2.3.1.2 Modélisation du comportement du système

Avant d'introduire des diagrammes, **UML** spécifie en premier lieu des concepts permettant d'exprimer des caractéristiques dynamiques. On fait la distinction entre les actions, les activités, les états, les interactions et les cas d'utilisation. Nous allons passer brièvement en revue leur utilisation pendant le processus de modélisation et indiquer les liaisons effectuées avec le modèle structurel pour maintenir la cohérence globale du modèle.

L'objectif ici est de pouvoir détailler le comportement de tous les éléments décrits de façon statique dans la structure du système et de montrer comment ils interagissent pour réaliser les fonctions attendues de l'ensemble.

Les actions. Comme précisé dans le standard, une action représente l'unité élémentaire des spécifications comportementales d'un système. Elle possède un ensemble d'entrées et active ou envoie un certain nombre de sorties en fonction du traitement qu'elle effectue.

Tout le travail de description du comportement des éléments structurels va donc consister à organiser ces actions au sein de diagrammes de comportement de façon à produire le comportement dynamique souhaité pour l'élément étudié.

Les activités. Un peu à l'image des structures composites, une activité contient un ensemble d'actions reliées entre elles qui permettent de réaliser le comportement global de l'activité, sa réaction aux sollicitations extérieures, l'envoi ou le stockage d'informations. Les activités sont décrites dans des ***diagrammes d'activité***. Lorsqu'une; activité est exécutée, les actions internes de l'activité vont être exécutées une ou plusieurs fois selon la façon dont elles sont connectées entre elles et avec les entrées et les sorties de l'activité parente.

Les partitions. Elles permettent d'intégrer un niveau de description supérieur en regroupant les actions ou activités qui ont des caractéristiques communes par exemple si elles sont réalisées dans un même contexte particulier. Les partitions sont très pratiques, par exemple pour modéliser un flux d'information entre différents services ou sites géographiques d'une société. On partitionne alors selon les services et on intègre les actions de chacun en définissant les flux de contrôle et les flux d'objets correspondant aux échanges entre les différents services (par exemple envoi de mails dans le cas d'un flux d'objets, ou ordre simple dans un flux de contrôle).

Les interactions. Alors que les activités et les actions vont souvent être utilisées pour décrire le comportement interne d'un élément structurel, les interactions ont pour vocation de permettre à l'utilisateur de représenter les échanges et les sollicitations qui vont être effectués entre ces éléments structurels. De plus, les diagrammes proposés pour spécifier ces interactions vont permettre d'introduire des contraintes de séquencement entre ces interactions pour que la description du comportement des échanges entre les objets soit précisément modélisée.

Il existe cinq diagrammes permettant la modélisation des interactions d'un système, le diagramme de séquence (sequence diagram), le diagramme de vue générale des interactions (interactions overview diagram), le diagramme de communication (communication diagram), le diagramme temporel (timing diagram) et le diagramme de tables d'interaction (interaction tables). Il n'est pas obligatoire de tous les utiliser dans une même spécification; le choix sera à effectuer en fonction des attentes de l'utilisateur au niveau de la forme de représentation et du type de comportement dynamique qu'il veut décrire [Mul 00].

Les machines à états. Le diagramme de séquence s'appuie sur une description du comportement basée sur le séquencement des échanges de messages entre objets et le diagramme d'activité centre sa représentation sur un enchaînement d'actions ou d'activités. Les machines à états proposent une approche supplémentaire et complémentaire pour la modélisation en permettant de décrire le comportement *discret* des éléments structurels avec la modélisation des différents états d'un élément ainsi que les conditions et les actions associées au passage d'un état à un autre.

Les cas d'utilisation. Les cas d'utilisation UML permettent de définir, dans un diagramme de cas d'utilisation, les interactions sous une forme fonctionnelle entre les éléments extérieurs du système et ce dernier.

2.3.2 Rational Unified Process (RUP)

2.3.2.1 Historique

Le processus unifié est le résultat de la fusion des méthodes Objectory d'Ivar Jacobson, Booch de Grady Booch et OMT de James Rumbaugh, enrichi de nombreux apports issus des travaux d'élaboration du standard UML. La figure 2.13 retrace la succession des produits qui en sont issus, depuis le processus Objectory, dont la première version est sortie en 1987, jusqu'au Rational Unified Process (sortie en 1998) en passant par le Rational Objectory Process (1997). Le Rational Unified Process a évolué au fil des années pour finir par incarner l'expérience de milliers de projets dans tous les domaines imaginables. Historiquement, le RUP est le successeur direct de Rational Objectory Process (version 4) (Figure 2.13).

2.3.2.2 Le RUP et le Développement Itératif

La plupart des équipes de développement logiciel continuent à appliquer un processus en cascade, dans lequel chaque phase observe une séquence stricte : spécification, analyse et conception, implémentation et intégration, et enfin les tests. On rencontre couramment une méthode en cascade modifiée, qui ajoute des boucles de rétroaction à ce flux global.

