Modélisation UML avec Rational Rose 2000

Terry Quatrani

Éditions Eyrolles

ISBN : 2-212-09173-7

2000

11

Considérations d'architecture

Dans ce chapitre

- De la nécessité d'une architecture
- L'équipe d'architectes
- Les « 4+1 » vues de l'architecture
- La vue logique
- La vue d'implémentation ou vue des composants
- La vue des processus
- La vue de déploiement
- La vue des cas d'utilisation
- Synthèse

Nécessité d'une architecture

Au fil des années, bien des définitions, parfois complexes, ont été données de l'architecture. Tantôt l'on entend par architecture le résultat du travail des architectes, tantôt l'on ironise en assimilant architecture et politique. Donner une définition de ce qu'est l'architecture d'un système est chose difficile. C'est un ensemble d'artefacts qui orientent les décisions stratégiques rela-

tives tant à la structure et au comportement d'un système qu'à la façon de faire collaborer entre eux ses éléments et encore qu'à son déploiement physique.

Le succès d'un projet orienté objet est tout à fait dépendant de l'existence d'une base architecturale solide. L'architecture d'un système est souvent négligée par manque de temps, parce que l'on se soucie surtout d'accélérer la mise sur le marché de l'application et aussi parce que l'on doute de sa valeur ajoutée. Dans tous les cas, cela conduit à anticiper la phase de codage, avec des conséquences désastreuses pour le projet tout entier : sans une architecture soignée, le logiciel qui doit être produit le sera dans du code inadéquat [1].

Le développement de l'architecture est un problème délicat, car on doit y procéder par itérations pendant la phase d'élaboration. L'architecture d'un système ne peut être créée du jour au lendemain. Elle requiert, entre autres, l'exploration des cas d'utilisation, un prototype de faisabilité et la prise en compte des exigences architecturales minimales au cours des phases d'étude d'opportunité et d'élaboration [2]. Pour tester les choix de conception, on construit des prototypes exécutables de l'architecture. Ce point est essentiel car l'équipe de développement est ainsi contrainte de confronter ses hypothèses de conception à la dure réalité [3].

L'équipe d'architectes

Tout projet devrait inclure un architecte en chef épaulé par une petite équipe. Le rôle de l'architecte couvre un large domaine. Il doit définir l'architecture du logiciel, veiller à ce que le logiciel s'y conforme, évaluer les risques techniques du projet, définir la portée et la planification des itérations successives. Il doit aussi délivrer des conseils aux équipes de conception, d'implémentation, d'intégration et d'assurance qualité et, pour finir, on attend de lui qu'il contribue à établir les tendances futures du marché [4].

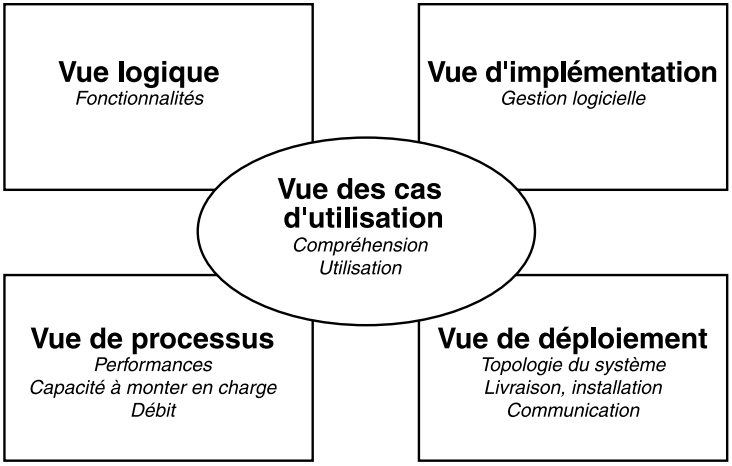
---

1. Booch, Grady. *Object Solutions.* Redwood City, CA : Addison-Wesley, 1995.
2. Jacbson, Ivar. *The Objector Software Development Process.* Édition provisoire.
3. *Ibid.*
4. Kruchten, Philippe. *Software Architecture and Iterative Development.* Santa Clara, CA : Rational Software Corporation. April 1994,p. 53.

