ENTERPRISE ARCHITECT

Série de Guides d'Utilisateur

Transformation du Modèle

Author: Sparx Systems
Date: 7/11/2024
Version: 17.0

CRÉÉ AVEC ENTERPRISE ARCHITECT

Table des Matières

Transformation du Modèle

L'un des grands avantages de la création de modèles est la possibilité de les manipuler pour produire des sorties, ce qui permet de gagner du temps et de réduire les risques d'erreurs. Enterprise Architect implémente des transformations MDA (Model Driven Architecture) à l'aide d'un système gabarit flexible et entièrement configurable. Les gabarits agissent comme des instructions pour une machine qui prend un modèle en entrée et le transforme en un modèle plus résolu en sortie. L'entrée peut être un modèle volumineux et complexe ou un élément unique et un modèle d'entrée peut être transformé en une variété de modèles de sortie.

Les transformations sont généralement unidirectionnelles et prennent un Modèle indépendant de la plate-forme (PIM) et le transforment en un ou plusieurs modèles spécifiques à la plate-forme (PSM). Un bon exemple de cas où cela est utile est lorsqu'un système doit être implémenté dans un certain nombre de systèmes de bases de données relationnelles différents. Un seul modèle conceptuel indépendant de la plate-forme (le PIM) peut être transformé en un certain nombre de modèles spécifiques à la plate-forme, par exemple Oracle, MySQL et SQLite. Pour une productivité accrue, une fois les modèles de sortie produits, ils peuvent également être convertis en code de programmation, en langage de définition de base de données ou en schémas. Enterprise Architect crée automatiquement une traçabilité qui peut être utilisée pour visualiser la manière dont les éléments du modèle d'entrée ont été transformés en éléments du modèle de sortie.

Facilités

Facilité	Description
Transformer les éléments 	Découvrez comment transformer des éléments sur un diagramme ou depuis une fenêtre Navigateur Paquetage .
Transformations intégrées 	Enterprise Architect propose un certain nombre de transformations intégrées qui support un large éventail de langues cibles. Chacune d'entre elles est entièrement personnalisable en fonction de vos besoins spécifiques.
Modifier Transformation Gabarits 	Apprenez à ajuster les gabarits de transformation pour produire des transformations spécifiques à votre système.
Transformations d'écriture 	Toutes les informations dont vous aurez besoin pour créer vos propres transformations.

Transformations prêtes à l'emploi

Le programme d'installation Enterprise Architect comprend un certain nombre de transformations intégrées de base, notamment :

- PIM à :
  - C#

- C++
- Éléments tableau DDL
- Bean d'entité EJB
- Bean de session EJB
- Java
- PHP
- VB.Net
-XSD

- Modèle de données vers diagramme de relation entre entités (ERD)
- diagramme de relation d'entité (ERD) vers Modèle de données
- diagramme Séquence vers diagramme Communication
- diagramme Communication vers diagramme Séquence
- Modèle Java vers modèle de test JUnit
- Modèle .NET vers modèle de test NUnit
- Modèle d'interface WSDL vers WSDL

D'autres transformations seront disponibles au fil du temps, soit intégrées, soit sous forme de modules téléchargeables à partir du site Web Sparx Systems .

Transformations personnalisées

Vous pouvez modifier les transformations intégrées ou définir les vôtres à l'aide du langage gabarit de génération de code simple d' Enterprise Architect . Cela implique un peu plus que l'écriture gabarits pour créer un fichier source intermédiaire simple ; le système lit le fichier source et le lie au nouveau PSM.

Dépendances de transformation

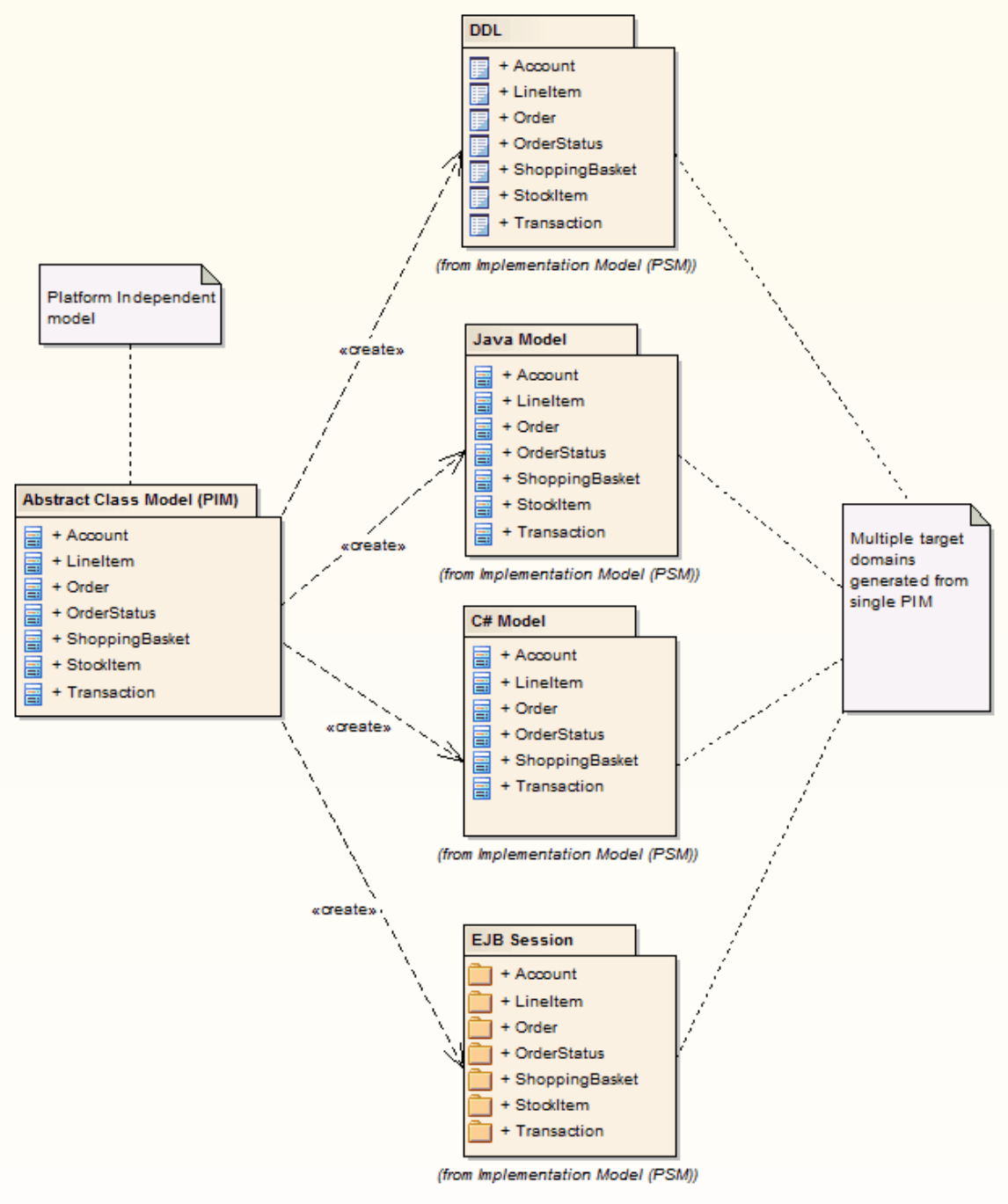
Lorsque vous exécutez une transformation, le système crée des liaisons internes (dépendances de transformation) entre chaque PSM créé et le PIM d'origine. Cela est essentiel, car cela permet de synchroniser plusieurs fois le PIM vers le PSM, en ajoutant ou en supprimant fonctionnalités au fur et à mesure ; par exemple, l'ajout d'un nouvel attribut à une classe PIM peut être synchronisé vers une nouvelle colonne du Modèle de données.

Vous pouvez observer les dépendances de transformation d'un Paquetage à l'aide de la fenêtre Traçabilité, pour vérifier l'impact des modifications apportées à un élément PIM sur les éléments correspondants dans chaque PSM généré, ou pour vérifier où une modification requise dans un PSM doit être initiée dans le PIM (et également pour se refléter dans d'autres PSM). Les dépendances de transformation sont un outil précieux pour gérer la traçabilité de vos modèles.

Enterprise Architect ne supprime ni n'écrase les fonctionnalités d'élément qui n'ont pas été générées à l'origine par la transformation ; par conséquent, vous pouvez ajouter de nouvelles méthodes à vos éléments et Enterprise Architect n'agit pas sur elles pendant le processus de génération directe.

Exemple de transformation

Ce diagramme montre comment fonctionnent les transformations et comment elles peuvent augmenter considérablement votre productivité.



Notes

- Si vous utilisez l'édition Corporate , Unified ou Ultimate , si la sécurité est activée, vous devez disposer de l'autorisation d'accès « Transformer Paquetage » pour effectuer une transformation MDA sur les éléments d'un Paquetage

Transformer les éléments

Une transformation de modèle est une fonction initiée par l'utilisateur qui démarre le processus de transformation d'un ou plusieurs éléments PIM (Platform Independent Modèle) en leurs éléments PSM (Platform Specific Modèle) correspondants. Ce processus se déroule conformément aux règles codifiées dans les Gabarits de transformation. La transformation peut être initiée en sélectionnant un Paquetage dans la fenêtre Navigateur ou un élément dans un diagramme .

Accéder

Ruban	Conception > Paquetage > Transformer > Transformer la sélection
Raccourcis Clavier	Ctrl+H (transformer les éléments sélectionnés) Ctrl+Maj+H (transformer Paquetage sélectionné)

Effectuer une transformation

Option	Action
Éléments	Liste tous les éléments individuels sélectionnés dans le diagramme ou contenus dans le Paquetage . Soit : • Cliquez sur un élément pour inclure uniquement cet élément dans la transformation • Maintenez la touche Ctrl et cliquez sur plusieurs éléments distincts pour les inclure dans la transformation, ou • Maintenez la touche Maj enfoncée et cliquez sur le premier et le dernier élément d'un bloc pour inclure ces éléments dans la transformation
Tous	Cliquez sur ce bouton pour sélectionner tous les éléments de la liste afin de les inclure dans la transformation.
Aucun	Cliquez sur ce bouton pour désélectionner tous les éléments de la liste.
Inclure paquetages enfants	(Si vous avez choisi de transformer des éléments dans un Paquetage .) Cochez cette case pour inclure (dans la liste « Éléments » et potentiellement dans la transformation) les éléments des Paquetages enfants du Paquetage sélectionné.
Transformations	Cochez la case correspondant à chaque type de transformation à effectuer. Lorsque vous cochez une case, la dialogue « Parcourir le projet » s'affiche ; recherchez et sélectionnez le Paquetage cible dans lequel générer les éléments transformés. Si vous souhaitez modifier un Paquetage cible sélectionné, cliquez sur le bouton  à droite du nom Paquetage et sélectionnez le nouveau Paquetage dans le dialogue .
Générer un code sur le résultat	Cochez cette case pour spécifier si vous souhaitez ou non générer automatiquement du code pour les classes transformées qui ciblent les langages de code.

	Si vous sélectionnez cette option, la première fois que vous transformez en classe, le système vous prompts à sélectionner un nom de fichier dans lequel générer du code ; les transformations suivantes génèrent automatiquement du code vers ce nom de fichier.
Effectuer des transformations sur le résultat	Cochez la case pour exécuter automatiquement les transformations précédemment effectuées sur la ou les classes cibles.
Fichier intermédiaire	Si vous souhaitez capturer le fichier de langue intermédiaire (par exemple, pour le déboguer), saisissez le chemin et le nom du fichier ou cliquez sur le bouton ... et recherchez et sélectionnez le fichier.
Écrire toujours	Cochez cette case pour toujours écrire le fichier intermédiaire sur le disque.
Écrivez maintenant	Cliquez sur ce bouton pour générer le fichier intermédiaire sans effectuer la transformation complète.
Transformez-vous	Cliquez sur ce bouton pour exécuter la transformation. Lorsque la transformation est terminée, la dialogue « Transformation du Modèle » se ferme.
Fermer	Cliquez sur ce bouton pour fermer la dialogue « Transformation du Modèle » sans effectuer la transformation.

Notes

- Lorsque le dialogue s'affiche, tous les éléments sont sélectionnés et toutes les transformations précédemment effectuées à partir de l'une de ces classes sont vérifiées.
- Cette procédure ne s'applique pas à la transformation diagramme Séquence / diagramme Communication , ni à la transformation diagramme Communication / diagramme Séquence

Enchaînement des transformations

L'enchaînement des transformations offre un degré supplémentaire de flexibilité et de puissance pour l'exécution des transformations. Par exemple, si deux transformations ont un élément commun, vous pouvez séparer cet élément dans sa propre transformation, puis exécuter les transformations d'origine à partir du point commun. La transformation séparée peut même produire elle-même un modèle utile.

Vous pouvez enchaîner les transformations en sélectionnant la case à cocher « Effectuer les transformations sur le résultat » dans la dialogue « Transformation du Modèle », afin que les transformations qui ont déjà été effectuées sur les classes cibles soient exécutées automatiquement la prochaine fois que cette classe est transformée.

Transformations intégrées

Enterprise Architect fournit un ensemble complet de transformations intégrées, couramment utilisées. Celles-ci s'avéreront utiles dans de nombreuses disciplines, de Modélisation de domaine à l'ingénierie de code. La facilité de transformation des modèles est un outil de productivité pratique et il est attendu que les modélisateurs voudront créer leurs propres transformations personnalisées. Les transformations intégrées fournissent des exemples utiles et constituent une référence précieuse pour le modélisateur.

