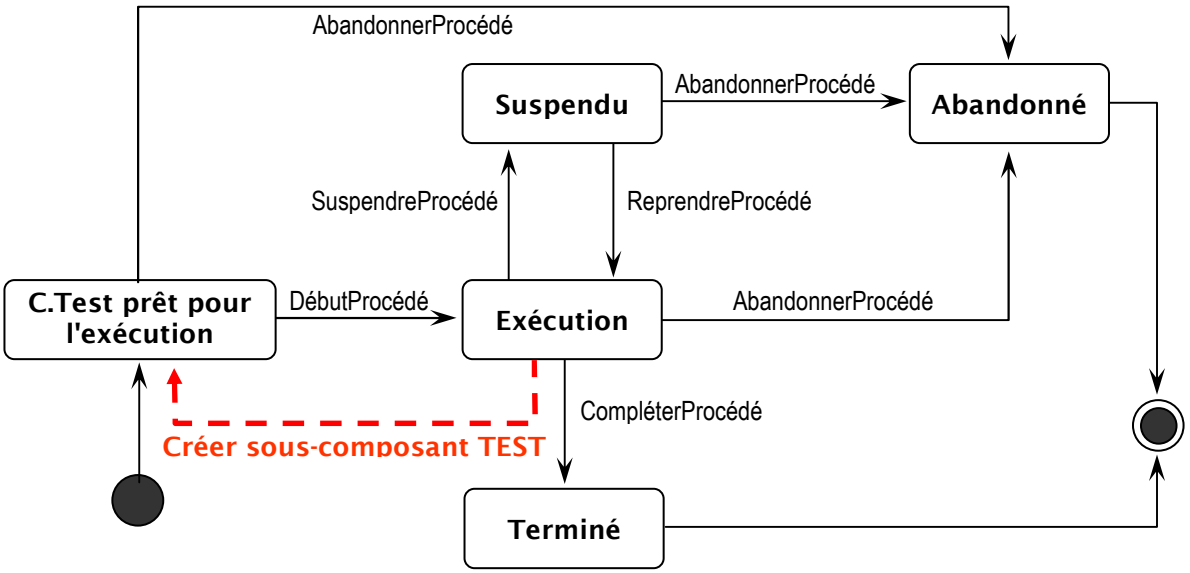


Figure 3.20 : *Diagramme de transition d'état pour le composant test.*

7. Conclusion.

L'architecture de notre approche, permettant la modélisation de procédés par composants, a été présentée dans ce chapitre à travers les concepts utilisés, la modélisation du FrameWork en UML via les cas d'utilisation, les diagrammes de séquences, les diagrammes d'états ainsi que les diagrammes de classes correspondants et enfin, l'élaboration d'un exemple pour la création et l'exécution du procédé TEST.

Dans le chapitre suivant, nous allons présenté l'expérimentation de notre approche à travers la mise en place d'un prototype permettant la mise en oeuvre du modèle de composants procédé, en présentons le fonctionnement global de l'architecture, les interfaces de dialogue pour la création d'un modèle par composants, et pour la gestion de l'exécution de chacun des composants procédés.

Chapitre -4-

Expérimentation de l'approche

Dans cette section nous allons présenter le prototype permettant la mise en œuvre de l'environnement de modélisation de procédés logiciels par composant baptisé PEDE (**P**rocess **E**nvironment for **D**evelopment and **E**xecution). Nous présentons l'architecture de l'environnement ainsi que les interfaces de dialogue pour la création d'un modèle par composants, et pour la gestion de l'exécution de chacun des composants procédés.

L'architecture est donc formée par :

- Une interface permettant à l'utilisateur de regarder le développement d'un procédé. L'information fournie par celle-ci, doit répondre en adéquation avec les besoins inhérents au rôle de l'utilisateur.
- Une Interface permettant la définition des procédés logiciel en utilisons Des Composant procèdes.
- Le moteur d'exécution, regroupe les différents modules de la définition d'un procédé et les exécutent.
- La base d'objets contient la définition des procédés stockés de façon persistante. Il y aura un gestionnaire de la base d'objets responsable de la cohérence et de la disponibilité des informations nécessaires aux autres composants de l'environnement.
- Le serveur des communications, suppose que des interfaces de communication soient créés pour les composants et que des protocoles de communication soient établis pour définir les modalités d'utilisation des services disponibles.

1. L'environnement PEDE v1.

Le module de définition doit doter la personne chargée de la définition des modèles de procédés, de tous les outils nécessaire, pour la spécification des procédés logiciels sous forme de composants. Ces derniers sont crées selon deux façon déférentes :

- Soit les crée directement, en spécifiant leurs caractéristiques,
- Soit en important des composants qui sont stockés dans la bibliothèque des composants et en les adaptant aux procédés en cours.

Le module de définition de procédé doit fournir les moyens de communication avec le moteur d'exécution afin de lui passer les instances de procédé qu'on veut exécuter. Nous présentons dans ce qui suit, une vue sur l'environnement principale de développement des procédés logiciel qu'on l'a nommé "**PEDE V1"** pour : **Process Environnement Development and Execution**, qui contient ce qui suit :