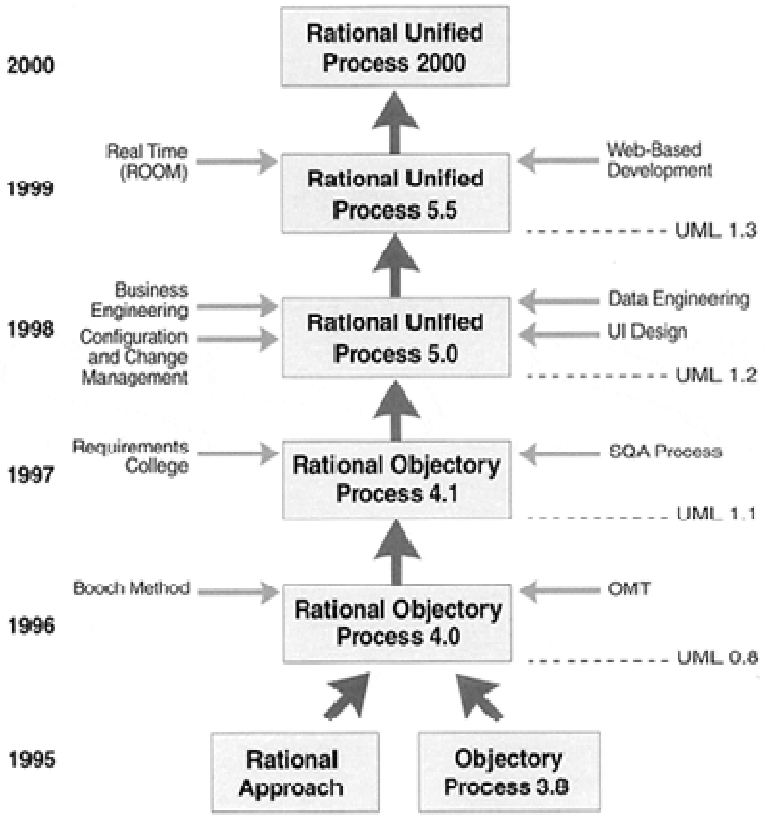

Figure 2.13 Historique du RUP [Kru 00]

De telles approches forcent des membres clés de l'équipe à de longues périodes d'inactivité, et diffèrent les tests en fin de cycle de vie du projet, moment où les problèmes ont tendance à être plus difficiles et plus coûteux à résoudre. Ils constituent ainsi de sérieuses menaces par rapport aux délais de livraison. A l'inverse, le RUP applique une démarche itérative, autrement dit une suite d'étapes incrémentales, ou itérations. Chaque itération comprend la plupart des disciplines du développement, sinon toutes (spécification, analyse, conception, implémentation, etc.), comme le montre la figure 2.14. Elle a un ensemble d'objectifs bien défini et produit une implémentation fonctionnelle partielle du système final. Chaque itération successive s'appuyant sur le travail effectué dans les précédentes, le produit final évolue et se raffine jusqu'à ce qu'il soit parachevé.

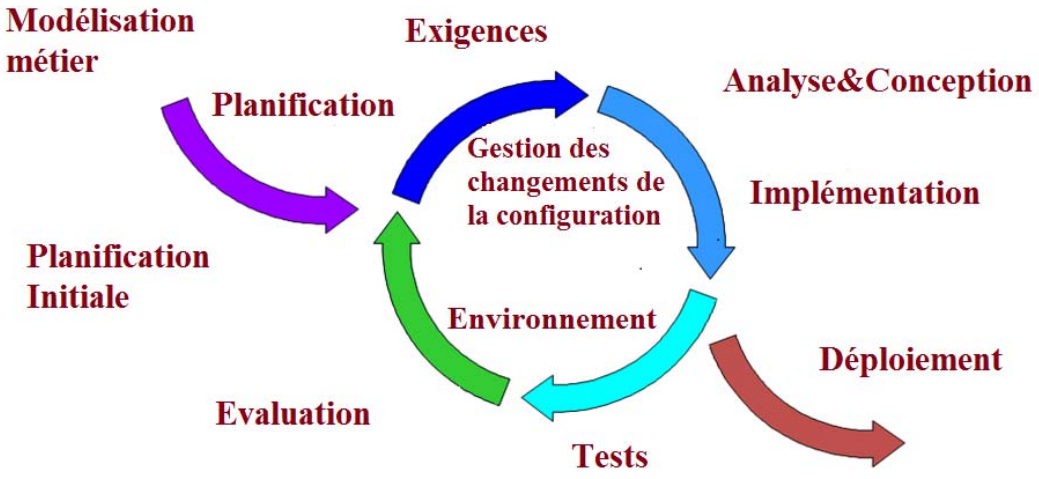

Figure 2.14 RUP et le développement itératif

L'approche itérative s'est montrée supérieure à la méthode en cascade pour plusieurs raisons :

- Elle s'adapte à l'évolution des besoins.
- L'intégration ne se résume pas à un « big bang » en fin de projet.
- Les risques sont généralement décelés lors des premières intégrations.
- Le management dispose d'un moyen d'apporter des modifications tactiques au produit.
- La réutilisation est facilitée.
- Les anomalies sont détectées au cours de plusieurs itérations.
- Les compétences sont mieux exploitées.
- L'apprentissage est permanent.
- Le processus de développement lui-même s'améliore continuellement.