Transformations intégrées

Transformation	Convertit
C#	Éléments Modèle indépendants de la plate-forme (PIM) vers des éléments de classe C# spécifiques au langage.
C++	Éléments PIM vers éléments de classe C++ spécifiques au langage.
Langage de définition des données	Un modèle logique vers un modèle de données ciblant le type de base de données par défaut, prêt pour la génération DDL.
Diagramme de relation d'entité vers Modèle de données	Un modèle logique ERD vers un modèle de données ciblant le type de base de données par défaut, prêt pour la génération DDL.
Modèle de données vers Diagramme de relation entre entités	Un modèle de données vers un modèle logique ERD.
Bean de session EJB	Un seul élément de classe pour les éléments d'une session EJB.
Bean d'entité EJB	Un seul élément de classe pour les éléments d'une entité EJB.
Java	Éléments PIM vers éléments de classe Java spécifiques au langage.
Unité JUnit	Un élément de classe Java existant avec des méthodes publiques vers une classe avec une méthode de test pour chaque méthode publique, ainsi que les méthodes requises pour configurer correctement les tests.
Unité N	Une classe compatible .NET existante avec des méthodes publiques vers une classe avec une méthode de test pour chaque méthode publique, ainsi que les méthodes requises pour configurer correctement les tests.
PHP	Éléments PIM vers éléments de classe PHP spécifiques au langage.
Séquence / Diagramme Communication	Tous les éléments et messages d'un diagramme Séquence ouvert sont transformés en éléments et messages correspondants dans un diagramme Communication , et vice versa.
VB.Net	Éléments PIM vers éléments de classe VB.Net spécifiques au langage.
WSDL	Un modèle simple vers un modèle étendu d'une interface WSDL, adapté à la

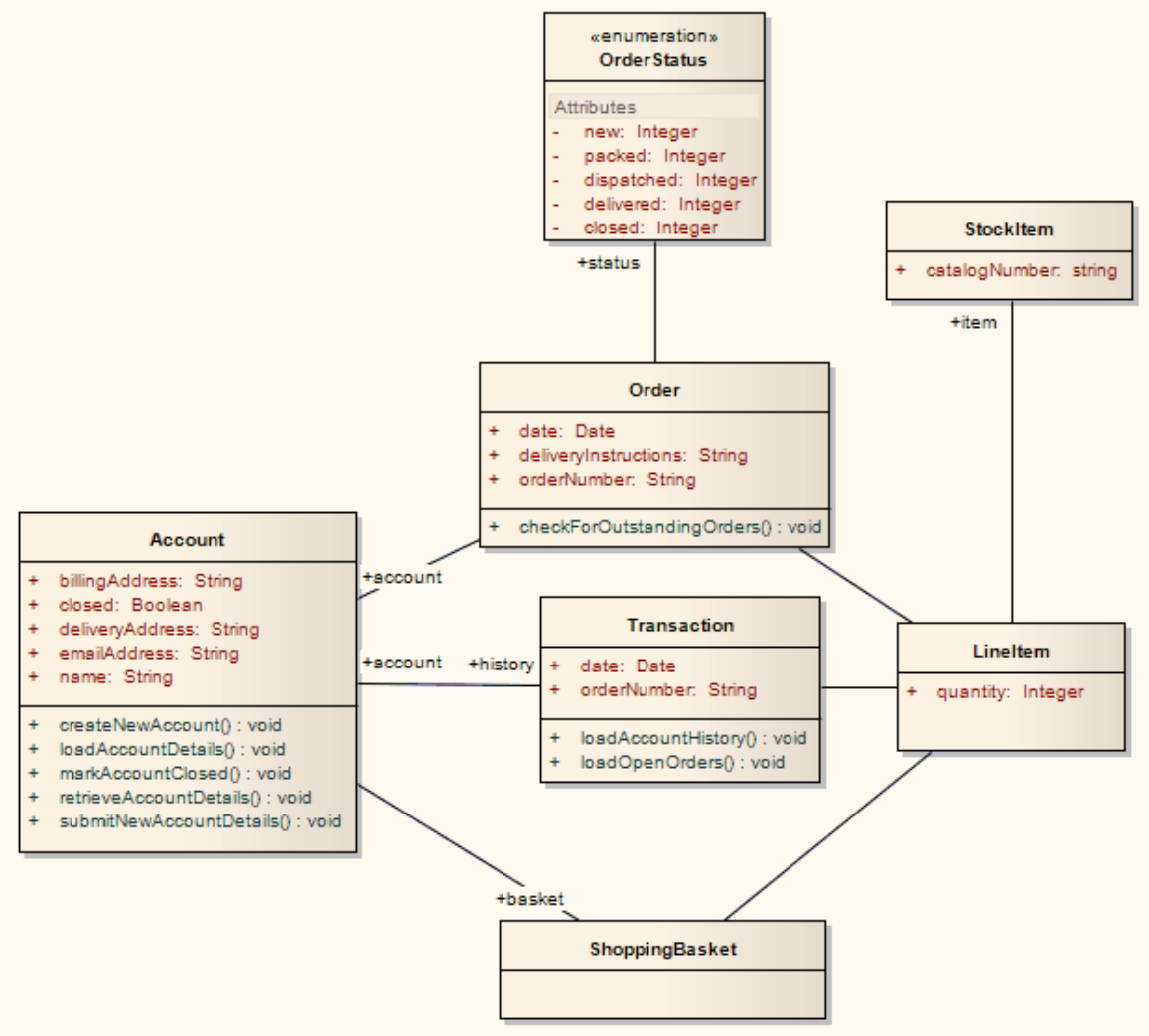
	génération.
XSD	Éléments PIM vers profil UML pour éléments XML, comme étape intermédiaire dans la création d'un schéma XML.

Transformation C#

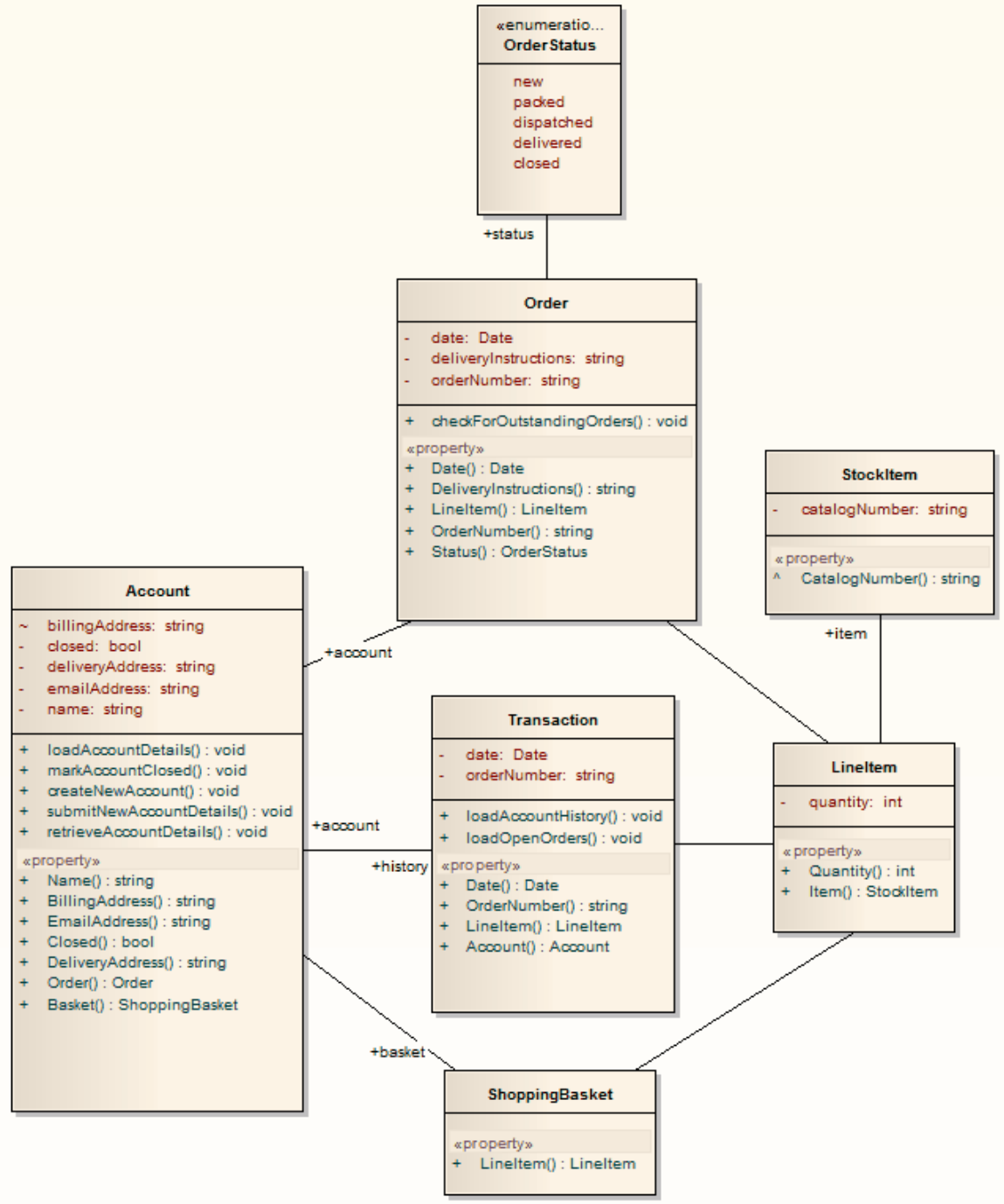
La transformation C# convertit les types d'éléments PIM (Platform-Independent Modèle) en types d'éléments de classe spécifiques à C# et crée une encapsulation en fonction des options système que vous avez définies pour la création de propriétés à partir d'attributs C# (sur la page « Spécifications C# » de la dialogue « Préférences »).

Exemple

Les éléments PIM



Après la transformation, devenez les éléments PSM

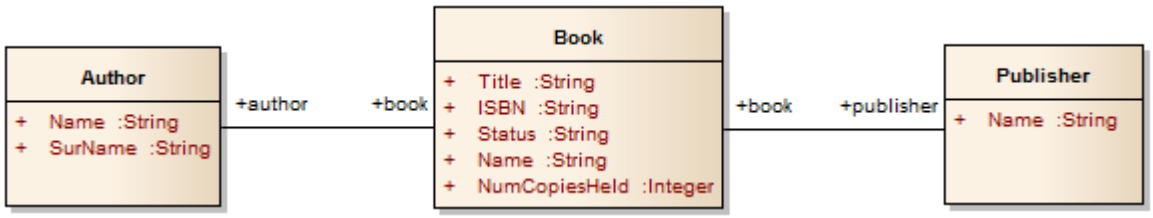

«enumeratio...
OrderStatus
new
packed
dispatched
delivered
closed
+status
Order
- date: Date
- deliveryInstructions: string
- orderNumber: string
+ checkForOutstandingOrders() : void
«property»
+ Date() : Date
+ DeliveryInstructions() : string
+ LineItem() : LineItem
+ OrderNumber() : string
+ Status() : OrderStatus
StockItem
- catalogNumber: string
«property»
^ CatalogNumber() : string
+item
Account
~ billingAddress: string
- closed: bool
- deliveryAddress: string
- emailAddress: string
- name: string
+ loadAccountDetails() : void
+ markAccountClosed() : void
+ createNewAccount() : void
+ submitNewAccountDetails() : void
+ retrieveAccountDetails() : void
«property»
+ Name() : string
+ BillingAddress() : string
+ EmailAddress() : string
+ Closed() : bool
+ DeliveryAddress() : string
+ Order() : Order
+ Basket() : ShoppingBasket
+account
+account
+history
Transaction
- date: Date
- orderNumber: string
+ loadAccountHistory() : void
+ loadOpenOrders() : void
«property»
+ Date() : Date
+ OrderNumber() : string
+ LineItem() : LineItem
+ Account() : Account
LineItem
- quantity: int
«property»
+ Quantity() : int
+ Item() : StockItem
+basket
ShoppingBasket
«property»
+ LineItem() : LineItem

Transformation C++

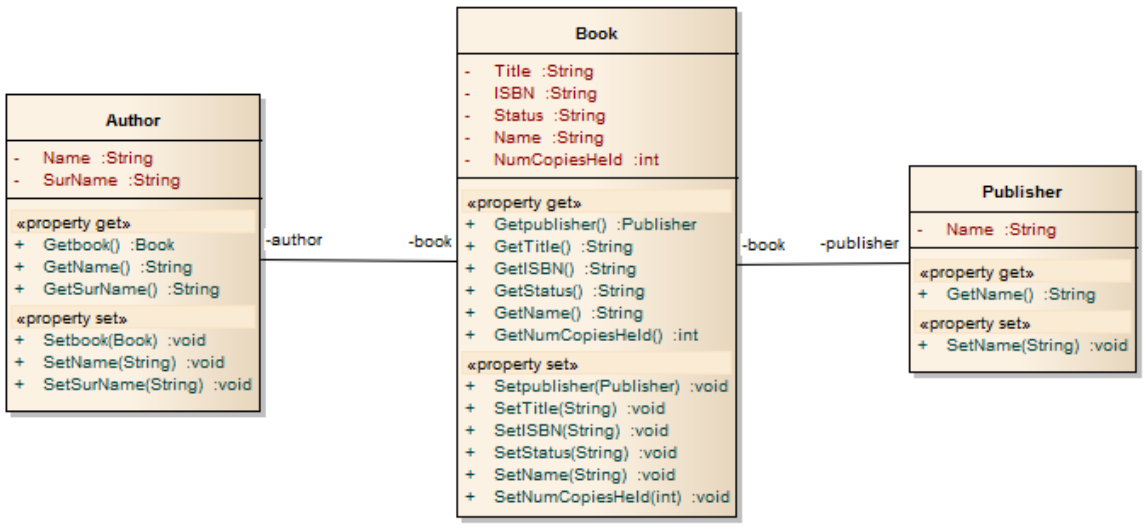
La transformation C++ convertit les types d'éléments PIM (Platform-Independent Modèle) en types d'éléments Class spécifiques à C++ et crée une encapsulation (produisant les getters et setters) selon les options que vous avez définies pour la création de propriétés à partir d'attributs C++ (sur la page « Spécifications C++ » de la dialogue « Préférences »). Note que les attributs publics dans le PIM sont convertis en attributs privés dans le PSM. Toutes les opérations sur une interface sont transformées en méthodes virtuelles pures sur une classe équivalente.

Exemple

Les éléments PIM



Après la transformation, devenez les éléments PSM



Transformation Modèle de données vers l'ERD

La transformation Modèle de données au diagramme de relations d'entités (ERD) crée un modèle logique ERD à partir d'un Modèle de données. Il s'agit de l'inverse de la transformation ERD en Modèle de données. Cette transformation utilise et démontre support dans le langage intermédiaire d'un certain nombre de concepts spécifiques à la base de données.