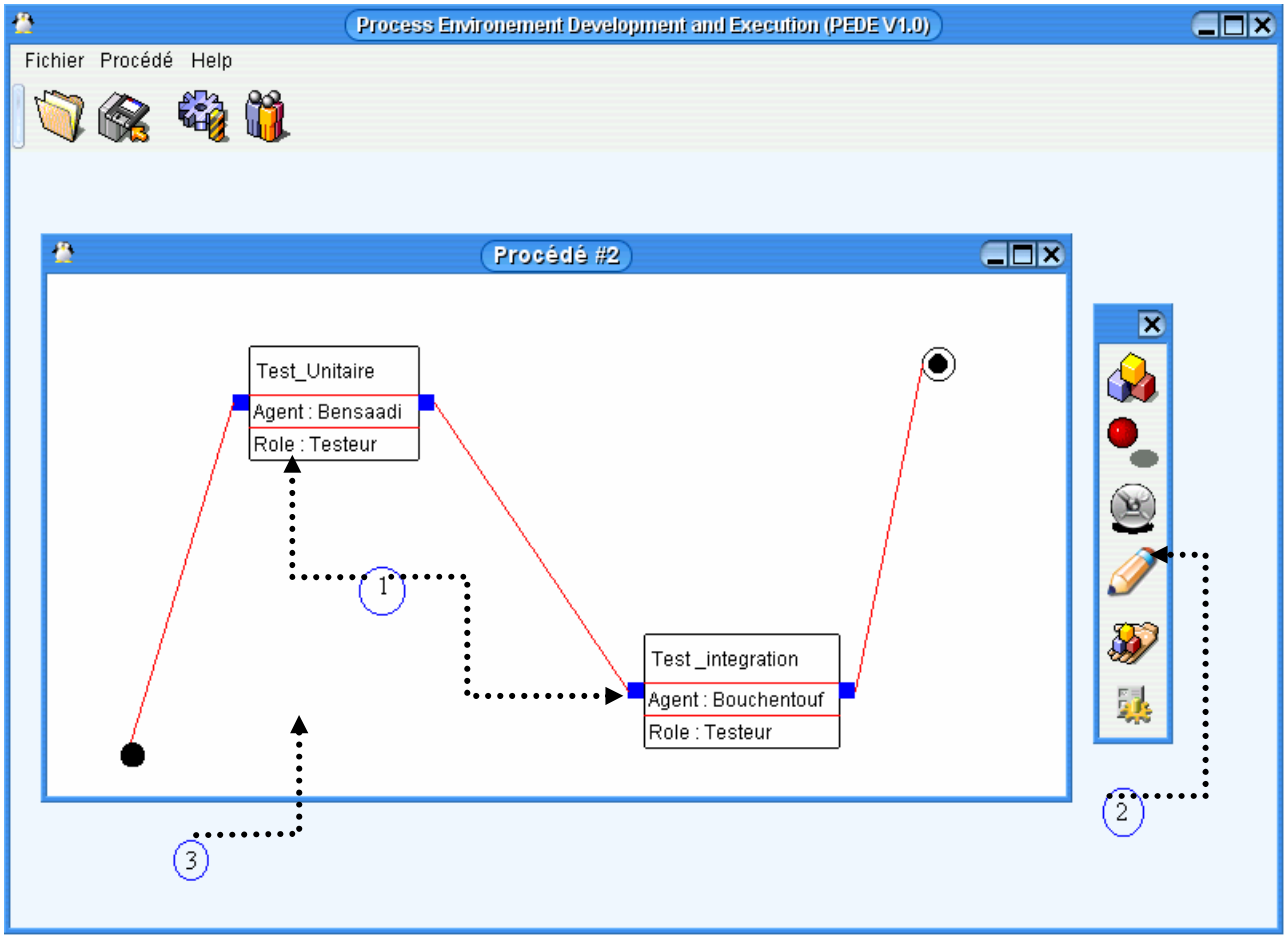


Figure 4.1 : *Vue Principale de PEDE v1.*

1. Deux Composants Procédé au sein d'un Model de Procèdes logiciel,
2. Outil de définition des procédé logiciel composée successivement par :
    a. Boite de création d'un nouveau procédé.
    b. Insertion un Begin_Point , c'est le point par le quel commence le procédé.
    c. Insertion d'un End_Point qui représente la fin du procédé logiciel.
    d. Boite de lien qui permet de définir les dépendances entre le composant d'un procédé.
    e. Boite d'importation des procédés
    f. Bouton de lancement qui permet de lancer une instance d'un procédé déjà défini.
3. Espace de définition des procédés logiciels.

2. Définition du procédé.

2.1. Création d'un Nouveau Composant.

Pour créer un nouveau composant, il faut spécifier :

- Le Nom de ce Composant.
- Le rôle nécessaire pour gérer l'activité dédiée à ce composant.
- Affecter ce composant à un agent qui remplit la condition du rôle.
- Les Artefacts en entrée de ce composant.
- Les Artefacts en Sortie de ce composant.
- Décrire le composant par une Série de Mots significatif, afin de faciliter sa réutilisation lors des futures importations de ce composant (le champ description, va réapparaître au niveau de la fenêtre importation des procédés).

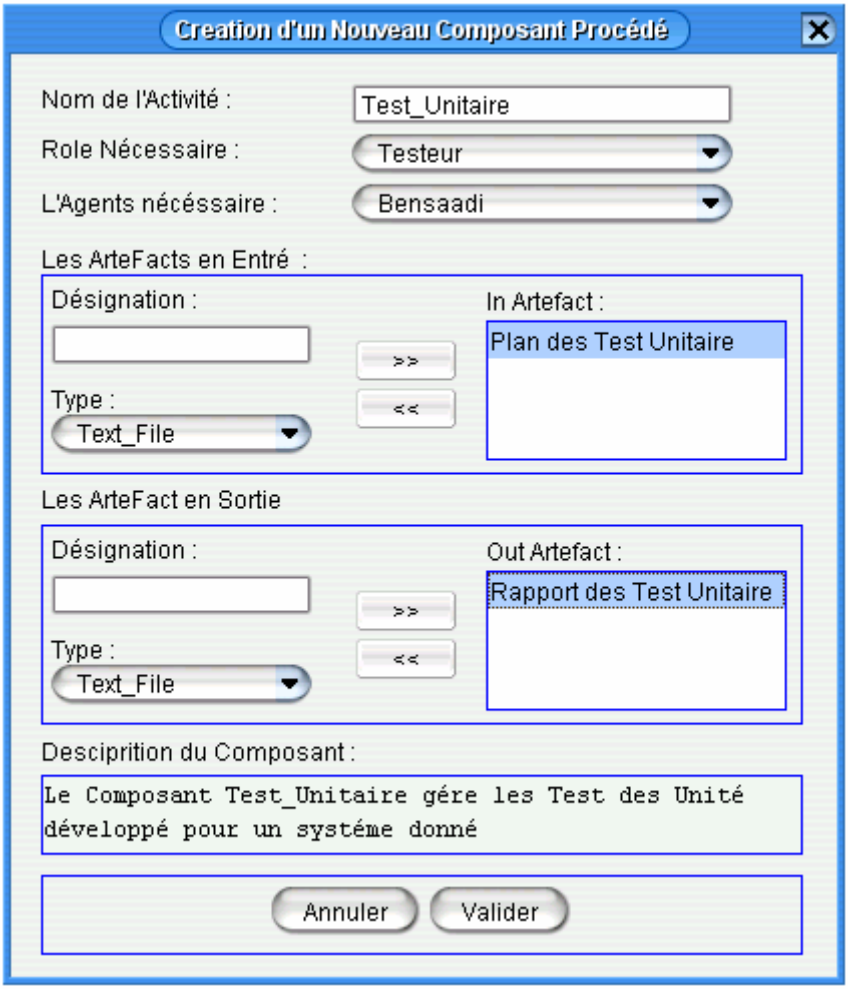


Figure 4.2 : *Boite de dialogue de création d'un nouveau composant.*

2.2. Importation d'un composant.

L'interface d'importation des composants procédés nous permet d'importer des composants qui ont été déjà développé et qui sont stockés au niveau de la base de donné PEDE v1, la seule information à ajouter est l'agent à qui doit être assigné le composant importé.