Les « 4+1 » vues de l'architecture

L'architecture d'un logiciel ne comporte pas qu'une dimension ; elle est au contraire constituée de plusieurs vues concurrentes [1] (voir Figure 11-1). Dans la suite de ce chapitre, seront abordés les différents éléments rencontrés dans chacune de ces vues et les notations UML employées pour représenter les choix architecturaux du système.

Figure 11-1
Les « 4+1 » vues de l'architecture.



La vue logique

Les besoins fonctionnels du système, autrement dit les services qu'il doit apporter aux utilisateurs, sont traités dans la vue logique de l'architecture. Celle-ci est constituée de diagrammes de classes qui contiennent les abstractions clefs du système, représentées sous la forme de classes et de relations. Les notations UML utilisées sont les classes, les associations, les relations d'agrégation, les relations de généralisation et les paquetages.

La vue logique est traitée au début de la phase d'élaboration ; elle produit les classes et les paquetages qui constituent les abstractions principales du domaine. Au fur et à mesure, des classes et des paquetages supplémentaires sont ajoutés au modèle pour prendre en compte les choix relatifs aux méca-

1. Ibid.

nismes clefs du système, qui concernent l'usage des règles et des standards habituels. Cette activité qui détermine les mécanismes clefs d'un système est habituellement nommée conception tactique. Une conception tactique de mauvaise qualité peut affecter la meilleure architecture, aussi l'équipe d'architectes doit-elle atténuer ces risques en identifiant très clairement les règles clefs du projet qu'il convient de suivre[1]. Parmi les mécanismes clefs habituels, citons les choix portant sur le langage d'implémentation, le support de la persistance des données, l'apparence de l'interface utilisateur, les mécanismes de gestion des erreurs, ceux de communication, de migration et de distribution des objets, et enfin le choix des réseaux utilisés.

Pour implémenter les choix relatifs aux mécanismes clefs du système, de nombreux canevas sont disponibles et il est fortement conseillé de les consulter avant d'en créer de toutes pièces.

Ces choix doivent refléter les concepts de cohésion, de fermeture et de réutilisation. Robert Martin, dans son livre *La Méthode Booch pour la conception orientée objet d'applications C++*, aborde l'implication des choix concernant les paquetages que l'on peut utiliser pour développer un système. Ce livre exploite les notations et le processus Booch, mais il peut très bien être utilisé dans le cadre du Processus unifié de Rational et de UML. Pour résumer, UML est tout à fait adapté à l'expression des décisions stratégiques prises dans le cadre d'un projet. L'ajout de paquetages et de classes à un modèle contribue à expliciter, implémenter, justifier et commenter ces décisions.

Quelques mécanismes clefs utilisés pour l'étude de cas

Le langage C++ a été retenu pour implémenter le système d'inscription. Ce choix se justifie par l'expérience que l'équipe de développement possède de ce langage et aussi par le fait que le système d'inscription devra intégrer des fonctionnalités couvrant d'autres domaines au sein de l'université. Pour gérer l'apparence des interfaces utilisateur, l'équipe d'architectes a pris la décision d'utiliser un paquetage dénommé contrôles GUI (contrôles d'interfaces graphiques). Afin de prendre en compte les problèmes de permanence des données, il a été décidé d'associer une classe dans la base de données à chacune des classes permanentes du système. D'autres approches sont possibles pour traiter cet

1. Booch, Grady. *Object Solutions, op. cit.*

aspect, par exemple l'utilisation de l'héritage. Cependant, cette approche simple a été adoptée afin de profiter de l'expertise des architectes dans ce domaine, et aussi pour limiter les risques. Les mécanismes de gestion des exceptions de C++, *try* et *catch*, seront employés conjointement avec un paquetage GestionDesErreurs, afin que chaque classe n'ait pas à connaître les détails du traitement et de la levée des exceptions. Pour finir, un ensemble de classes de base disponible sur le marché sera utilisé. Les paquetages qui concrétisent les choix de ces mécanismes clefs pour l'étude de cas sont présentés Figure 11-2.