Concepts pris en charge

Concept	Effet
Entité	Mappé un à un sur les éléments Tableau .
Attribut	Mappé un à un sur les colonnes.
Primary Key	Dérivé du type de colonne PrimaryKey.

Notes

- Parfois, vous souhaiterez peut-être limiter l'étirement des connecteurs de relation en forme de losange ; choisissez simplement un connecteur de relation, cliquez-droit pour afficher le menu contextuel et sélectionnez l'option « Plier la ligne au curseur ».

Transformation DDL

La transformation DDL convertit le modèle logique en un modèle de données structuré pour être conforme à l'un des SGBD pris en charge. Le type de base de données cible est déterminé par le SGBD défini comme base de données par défaut dans le modèle (voir la rubrique d'aide *Types de données de base de données* , option « Définir comme base de données par défaut »). Le modèle de données peut ensuite être utilisé pour générer automatiquement des instructions DDL à exécuter dans l'un des produits de base de données pris en charge par le système.

La transformation DDL utilise et démontre support dans le langage intermédiaire d'un certain nombre de concepts spécifiques à la base de données.

Concepts

Concept	Effet
Tableau	Mappé un à un sur les éléments de classe. Les relations « plusieurs à plusieurs » sont prises en charge par la transformation, créant tableaux de jointure.
Colonne	Mappé un à un sur les attributs.
Primary Key	Répertorie toutes les colonnes impliquées afin qu'elles existent dans la classe et crée une méthode Primary Key pour elles.
Foreign Key	Un type spécial de connecteur, dans lequel les sections Source et Cible répertorient toutes les colonnes impliquées de sorte que : • Les colonnes existent • Une Primary Key correspondante existe dans la classe de destination, et • La transformation crée la Foreign Key appropriée

MDG Technologie pour personnaliser les mappages par défaut

Les transformations DDL qui ciblent un nouveau SGBD défini par l'utilisateur nécessitent une MDG Technologie pour mapper les types de données PIM au nouveau SGBD cible.

Pour cela, créez un fichier .xml MDG Technologie nommé 'UserDBMS Types.xml', en remplaçant UserDBMS par le nom du SGBD ajouté. Placez le fichier dans le dossier EA\MDGTechnologies. Le contenu du fichier MDG Technologie doit avoir cette structure :

<MDG.Technology version="1.0">

<Documentation id="UserdataTypes" name="Userdata Types" version="1.0" notes="Mappage Type de bases de données pour UserDBMS"/>

<Modules de code>

<CodeModule langage="Données utilisateur" notes="">

<Options de code>

<CodeOption name="DBTypeMapping-bigint">BIGINT</CodeOption>

<CodeOption name="DBTypeMapping-blob">BLOB</CodeOption>

<CodeOption name="DBTypeMapping-boolean">TINYINT</CodeOption>

<CodeOption name="DBTypeMapping-text">CLOB</CodeOption>

...

</CodeOptions>

</CodeModule>

</CodeModules>

</MDG.Technology>

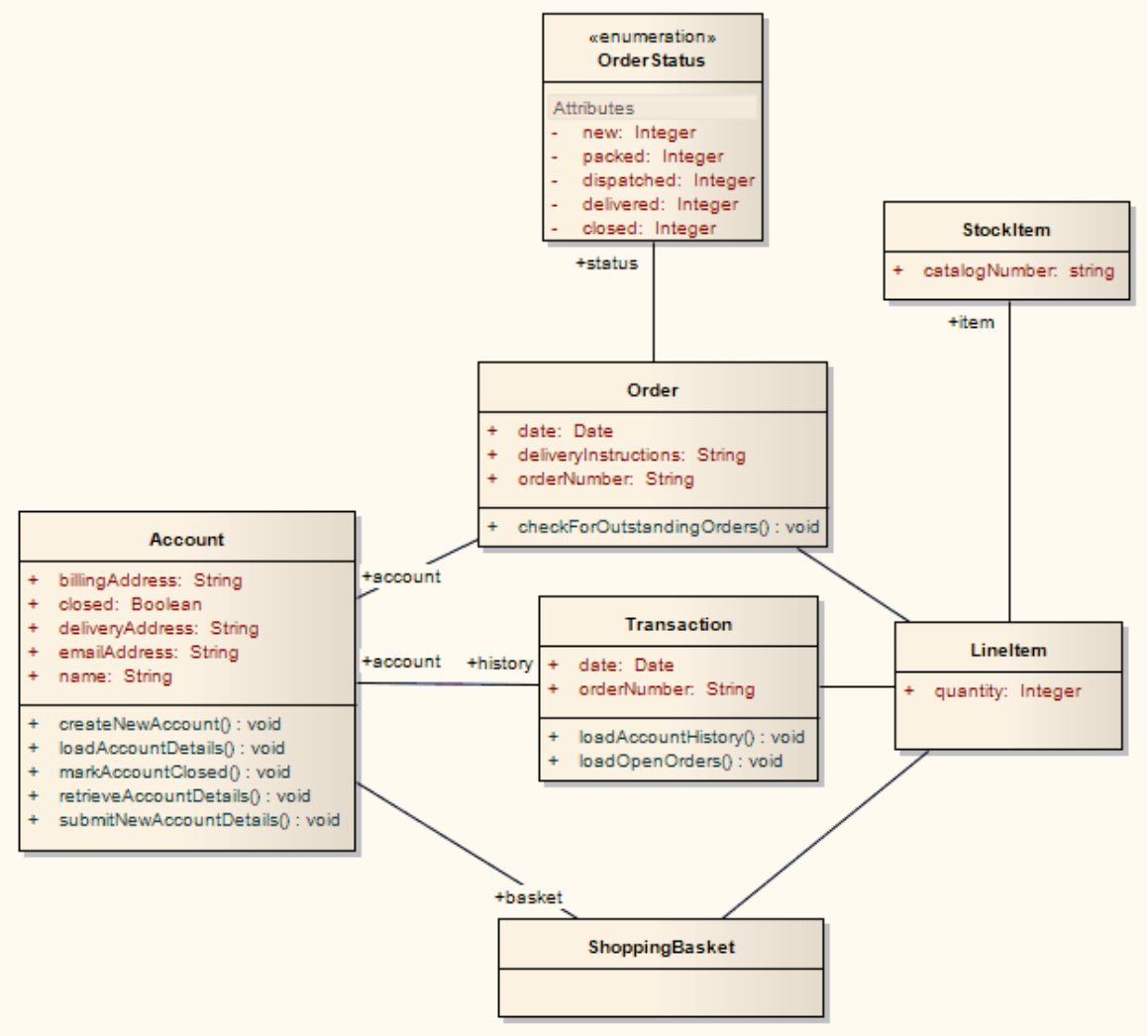
À titre d'exemple, « texte » est un Type commun (comme indiqué dans la dialogue « Types de données de base de données ») qui correspond au type de données « CLOB » d'un nouveau SGBD.

Notes

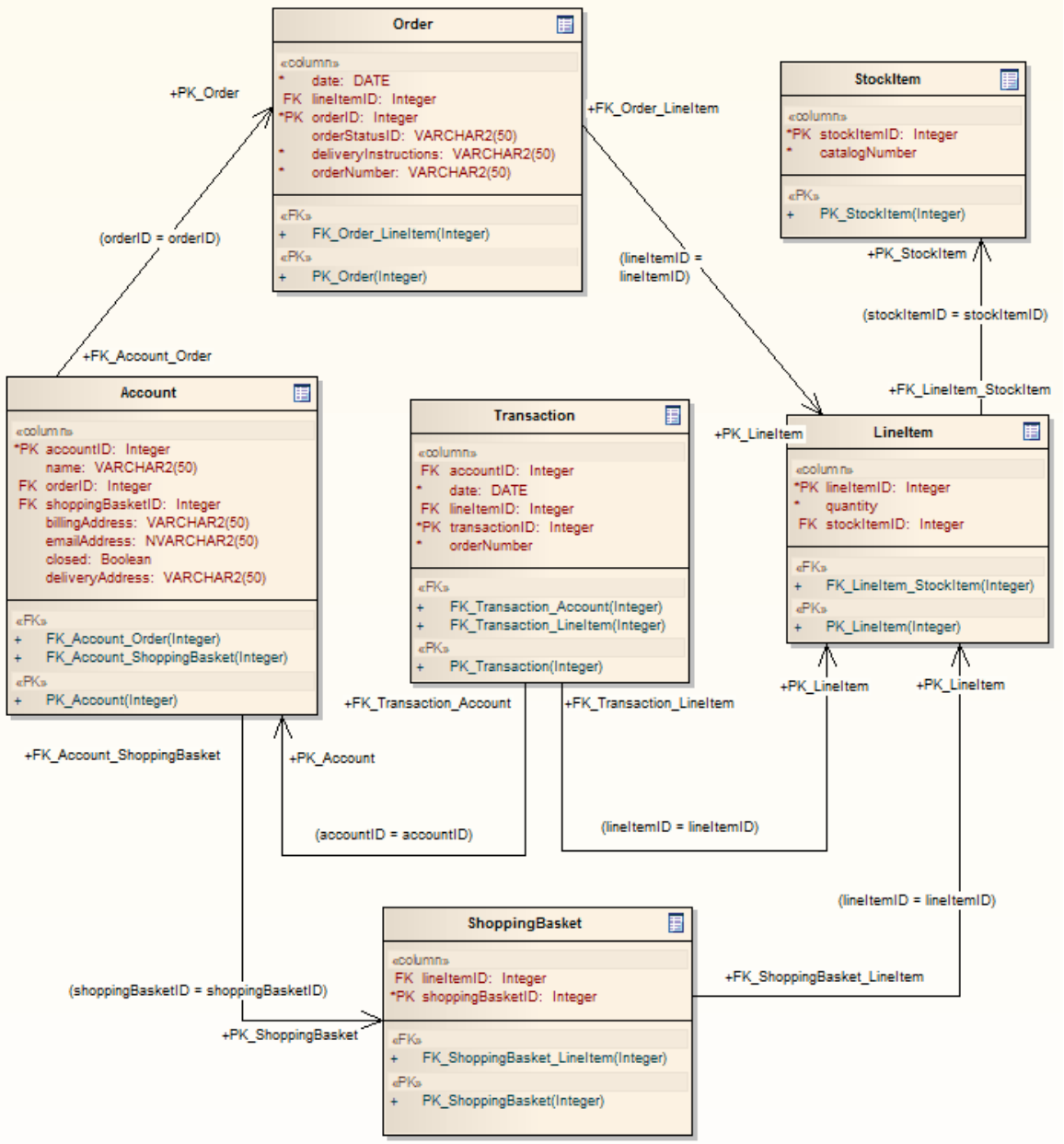
- Vous pouvez définir des aspects spécifiques au SGBD non représentés dans un modèle logique, tels que les procédures stockées, Déclencheurs , Vues et les contraintes de vérification, après la transformation ; voir la rubrique d'aide *Modèle de données physiques*

Exemple

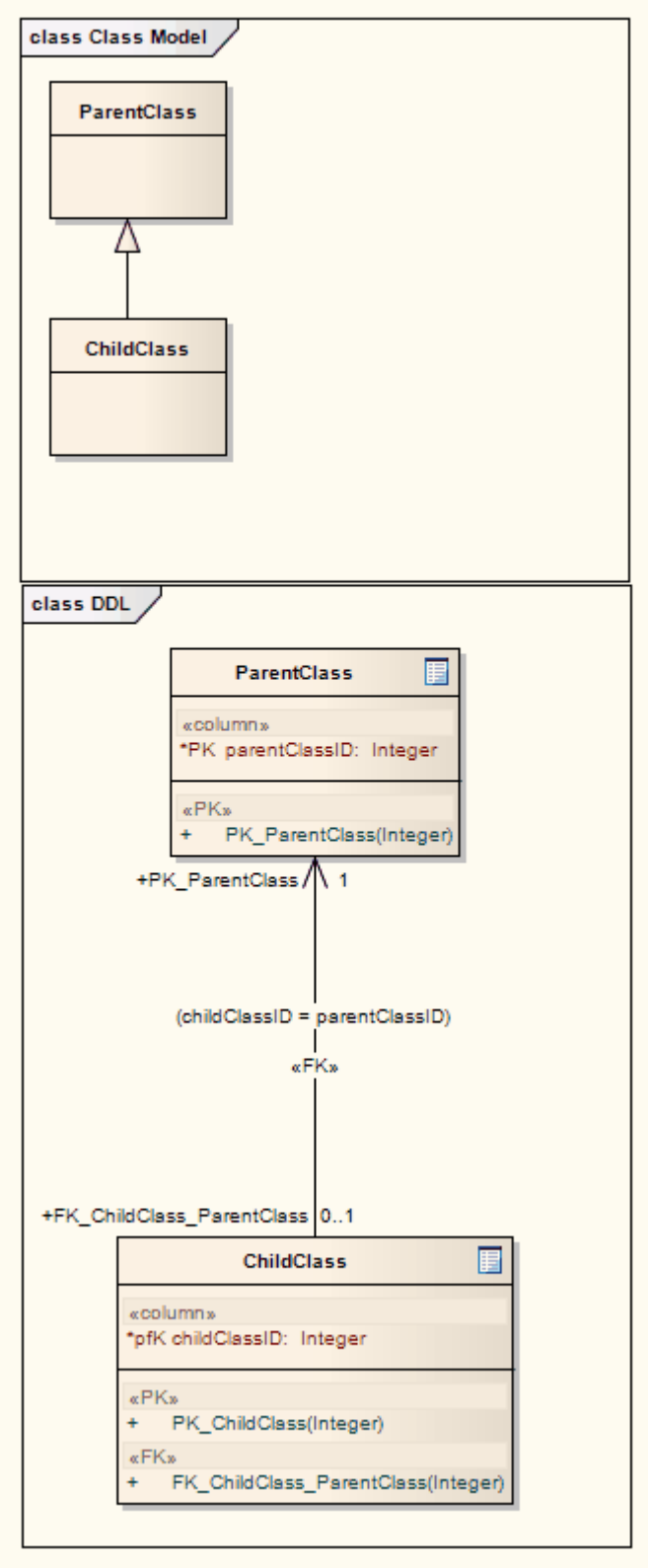
Les éléments PIM



Après la transformation, devenez les éléments PSM



Les généralisations sont gérées en fournissant à l'élément enfant une Foreign Key vers l'élément parent, comme indiqué. L'héritage par copie n'est pas pris en charge.


class Class Model
ParentClass
ChildClass
class DDL
ParentClass
«column»
*PK parentClassID: Integer
«PK»
+ PK_ParentClass(Integer)
+PK_ParentClass
1
(childClassID = parentClassID)
«FK»
+FK_ChildClass_ParentClass
0..1
ChildClass
«column»
*pfK childClassID: Integer
«PK»
+ PK_ChildClass(Integer)
«FK»
+ FK_ChildClass_ParentClass(Integer)

Transformations EJB

Les transformations EJB Session Bean et EJB Entity Bean réduisent le travail nécessaire à la génération des composants internes des Enterprise Java Beans. Vous pouvez donc vous concentrer sur modélisation à un niveau d'abstraction plus élevé.