2.3.2.3 Architecture du Rational Unified Process

RUP a été conçu grâce à des techniques analogues à celles de la conception logicielles. Il a été en particulier modélisé en appliquant le modèle SPEM (Software Process Engineering Metamodel) — un standard de modélisation de processus basé sur UML (Unified Modeling Language). La figure 2.15 représente l'architecture globale de RUP. Le processus possède deux structures ou (deux dimensions) :

- ***Une structure dynamique.*** La dimension horizontale représente la structure dynamique, autrement dit la dimension temporelle du processus. Montre comment

celui-ci, exprimé en termes de cycles, de phases, d'itérations et jalons, se déroule au cours de la durée de vie du projet.

- ***Une structure statique.*** La dimension verticale représente la structure statique du processus. Elle décrit la façon dont ses éléments – activités, disciplines, artefacts et rôles – sont regroupés logiquement en disciplines principales, ou enchaînements d'activités (workflow).

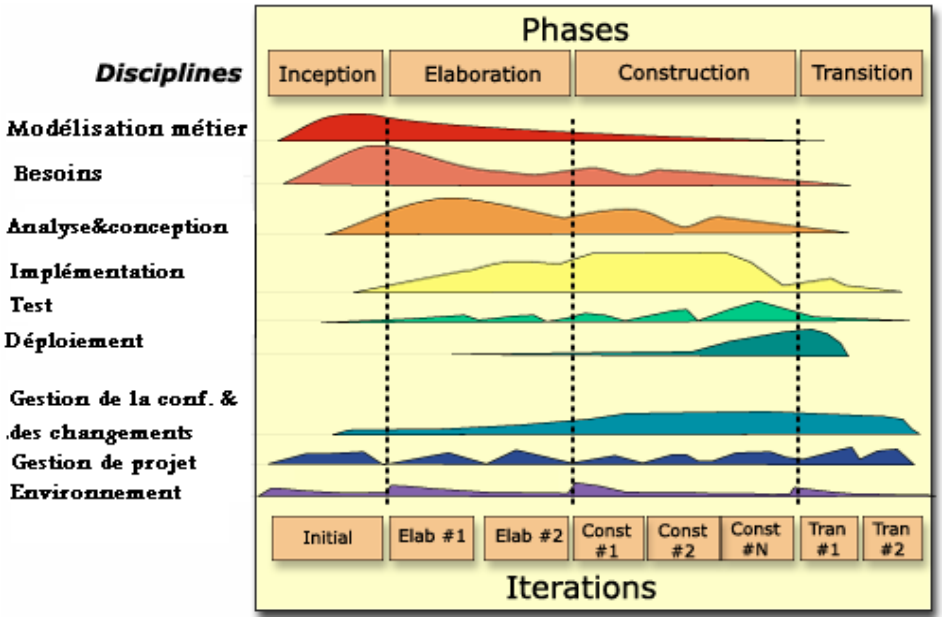

Figure 2.15 Les deux Dimensions de RUP

A - La structure dynamique du RUP(Cycle de vie du RUP: phases, objectifs et jalons)

A.1 - Concepts fondamentaux

Cycles. Un cycle de développement est le laps de temps qui s'écoule entre le tout début du projet et la livraison du produit (ou l'annulation du projet). Il comprend toutes les activités exécutées durant cette période.

Phases. Chaque cycle du RUP est décomposé en une suite de quatre **phases**, nommées Inception, Elaboration Construction et Transition.

Itérations. Chaque phase peut comprendre une ou plusieurs itérations. Chaque itération développe une partie du logiciel -- une version interne ou externe – et se conclut par un jalon mineur, qui constitue un point permettant d'évaluer l'avancement du projet. Le produit logiciel croît de façon incrémentale au fur et à mesure des itérations.

La structure dynamique concerne le cycle de vie, ou dimension temporelle, d'un projet. Le RUP applique une approche structurée au développement itératif, divisant un projet en quatre phases : inception, élaboration, construction et transition.

A.2 - Jalons des phases du cycle de vie du RUP.

Le cycle de vie du RUP compte quatre phases: inception, élaboration, construction et transition. Il se termine avec la livraison d'un produit logiciel complet. Chacune des quatre phases du RUP possède un jalon et un ensemble bien défini d'objectifs. Ceux-ci nous permettent de déterminer les activités à effectuer et les artefacts à produire (Figure 2.16).

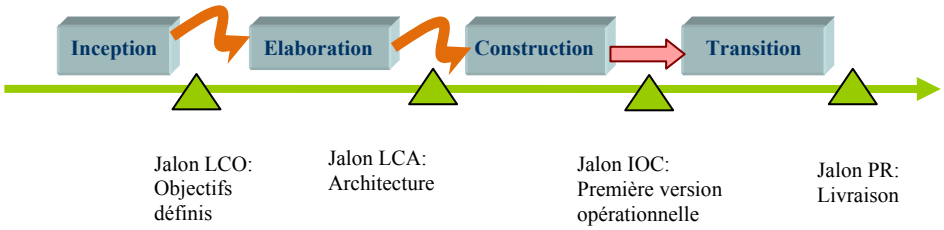

Figure 2.16 Les jalons des phases du cycle de vie du RUP

Chaque itération doit être précisément centrée sur ce qui est strictement nécessaire pour atteindre les objectifs métiers de la phase en question. Chaque phase comprend une ou plusieurs itérations, consacrées à la production des livrables techniques requis pour atteindre les objectifs métiers de cette phase.

A.3 - Les phases du cycle de vie du RUP.