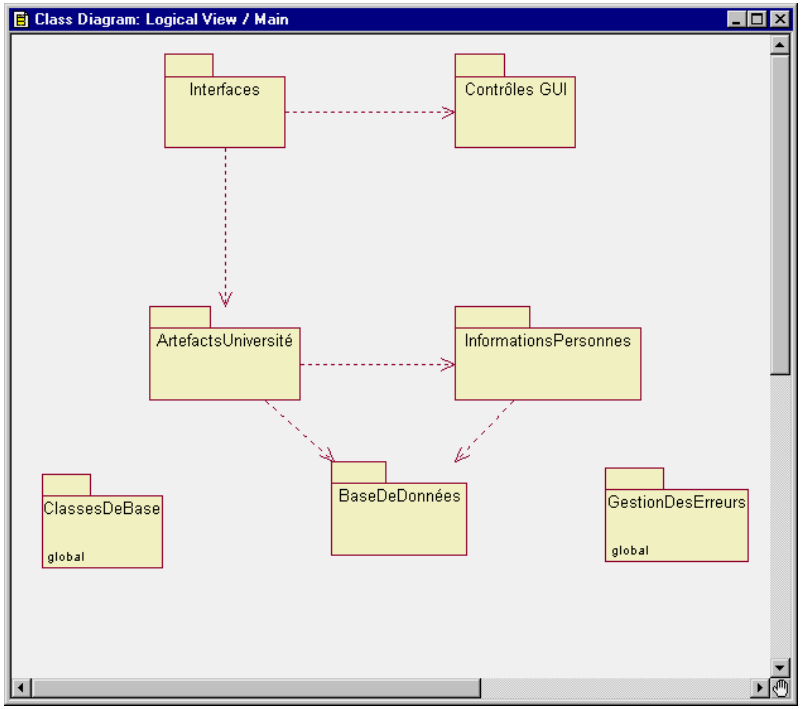


Figure 11-2
Système d'inscription de l'ESU.

Les paquetages GestionDesErreurs et ClassesDeBase possèdent la valeur étiquetée *global* car ils sont utilisés par tous les paquetages du système.

Comment rendre un paquetage *global avec Rational Rose*

1. Cliquer avec le bouton droit de la souris pour sélectionner un paquetage sur un diagramme de classes.
2. Sélectionner l'option de menu Open Specification.
3. Sélectionner l'onglet Detail.
4. Cocher la case global.
5. Cliquer sur le bouton OK pour fermer la fenêtre de spécification.

La vue d'implémentation ou vue des composants

Cette vue de l'architecture décrit l'organisation des composants logiciels au sein de l'environnement de développement. Elle prend en compte des besoins auxiliaires comme la facilité d'utilisation, la gestion du logiciel, la réutilisation, les contraintes imposées par le langage de programmation et les outils de développement. Les paquetages, les composants et les relations qui les lient constituent les éléments de modélisation employés pour décrire la vue des composants de l'architecture.

Dans ce contexte, un paquetage correspond à un constituant physique du système. Les paquetages sont organisés hiérarchiquement en couches et chacune d'entre elles possède une interface clairement définie. Il n'est pas surprenant qu'un système orienté objet soit structuré en couches. Cela découle de la définition même d'un objet, qui stipule qu'un objet doit représenter une seule abstraction. Un exemple classique de système structuré en couches est présenté Figure 11-3.

Tout comme il en va pour un paquetage de la vue logique, un paquetage de la vue des composants est représenté en UML par un dossier (voir Figure 11-4).

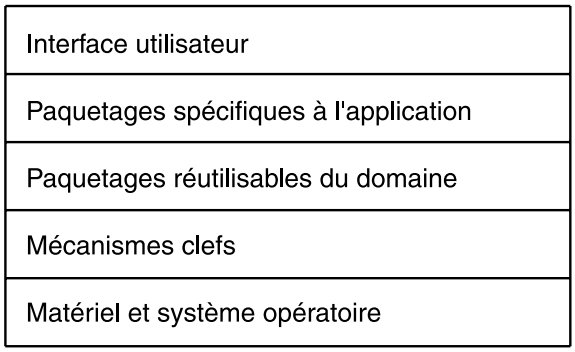


Figure 11-3
Système structuré en couches.

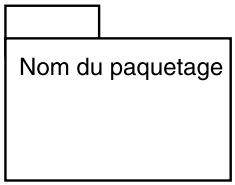


Figure 11-4
Notation UML pour un paquetage.