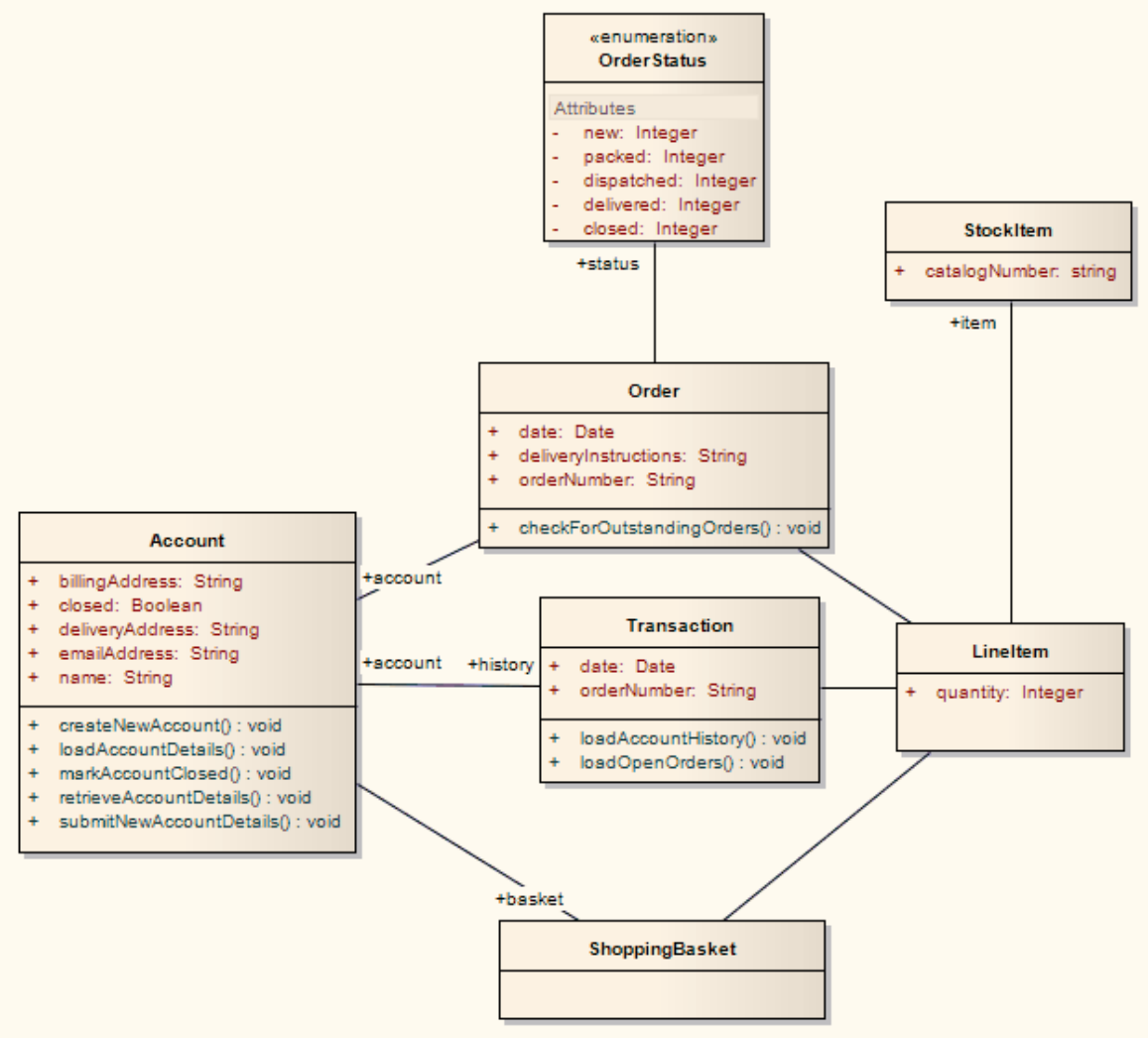
Transformations

Les deux transformations génèrent également un Paquetage META-INF contenant un élément descripteur de déploiement.

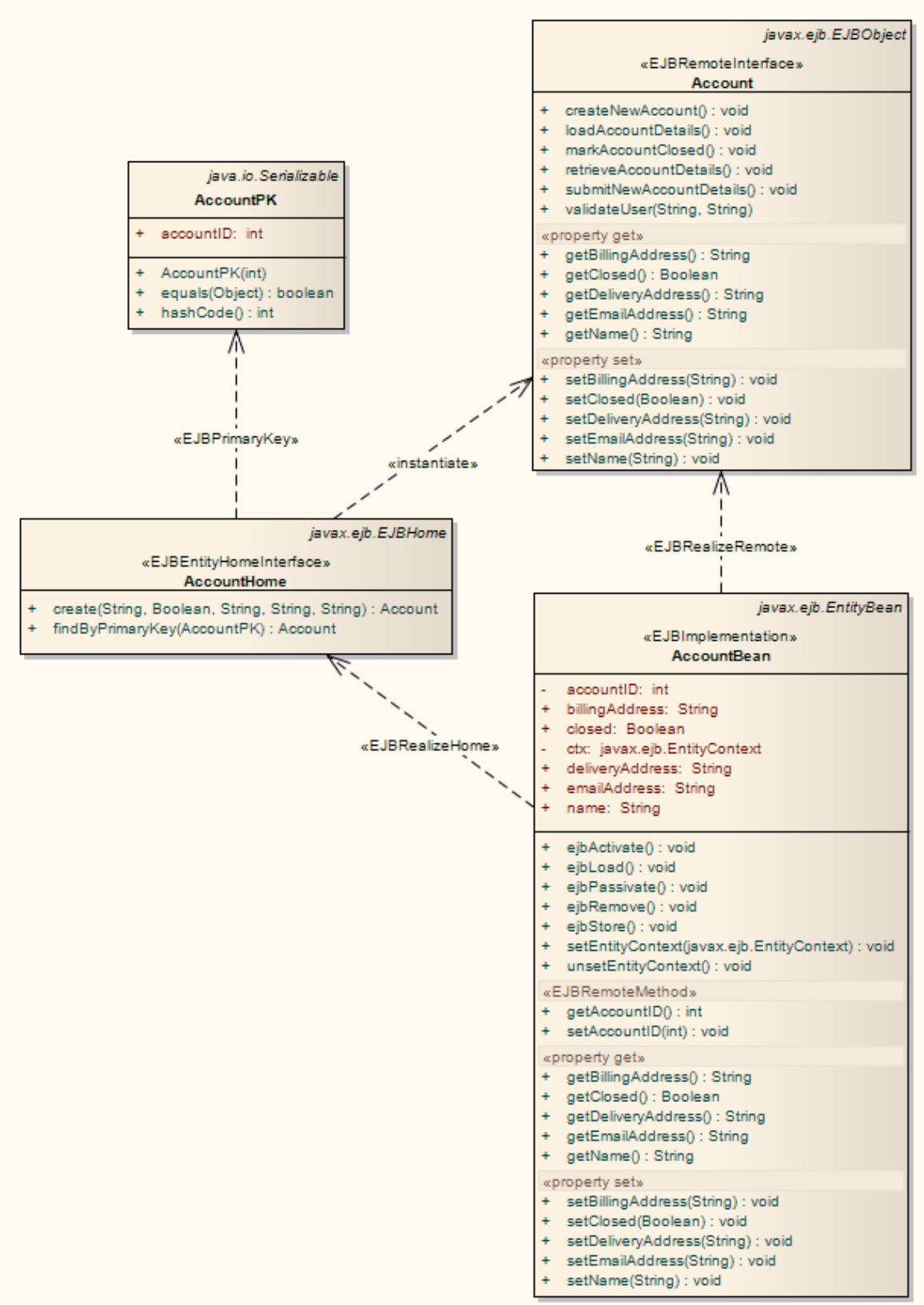
Transformation	Détail
Bean de session EJB	Cette transformation convertit un seul élément de classe (contenant les attributs, les opérations et les références requis pour la génération de code par le Paquetage javax.ejb.*) en • Un élément de classe d'implémentation • Un élément d'interface domestique • Un élément d'interface distant
Bean d'entité EJB	Cette transformation convertit un seul élément de classe (contenant les attributs, les opérations et les références requis pour la génération de code par le Paquetage javax.ejb.*) en : • Un élément de classe d'implémentation • Un élément d'interface domestique • Un élément d'interface distant • Un élément Primary Key

Exemple

Les éléments PIM



Après la transformation, générez un ensemble d'entités Beans, où chacune prend cette forme (pour la classe Account) :


javax.ejb.EJBObject
«EJBRemoteInterface»
Account
+ createNewAccount() : void
+ loadAccountDetails() : void
+ markAccountClosed() : void
+ retrieveAccountDetails() : void
+ submitNewAccountDetails() : void
+ validateUser(String, String)
«property get»
+ getBillingAddress() : String
+ getClosed() : Boolean
+ getDeliveryAddress() : String
+ getEmailAddress() : String
+ getName() : String
«property set»
+ setBillingAddress(String) : void
+ setClosed(Boolean) : void
+ setDeliveryAddress(String) : void
+ setEmailAddress(String) : void
+ setName(String) : void
java.io.Serializable
AccountPK
+ accountID: int
+ AccountPK(int)
+ equals(Object) : boolean
+ hashCode() : int
«EJBPrimaryKey»
«instantiate»
javax.ejb.EJBHome
«EJBEntityHomeInterface»
AccountHome
+ create(String, Boolean, String, String, String) : Account
+ findByPrimaryKey(AccountPK) : Account
«EJBRealizeRemote»
«EJBRealizeHome»
javax.ejb.EntityBean
«EJBImplementation»
AccountBean
- accountID: int
+ billingAddress: String
+ closed: Boolean
- ctx: javax.ejb.EntityContext
+ deliveryAddress: String
+ emailAddress: String
+ name: String
+ ejbActivate() : void
+ ejbLoad() : void
+ ejbPassivate() : void
+ ejbRemove() : void
+ ejbStore() : void
+ setEntityContext(javax.ejb.EntityContext) : void
+ unsetEntityContext() : void
«EJBRemoteMethod»
+ getAccountID() : int
+ setAccountID(int) : void
«property get»
+ getBillingAddress() : String
+ getClosed() : Boolean
+ getDeliveryAddress() : String
+ getEmailAddress() : String
+ getName() : String
«property set»
+ setBillingAddress(String) : void
+ setClosed(Boolean) : void
+ setDeliveryAddress(String) : void
+ setEmailAddress(String) : void
+ setName(String) : void

ERD To Data Transformation du Modèle

La transformation Diagramme de relation d'entité (ERD) en Modèle de données convertit un modèle logique ERD en un modèle de données ciblé sur le type de base de données par défaut, prêt à générer des instructions DDL à exécuter dans l'un des produits de base de données pris en charge par le système. Avant d'effectuer la transformation, vous définissez le type de données commun pour chaque attribut et sélectionnez un type de base de données comme base de données par défaut. Vous pouvez ensuite générer automatiquement le diagramme Modélisation des données.

La transformation utilise et démontre support dans le langage intermédiaire d'un certain nombre de concepts spécifiques à la base de données.

Concepts

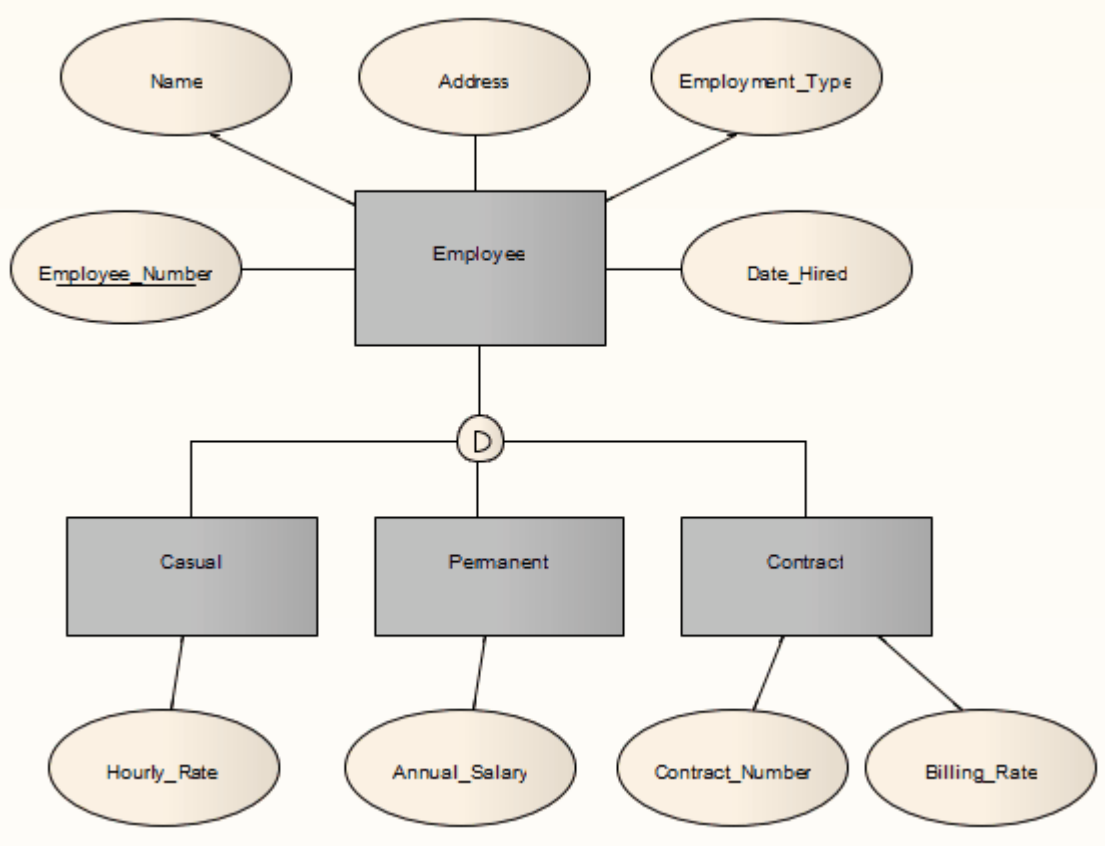
Concept	Définition
Tableau	Mappé un à un sur les éléments de classe.
Colonne	Mappé un à un sur les attributs.
Primary Key	Répertorie toutes les colonnes impliquées afin qu'elles existent dans la classe et crée une méthode Primary Key pour elles.
Foreign Key	Un type spécial de connecteur, dans lequel les sections Source et Cible répertorient toutes les colonnes impliquées de sorte que : • Les colonnes existent • Une Primary Key correspondante existe dans la classe de destination, et • La transformation crée la Foreign Key appropriée