Le cycle de vie du RUP compte quatre phases : inception, élaboration, construction et transition. Il se termine par une livraison d'un produit logiciel complet. Même si ces quatre phases sont exécutées séquentiellement, son cycle de vie est fondamentalement **itératif et piloté par les risques**. Il est vrai que les premières semaines ou les premiers mois d'un projet mettent plutôt l'accent sur les exigences et qu'il y aura plus de tests et de finitions en fin de projet. C'est précisément ce changement de priorité au cours du cycle de vie que représentent les « bosses » sur le graphique des itération (voir Figure 2.15), et chacune d'elles comprend de nombreuses activités de développement produisant du code testé qui devient une version interne, et plus tard externe.

Phase Inception.

L'inception est la première des quatre phases du cycle vie du RUP. Elle est entièrement dédiée à l'étude du périmètre et des objectifs du projet, et doit permettre de récolter suffisamment d'information. Elle vise les objectifs suivants.

Objectif 1 : Comprendre le système à construire. Déterminer la vision, le périmètre du système et ses limites, autrement dit ce qui en fait partie et ce qui n'en fait pas partie. Identifier les personnes qui veulent le système et la valeur qu'elles lui accordent.

Tous les intervenants (clients, responsables, analystes, développeurs, rédacteurs techniques et autres personnes impliqués) doivent s'accorder sur une définition commune du succès. Pour ce faire, voici les mesures à prendre :

- **Convenir d'une vision de haut niveau**. La solution s'exprime par la production d'un document. Pour un petit projet, ce document peut être informel, voire se réduit à un message qui résume une précédente discussion autour d'un tableau blanc. Pour les projets de taille moyenne, ce document peut comporter quelques pages.
- **Fournir une vision large et superficielle du système**. Décrire ce que le système doit faire sans trop détailler, pour éviter de s'enliser ou de voir certains intervenants, responsables ou clients par exemple, perdre de vue l'essentiel de l'information parce qu'elle est enfouie sous une masse de documentation.
- **Détailler quels sont les acteurs et les cas d'utilisation essentiels** pour que tous les intervenants les comprennent facilement et que les membres de l'équipe puissent les utiliser directement.

Objectif 2 : Identifier la fonctionnalité essentielle du système. Il importe de décider quels sont les cas d'utilisation essentiels ou les plus significatifs du point de vue architectural pour pouvoir consacrer plus de temps dès le départ aux points les plus critiques. Décider quels sont les cas d'utilisation (les interactions avec les systèmes) les plus critiques.

Chef de projet et architecte doivent collaborer étroitement, impliquer tous les autres intervenants nécessaires et recourir à plusieurs critères pour déterminer quels sont les cas d'utilisation les plus critiques.

La fonctionnalité est le noyau de l'application, ou elle représente des interfaces clés, et a donc un impact majeur sur l'architecture. Généralement un analyste identifie ces cas d'utilisation en analysant la stratégie de gestion de redondance, les risques de contention des ressources, les risques de performances, les stratégies de sécurité des données, etc.

La fonctionnalité doit être livrée. La fonctionnalité résume l'essentiel du système, et livrer une application dont elle serait absente n'aurait pas de sens. En général, les experts du domaine savent de quelle fonctionnalité il s'agit du point de vue de l'utilisateur (comportement primaires, transaction de données en pics, transactions de contrôle critiques, etc.).

La fonctionnalité traite un élément de l'architecture qui n'est couvert par aucun cas d'utilisation. Il faut s'assurer de traiter tous les risques majeurs et comprendre chaque élément du système. Même si un élément donné de l'architecture semble présenter peu de

risques, il peut cacher des difficultés techniques inattendues, qu'il est possible de détecter en concevant, implémentant et testant une partie de la fonctionnalité qui lui est associée.

Les critères 1 et 3 sont particulièrement importants pour l'architecte, tandis que les chefs de projet se concentreront particulièrement sur les points 1 et 2.

Dans un système comprenant vingt cas d'utilisation, généralement trois ou quatre d'entre eux sont critiques. Dans certains systèmes, un ou deux cas d'utilisation peuvent constituer le cœur de l'application, et les autres sont des cas d'utilisation "auxiliaires" permettant leur exécution. Dans ce cas, moins de 20 à 30% des cas d'utilisation ont une signification architecturale, et l'équipe n'implémentera généralement que quelques scénarios pour ceux qui font partie du noyau.

Objectif 3 : Déterminer au moins une solution possible. L'objectif global de la phase d'inception est de déterminer si la poursuite du projet est possible, et de s'assurer de l'existence d'au moins une architecture potentielle permettant de construire le système sans trop de risques et avec un coût raisonnable. Dans certains cas, il faut acquérir ou implémenter certains éléments clés de l'architecture, ou d'autres suggestions d'architecture, afin de mieux comprendre les risques auxquels l'équipe est confrontée et les options dont elle dispose. Dans le cas d'application où les différents intervenants peuvent avoir du mal à se représenter le système final, l'équipe doit également consacrer un certain temps à développer quelques prototypes fonctionnels, suffisamment riches pour vérifier que la vision a un sens. A la fin de la phase d'inception, l'équipe doit avoir une idée globale des risques auxquels elle est confrontée.