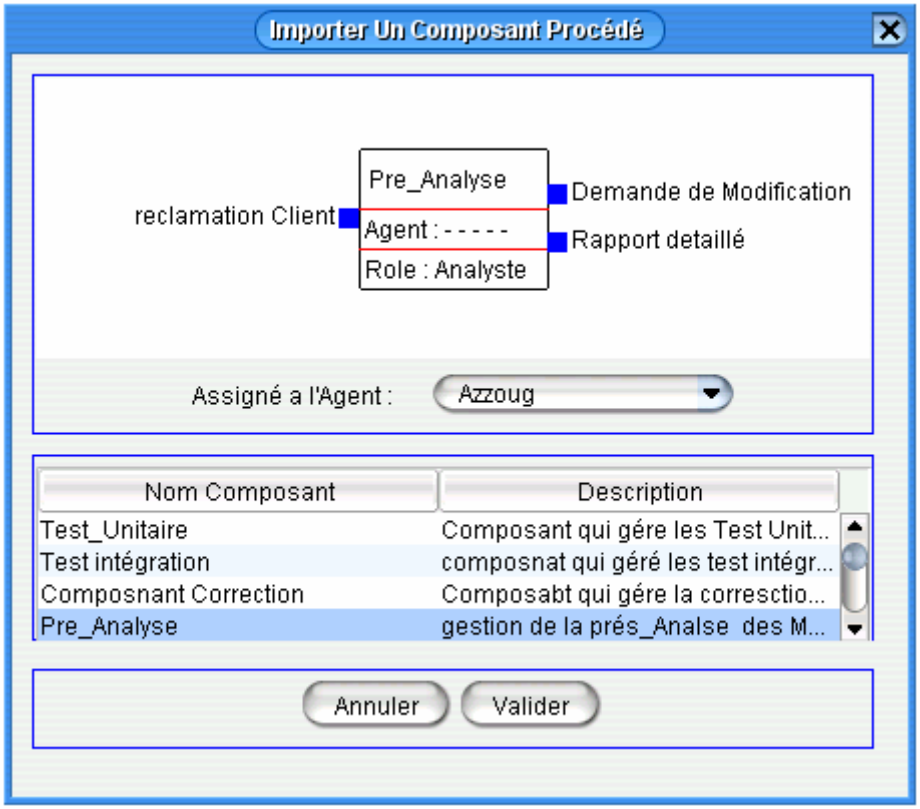
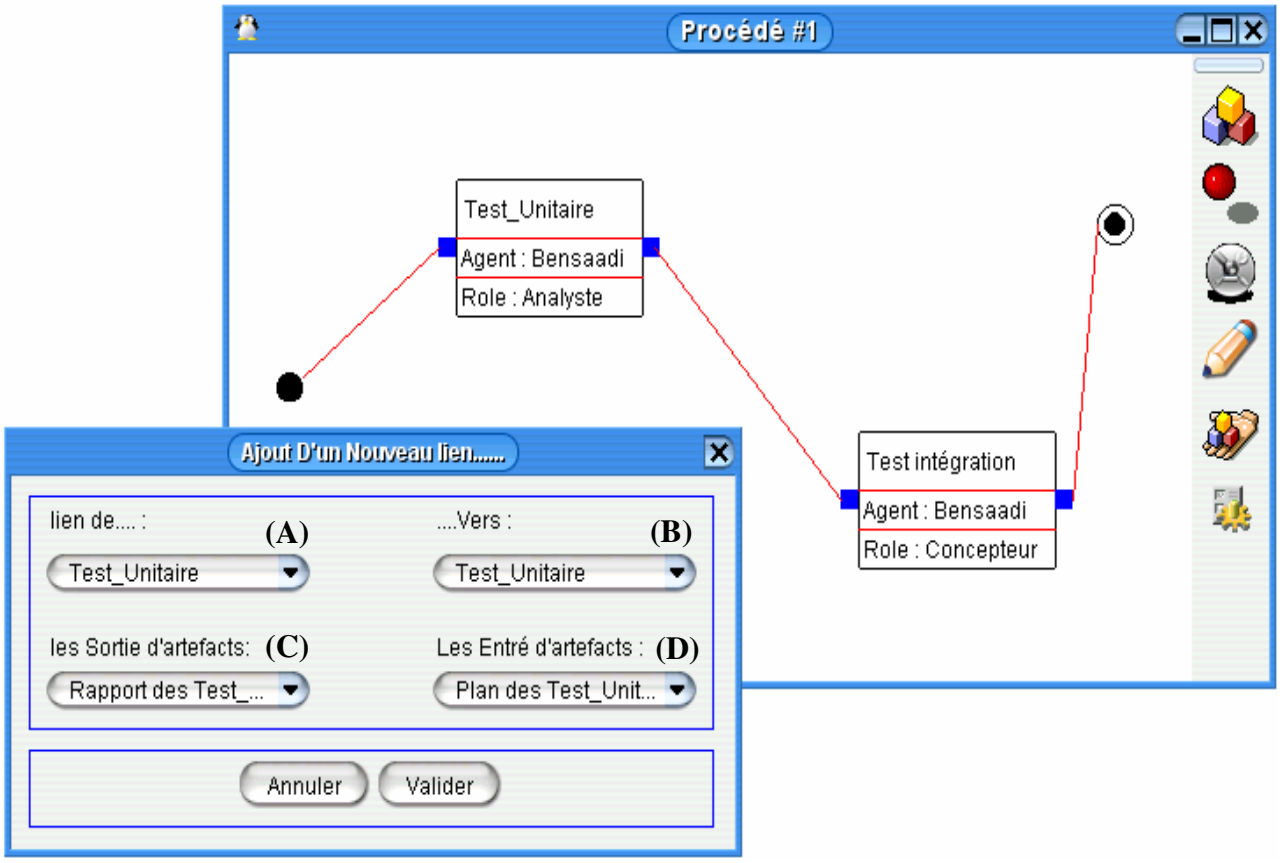