Généralisation

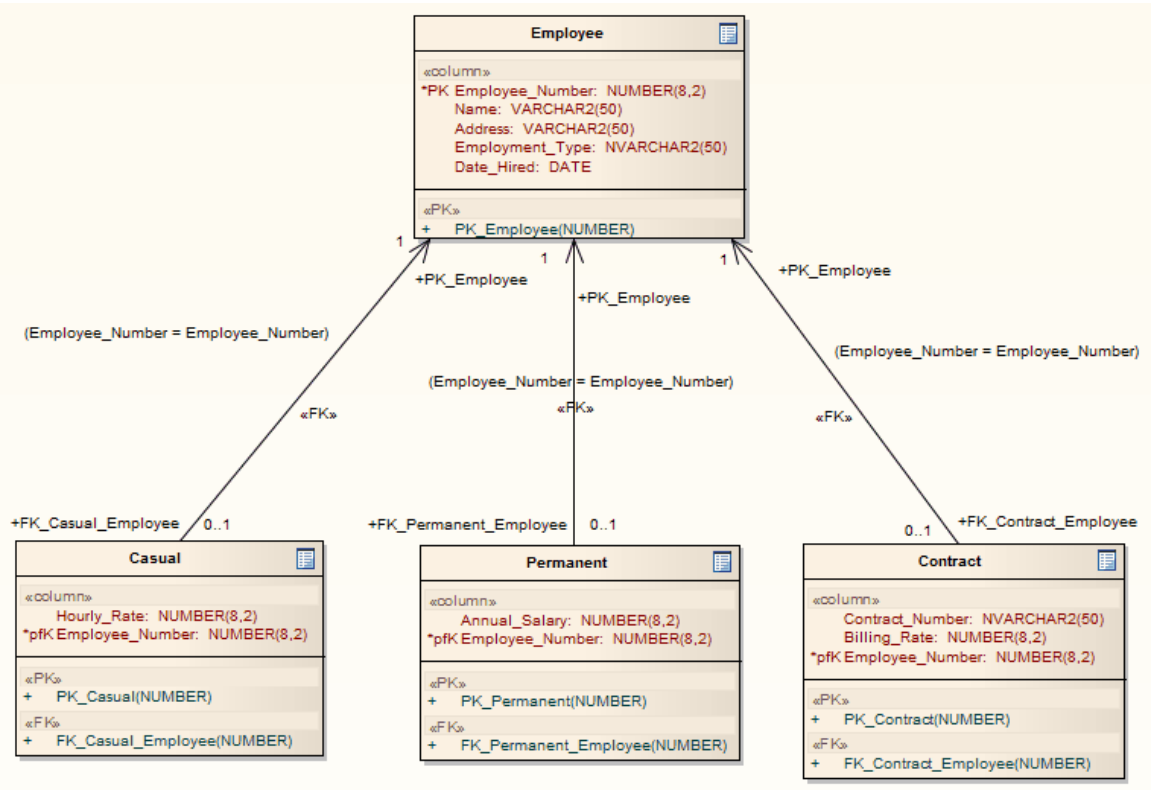
La technologie ERD peut gérer la généralisation, comme illustré. Note que l'héritage par copie vers le bas n'est actuellement pris en charge qu'avec deux niveaux.

Exemple

Les éléments ERD



Après la transformation, ils deviennent les éléments Modèle de données



Notes

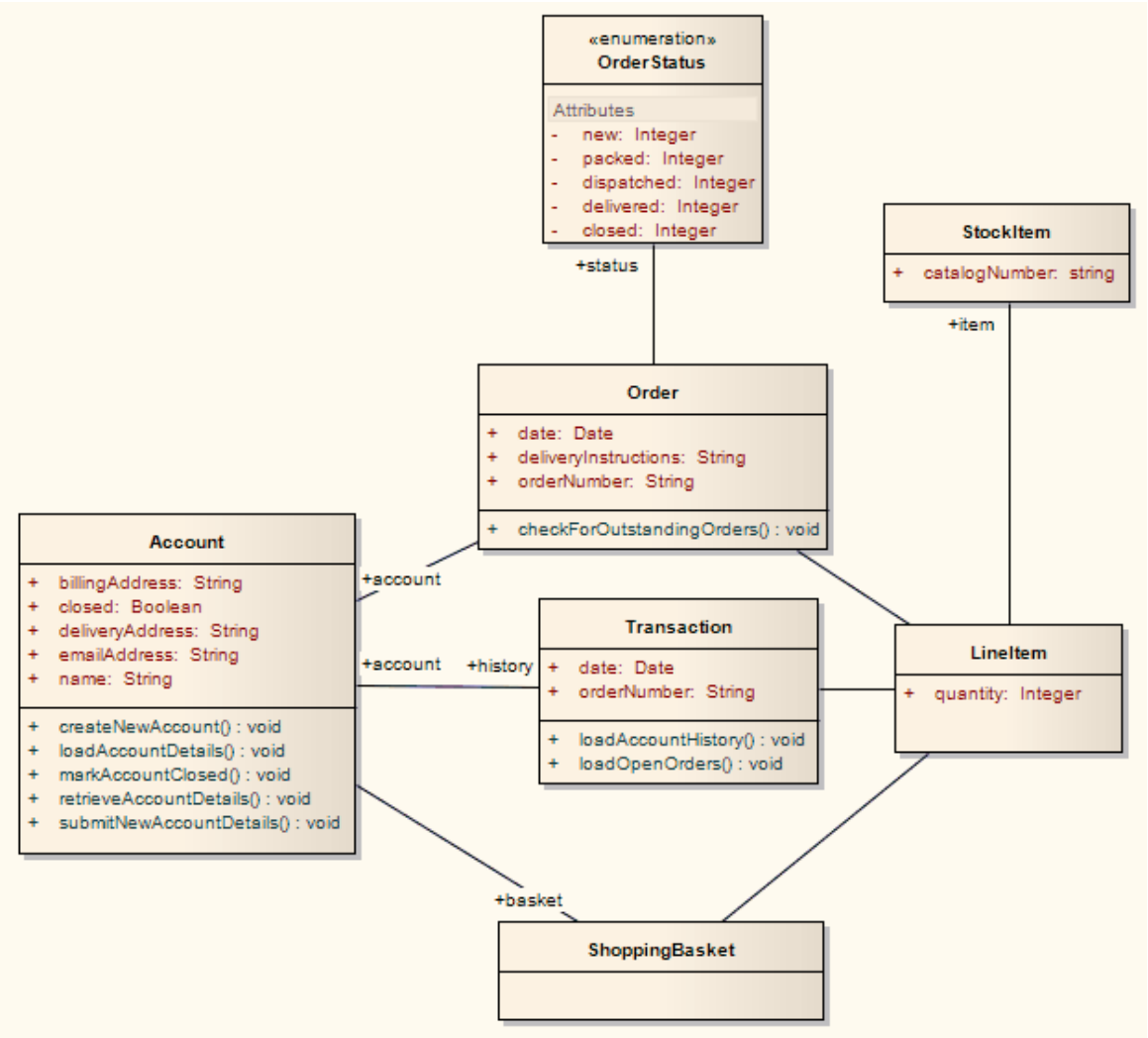
- Parfois, vous pouvez revenir à l'ERD, apporter des modifications et ensuite avoir besoin d'effectuer une autre transformation ; dans ce cas, pour obtenir de meilleurs résultats, supprimez toujours le Paquetage de transformation précédent avant d'effectuer la transformation suivante

Transformation Java

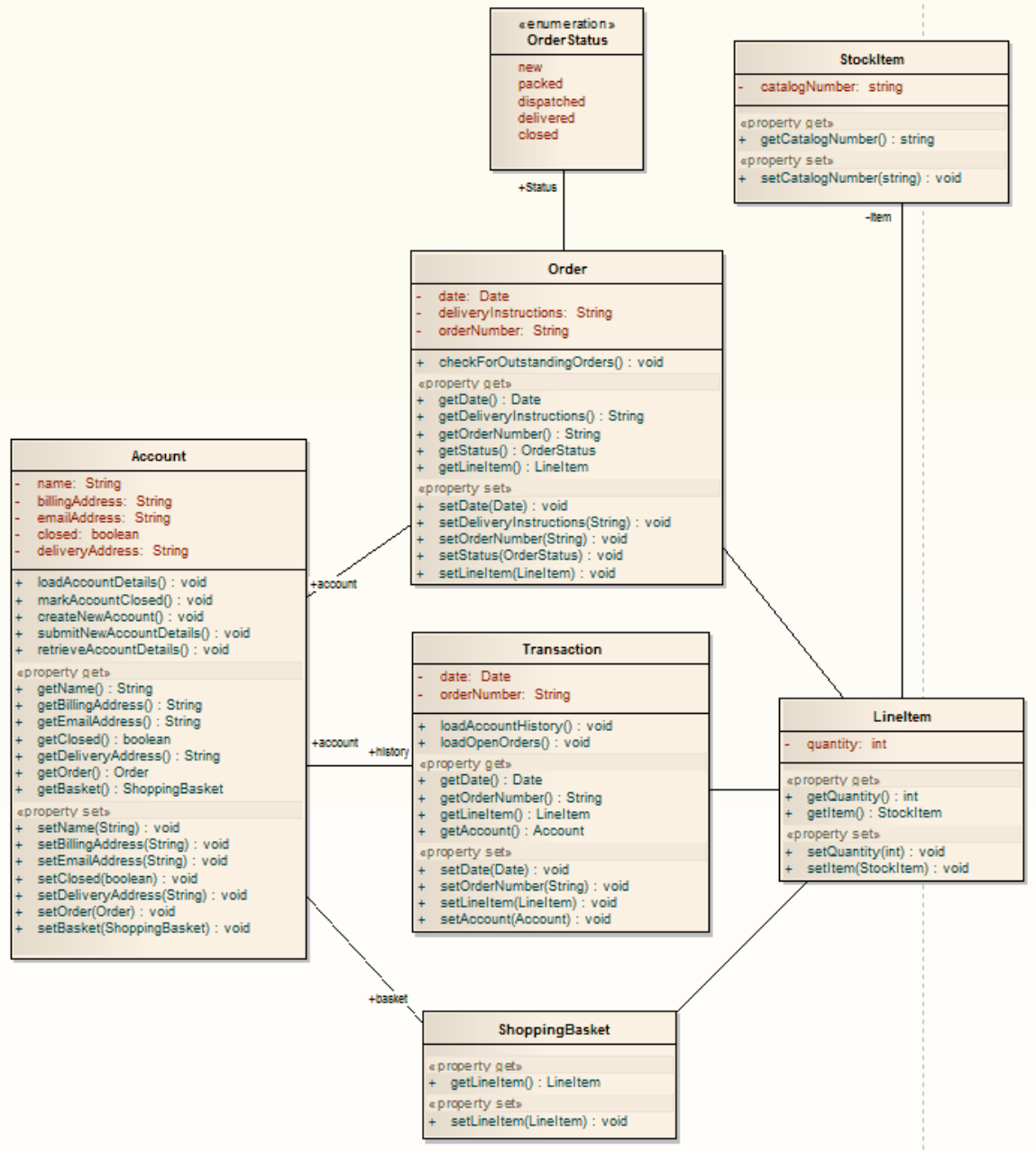
La transformation Java convertit les types d'éléments PIM (Platform-Independent Modèle) en types d'éléments Class spécifiques à Java et crée une encapsulation (produisant les getters et setters) selon les options que vous avez définies pour la création de propriétés à partir d'attributs Java (sur la page « Spécifications Java » de la dialogue « Préférences »). Note que les attributs publics dans le PIM sont convertis en attributs privés dans le PSM. Toutes les opérations de l'interface sont transformées en méthodes virtuelles pures.