Objectif 4 : Comprendre les coûts, les délais et les risques associés au projet. Comprendre le système à construire est essentiel, mais définir la façon de le construire et à quel coût est également crucial. Pour déterminer s'il faut poursuivre un projet, il faut avoir une idée globale de son coût. La plupart des coûts sont associés aux ressources dont l'équipe aura besoin et au temps nécessaire pour terminer le projet. En combinant ces données avec ce que l'équipe connait des fonctionnalités nécessaires et ce qu'elles représentent pour les utilisateurs, elle est à même de construire l'étude de rentabilité. Cette dernière documente la valeur économique du projet exprimée en termes quantitatifs, de retour sur investissement (ROI) par exemple. C'est l'instrument qui permet d'obtenir un financement adéquat. Il décrit également les principaux risques non réduits, et donc le degré d'incertitude qui demeure.

Objectif 5 : Décider du processus à appliquer et des outils à utiliser. Il importe que l'équipe partage une vision commune de la façon dont elle va développer le logiciel, autrement dit du processus qu'elle va appliquer. Il faut s'assurer que le processus est aussi léger que possible pour minimiser les surcoûts, et qu'il répond aux besoins spécifiques du projet. Un petit projet supporte que les décisions sur le processus exact à suivre soient prises à

mesure, mais un projet plus important nécessite plus de temps préalable à consacrer à son choix.

L'idée est d'obtenir lors de la première itération un processus et un environnement d'outils fonctionnels. En deuxième itération, le processus et les outils seront déployés, et on obtient un feed-back immédiat sur ce qui fonctionne et ce qui ne fonctionne pas. En fonction de ce *feed-back*, on actualise le processus et ses outils, et on continue à itérer jusqu'à ce qu'on soit satisfait de son environnement.

Une fois décidé d'un processus, on peut sélectionner des outils. Dans certains cas, l'environnement sera prédéterminé, pour correspondre par exemple à un standard de l'entreprise. Sinon, on doit choisir notamment un environnement de développement intégré (EDI), un outil de modélisation graphique, un outil de gestion de la configuration et des changements. Il importe que ces outils permettent d'automatiser efficacement le processus qu'on a choisit. On devra peut-être alors personnaliser les outils et les modèles, installer différents répertoires pour le projet, etc.

Revue de projet : le jalon LCO (objectifs définis). La phase inception se termine par le premier grand jalon du projet, le jalon LCO. A ce stade, on examine les objectifs liés au cycle de vie du projet. Si ce jalon n'est pas atteint, le projet doit être annulé ou reconsidéré. S'il est voué à l'échec, mieux vaut en rendre compte tôt que tard, et l'approche itérative combinée à ce jalon peut révéler que c'est son destin.

Le projet peut être annulé ou considérablement repensé s'il échoue à passer ce cap.

Phase Elaboration

L'élaboration est la deuxième des quatre phases de l'approche RUP. Son objectif est de définir l'architecture du système et de l'établir comme référentiel afin de fournir une base stable au gros de l'effort de conception et d'implémentation en phase de construction. Celle-ci émerge de l'analyse des exigences les plus significatives et de l'évaluation des risques.

Cet objectif général se traduit par quatre sous-objectifs majeurs qui traitent chacun une zone de risque (exigences, architecture, coût, calendrier, Etc.). Cela permet de passer à la phase de construction avec un minimum de risques et de problèmes.

Les objectifs de la phase élaboration sont :

- Objectif 1 : Comprendre en détail les exigences
- Objectif 2 : Concevoir, implémenter, valider l'architecture et en établir une version de référence.
- Objectif 3 : Réduire les risques essentiels et estimer plus exactement les délais et le budget.
- Objectif 4 : Affiner le plan de développement et mettre en place l'environnement de développement

Revue de projet : le jalon LCA. Le jalon LCA termine la phase d'élaboration. Parvenu à ce stade, on examine les objectifs détaillés du système et son périmètre, le choix de l'architecture et la résolution des risques majeurs. Si le projet n'atteint pas ce jalon, il peut être annulé, ou doit être au moins sérieusement reconsidéré. Combinée avec ce jalon, la démarche itérative force une telle décision.

Phase Construction

La construction est la troisième des phases du cycle de vie du RUP. Cette phase est centrée sur la conception détaillée, l'implémentation et les tests qui permettront d'étoffer le système et d'en faire un produit fini.

La construction est entièrement dédiée au développement dans des conditions rentables d'un produit complet – une version opérationnelle du système -- qui puisse être déployé dans la communauté des utilisateurs. Cela se traduit par les objectifs suivants :

- Objectif 1 : Minimiser les coûts de développement et obtenir un certain degré de parallélisme
- Objectif 2 : Développer d'une manière itérative un produit prêt à la transition vers la communauté des utilisateurs.

Revue de projet : le jalon IOC (Initial Operational Capability Milestone) : Architecture définie. La phase de construction se termine par un jalon important : le jalon IOC qui sert à déterminer s'il est possible de déployer le produit dans un environnement de test bêta, en répondant notamment aux questions suivantes :

- Cette version du produit est-elle stable et suffisamment mature pour être déployée dans la communauté des utilisateurs ?
- Tous les intervenants sont-ils prêts pour la transition ?
- Les dépenses en ressources comparées aux dépenses prévisionnelles sont-elle toujours acceptables ?

Si le projet n'atteint pas ce jalon, la phase de transition peut être différée en ajoutant une itération à la phase de construction.