Figure 4.3 : *Boite d'importation des Composants Procédés.*

2.3. Dépendances des Composants.

L'interface de dépendances ou la boite de création des liens, permet de spécifier les dépendances entre les différents composants créés pour un procédé donné, les dépendances peuvent être de deux types :

- Dépendances entre Artefacts : les liens qui relient les artefacts de sortie d'un composant avec les artefacts d'entrée d'un autre composant.
- Dépendances entre composant : ce sont les relations qui définissent l'enchaînement des composants procédés



(A) Composant de départ (qui génère un output artefact).

(B) : Composant d'arrivée (qui reçoit un input artefact).

(C) : L'artefact en sortie.

(D) : L'artefact en entrée

Figure 4.4 : *Création des liens.*

3. Exécution des procédés.

Afin d'exécuter des procédés définis dans le Module de Définition, il faut convertir le formalisme du procédé développé vers le langage supporté par le moteur d'exécution, dans notre cas le moteur d'exécution qu'on va implémenter est basé sur un langage dérivé de XML, le choix de langage XML et justifier par la souplesse et la richesse qu'offre XML. XML nous permet de créer nos propres balises et de les utiliser pour satisfaire nos propres besoins, dans notre cas on a besoin de représenter des procédés logiciels qui sont composés de composants, ces derniers ont des relations entre eux et qui doivent respecter certain ordre durant leur exécution.

Dans ce qui suit, un extrait de langage sur lequel est basé notre Moteur d'exécution, est illustré.

```
<Procédé Name_Procédé=#Nom_du_Procédé>
        <Begin_Point To_Composant=#Id_Composant In_Artefact=#Artefact_Désignation/>
        <Compsant IdComposant=#idComposant Agent=#Id_Agent Role=#Id_Role>
                <In_Artefacts>
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                        ... ... ...
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                </In_Artefacts>
                <Out_Artefact>
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                        ... ... ...
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                </Out_Artefact>
        </Compsant>
        <Compsant IdComposant=#idComposant Agent=#Id_Agent Role=#Id_Role>
                <In_Artefacts>
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                        ... ... ...
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                </In_Artefacts>
                <Out_Artefact>
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                        ... ... ...
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                </Out_Artefact>
        </Compsant>
        ... ... ...
        <Compsant IdComposant=#idComposant Agent=#Id_Agent Role=#Id_Role>
                <In_Artefacts>
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                        ... ... ...
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                </In_Artefacts>
                <Out_Artefact>
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                        ... ... ...
                        <Designation=#Designation Type=#Type_ArteFact Description=#Description/>
                </Out_Artefact>
        </Compsant>
        <End_Statement>
                <End_Point Label=#label Out_From=#Id_Composant Out_Artefact=#Artefact_Désignation/>
                <End_Point Label=#label Out_From=#Id_Composant Out_Artefact=#Artefact_Designation/>
                ... ... ...
                <End_Point Label=#label Out_From=#Id_Composant Out_Artefact=#Artefact Designation/>
        </End_Statement>
</Procédé>
```

Les procédés définis dans le module de définition, sont représentés graphiquement (flots de donné entre les différents composants du procédé), cette représentation est appelée **formalisme graphique d**e représentation, ce dernier et différent de celui utilisé par le moteur d'exécution, donc afin de faire communiqué le module de définition avec le moteur d'exécution, il faut convertir la représentation graphique du procédé en utilisant le langage LDP défini précédemment.

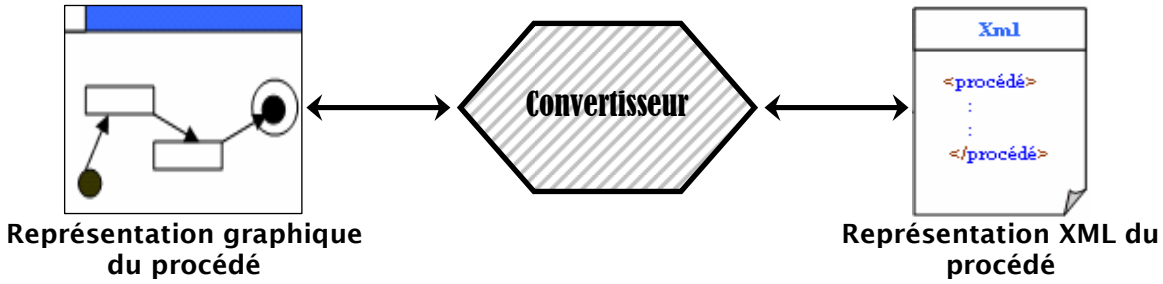


Figure 4.5 : *Le Convertisseur de formalismes.*

3.1. Le Module d'exécution.

Le module d'exécution est responsable d'exécuter les procédés définis dans le module de définition, pour cela le moteur d'exécution intègre un serveur d'événement (observateur d'événement) qui est chargé d'observer le comportement des composants et d'informer le module d'exécution, sur les changements des états des différents composants procédés qui sont en cours d'exécution. Par exécution en sous entend l'interprétation, en effet le procédé est interprété par le module d'exécution : pour chaque balise du langage LDP le module associé une action. Soit la première balise du code d'un procédé logiciel représenté comme suit :

```
<Begin_Point To_Composant="Test_Unitaire" To_In_Artefacts="Plan de Test"/>
```

Le module d'exécution génère les actions suivantes :

Actions générées pour l'exemple	Description des actions
`get_Composant_Location("Test_Unitaire")`	Localisé le Composant (obtenir l'adresse de son emplacement sur le réseaux sou forme d'une adresse IP, ou sous la forme de nom de Machine
`get_Agent_of_Composant("Test_Unitaire")`	obtenir le Nom de l'agent Agent_Name associer a ce Composant.
`Verify_Connectivity_of_Agent (Agent_Name)`	Vérifie si l'agent Agent_Name est connecté

