

LEONARDI

Présentation Générale

Objet

Ce document a pour objet de présenter de manière générale le produit LEONARDI et la société LYRIA. Il n'a pas pour objet d'être exhaustif sur les possibilités de cet environnement mais d'en présenter les concepts directeurs.

La première partie expose dans quel contexte a été créé le produit LEONARDI.

Les parties suivantes présente l'architecture générale d'une application LEONARDI et le rôle des composants principaux constituant cette architecture et illustre par des exemples la mise en oeuvre de cette architecture en fonction des besoins d'applications types.

Les dernières parties résumant les avantages clients procurés par le produit LEONARDI et portés par la société LYRIA.

Table des matières



Introduction	3
Architecture LEONARDI	4
Principe général	4
Modèle descriptif	5
Connecteurs.....	6
Afficheurs.....	7
Noyau	7
<i>Gestion des données</i>	7
<i>Gestion des actions</i>	8
Comportements.....	8
Exemples d'architecture	10
Synthèse technique	11
Bénéfices client	12
Versions futures	14

Introduction



Dans une étude publiée en 1995 par le Human Computer Interaction Institute (Myers, Brad A. "User Interface Software Tools."), l'auteur indique que « Les interfaces utilisateurs sont souvent de taille importante, complexes et difficiles à développer, à maintenir et à modifier. »

Cette même étude note que les interfaces utilisateur représentent plus de 50% du temps de développement d'une application et que ces chiffres sont en augmentation, car plus l'interface devient facile à utiliser, plus elles deviennent difficiles à réaliser.

Cette information est corroborée par une étude menée par la grande entreprise de télécommunications NTT (Nippon Telegraph & Telephone Corporation), pour laquelle, le développement des interfaces utilisateurs représente entre 50% et 70% du temps total de développement des applications.

La méthode la plus utilisée pour construire une interface homme-machine consiste à assembler à l'aide d'un outil spécialisé des composants graphiques de plus ou moins haut niveau pour construire la partie visible de l'interface, puis de faire le lien avec les données et les traitements. Il s'agit alors de produire avec un simple éditeur le code nécessaire pour alimenter les objets graphiques avec les données, contrôler les saisies effectuées, réaliser les modifications, déclencher ou implémenter des traitements applicatifs et mettre à jour l'information lorsque cela est nécessaire.

Notre opinion est que ce processus est extrêmement coûteux car essentiellement manuel et donc source de nombreuses erreurs, que celui-ci n'offre aucune garantie de qualité et qu'en outre le résultat obtenu peut s'avérer difficilement maintenable et peu évolutif.

Cette solution est, à notre avis, acceptable essentiellement pour des applications de petite taille manipulant peu de données et offrant un nombre limité d'interactions sur celles-ci. Cette méthode devient vite fastidieuse, voire rédhibitoire et très coûteuse dès que l'application considérée adresse un système d'informations complexe.

De plus, une étude récente de l'IDC souligne que 80% du temps de développement des applications est consacré à la réalisation de l'infrastructure technique de l'application contre seulement 20% à son aspect fonctionnel. L'IDC prévoit que les outils de développement informatiques devront évoluer en particulier en adoptant une démarche MDA (Model-Driven Architecture) afin de parvenir à inverser ces proportions entre développement technique et développement métier et offrir ainsi des gains substantiels de productivité.

C'est porteur de cette même idée et avec cette même volonté que LYRIA a créé LEONARDI...

Architecture LEONARDI



Principe général

L'idée fondatrice de LEONARDI est la fourniture de services génériques à partir de la description des données logiques présentées à l'utilisateur final.

Ce concept est porté par une architecture mettant en œuvre cinq composants principaux :

- le modèle descriptif des données de l'application,
- les afficheurs LEONARDI,
- les connecteurs LEONARDI,
- le noyau LEONARDI,
- les comportements spécifiques de l'application.