Phase Transition

L'objectif de la phase **transition** est d'assurer que le logiciel réponde parfaitement aux besoins de ses utilisateurs. Elle se déroule normalement en une ou deux itérations, qui consistent à tester le produit en préparation de sa livraison, et à apporter quelques ajustements en fonction du *feed-back* des utilisateurs. A ce stade du cycle de vie, tous les problèmes structurels majeurs doivent être résolus, et ce feed-back ne doit pas porter principalement que sur des questions d'optimisation, de configuration, d'installation et d'ergonomie. La phase de transition des projets plus complexes peut comprendre plusieurs itérations, chacune produisant une version déployable plus élaboré du système.

- Objectif 1 : Comprendre en détail les exigences

- Objectif 2 : Former les utilisateurs
- Objectif 3 : Préparer le site de déploiement et convertir les bases de données opérationnelles
- Objectif 4 : Préparer le lancement
- Objectif 5 : Obtenir le consensus des intervenants
- Objectif 6 : Améliorer les performances futures

Revue de projet : le jalon PR: La phase de transition se termine sur le quatrième grand jalon, le jalon PR. Il permet de déterminer si les objectifs sont atteints et si on doit commencer un autre cycle de développement. A ce jalon, le produit est en production, et le processus d'exploitation, de maintenance et de support commence. Cela peut impliquer d'entamer un nouveau cycle de développement pour apporter des améliorations majeures, ou aller vers une version de maintenance pour corriger certains défauts.

B - La structure statique du RUP

La structure statique concerne la façon dont les différents éléments du processus –rôle, artefacts, activités et disciplines – sont logiquement regroupés en enchaînements d'activités (workflows). Un processus décrit ***qui*** fait ***quoi***, ***comment*** et ***quand***. Le RUP est représenté à l'aide de quatre éléments clés :

- Activités. Comment ?
- Artefacts. Quoi ?
- Enchaînements d'activités. Quant
- Rôles. Qui ?

Rôle. Un rôle définit simplement la façon dont un travail doit être effectué, et spécifie la compétence que les individus qui le jouent doivent posséder et les responsabilités qui leur incombent. Une personne joue généralement un ou plusieurs rôles, et plusieurs personnes peuvent jouer le même rôle.

Activités. Une activité liée à un rôle spécifique constitue une unité de travail qu'un individu, à qui ce rôle est attribué, peut être amené à exécuter. Elle a une fin explicite, habituellement exprimée en termes de création ou d'actualisation d'un artefact donné, par exemple un modèle, un composant ou un plan. Chaque activité est affectée à un rôle spécifique. Une activité doit pouvoir servir d'éléments de planification et de mesure de la progression.

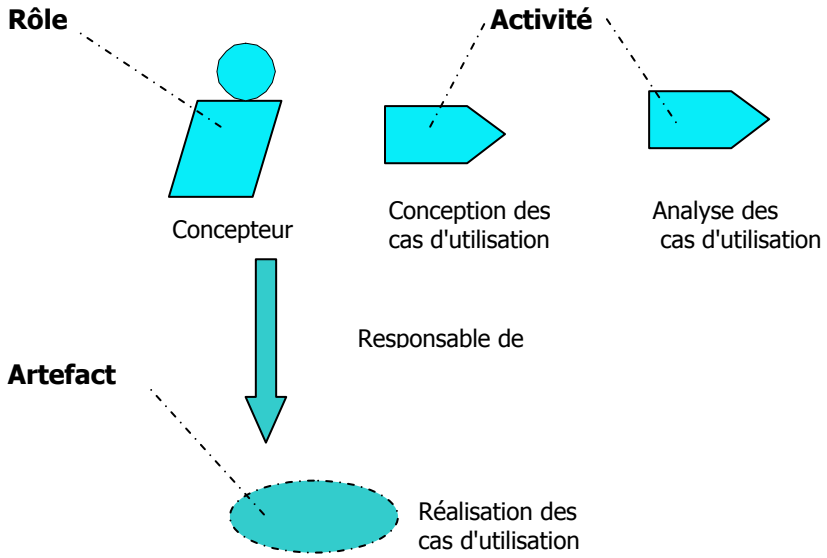

Figure 2.17 Rôles, activités et artefacts

Etapes. Les activités sont décomposées en étapes qui se divisent en trois grandes catégories :

- Les étapes de **réflexion**, où la personne qui joue le rôle étudie la nature de la tâche, recueille et examine les artefacts entrants, et formule un résultat à atteindre.
- Les étapes de **réalisation**, où le rôle crée ou met à jour des artefacts.
- Les étapes d'**évaluation**, où le rôle inspecte le résultat en fonction de certains critères.

Artefacts. Les **artefacts** sont des unités d'information produites, modifiées ou utilisées par un processus. Ce sont les éléments tangibles du projet, ceux qu'il produit ou exploite pour parvenir au produit final. Les artefacts sont utilisés en entrée par des rôles pour réaliser une activité et sont le résultat (ou la sortie) d'autres activités. Ils prennent différentes formes :

- un ***modèle***, comme le modèle des cas d'utilisation ou le modèle de conception ;
- un ***élément de modèle***, comme la vision ou l'étude de rentabilité ;
- un programme source ;
- un ***exécutable***, par exemple un prototype exécutable.