`if_Agent_Connected`
`  Send_In_Artefact("Plan de Test")`
`else`
`add_To_Waiting_Table_Of_Connection`
`(Agent_Name,"Plan de Test")`
	si l'agent est Connecté le Moteur lui envoie les Plan des TestUnitaire qui correspond au Input_artefact. Dans le cas ou l'agent n'est pas connecté le Moteur d'exécution le met dans la table des Agent qui ont des Composant qui leur sont affecter et met l'état du composant en attente. dans ce cas la connexion de l'agent au Moteur de procédé génère un événement qui active le Composant

`add_To_Waiting_Table_Of_OutputArtefact` `("Test_Unitaire")`	ici le moteur active un observateur pour le composant Test_Unitaire, la mission de cet observateur et d'informer le moteur d'exécution sur les événements qui vont être émis par ce composant.

Après avoir expliqué d'une manière brève le fonctionnement du module d'exécution nous montrons quelques captures d'écran de ce module

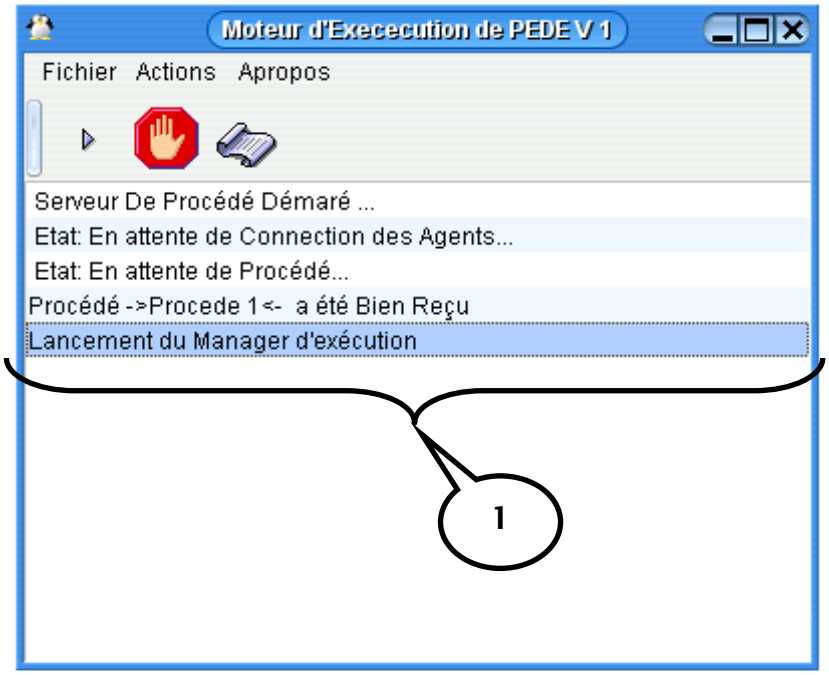


Figure 4.6 : *Vue générale du moteur d'exécution.*

1. Les trois premières lignes indiquent le bon démarrage des services du moteur : l'observateur des agents et l'observateur des procédés. La quatrième ligne indique qu'un procédé a été reçu par le moteur de procédés et que le moteur a lancé le manager d'exécution du procédé. Le Bouton nous affiche les détails du procédé reçu :

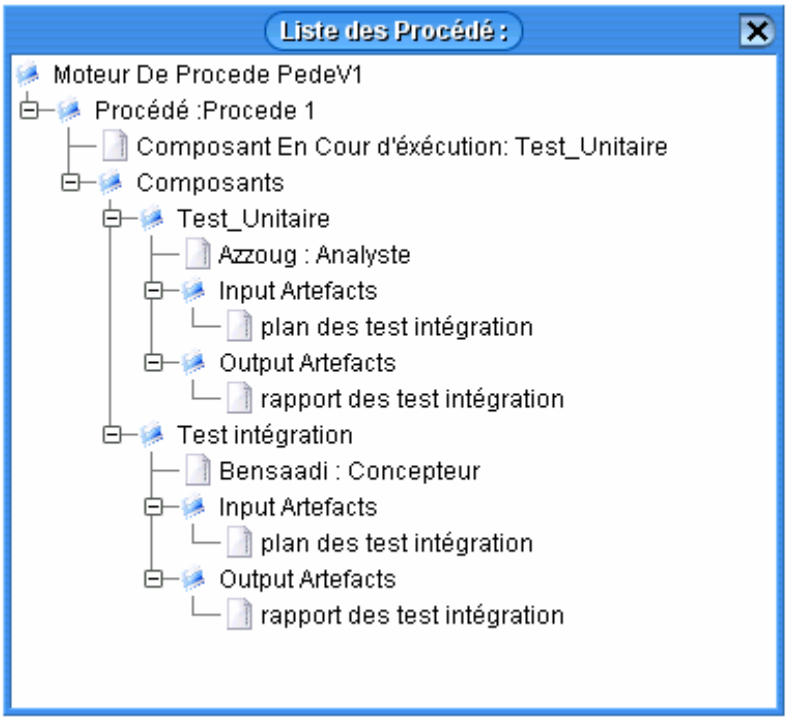


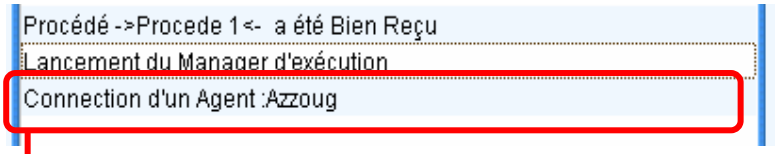
Figure 4.7 : *Détails des procédés reçus par le moteur d'exécution.*

3.2. Le Module client.

Le module client appelé aussi **ProcViewer** permet aux agents, de visualiser les composants procédés qui leur sont attribué, il leur permet aussi de recevoir et de déposer les Inputs et les outputs des artefacts. A titre d'exemple, la figure suivante illustre la connexion de l'agent AZZOUG au serveur de procédés.

Figure 4.8 : *le Module Client.*

La connexion des agents au moteur de procédés génère des événements `On_Agent_Connect` qui à leur tour activeront une série d'actions exécutées par le moteur.