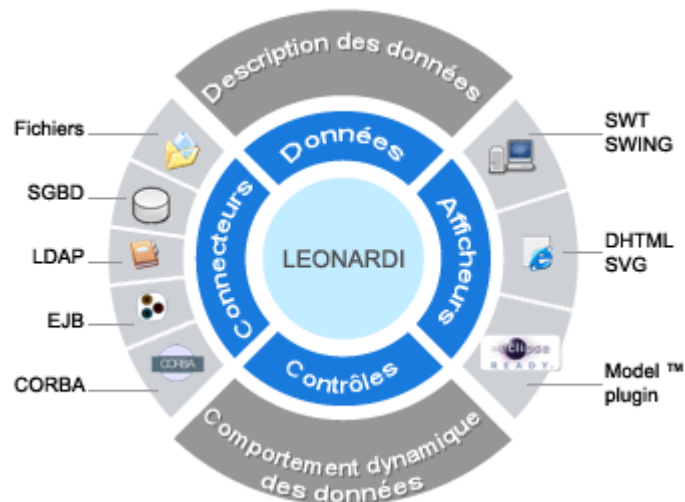


Figure 1 : Architecture LEONARDI

Le modèle descriptif des données d'une application LEONARDI est un ensemble de fichiers XML décrivant les classes de l'application, les attributs constituant ces classes, les actions autorisées sur les instances de ces classes...

Cette description sert de base à la fourniture de services génériques élaborés :

- accès aux données et envoi de requêtes par le biais de connecteurs liés à des sources de données,
- production dynamique des vues présentées sur différents supports d'affichage par le biais d'afficheurs,
- contrôles des saisies effectuées par l'utilisateur,
- traitements des événements sur les affichages permettant le déclenchement des actions requises par l'utilisateur,
- traitements des événements sur les données permettant la mise à jour de l'information présentée à l'utilisateur,
- appels des comportements spécifiques programmés en Java permettant de spécialiser l'application selon sa logique fonctionnelle.

Modèle descriptif



Le **modèle XML descriptif des "entités logiques"** est la pierre angulaire de l'architecture LEONARDI. C'est par là que débute toute application.

Le modèle contient principalement la description des classes de l'application et l'arbre des actions proposées sur le poste client. Ce modèle est structuré en projets.

Un **projet** est notamment constitué de :

- une action racine (par exemple une action de connexion utilisateur/login),
- des classes logiques,
- des actions,
- des sous-projets.

Les **classes logiques** sont les classes telles qu'elles sont présentées à l'utilisateur final, c'est à dire, décorrélées de leur représentation physique réelle.

Outre des informations propres à la classe logique (nom, image associée...), la description d'une classe logique comprend :

- les informations de liaison avec le niveau physique permettant au connecteur de transformer une requête sur un objet logique en requête sur des données physiques,
- la liste des champs constituant la classe,
- la liste des actions autorisées sur les objets logiques.

Les **champs** sont des éléments typés permettant de décrire un attribut d'une classe logique. Les types supportés sont : Texte, Nombre, Date, Choix simple ou multiple, Relation simple ou multiple, Chemin d'accès à un fichier, Structure, Tableau.

Une description de champ comprend différentes informations en fonction du type du champ. Les informations communes sont en particulier un nom et des propriétés. Ces propriétés sont des valeurs booléennes permettant de spécifier des comportements attendus pour un champ tels que : le champ est-il identifiant ? Fait-il partie du nom de l'objet ? Est-il présent lors de la création ? Obligatoire en saisie ? Présenté en affichage tableau ? ... Les informations spécifiques sont, par exemple, pour un champ Nombre la valeur minimale, la valeur maximale, la valeur par défaut, le pas d'incrémentation, l'unité, la formule de calcul... Ou encore, pour un champ relation, on trouve la liste des classes pointées par la relation.

Les **actions** déterminent les traitements applicables soit indépendamment de toute sélection, soit sur une sélection d'un ou de plusieurs objets logiques.

Des actions standard sont pré-implémentées. Elles exploitent la description des classes logiques et des propriétés attachées aux champs de la classe. Les actions standards proposées sont : consultation, création, modification, suppression, impression, import, export, affichage d'une barre d'actions, d'une liste, d'un arbre, d'une vue graphique, connexion, déconnexion...

Il est possible de créer de nouvelles actions, en utilisant ou non une action existante comme modèle.

En particulier, on utilise un modèle pour modifier le déroulement standard d'une action générique en attachant à l'action le nom de la classe Java qui implémente le comportement spécifique voulu par héritage du comportement générique usuel.

D'autres éléments peuvent également être définis dans le modèle descriptif d'une application tels que des **filtres**, permettant de limiter le contenu des vues présentées à l'utilisateur final, ou des **tris**, permettant d'ordonner l'information proposée.

Connecteurs



Les connecteurs ont pour rôle de transformer des requêtes sur les données logiques définies dans le modèle descriptif des données en requêtes sur des données physiques.

Ils ont donc pour rôle l'encodage des requêtes et le décodage des réponses à ces requêtes.