Création de paquetages pour la vue des composants avec Rational Rose

1. Dans le navigateur, cliquer avec le bouton droit de la souris sur le paquetage Component View pour faire apparaître le menu contextuel.
2. Sélectionner l'option de menu New :Package, ce qui aura pour effet de créer un nouveau paquetage dénommé NewPackage dans le navigateur.
3. Le paquetage étant sélectionné, saisir son nom.

Le diagramme principal de composants est habituellement une vue qui présente l'ensemble des paquetages définis pour le système.

Création du diagramme principal de composants avec Rational Rose

1. Dans le navigateur, double-cliquer sur le diagramme Main Diagram du paquetage Component View.
2. Sélectionner un paquetage et le faire glisser sur le diagramme.
3. Répéter l'étape 2 pour chaque paquetage supplémentaire requis.

4. Pour ajouter des relations de dépendance, il faut sélectionner l'icône Dependency sur la barre d'outils, cliquer sur le paquetage client et étirer la flèche vers le paquetage fournisseur.

Le diagramme de composants principal de l'étude de cas apparaît Figure 11-5.

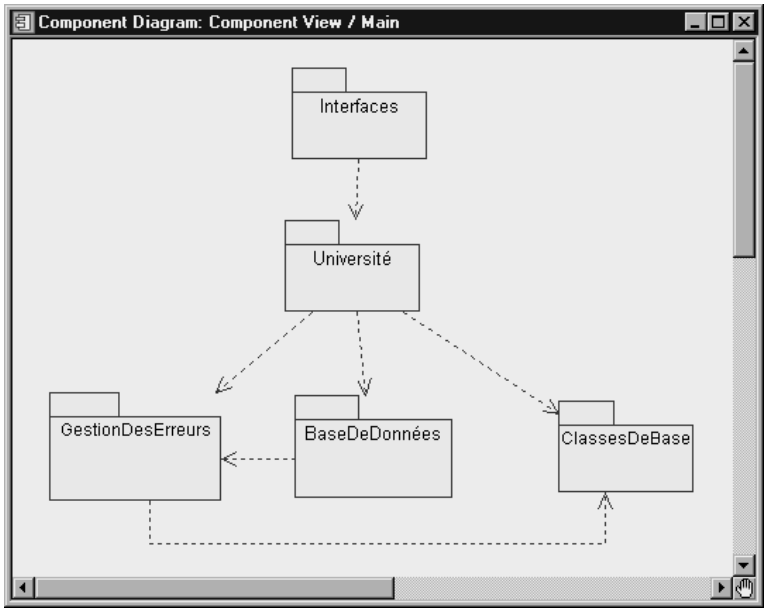


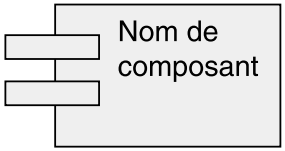
Figure 11-5
Diagramme principal de composants.

Composants de code source

Un fichier logiciel contenu dans un paquetage est représenté par un composant de code source dans la vue des composants du modèle. Le type du fichier dépend du langage de programmation. En C++, les composants correspondent à des fichiers avec une extention `.h` ou `.cpp`, en Java, ce sont des fichiers suffixés par `.java` et en Powerbuilder un composant correspond à un fichier suffixé par `.pbl`. Un langage de programmation est défini pour chaque composant, ce point est traité dans les annexes du livre. Les classes de la vue logique sont assignées aux composants de la vue des composants. En C++, cette mise en relation est habituellement une bijection, c'est-à-dire qu'une classe correspond à un composant. Il est cependant possible d'assigner plusieurs classes à un composant, notamment lorsqu'il existe un fort couplage entre celles-ci. Par exemple, une classe conteneur et son itérateur seront inclus

dans un fichier `.h` et un fichier `.cpp`, c'est-à-dire dans un composant. Un autre exemple concerne le regroupement dans un même composant physique des classes qui constituent un canevas de collaboration. Un exemple de notation UML est présenté Figure 11-6.

Figure 11-6
Notation UML pour un composant.