L'observateur des agent "Agent_Litener" a généré l'événement `On_Agent_Connect` qui sera notifié au gestionnaire intéressé par cet évènement.

Figure 4.9 : *Connexion d'un agent.*

L'évènement `On_Agent_Connect` activera l'action de vérification de composant : `Has_Composant_For_Agent (Agent_Name)` qui retourne une valeur

- *Vrais* si un composant a été attribué à l'agent `Agent_name`, ou
- *False* dans le cas contraire.

L'agent va être informé de la réponse du serveur de procédés comme le montre les figures suivantes.

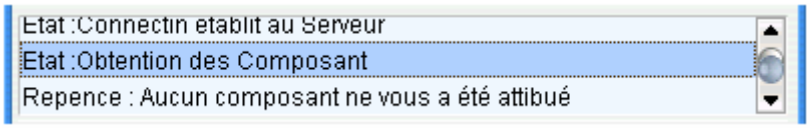


Figure 4.10 : *Réponse négative.*

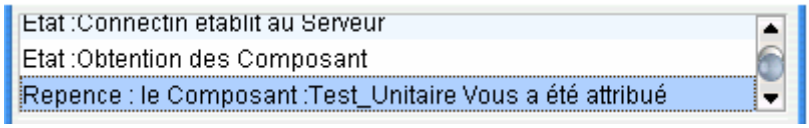


Figure 411: *Réponse positive.*

Chapitre -5-

Conclusion générale et perspectives

Ce travail s'inscrit dans le cadre d'une recherche sur le support automatisé aux procédés logiciels.

Une approche à base de composants est proposée pour distribuer, modéliser, réutiliser et exécuter des procédés. Les bénéfices attendus de cette recherche sont :

1. *La distribution* : Le modèle de procédé est géographiquement distribué sur des plateformes hétérogènes,
2. *La réutilisation* : Un composant procédé est une unité qui peut être décrite, déployée et manipulée indépendamment d'autres composants.
3. *Le support aux procédés dynamiques*: Un composant est défini avec le modèle et les informations d'instanciation permettant au composant de gérer sa propre évolution. Un composant possède des informations génériques (FrameWork), spécialisées (adaptation) et d'instanciation. Ainsi un composant peut s'adapter aux situations dynamiquement.
4. *L'exécution* : le modèle de procédés est exécuté en interprétant la représentation XML générer.

Le but de notre travail est une contribution à cette recherche. Nous proposons un FrameWork orienté objet pour le support aux procédés. Les principales contributions de notre travail sont :

1. La présentation des concepts et de la terminologie qui caractérise la technologie de procédés.
2. La proposition d'un FrameWork orienté objet pour le support de procédés. Les principales caractéristiques de notre FrameWork sont :
    a. Extensible pour couvrir les fonctionnalités d'une application particulière.
    b. Interopérabilité par événements. L'encapsulation du composant permet de l'utiliser sans avoir besoin de connaître son fonctionnement.
    c. Intégration dynamique de composants car chaque composant est indépendant.
    d. Un prototype qui présente l'interface graphique pour l'environnement supportant la modélisation par composants.

Les perspectives, qu'on peut dégagés de ce travail, peuvent se résumées par :

1. L'enrichissement du langage de définition de composants basé sur XML afin de permettre l'interopérabilité syntaxico-sémantique.
2. La construction de composants procédés. Prendre différents modèles existants, les encapsuler et leur fournir une interface simple pour leur utilisation.
3. La création d'une bibliothèque de composants réutilisables dans laquelle l'utilisateur sélectionnera des composants pour sa modélisation.
4. La mise en oeuvre du protocole d'interopérabilité par contrôle basé sur le standard **WF-XML**.

Références bibliographiques

[ALLOUI et al. 94] I. Alloui, S. Arbaoui, F. Oquendo "Process Centred Environments : Support for Human-Environment Interation and Environment-Mediated Human Cooperation", Proceedings of the 9th International Sofware Process Workshop ISPW9, Airlie, Virginie, IEEE Computer Society Press, septembre 1994.

[AMBRIOLA et al. 90] V. Ambriola, P. Ciancarini, C. Montangero, "Software Process Enactment in Oikos", Proceedings of the 4th ACM SIGSOFT Symposium on Software Development Environments, Irvine, Californie (EU), décembre 1990.

[AMBRIOLA et al.97] V. Ambriola, R. Conradi, A. Fuggetta, "Assessing Process-Centred Software Engineering Environments", ACM trsansactions on Software Engineering and Methodology, Vol. 6, No 3, juillet 1997, pages 283-328.

[AMIOUR 99] Amiour, Mahfoud, "Vers une fédération de composants interopérables pour les environnements centrés procédés logiciels", Thèse de Doctorat, Université Joseph Fourier, Grenoble I, juin 1999.

[AMIOUR et al. 97] M. Amiour, S. Dami, J. Estublier, "APEL V4: Formalism and Environment Specification", Technical Report, Adele Team, LSR Laboratory, Grenoble, France, juin 1997.

[AMIOUR et ESTUBLIER 98b] M. Amiour, J. Estublier, "PIE Interoperability Support: An Approach Based on a Federation of Heterogeneous and Interoperable Components", Deliverable IS-DDOC, PIE ESPRIT LTR IV Project No. 24840, mai 1998.

[ARBAOUI et al. 92] S. Arbaoui, S. Mouline, F. Oquendo, G. Tassart " PEACE : Describing and Managing Evolving Knowledge in the Software Process ", Proceedings of the Second European Workshop on Software Process Technology, Trondheim (Norvège), septembre 1992.

[AVRILIONIS 96] D. Avrilionis. “Mécanismes pour l’évolution et la réutilisation des processus de production de logiciels”. Thèse de doctorat de l’université Joseph Fourier-Grenoble 1. Octobre, 1996.

[BANDINELLI et al. 92] S. Bandinelli, A. Fuggetta, C. Ghezzi, S. Grigolli "Process Enactment in SPADE", Proceedings of the Second Europeean Workshop on Software Process Technology, Trondheim (Norvège), Springer Verlag, septembre 1992

[BARGHOUTI et KRISHNAMURTHY 93a] N.S. Barghouti, B. Krishnamurthy, "An Open Environment for Process Modeling and Enactment", In [Schäfer 1993], pages 33-36.