Un artefact peut être documenté de façon formelle (à l'aide d'un outil spécial ou informelle (dans un message électronique ou un tableau blanc).

Enchaînements d'activités. La simple liste de tous les rôles, activités et artefacts, ne constitue pas encore un processus. Il faut disposer de moyen de décrire des séquences significatives d'activités qui produisent un résultat, et de mettre en évidence les interactions entre rôles. C'est précisément l'objectif des enchainements d'activités (Workflows). Ils se présentent sous plusieurs formes, les deux plus courantes étant des ***disciplines***, qui sont des enchaînements d'activités de haut niveau, et les ***détails d'enchainement d'activité*** qui sont des enchaînements d'activités au sein d'une discipline. UML permet d'exprimer un enchaînement d'activités sous forme de diagramme de séquences, de collaboration ou d'activité [Kro 05].

Autre éléments du processus. Les rôles, les activités (organisées en enchaînements d'activités) et les artefacts représentent l'épine dorsale de la statique du RUP. Mais certains autres éléments s'y ajoutent, pour rendre le processus plus facile à comprendre et mettre en œuvre et mieux guider les praticiens (Figure 2.18). Les éléments supplémentaires sont les suivants :

- des **principes et conseils**, qui fournissent des règles, des recommandations ou des heuristiques pour appuyer la réalisation des activités, des étapes et des artefacts ;
- des **modèles** (Templates) pour les différents artefacts ;
- des **guides d'utilisation d'outils**, pour établir un lien avec les outils de développement et fournir une aide à leur utilisation ;
- des **concepts**, pour présenter les définitions et les principes clés ;
- des **cartes**, pour guider l'utilisateur d'un point de vue donné.

Disciplines. Tous les éléments du processus—rôles, activités, artefacts, concepts associés, lignes directrices et modèles—sont regroupés dans des conteneurs logiques nommés **disciplines**. Le produit RUP standard compte neuf disciplines (voir l'encadré suivant). La liste n'est pas définitive, et toute entreprise peut apporter des extensions au RUP en ajoutant des disciplines.

Voici les neufs disciplines :

1. Modélisation métier
2. Gestion des exigences
3. Analyse et conception
4. Implémentation
5. Déploiement
6. Tests
7. Gestion de projet
8. Gestion de la configuration et changements
9. Environnement

Bien que les noms des six disciplines noyau d'ingénierie puissent évoquer les phases séquentielles dans un processus traditionnel en cascade, nous devrions maintenir dans l'esprit que les phases d'un processus itératif sont différentes et que ces enchaînements d'activités sont revisités à plusieurs reprises tout au long du cycle de vie. L'enchaînement d'activités complet actuel d'un projet intercale (Interleaves) ces neuf enchaînements d'activités noyau, et les répète durant les différentes phases avec une intensité à chaque itération (Figure 2.19).

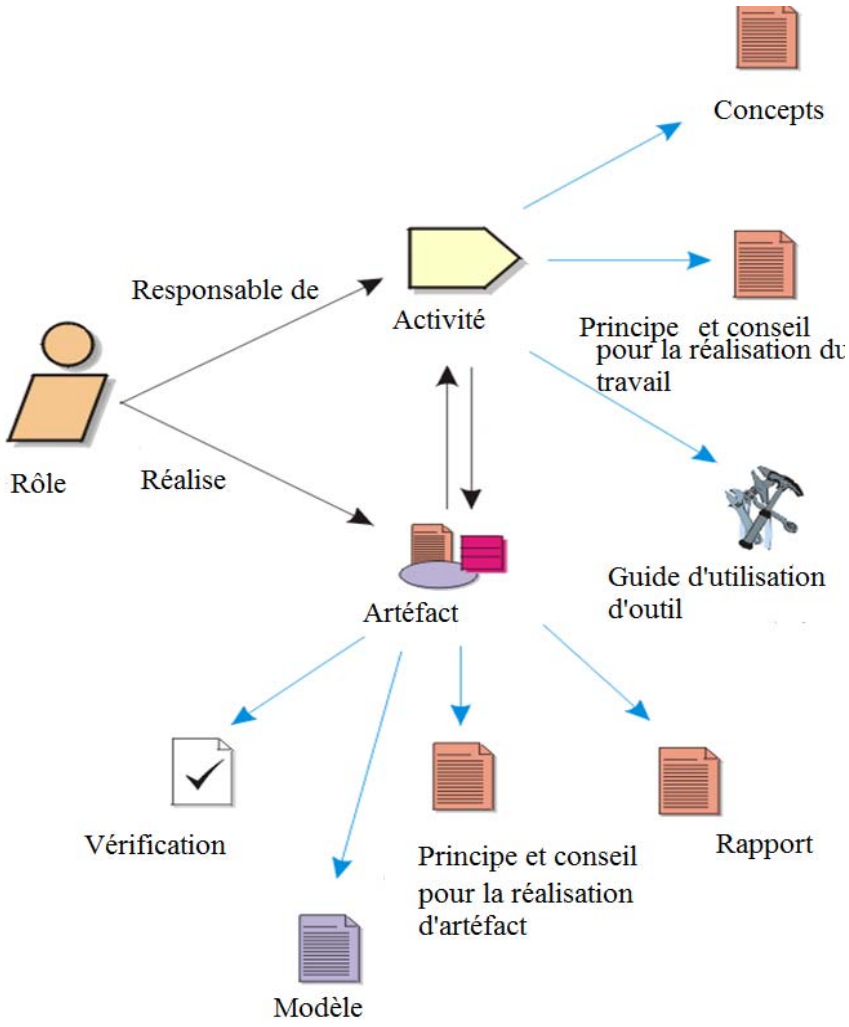

Figure 2.18 Aperçu sur les concepts du RUP [Kru 00]

2.3.2.4 Le RUP- Un produit personnalisable

Chaque projet et chaque entreprise ont des besoins uniques qui nécessitent un processus adapté à leur situation spécifique. Pour répondre à cette exigence, le produit RUP constitue un processus générique complet composé de plusieurs parties intégrées.