Exemple

Les éléments PIM



Après la transformation, devenez les éléments PSM

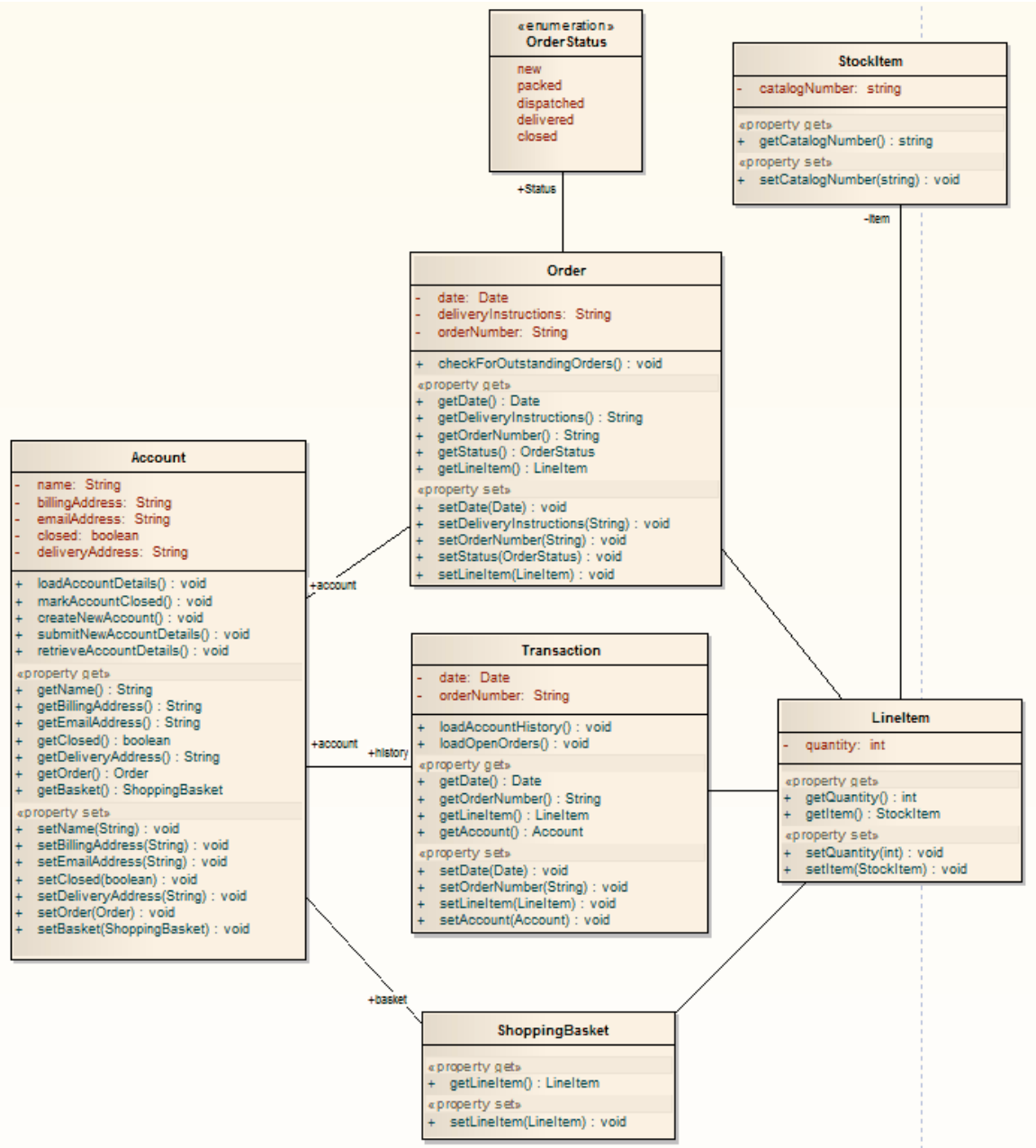

«enumeration»
OrderStatus
new
packed
dispatched
delivered
closed
+Status
StockItem
- catalogNumber: string
«property get»
+ getCatalogNumber() : string
«property set»
+ setCatalogNumber(string) : void
-Item
Order
- date: Date
- deliveryInstructions: String
- orderNumber: String
+ checkForOutstandingOrders() : void
«property get»
+ getDate() : Date
+ getDeliveryInstructions() : String
+ getOrderNumber() : String
+ getStatus() : OrderStatus
+ getLineItem() : LineItem
«property set»
+ setDate(Date) : void
+ setDeliveryInstructions(String) : void
+ setOrderNumber(String) : void
+ setStatus(OrderStatus) : void
+ setLineItem(LineItem) : void
Account
- name: String
- billingAddress: String
- emailAddress: String
- closed: boolean
- deliveryAddress: String
+ loadAccountDetails() : void
+ markAccountClosed() : void
+ createNewAccount() : void
+ submitNewAccountDetails() : void
+ retrieveAccountDetails() : void
«property get»
+ getName() : String
+ getBillingAddress() : String
+ getEmailAddress() : String
+ getClosed() : boolean
+ getDeliveryAddress() : String
+ getOrder() : Order
+ getBasket() : ShoppingBasket
«property set»
+ setName(String) : void
+ setBillingAddress(String) : void
+ setEmailAddress(String) : void
+ setClosed(boolean) : void
+ setDeliveryAddress(String) : void
+ setOrder(Order) : void
+ setBasket(ShoppingBasket) : void
+account
Transaction
- date: Date
- orderNumber: String
+ loadAccountHistory() : void
+ loadOpenOrders() : void
«property get»
+ getDate() : Date
+ getOrderNumber() : String
+ getLineItem() : LineItem
+ getAccount() : Account
«property set»
+ setDate(Date) : void
+ setOrderNumber(String) : void
+ setLineItem(LineItem) : void
+ setAccount(Account) : void
+account
+history
LineItem
- quantity: int
«property get»
+ getQuantity() : int
+ getItem() : StockItem
«property set»
+ setQuantity(int) : void
+ setItem(StockItem) : void
+basket
ShoppingBasket
«property get»
+ getLineItem() : LineItem
«property set»
+ setLineItem(LineItem) : void

Transformation JUnit

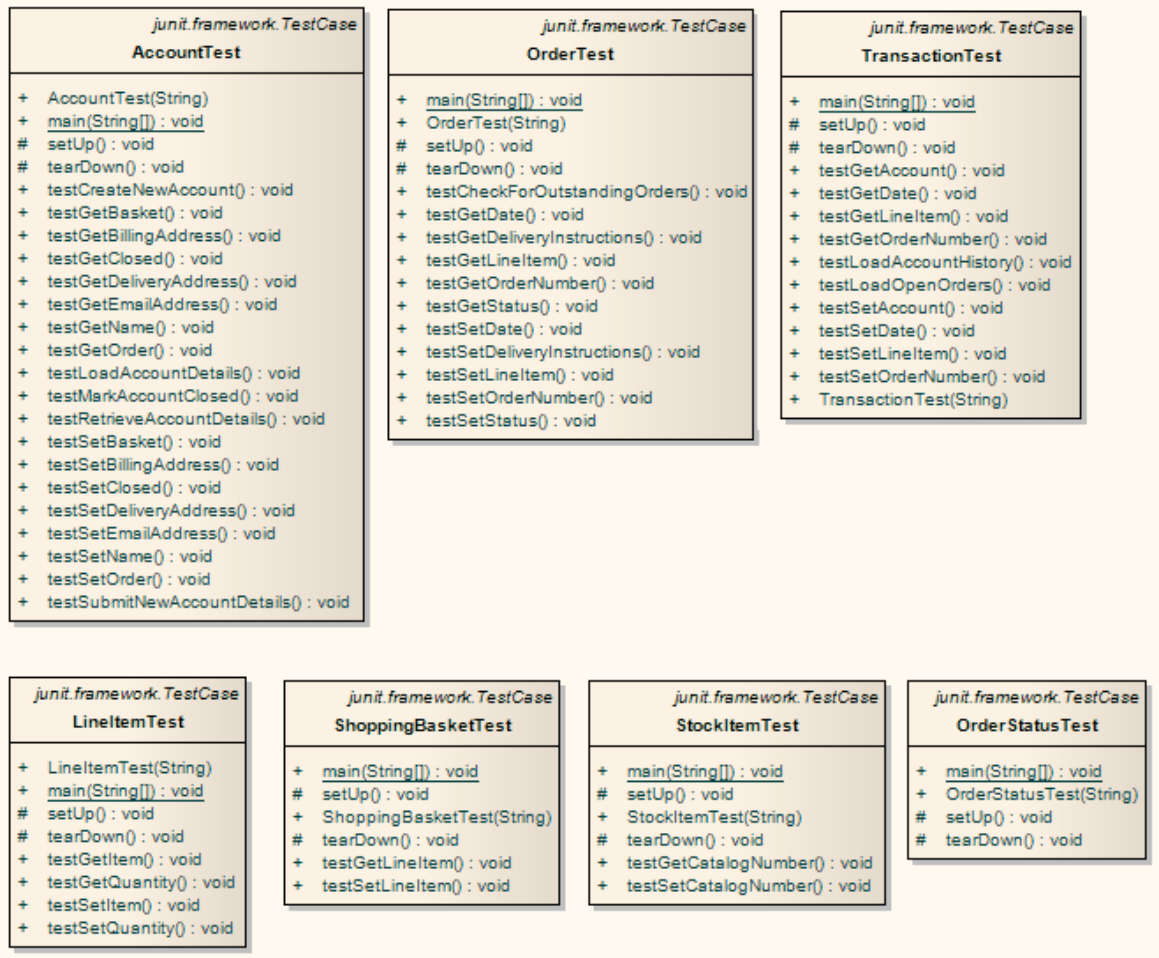
La transformation JUnit convertit une classe Java existante avec des méthodes publiques en une classe avec une méthode de test pour chaque méthode publique. La classe résultante peut alors être générée et les tests remplis et exécuter par JUnit.

Exemple

Les éléments du modèle Java (initialement transformés à partir du PIM)



Après la transformation, devenez les éléments PSM



Notes

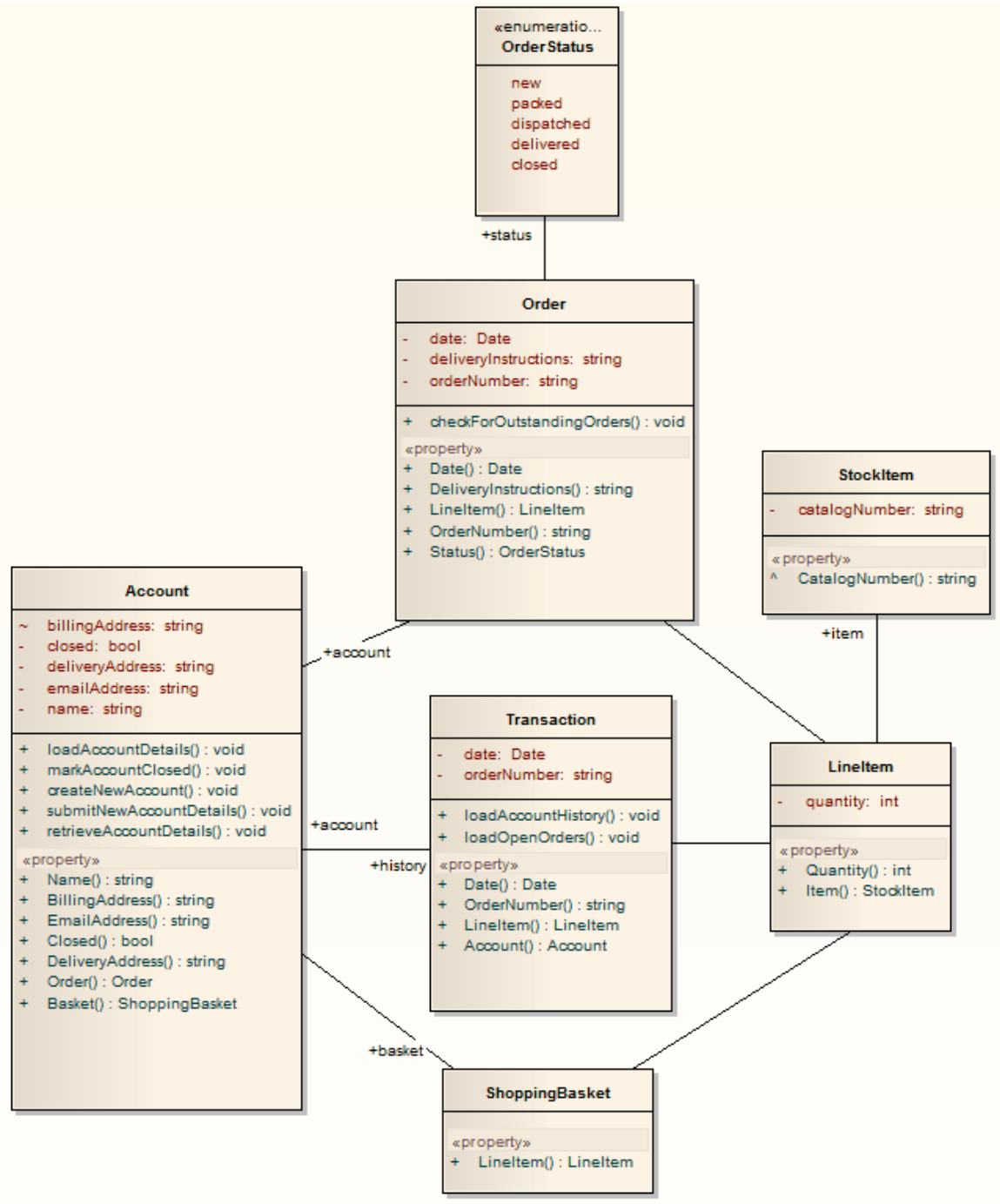
- Pour chaque classe du modèle Java, une classe Test correspondante a été créée contenant une méthode de test pour chaque méthode publique de la classe source, ainsi que les méthodes requises pour configurer correctement les tests ; vous remplissez les détails de chaque test

Transformation NUnit

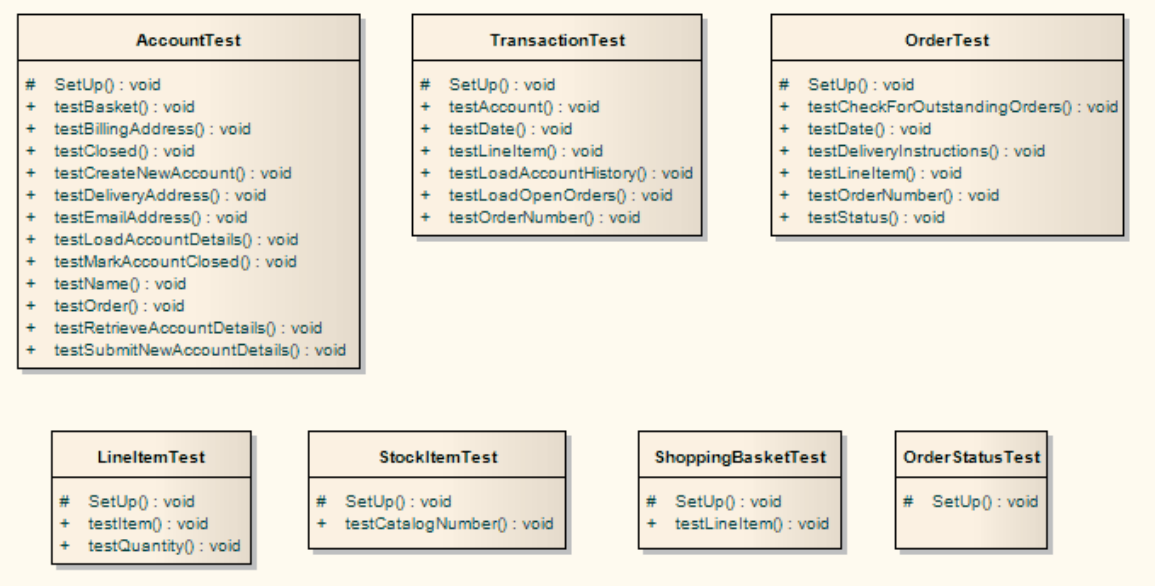
La transformation NUnit convertit une classe compatible .NET existante avec des méthodes publiques en une classe avec une méthode de test pour chaque méthode publique. La classe résultante peut ensuite être générée et les tests remplis et exécuter par NUnit.