Les requêtes supportées sont :

- création d'un objet,
- modification d'un ou plusieurs objets,
- suppression d'un ou plusieurs objets,
- chargement de listes d'objets (triées, filtrées),
- comptage d'objets (avec ou sans filtre).

Le service attendu des connecteurs est offert via une interface abstraite d'accès et de modification de données. On trouve une implémentation par type de technologie (HIBERNATE, Workflow, GED, SGBDR, SGBDO, Corba, fichiers, ...).

Une même application peut utiliser simultanément plusieurs connecteurs pour accéder à plusieurs sources d'information.

Un certain nombre de connecteurs est proposé en standard. D'autres seront proposés dans le futur. Il est également possible de créer des connecteurs spécifiques pour d'autres types de sources de données.

Afficheurs



*Le rôle d'un afficheur est de traiter la **problématique d'affichage pour une technologie donnée**.*

Les afficheurs prennent en entrée des fichiers XML décrivant des vues indépendamment du support d'affichage. L'afficheur a en charge de transformer cette description XML en objets graphiques correspondant à la technologie d'affichage.

On trouve donc un afficheur par type de cible d'affichage (AWT, Swing, SWT, HTML, DHTML ou DHTML/STRUTS). Un certain nombre d'afficheurs sont présents de manière standard. D'autres seront ajoutés dans le futur. Le cas échéant, il est possible de créer un afficheur spécifique pour une technologie particulière ou pour des bibliothèques d'objets graphiques données.

Les afficheurs ont également pour rôle de transformer les événements physiques sur les objets graphiques en événements logiques abstraits indépendants de la technologie sous-jacente.

Les vues proposées en entrée des afficheurs peuvent être mises à jour soit en temps réel, soit en temps différé selon la capacité de l'afficheur.

Les afficheurs prennent en charge les contrôles élémentaires sur les saisies, dits "contrôles syntaxiques", tels que, par exemple, la vérification du bon format d'une date.

Les afficheurs ont la capacité de gérer le multilinguisme. A cet effet, les vues produites ne contiennent pas directement des chaînes de caractères mais des entrées dans des dictionnaires de chaînes afin de permettre leur remplacement au moment de l'affichage. De cette façon, l'application peut non seulement être lancée dans différentes langues et, en outre, plusieurs utilisateurs peuvent être connectés simultanément dans des langues différentes.

En dernier lieu, il est possible de personnaliser les affichages pour des besoins propres en utilisant XSL pour filtrer les vues XML produites en standard.

Noyau



Le noyau LEONARDI est le cœur de l'application. C'est lui qui fait le lien entre le modèle descriptif des données, les connecteurs et les afficheurs.

Le noyau a 2 fonctions majeures à assurer :

- la **gestion des données**
- la **gestion des actions**.

Gestion des données

Les données logiques sont obtenues ou créées suite à des requêtes effectuées sur les connecteurs spécifiés pour chacune des classes logiques.

En premier lieu, le noyau a pour rôle de conserver en mémoire les données utilisées en fonction de la politique de cache définie pour la classe correspondante et d'accéder aux

connecteurs lorsque des traitements sur les données sont demandés : chargement de données, création, modification ou suppression. Ces traitements peuvent être transactionnés de manière logique ou physique.

En second lieu, le noyau a pour tâche de maintenir ces informations à jour conformément aux opérations effectuées sur les données et aux événements externes reçus sur ces données.

Des mécanismes internes d'abonnement permettent la mise à jour simultanée de toutes les vues présentant les mêmes données lorsque celles-ci sont modifiées de manière interne ou externe (après réception de messages en provenance d'un autre poste ou d'un autre processus du système). Cela signifie que l'affichage reste à jour quelque soit l'origine de la modification.

Lorsque plusieurs utilisateurs sont connectés sur une même application, il est possible d'offrir une vision partielle des données en fonction d'habilitations, de critères géographiques ou organisationnels.

Gestion des actions

Le noyau a en charge de rendre le service générique à partir de la description des données et en particulier de fournir l'implémentation des actions standard citées précédemment.

La prise en charge d'une action est effectuée par un contrôleur qui est une classe Java dédiée qui implémente l'ensemble du processus de traitement d'une action de son déclenchement jusqu'à sa fin.

Pour ce faire, le contrôleur s'appuie notamment sur un bâtisseur, ou *builder*, qui est une classe Java qui a en charge la construction de la vue correspondant à l'action demandée. De manière spécifique par paramétrage de l'action correspondante, il est possible de fournir un bâtisseur particulier pour une action ou de spécifier directement la vue à utiliser au format XML.