Composants logiciels de l'étude de cas

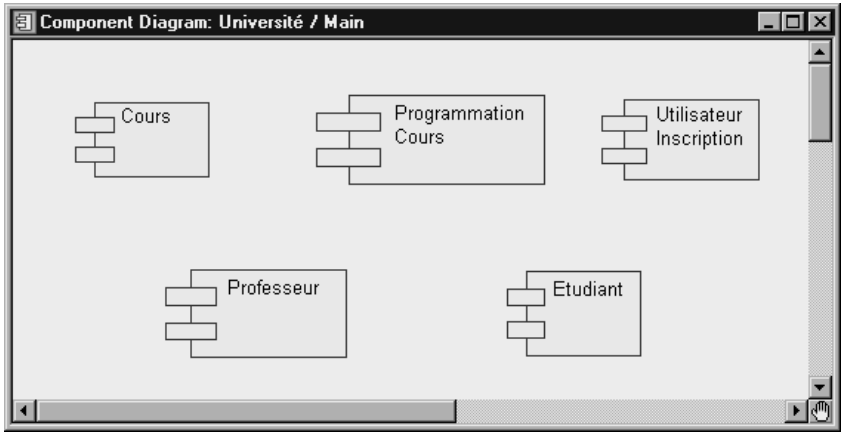
Ce système est relativement simple, chaque classe, décrite par un fichier `.h` et un fichier `.cpp`, est associée à un composant.

Création de composants avec Rational Rose

1. Ouvrir le diagramme de composants.
2. Sélectionner l'icône Component sur la barre d'outils.
3. Cliquer sur le diagramme pour y placer le composant. Cela aura pour effet d'ajouter le composant dans le navigateur.
4. Le composant étant sélectionné, saisir son nom.

Un exemple de diagramme de composants est présenté Figure 11-7.

Figure 11-7
Composants logiciels.



Assignation des classes aux composants avec Rational Rose

1. Dans le navigateur, sélectionner le composant avec le bouton droit de la souris pour faire apparaître le menu contextuel.
2. Sélectionner l'option de menu Open Specification.
3. Sélectionner l'onglet Realizes.
4. Sélectionner la classe avec le bouton droit de la souris pour faire apparaître le menu contextuel.
5. Sélectionner l'option de menu Assign.
6. Cliquer sur le bouton OK pour fermer la fenêtre de spécification.

REMARQUE Une classe peut aussi être assignée à un composant en la sélectionnant dans le navigateur et en la faisant glisser sur le composant (dans le navigateur ou sur le diagramme de composants).

La spécification du composant ProgrammationCours apparaît Figure 11-8.

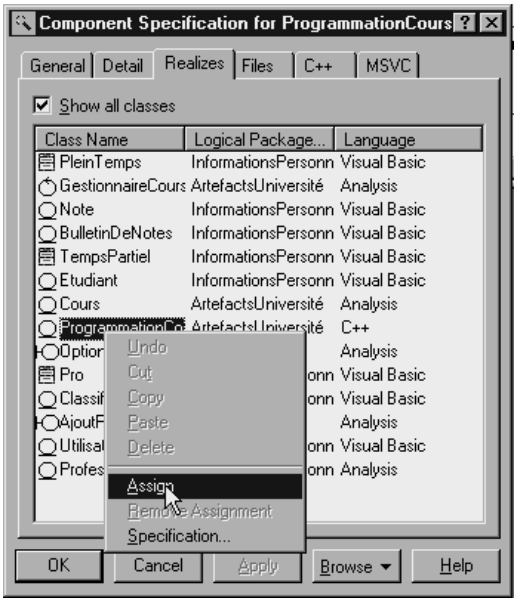
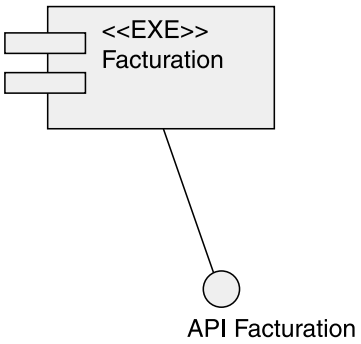


Figure 11-8
Spécification du composant ProgrammationCours.