Exemple

Les éléments C# (initialement transformés à partir du PIM)



Après la transformation, devenez les éléments PSM



Notes

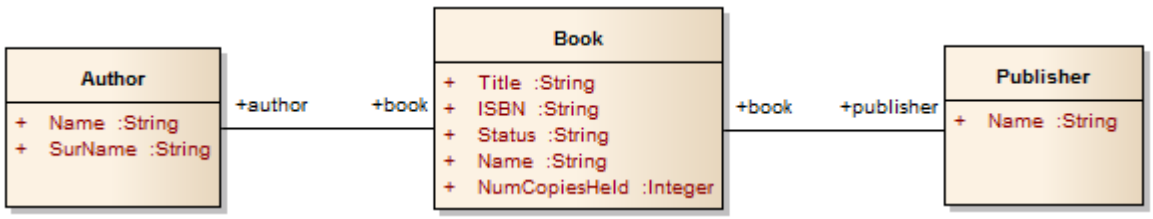
- Pour chaque classe du modèle C# , une classe Test correspondante a été créée contenant une méthode de test pour chaque méthode publique de la classe source, ainsi que les méthodes requises pour configurer correctement les tests ; vous remplissez les détails de chaque test

Transformation PHP

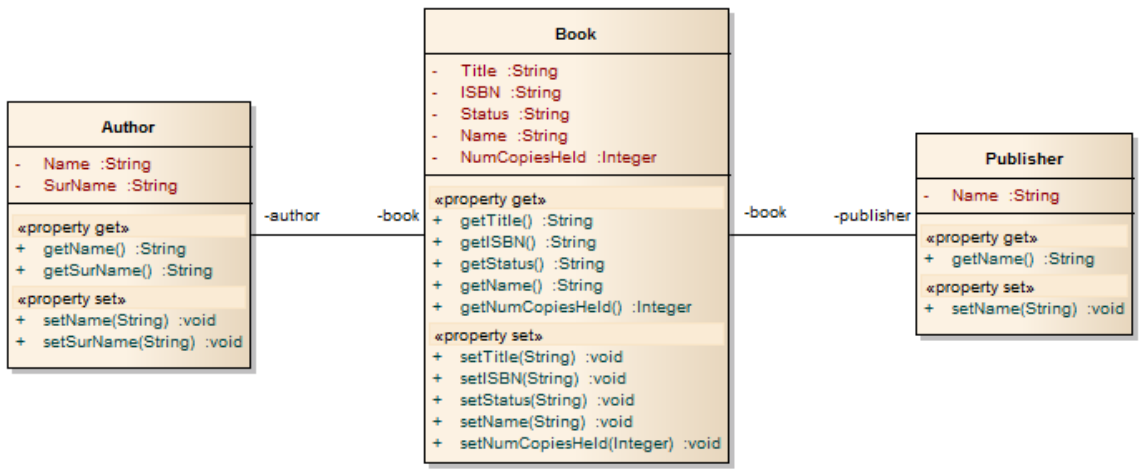
La transformation PHP convertit les types d'éléments PIM (Platform-Independent Modèle) en types d'éléments de classe PHP spécifiques au langage et crée une encapsulation (produisant les getters et setters) selon les options que vous avez définies pour la création de propriétés à partir d'attributs PHP (sur la page « Spécifications PHP » de la dialogue « Préférences »). Note que les attributs publics dans le PIM sont convertis en attributs privés dans le PSM.

Exemple

Les éléments PIM



Après la transformation, devenez les éléments PSM



Transformations de Séquence / Diagramme Communication

diagrammes Séquence peuvent être transformés en diagramme Communication et diagrammes Communication en diagrammes Séquence . Dans chaque cas, chaque élément ou message du type diagramme source est mappé 1:1 à un élément ou message correspondant dans le diagramme cible.

Accéder

Ruban	Conception > Paquetage > Transformer > Transformer la sélection
Raccourcis Clavier	Ctrl+Maj+H (transformer Paquetage actuel) Ctrl+H (transformer les éléments sélectionnés)

Effectuer une transformation

Le diagramme en cours de transformation doit être ouvert dans la vue diagramme principale pour que les options ' Communication ' ou ' Séquence ' apparaissent dans la dialogue Transformation du Modèle .

Étape	Action
1.	Ouvrir et sélectionner dans la vue diagramme , le diagramme à transformer.
2.	Ouvrez la dialogue *Transformation Modèle* en utilisant : Conception > Paquetage > Transformer > Transformer la sélection (Ctrl+Maj+H).
3.	Dans la liste *Éléments* , mettez en surbrillance tous les éléments du diagramme qui seront inclus dans la transformation.
4.	Dans la liste *Transformations* , sélectionnez : • La case à cocher « Communication », si vous transformez un diagramme Séquence en diagramme Communication , ou • La case ' Séquence ', si vous transformez un diagramme Communication en diagramme Séquence La dialogue « Parcourir le projet » s'affiche. Recherchez et sélectionnez le Paquetage cible dans lequel le diagramme cible sera créé, puis cliquez sur le bouton *OK* .
5.	Cliquez sur le bouton *Do Transformer* pour exécuter la transformation. Le diagramme cible est créé et répertorié dans la fenêtre Navigateur sous le Paquetage cible avec le nom (selon la transformation que vous avez exécutée) : <nom diagramme source> Communication ou <nom diagramme source> Séquence

Notes

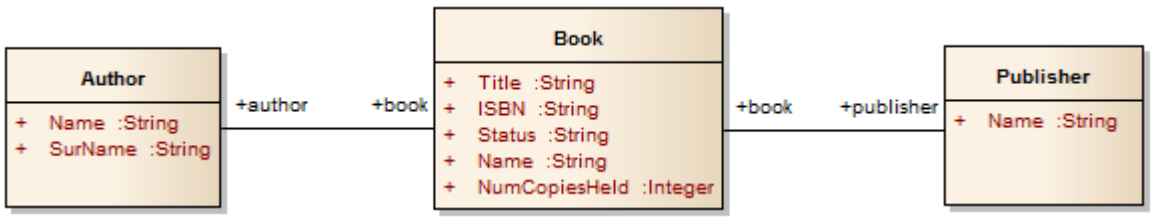
- Une fois que vous avez coché la case « Communication » ou « Séquence », ces transformations ignorent tout autre paramètre de champ dans le dialogue à l'exception de « Paquetage cible », et effectueront une transformation directe de chaque élément du diagramme source.

Transformation VB.Net

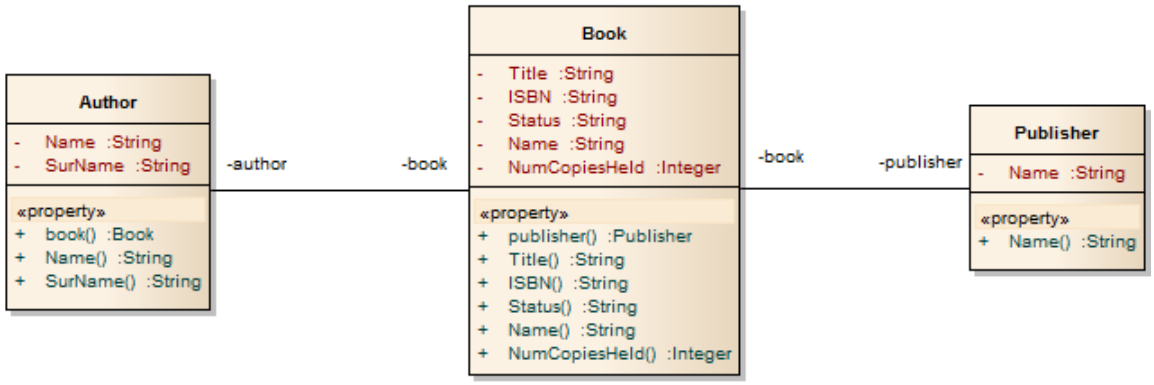
La transformation VB.Net convertit les types d'éléments PIM (Platform-Independent Modèle) en types d'éléments de classe VB.Net spécifiques au langage et crée une encapsulation en fonction des options que vous avez définies pour la création de propriétés à partir d'attributs VB.Net (sur la page « Spécifications VB.Net » de la dialogue « Préférences »). Note que les attributs publics dans le PIM sont convertis en attributs privés dans le PSM.

Exemple

Les éléments PIM

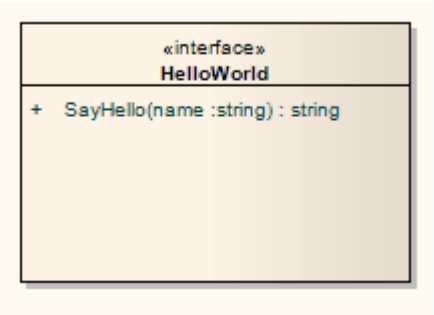


Après la transformation, devenez les éléments PSM



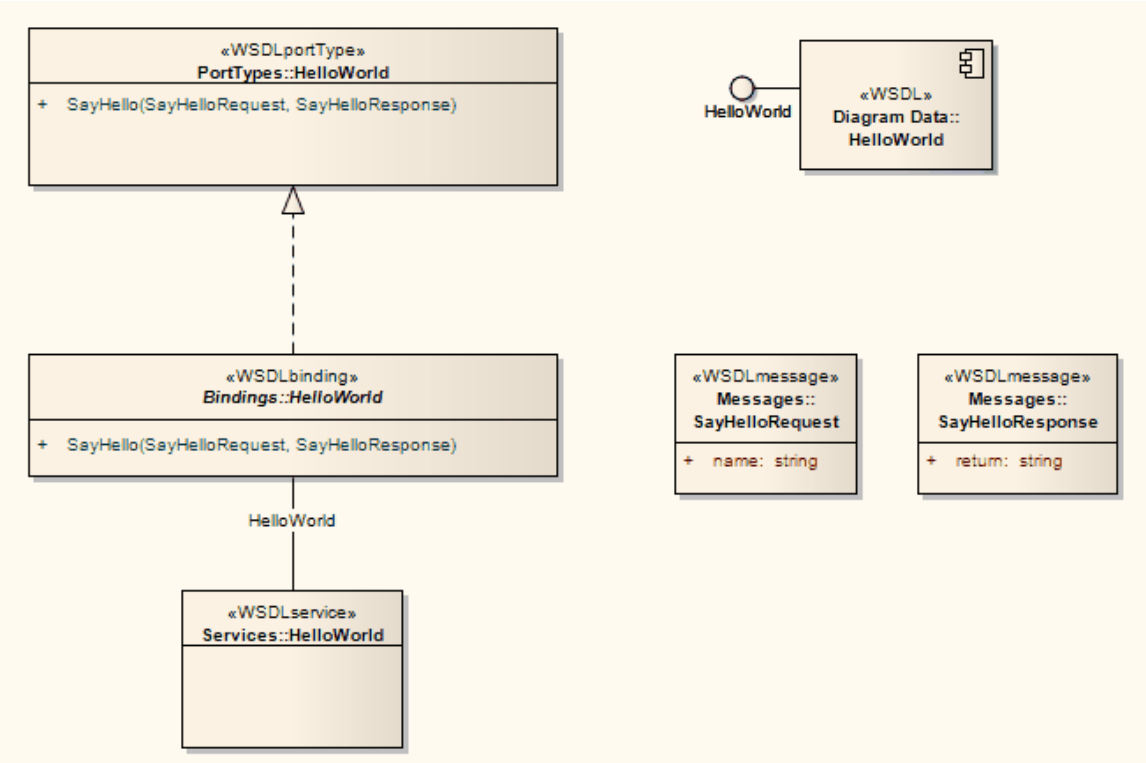
Transformation WSDL

La transformation WSDL convertit un modèle simple en un modèle étendu d'une interface WSDL adaptée à la génération. Par exemple :



La transformation de ceci génère le composant WSDL, le service, Type de port, la liaison et les messages correspondants :

- Les classes sont traitées de la même manière que dans la transformation XSD
- Tous les paramètres « in » sont transformés en parties de message WSDL dans le message de demande
- La valeur de retour et tous les paramètres « out » et « return » sont transformés en parties de message WSDL dans le message de réponse
- Toutes les méthodes où une valeur est renvoyée sont transformées en opérations de requête-réponse, et toutes les méthodes ne renvoyant pas de valeur sont transformées en opérations OneWay
- La transformation ne gère pas la génération des méthodes Solicit-Response et Notification ni les erreurs