Une fois la vue construite, le contrôleur assure le remplissage de la vue avec les données logiques et la mise à jour ultérieure de l'information présentée en fonction des abonnements souscrits sur le module interne de gestion des données.

Par la suite, lorsque l'utilisateur interagit sur la vue proposée, le contrôleur assure le traitement des événements logiques remontés par les afficheurs et la fourniture des réponses aux demandes de l'utilisateur par mise à jour de l'affichage, déclenchement de nouvelles actions, envoi de requêtes sur les données...

Dans les vues autorisant la saisie, le contrôleur a notamment en charge la validation globale des saisies effectuées. Contrairement aux contrôles effectués par les afficheurs, ces contrôles ont pour but de vérifier complètement la validité d'une opération avant impact des données réelles (« contrôles sémantiques »).

L'accès aux différentes actions peut être offert au travers d'une gestion de profils permettant de déterminer si une action est accessible ou non à un utilisateur selon les habilitations qui lui ont été octroyées.

Comportements



Le contrôleur, on l'a vu dans le paragraphe précédent, a en charge le processus complet de traitement d'une action.

En standard, il y a donc autant de contrôleurs proposés que d'actions standard pré-implémentées. Il est bien entendu possible de créer de nouveaux contrôleurs pour des actions totalement nouvelles. Mais il est également possible de spécialiser les actions existantes afin de leur apporter une dimension « métier », spécifique à une application ou à un domaine.

En standard, le contrôleur offre un traitement purement générique guidé par la modélisation des données et par les propriétés affectées à ces données. Lors de ce traitement, le contrôleur fait appel, à différents moments clés, à un contrôleur de comportement permettant d'ajouter une logique spécifique d'une problématique métier.

Le contrôleur de comportement est une classe Java offrant le comportement standard attendu sur l'action concernée. Il est possible d'hériter de ce contrôleur, de modifier le comportement usuel et de l'attacher à l'action dans le modèle descriptif des données afin de spécialiser la prise en compte d'une action.

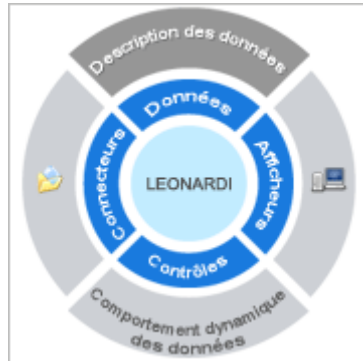
Le contrôleur définit donc le canevas global de traitement d'une action et le contrôleur de comportement permet de spécialiser dynamiquement ce traitement au moment voulu.

Le but de la spécialisation peut être de plusieurs natures :

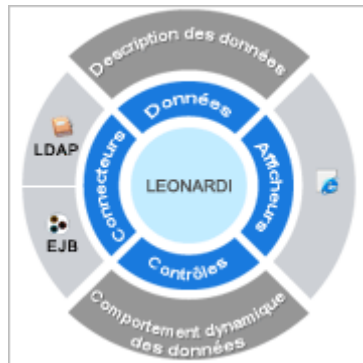
- offrir des contrôles sémantiques plus élaborés sur les saisies,
- gérer des modifications dynamiques sur les affichages,
- limiter les données présentées,
- intégrer l'appel à des produits tiers...

Si le code Java produit s'appuie sur les services offerts par LEONARDI, celui-ci restera alors totalement indépendant tant de la source de données que de la cible d'affichage. Cette approche permet de rendre ce « code métier IHM » valable quelque soit le mode de déploiement de l'application et de s'affranchir des problèmes techniques sous-jacents pour se focaliser sur la vision fonctionnelle du besoin à couvrir.

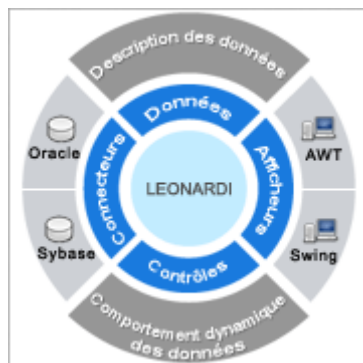
Exemples d'architecture



Maquette réalisée avec un afficheur type client lourd (AWT, Swing ou SWT) et données sur fichiers plats sans codes spécifique. Seul le modèle descriptif des données est fourni.