[BARGHOUTI et KRISHNAMURTHY 93b] N. S. Barghouti, B. Krishnamuthy, "Provence: A Process Visualization and Enactment Environment", In Proceedings of the 4th European Software Engineering Conference, Garmish-Partenkirchen, Allemagne, Springer-Verlag, pp.451-465, septembre 1993.

[BELKHATIR et MELO 92] N. Belkhatir, W.L. Melo, "Tempo: a software process model based on object context behavior", In the Proceedings of the 5th International Software Process Engineering and its Applications, pages 733-742, Toulouse, France, 7-11 décembre 1992.

[BEN-SHAUL et KAISER 93] I.Z. Ben-Shaul, G.E. Kaiser, "Process evolution in the Marvel environment", dans [Schäfer 93].

[BEN-SHAUL et KAISER 95] Ben – Shaul, I. and Kaiser, G. "An Interoperability Model for Process Centered Software Engineering Environments and its Implementation in OZ". Technical Report CUCS – 034 – 95, Computer Science Department, Culumbia University, 1995.

[BEN-SHAUL et KAISER 98] I.Z. Ben-Shaul, G.E. Kaiser, "Federating Process-Centred Environments: the Oz Experience", ASE journal, vol. 5, Issue 1, janvier 1998, Kluwer Academic Publishers.

[BOEHM 81] B.W. Boehm Software Engineering Economics, Prentice Hall ed., 1981

[BOLCER et TYLOR 96] Bolcer, G-A, Taylor, R-N, "Endeavors: A Process System Integration Infrastructure", In procedings of the IEEE, August 1996.

[BRUYNOOGHE et al. 94] R.F. Brynooghe, R.M. Greenwood, I. Robertson, B.C. Warboys, "PADM: Towards a Total Process Modelling System", dans [Promoter 94], 1994.

[CANALS et al. 94] G. Canals, N. Boudjlida, J.-C; Derniame, C. Godart, J. Longchamp, "ALF: A Framework for Building Process-Centred Software Engineering Environments", pages 153-185. Dans [Promoter 94], 1994.

[CHAPPELL 96b] D. Chappell, "Understanding ActiveX and OLE", Microsoft Press Redmond, 1996.

[CIMPAN 00] S. Cîmpan, "Omega : Un formalisme et un système pour le monitoring des processus dans le cadre des environnements de génie logiciel", thèse de doctorat, Université de Savoie, janvier 2000.

[CÎMPAN et OQUENDO 98] S. Cîmpan, F. Oquendo "Fuzzy indicators for monitoring software processes", Proceedings of the 6th European Workshop on Software Process Technology, EWSPT'98, Springer Verlag, Weybridge (Angleterre), septembre 1998.

[COAD et YOURDON 91a] P. Coad, E. Yourdon, "Object Oriented Analysis", Second Edition, Yourdon Press, Prentice Hall, 1991.

[COAD et YOURDON 91b] P. Coad, E. Yourdon, "Object Oriented Design", Yourdon Press, Prentice Hall, 1991.

[CONRADI et al. 92] R. Conradi, C. Fernstrom, A. Fuggetta and R. Snowdon. "Towards a Reference Framework for Process Concepts". Second European Workshop on Software Process Technology, EWSPT'92, Lecture Notes in Computer Science 635, J. C. Derniame Ed. 1992.

[CONRADI et al. 94] R. Conradi, M. Hagaseth, J.O. Larsen, M.N. Nguyen, B.P. Munch, P.H. Westby, W. Zhu, M.L. Jaccheri, C. Liu, "EPOS: Object-Oriented Cooperative Process modelling", dans [Promoter 94]

[DERNIAME et al. 94] J.C. Derniame et al., "Life Cycle Process Support in PCIS", In Proceedings of the PCTE'94 Conference, 1994.

[DIENG et al. 00] R. Dieng, O. Corby, A Giboin, J. Golebiowska, N. Matta, M. Ribière, "Méthodes et outils pour la gestion des connaissances", Dunod, Paris, 2000.

[ESTUBLIER et al. 97b] J. Estublier, S. Dami, M. Amiour, "High Level Modeling for SCM", 7th International Workshop on Software Configuration Management (SCM7), Boston (EU), mai 1997.

[ESTUBLIER et al. 98a] J. Estublier, S. Dami, M. Amiour "APEL: a Graphical Yet Executable Formalism for process Modelling", ASE journal (Automated Software Engineering), vol. 5, Issue 1,1998 Kluwer Academi Publishers.

[ESTUBLIER et al. 99] J. Estublier, M. Amiour, S. Dami, "Building a Federation of Process Support Systems", WACC'99, San-Francisco, CA, février 1999.

[ESTUBLIER et CASALAS 94] J. Estublier and R. Casallas."The Adele Software Configuration Manager". Chapter 4, pages 99–139. Trends in Software. J. Wiley and Sons, Baffins Lane, Chichester West Sussex, PO19 1UD, England, 1994.

[ESTUBLIER et VERJUS 99] J. Estublier, H. Verjus, "Definition of the behaviour paradigms of a heterogeneous federation of evolving process components", PIE LTR ESPRIT Project 34840, Deliverable D2.01, juin 1999.

[FELDMAN 79] S. I. Feldman, "Make – A program for maintaining computer programs", Software Practice and Experience, 1979.

[FINKELSTEIN et al. 94] B. N. A. Finkelstein, J. Kramer, editor. "Software process modelling and technology", volume 1. John Willey and Son Inc. Research Study Press, Tauton Somerset, England, 1994.

[FUGGETTA et Al. 93] A. Fuggetta, "A classification of CASE technology", IEEE Computer, 26(12):25-38, décembre 1993.

[GAMMA et al. 99] Gamma E., Helm R., Johnson R., Vlissides J., "Design Patterns : Catalogue de modèles de conception réutilisables", Traduction de Jean-Marie Lasvergères, Ed. Vuibert Informatique, 1999, 459 pages.

[GOLDBERG 84] A. Goldberg, "Smaltalk-80 : The Interactive Programming Environment", Reading MA, Addison Weisley Publishing Company, 1984.

[GRAW et GRUHN 95] G. Graw, V. Gruhn, "Process Management in-the-many", 4th European Workshop, EWSPT'95, avril 1995.