La vue des processus

Dans cette vue de l'architecture, l'accent est mis sur la structure de l'implémentation du système lors de l'exécution. La vue des processus prend en compte les besoins non fonctionnels comme la performance, la fiabilité, la capacité à monter en charge, l'intégrité, la gestion du système et les synchronisations. On utilise aussi les composants dans cette vue pour visualiser l'environnement d'exécution et les composants exécutables du système. Les composants sont reliés par des relations de dépendance. Les composants de l'environnement d'exécution représentent la façon dont les classes sont assignées à des bibliothèques telles que des applets Java, des composants ActiveX et des bibliothèques de liens dynamiques (DLL). Les composants exécutables sont accompagnés de leurs interfaces et de leurs dépendances d'appel vers d'autres exécutables. On peut utiliser des stéréotypes pour faire des catégories de composants. Par exemple, les interfaces (classes du stéréotype interface) créées pendant l'étape de conception sont représentées en utilisant une notation en forme de sucette (voir Figure 11-9).

Figure 11-9
Interface d'un composant.

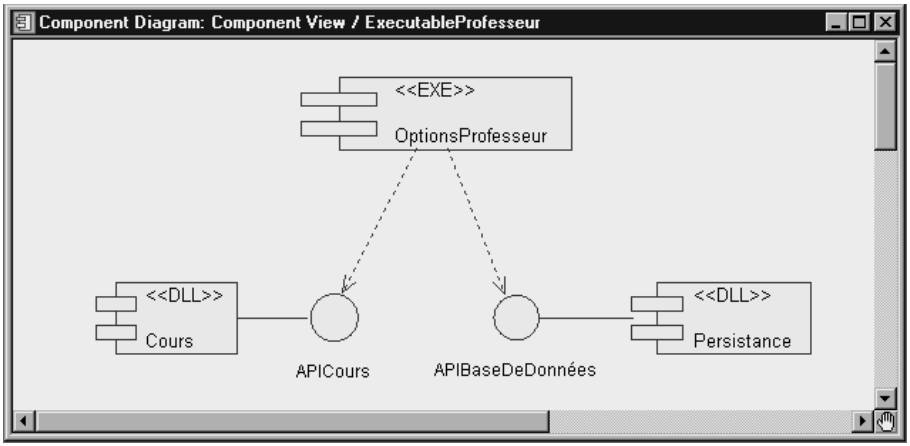


Les informations relatives à la vue architecturale des processus sont représentées sur des diagrammes de la vue Component View de Rational Rose. Ces diagrammes contiennent des composants exécutables et des composants de l'environnement d'exécution, éventuellement liés par des relations de dépendance.

Pour l'étude de cas, les architectes ont fait le choix de créer deux bibliothèques de liens dynamiques, l'une incluant les informations relatives aux cours et à leur programmation, l'autre incluant les informations relatives aux bases

de données. Ce choix a été motivé par le fait que la structure des cours, ainsi que les décisions stratégiques pour les bases de données, seront amenées à évoluer. En effet, pour implémenter les changements, il suffira de remplacer les bibliothèques qui auront été modifiées. Trois exécutables composent l'application. Le premier permet à la scolarité de créer et de gérer le système, les deuxième et troisième en fournissent respectivement l'accès aux étudiants et aux professeurs. Le diagramme de composants de l'exécutable utilisé par les professeurs (OptionsProfesseur.exe) est représenté Figure 11-10.

Figure 11-10
L'exécutable Professeur.



La vue de déploiement

L'objet de cette vue est l'assignation des parties exécutables du logiciel aux nœuds physiques que sont les machines et les processeurs. Elle révèle la topologie constituée par les éléments physiques de calcul et la configuration des processus logiciels qui leur sont affectés. Elle prend en compte les besoins de disponibilité du système, sa fiabilité, ses performances et sa capacité à monter en charge. Des diagrammes de déploiement sont utilisés pour représenter les nœuds et leurs connexions. Ce sont des représentations visuelles de la répartition des composants au sein de l'entreprise, dans lesquelles les éléments de calcul ne sont autres que les nœuds d'un graphe et où les arcs correspondent aux chemins de communications entre ces éléments de calcul. Les processus logiciels apparaissent sous forme textuelle et décorent un nœud ou un groupe de nœuds.

Ces diagrammes permettent aux architectes de comprendre la topologie du système et d'assigner les composants à des processus exécutables. Ils prennent en compte l'architecture des processeurs, leur vitesse, leur puissance de calcul, leur bande passante pour les communications inter-processus, la localisation physique du matériel et les mécanismes de calcul réparti.