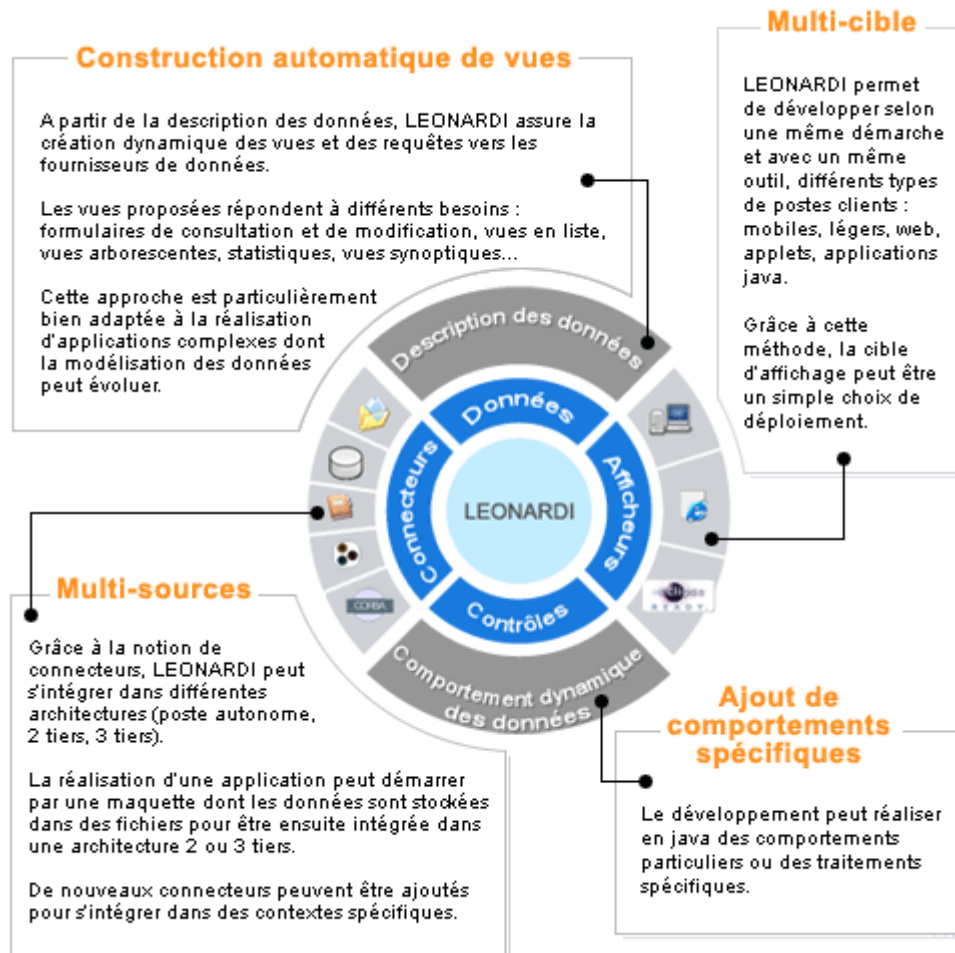


Application 3 tiers déployée avec un afficheur client léger HTML/Javascript (fonctionnant sur navigateurs MSIE et Netscape) et connectée à 2 sources de données : serveur d'application/EJB et annuaire LDAP. L'application comporte des contrôleurs de comportement spécifiques implémentant la logique métier de l'application.



Application déployée avec 2 afficheurs, client lourd (AWT et Swing), et connectée à 2 bases de données (Oracle et Sybase). L'application comporte des contrôleurs de comportement spécifiques implémentant la logique métier de l'application.

Synthèse technique



Le propos de l'architecture **LEONARDI** est de permettre l'automatisation de la réalisation de postes clients évolués, portables et indépendants de la cible d'affichage et de la source de données.

L'automatisation du développement est réalisée par l'écriture d'un modèle XML décrivant les données et les actions de l'application visée et par la fourniture de code générique exploitant cette description afin de fournir un service élaboré de bout en bout.

La portabilité des applications est réalisée par l'utilisation exclusive de technologies portables : Java et XML.

L'indépendance vis à vis de la cible d'affichage et de la source de données est obtenue grâce à la fourniture d'afficheurs et de connecteurs en charge de l'implémentation sur une technologie particulière d'interfaces abstraites masquant les problèmes techniques.

Cette approche a le double avantage de permettre d'emblée au développeur de se focaliser sur la problématique fonctionnelle de son application et d'obtenir des résultats dès que ses données sont décrites.

Le développeur peut ensuite par itérations successives intervenir à plusieurs niveaux pour spécialiser sa réalisation en fonction du besoin et en particulier attacher des comportements spécifiques liés à une logique purement métier.