[GRUBER 93] T. Gruber, "A translation approach to portable ontology specifications", Knowledge Acquisition, 5(2):199-220, 1993.

[INOUE et al. 89] K. Inoue, T. Ogihara, T. Kikuno, K. Torii, "A Formal Adaptation Method for Process Description", In Proceedings of the 11th International Conference on Software Engineering, 1989.

[JUNKERMANN et al. 94] G. Junkermann, B. Peuchel, W. Schaefer, S. Wolf, "Merlin: Supporting Cooperation in Software Development Through a Knowledge-Based Environment", dans [Promoter 1994], 1994.

[KUVAJA 94] P. Kuvaja Software Process assesment & improvement : The bootstrap approach. Blackwell, Oxford, Angleterre, 1994.

[LATROUS et OQUENDO 95] S. Latrous, F. Oquendo " PEACE+/PECAM : une machine virtuelle pour l'exécution et l'évolution des processus logiciels ", Proceedings of the 8th International Conference on Software Engineering and its Applications GL'95, Paris, France, novembre 1995.

[LONGCHAMP 93] J. Longchamp, "A structured conceptual and terminological framework for software process engineering", In Proceedings of the 2nd International Conference on the Software Process, pages 41-53, IEEE Computer Society Press, février 1993.

[MOORE 88] R.E. Moore " Autoepistemic logic ", Non-Standard Logics for Automated Reasoning, P. Smets, A. Mamdani, D. Dubois et H. Prade (Eds.), Academic Press, Londre, 1988.

[MULLER 96] P.A. Muller, "Modélisation Objet avec UML", Editions Eyrolles, 1996.

[OQUENDO 95] F. Oquendo, " SCALE: Process Modelling Formalism and Environment Framework for Goal-directed Cooperative Processes ", Proceedings of the Seventh International Conference on Software Engineering Environments (SEE'95), Noordwijkerhout, Pays-Bas, avril 1995. IEEE Computer Society Press.

[OSTERWEIL 87] L. Osterweil "Software Processes are Software Too", Proceedings of the 9th International Conference on Software Engineering, Monterey, mars 1987.

[PAULK et al. 91] M.C. Paulk, B. Curtis, M.B. Chrissis et coll. Capability Maturity Model for Software. Software Engineering Institute, CMU/SEI-91-TR-24, ADA240603, août 1991.

[PROMOTER 99] Software Process : Principles, Methodology, Technology. J.C. Derniame, A.B. Kaba, D. Wastell (Eds), Springer-Verlag 1999, LNCS 1500, ISBN 3-540-65516-6199.

[ROUT 95] T. Rout : "SPICE : A Framework for Software Process Assesment". Software Process : Improvement and Practice, 1(1), 1995.

[ROYCE 70] W. Royce, "Managing the development of large software systems", In Proceedings of WESTON, San Francisco, 1970.

[RUMBAUGH et al. 91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, "Object-Oriented Modeling and Design", Prentice Hall, 1991.

[SOMMERVILLE 89] I.Sommerville, "Software Engineering (3rd edition)", Addison-Wesley, Wokingham, 1989.

[SUTTON et al. 97] S.M. Sutton, B.S. Lerner, L.J. Osterweil, "Experience Using the JIL Process Programming Language to Specify Design Processes", Technical Report 97-68, Department of Computer Science, University of Massachussetts at Amherst, 6 septembre, 1997.

[SUZUKI et KATAYAMA 91] M. Suzuki, T. Katayama, "Meta operations in the process model HFSP for the dynamics and flexibility of software processes", In. Proceedings of the 1st International Conference on the Software Process, IEEE Computer Society, 1991.

[TAYLOR et al. 88] R.N. Taylor, F.C. Belz, L.A. Clarke, L. Osterweil, R.W. Selby, J.C. Wileden, A.L. Wolf, M. Young "Foundation of the Arcadia Environment", Proceesings of ACM SIGSOFT '88 : Third Symposium on Software Development Environments, pp. 1-13, Boston, novembre 1988.

[TEITLEMAN et MASINTER 84] W. Teitleman, L. Masinter, "The Interlisp programming environment", In Interactive Programming Environments. D.R. Barstow, H.E. Shrobe, E. Sandewall ed. New York: McGraw-Hill, 1984.

[TIAKO 98] P.F. Tiako, "Modelling the Federation of Process Sensitive Engineering Environments: Basic Concepts and Perspectives", In Proceedings of the 6th European Workshop on the Software Process Technology, EWSPT'98, Weybridge, Angleterre, septembre 1998.

[TIAKO 99] P.F. Tiako, "Modélisation de la Fédération des Environnements Sensibles aux Procédés de Développement de Logiciels", thèse de doctorat, Institut National Polytechnique de Lorraine, juin 1999.

[USCHOLD et GRUNINGER 96] M. Unschold, M. Gruninger, "Ontologies: principles methods and applications knowledge", Engineering Review, Vol. 11, No 2, juin 1996.

[VERJUS et OQUENDO 98] H. Verjus, F. Oquendo, "Fédérations d'environnements centrés processus logiciels : une approche basée composants", Proceedings of the 11th International Conference on Software Engineering and its Applications (GL'98), Paris, décembre 1998.

[WFMC 98] WfMC. "Interface 1: Process Definition Interchange Process model", Workflow Management Coalition, Document N° WfMC TC-1016-P (Official release), November 12, 1998.

Une approche de modélisation de procédés logiciels à base de composants

Présenté par
Mr BELKASMI Djamel

Dirigé par
Mr. AHMED NACER Mohamed (Professeur U.S.T.H.B)

Résumé

Le domaine des procédés logiciels est vaste et les procédés logiciels sont intrinsèquement complexes. De nombreux travaux de recherches poursuivent leurs efforts pour caractériser et pour mieux cerner les procédés. Des approches ont été proposées pour les modéliser et des environnements ont été développés pour les supporter.

Dans ce travail, Nous proposons une approche de modélisation de procédés logiciels à base de composants permettant à des équipes distribuées de coopérer dans la modélisation et l'exécution des procédés en utilisant des environnements hétérogènes.

Un FrameWork basé sur le langage XML a été développé pour supporter l'approche proposée.

Mots clés

Procédé logiciel, modèle de procédé logiciel, environnement centré procédé logiciel, composant procédé.