Diagrammes de déploiement pour l'étude de cas

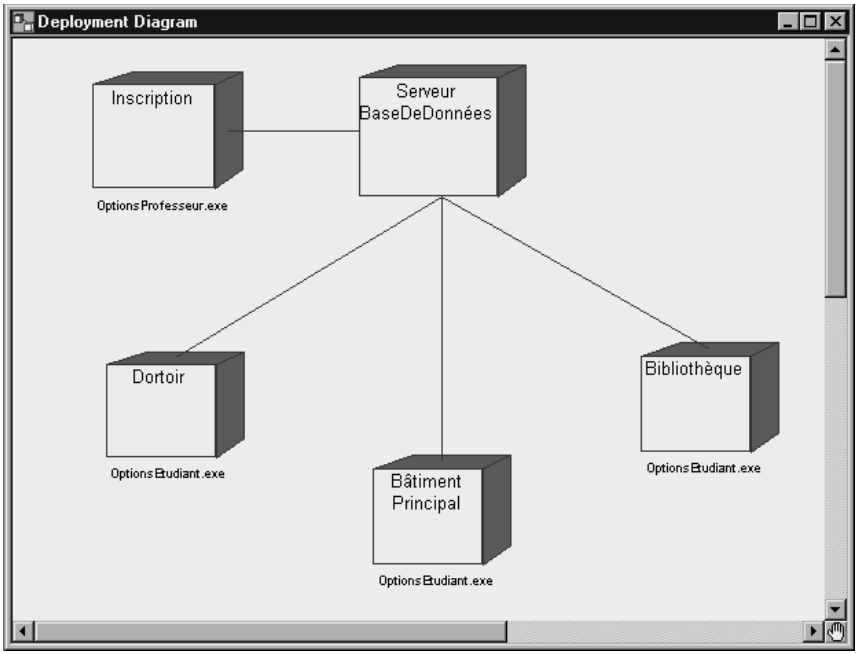
Après une étude des paquetages de composants du système, un examen du matériel existant et une évaluation de la charge supportée par le système en période d'inscription, les architectes ont estimé que cinq processeurs seraient nécessaires. Un premier accueillera l'exécutable utile aux professeurs, un autre tiendra lieu de serveur pour les bases de données et les trois derniers seront destinés à l'inscription des étudiants.

Création de diagrammes de déploiement avec Rational Rose

1. Le diagramme de déploiement est créé automatiquement par l'outil Rational Rose. Pour l'ouvrir, double-cliquer dans le navigateur sur Deployment Diagram.
2. Pour créer un nœud, sélectionner l'icône Processor et cliquer sur le diagramme pour le placer.
3. Le nœud étant sélectionné, saisir son nom.
4. Pour créer une connexion entre deux nœuds, sélectionner l'icône Connection sur la barre d'outils, cliquer sur le nœud origine et étirer la connexion vers le nœud cible.

Le diagramme de déploiement de l'étude de cas apparaît Figure 11-11.

Figure 11-11
Diagramme de déploiement.



La vue des cas d'utilisation

Cette dernière vue de l'architecture valide la vue logique, la vue des composants, la vue des processus et la vue de déploiement. Des diagrammes de séquence et des diagrammes de collaboration complètent la vue des cas d'utilisation pour illustrer la façon dont les différents éléments de la conception interagissent afin que le comportement escompté du système soit concrétisé.

Synthèse

L'architecture d'un logiciel ne comporte pas qu'une seule dimension mais est constituée de plusieurs vues concurrentes, respectivement, la vue logique, la vue d'implémentation, la vue des processus (ou vue des composants) et la vue de déploiement. Des scénarios viennent compléter la vue des cas d'utilisation et valider les quatre autres vues. Les architectures doivent être structurées en couches d'abstractions bien définies, dans lesquelles l'interface et

l'implémentation sont clairement séparées. Les mécanismes clefs concrétisent les décisions qui concernent les standards et les règles habituelles. L'utilisation de paquetages met en évidence l'arrangement architectural du système. Les diagrammes de composants décrivent les éléments qui constituent l'implémentation physique de l'application. Les diagrammes de déploiement représentent quant à eux la configuration matérielle sur laquelle sera exécuté le système.