Bénéfices client



Notre principale force par rapport à nos concurrents provient d'une part, de l'approche d'automatisation prônée par notre produit, en rupture par rapport aux procédés plus classiques proposés par les outils concurrents et d'autre part, de l'architecture du logiciel conçu pour être évolutif et ouvert.

Ces caractéristiques permettent :

- la réduction des coûts et délais de développement,
- la réduction induite des coûts de maintenance,
- l'évolutivité des applications réalisées,
- l'ouverture et l'extensibilité de la solution,
- accessibilité
- la portabilité tous azimuts (affichage, accès à l'information, matériel),
- la qualité du résultat.

Réduction des délais et des coûts

Les bénéfices clients de l'approche LEONARDI sont multiples. En premier lieu, elle permet de diminuer de manière considérable (jusqu'à 70%) les coûts et délais consacrés au développement et de manière induite, les coûts d'intégration et de maintenance.

Evolutivité

Ensuite, le développement ainsi réalisé s'avère plus pérenne. L'application peut d'une part évoluer d'un point de vue fonctionnel en modifiant son modèle de données et d'autre part, l'aspect technique étant majoritairement masqué par le produit LEONARDI, une même application peut aisément passer à de nouvelles architectures en remplaçant les afficheurs et les connecteurs employés.

Extensibilité

Cette capacité lui permet en outre de résister aux futures évolutions technologiques dès lors que celles-ci sont intégrées au produit LEONARDI.

Ouverture

Le produit LEONARDI a été conçu pour être totalement ouvert, l'ensemble des comportements génériques peut être surchargé. L'intégration d'actions et de vues spécifiques peut être réalisé aisément. L'intégration de produits tiers de toute nature reste possible. Au besoin, la création de nouveaux connecteurs, voire de nouveaux afficheurs est également possible.

Plus faible technicité requise

Par ailleurs, le développement s'opérant à un niveau plus fonctionnel, la technicité requise s'avère moins importante, ce qui permet à des développeurs accoutumés à d'autres technologies de devenir plus rapidement opérationnels sur des projets basés sur les nouvelles technologies.

Portabilité

Grâce aux concepts de connecteurs et d'afficheurs ainsi qu'au choix de technologies portables (plate-forme Java et XML), les applications réalisées avec LEONARDI bénéficient d'une portabilité totale : technologie d'affichage, technologie d'accès à l'information, système d'exploitation.

Qualité

La production dynamique des vues de l'application permet d'assurer la qualité, l'homogénéité et l'ergonomie des affichages proposés à l'utilisateur final. Ces vues peuvent par ailleurs être personnalisées de différentes façons (spécialisation des ressources graphiques, ajout de filtres d'affichage, surcharge de la production de vues, utilisation de vues spécifiques...).

Multilinguisme

Grâce à une gestion dans des dictionnaires des chaînes de caractères présentées à l'utilisateur et à la construction dynamique des vues de l'application, LEONARDI permet de créer des applications gérées dans différentes langues. En mode client léger, plusieurs utilisateurs peuvent être connectés en simultané dans des langues différentes.

Versions futures



Pour les versions suivantes, les évolutions envisagées sont les suivantes :

Extension de la couverture technique :

La couverture technique du produit peut être étendue de manière aisée par le biais d'ajout de nouveaux connecteurs (mainframes, SNMP, JCA, ...) ou de nouveaux afficheurs (PDA en particulier). Il faut toutefois noter que l'ajout d'un afficheur pour assistant personnel reste un sujet délicat à traiter car les assistants personnels ont encore des capacités très limitées.

Versions métiers :

Compte tenu de l'approche proposée par le produit et de l'isolation claire des composants fonctionnels propres aux applications réalisées, il est possible de proposer le produit LEONARDI avec des objets métiers prêts à l'emploi réutilisables pour construire des applications spécifiques. Une telle évolution aurait une valeur ajoutée très importante car elle permettrait de réaliser des applications métiers complexes en quelques jours.

Modules de production documentaire :

La version V4.2 introduit le concept de génération de la documentation à partir du modèle métier. Le rôle de ces modules est la production automatique de squelettes de documents (manuels d'installation, manuels utilisateurs, dossiers de rétro-conception) grâce à l'exploitation des informations contenues dans le modèle descriptif des actions et des données de l'application. Cette fonctionnalité apporterait une forte valeur ajoutée car la rédaction et la mise à jour de ces documents sont des tâches fastidieuses et coûteuses.