- **Meilleure pratique.** Le RUP est accompagné d'une bibliothèque de meilleures pratiques, produites par IBM Software et ses partenaires.

- **Outils de mise à disposition de processus.** Le RUP est littéralement à portée de main des développeurs, parce qu'il est livré en ligne par l'intermédiaire de la technologie Web et non sous forme de livres ou de dossiers.
- **Outils de configuration.** La forme modulaire et électronique du RUP offre la possibilité de le personnaliser et de le configurer pour répondre aux besoins spécifiques d'une organisation de développement.
- **Outils de création de processus RPW** (Rational Process Workbench)
- **Communauté/Marché.** La communauté virtuelle RDN *(Rational Developer Network)* permet aux utilisateurs d'échanger leurs expériences avec leurs pairs et avec des experts, et d'accéder aux dernières informations.

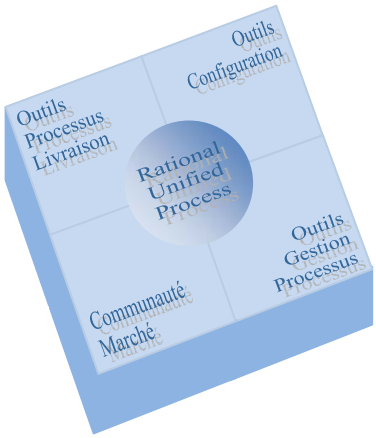

Figure 2.19 Le processus générique du RUP

2.4 Démarches de Développement Actuelles et Ontogénèse

L'évolution est un terme qui désigne tout changement appliqué à un modèle. A travers une littérature abondante, le terme évolution est utilisé comme synonyme de changement ou modification. Actuellement, il n'existe pas de définition standard de l'évolution. Alors que certains lui donne une portée large en considérant que l'évolution d'un système touche toute les phases : aussi bien le développement que la maintenance, d'autres l'utilise comme substitut préférable de la maintenance [Mes 06].

Par définition l'évolution non anticipée des logiciels n'est pas quelque chose qu'on puisse préparer lors de la conception d'un système logiciel. Par contre, l'évolution anticipée est prise en charge durant le développement du système pour être exécutée après sa mise en œuvre.

Pour rester utile, un système logiciel doit constamment évoluer. Ceci est principalement dû au changement et à l'accroissement des exigences de ses utilisateurs. Faire évoluer un

système est un réel challenge impliquant la satisfaction des besoins suivants : découvrir la partie du logiciel concernée par cette évolution, trouver un moyen de la réaliser sans entraîner la régression du système et enfin, pouvoir valider cette évolution. Un autre challenge serait de répondre aux besoins cités précédemment au moindre coût.

Modéliser, même partiellement, le processus de l'évolution par des concepts et des mécanismes qui lui sont dédiés semble nécessaire. Bien que plusieurs approches soient actuellement proposées comme des solutions aux problèmes de l'évolution, elles demeurent néanmoins des palliatifs. Très peu sont les approches qui redonnent à l'évolution un statut important et un paradigme de base, nous pouvons citer à titre d'exemple : "autonomic computing" [Mur 04], les systèmes biomorphiques [Lod 04], le projet « Reconfigurable POEtic Tissue » [Tem 02], et le paradigme Mage [Mes 04, Mes 06].

A notre connaissance, sur le plan des démarches de développement, il n'en existe pas encore qui soient dédiées à ces approches. On trouve dans la littérature plusieurs approches qui tentent de modéliser les évolutions. Néanmoins, toutes ses recherches [Eck 96, Eti 04, Sal 04, Pim 11], reste au niveau de l'analyse des besoins.

2.5 Conclusion

Au fil des années, plusieurs modèles de développement logiciel on été élaborés. Or, quand on pense au modèle en cascades, au modèle en spirale ou un modèle itératif, force est de constater que le développement logiciel est difficile à maîtriser. Des années 1970 à 2000, les processus de développement étaient principalement prédictifs et basés sur la production d'artefacts, souvent dans le but de satisfaire des normes. À l'opposé, principalement après les années 2000, les méthodologies agiles, qui sont réactives et qui mettent l'accent sur les ressources humaines ont gagné en popularité. Nous avons présenté dans ce chapitre RUP, une des méthodes agiles jouissant d'une notoriété mondiale et qui fait le noyau de la démarche proposée dans ce travail de thèse. Nous avons opté pour l'extension du RUP l'adjonction de deux autres disciplines. Le chapitre suivant constitue un état de l'art sur l'ingénierie des besoins des systèmes logiciels et une description de modèles existants dans la littérature pour l'anticipation des changements/évolutions des besoins.