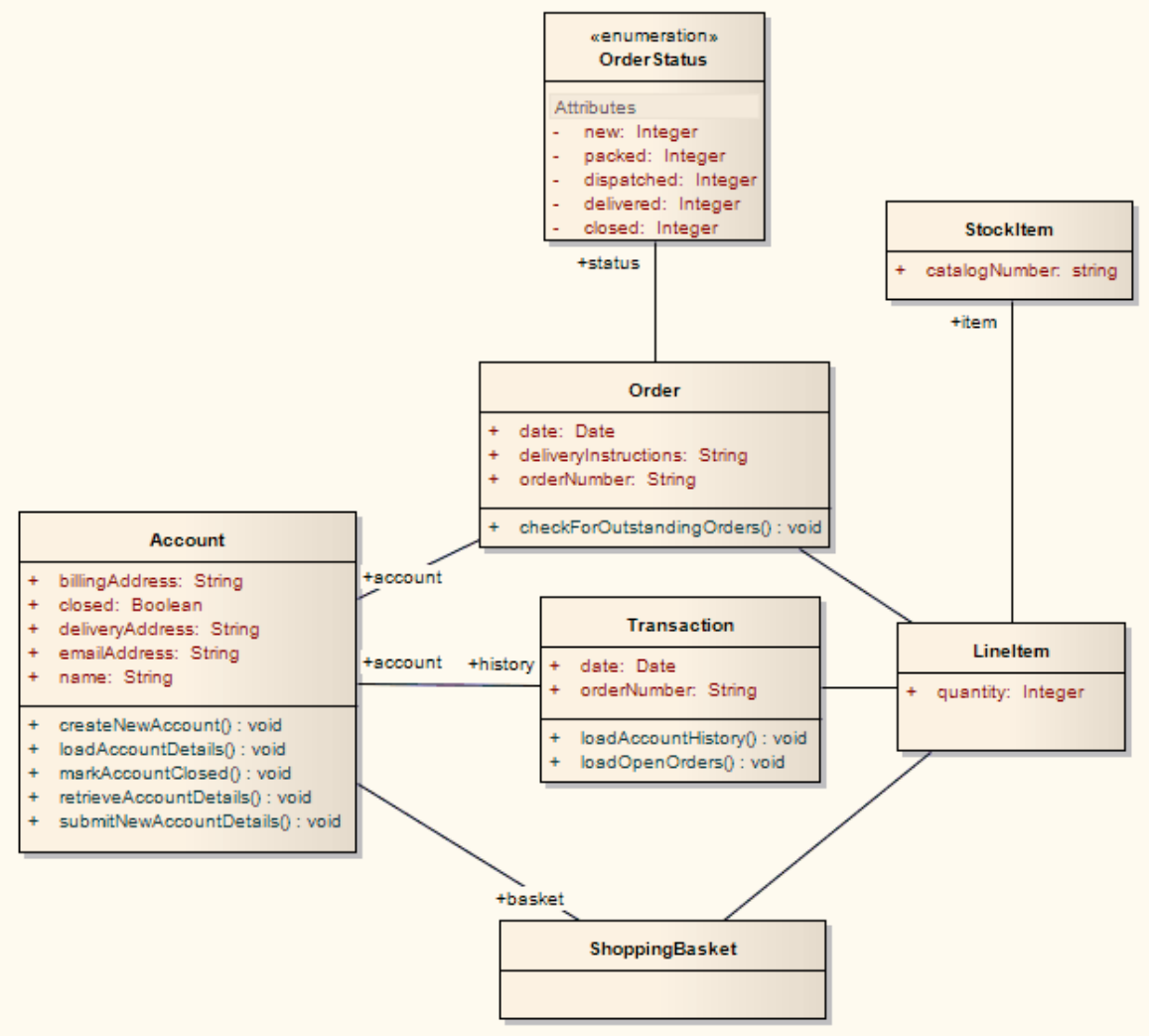
Dans le Paquetage résultant, vous pouvez ensuite remplir les détails à l'aide des capacités d'édition WSDL d' Enterprise Architect , et enfin générer le Paquetage à l'aide des outils de génération WSDL.

Transformation XSD

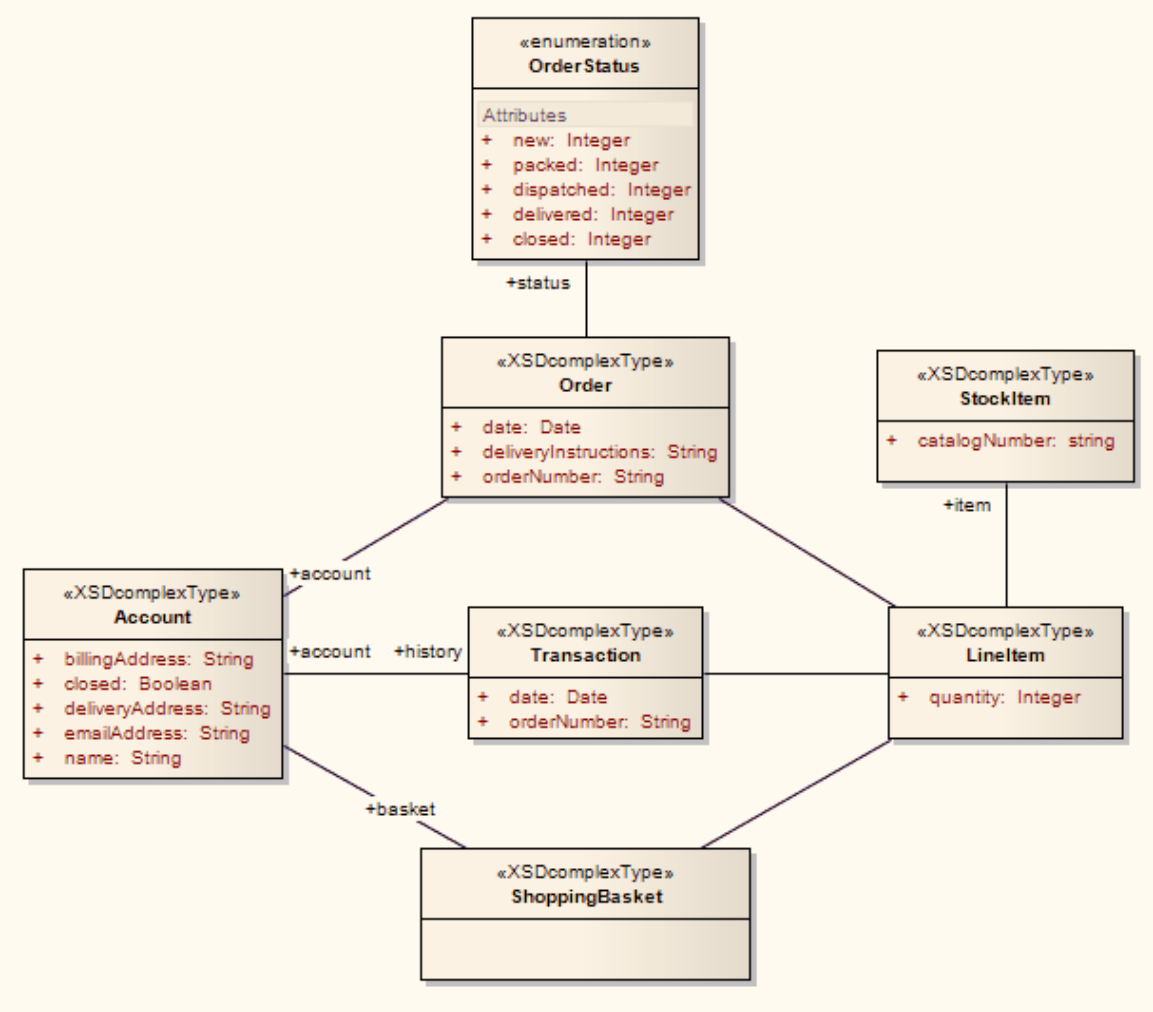
La transformation XSD convertit les éléments PIM (Platform-Independent Modèle) en éléments UML Profile for XML comme étape intermédiaire dans la création d'un schéma XML. Chaque élément PIM Class sélectionné est converti en élément « XSDcomplexType ».

Exemple

Les éléments PIM



Après la transformation, ils deviennent les éléments PSM



Ceux-ci génèrent à leur tour ce XSD

<?xml version="1.0" encoding="ISO-8859-1"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="Compte" type="Compte"/>

<xs:complexType name="Compte">

<xs:séquence>

<xs:élément nom="nom" type="xs: string "/>

<xs:element name="billingAddress" type="xs: string "/>

<xs:element name="adresse e-mail" type="xs: string "/>

<xs:élément nom="fermé" type="xs:booléen"/>

<xs:element name="adresse de livraison" type="xs: string "/>

<xs:element ref="Commande"/>

<xs:element ref="Panier"/>

</xs:séquence>

</xs:complexType>

<xs:element name="LineItem" type="LineItem"/>

<xs:complexType name="Ligne">

<xs:séquence>

<xs:élément nom="quantité" type="xs: integer "/>

<xs:element ref="Article en stock"/>
</xs:séquence>
</xs:complexType>
<xs:element name="Commande" type="Commande"/>
<xs:complexType name="Commande">
<xs:séquence>
<xs:élément nom="date" type="xs:date"/>
<xs:élément nom="deliveryInstructions" type="xs: string "/>
<xs:élément nom="numéro de commande" type="xs: string "/>
<xs:element ref="Ligne"/>
<xs:élément nom="status" type="OrderStatus"/>
</xs:séquence>
</xs:complexType>
<xs:simpleType name="État de la commande">
<xs:restriction base="xs: string ">
<xs:enumeration value="nouveau"/>
<xs:enumeration value="emballé"/>
<xs:enumeration value="envoyé"/>
<xs:enumeration value="livré"/>
<xs:enumeration value="fermé"/>
</xs:restriction>
</xs:simpleType>
<xs:element name="Panier" type="Panier"/>
<xs:complexType name="Panier">
<xs:séquence>
<xs:element ref="Ligne"/>
</xs:séquence>
</xs:complexType>
<xs:element name="StockItem" type="StockItem"/>
<xs:complexType name="Article en stock">
<xs:séquence>
<xs:element name="catalogNumber" type="xs: string "/>
</xs:séquence>
</xs:complexType>
<xs:element name="Transaction" type="Transaction"/>
<xs:complexType nom="Transaction">
<xs:séquence>
<xs:élément nom="date" type="xs:date"/>
<xs:élément nom="numéro de commande" type="xs: string "/>
<xs:element ref="Compte"/>
<xs:element ref="Ligne"/>
</xs:séquence>

```
</xs:complexType>
</xs:schema>
```

Modifier Transformation Gabarits

Une transformation unique applique un certain nombre de gabarits de transformation, chacun définissant un type d' object sur lequel la transformation agit, ainsi que les actions exécutées sur les objets de ce type. Le système fournit une gamme de gabarits par défaut intégrés, et chaque type de transformation utilise un sous-ensemble spécifique de ces gabarits . En règle générale, le type de transformation et le sous-ensemble de gabarits sont adaptés à la langue cible. Certains gabarits par défaut d'un ensemble n'ont aucun contenu ; ils sont « latents » et représentent le potentiel d'agir sur un object qui n'est généralement pas inclus dans la transformation, mais qui est parfaitement valide si vous souhaitez l'inclure. Un exemple de gabarit latent est le gabarit de base de classe liée dans la transformation **C#** .

Vous pouvez personnaliser les gabarits de transformation de différentes manières, notamment :

- Ajuster le code dans un ou plusieurs gabarits d'un ensemble par défaut
- Ajouter du code à un gabarit par défaut « latent »
- Ajoutez un nouveau gabarit personnalisé, basé sur l'une des valeurs par défaut, mais servant un objectif différent que vous définissez
- Ajouter un nouveau type de transformation contenant - initialement - un ensemble de base de gabarits par défaut
- Ajouter (ou supprimer) un remplacement stéréotypé pour un gabarit

Une substitution stéréotypée ordonne à la transformation d'utiliser le gabarit modifié uniquement si l'élément et/ou fonctionnalité sont des types stéréotypés spécifiés. Si l' object ou fonctionnalité ne sont pas de ce type, la transformation applique le gabarit de base.

Accéder

Ruban	Conception > Paquetage > Transformation > Transformation Gabarits
Raccourcis Clavier	Ctrl+Alt+H

Modifier Transformation Gabarits

Option	Action
Langue	Cliquez sur la flèche déroulante et sélectionnez le nom de la transformation.
Nouveau Type de transformation	Cliquez sur ce bouton si vous souhaitez créer une nouvelle transformation. Une prompt s'affiche pour le nom de la transformation ; saisissez le nom et cliquez sur le bouton OK . La liste « Gabarits » affiche l'ensemble par défaut de gabarits intégrés, à partir desquels vous pouvez développer votre transformation. Votre transformation personnalisée n'est pas enregistrée ni disponible à l'utilisation à moins que vous n'ajoutiez et/ou ne modifiiez un ou plusieurs gabarits dans la transformation.
Gabarits	Répertorie les gabarits de transformation pour la transformation courante. Cliquez sur le nom d'un gabarit pour le mettre en surbrillance et afficher son contenu dans le panneau Gabarit . La colonne « Modifié » indique si vous avez modifié le gabarit pour cette transformation.

Gabarit	Affiche le contenu du gabarit actuellement sélectionné, et fournit l'éditeur facilités pour modifier le gabarit (cliquez-droit sur le texte du code).
Les stéréotypes supplantent	Liste les gabarits stéréotypés, pour le gabarit de base actif. La colonne « Modifié » indique si vous avez modifié un gabarit stéréotypé.
Ajouter un nouveau Gabarit personnalisé	Cliquez sur ce bouton pour créer un gabarit personnalisé à ajouter à la transformation actuelle. Une dialogue s'affiche, vous invitant à spécifier : • Le type object (type gabarit de base) auquel ce nouveau gabarit répondra - cliquez sur la flèche déroulante et sélectionnez le nom (les types gabarit personnalisés ne sont pas inclus dans cette liste) • Le nom du nouveau gabarit - saisissez le texte approprié Cliquez sur le bouton OK . Le nouveau nom gabarit est ajouté à la liste des gabarits et il est ouvert dans l'éditeur gabarit prêt à recevoir son code.
Ajouter un nouveau remplacement stéréotypé	Cliquez sur ce bouton pour ajouter un nouveau remplacement de stéréotype pour le gabarit actuellement sélectionné. Une dialogue s'affiche, vous invitant à spécifier : • Classe de base (type de classe stéréotypé - cliquez sur la flèche déroulante et cliquez sur le type dans la liste) et/ou • Fonctionnalité (cliquez sur la flèche déroulante et cliquez sur la fonctionnalité stéréotypée dans la liste) Cliquez sur le bouton OK . Le remplacement est ajouté à la liste « Remplacements de stéréotypes ».
Obtenir Gabarit par défaut	Cliquez sur ce bouton pour mettre à jour l'affichage de l'éditeur avec la version par défaut du gabarit intégré actuel ou pour effacer le contenu du gabarit personnalisé actuel. Si vous avez enregistré le gabarit modifié, le rétablissement de la version par défaut est un changement, donc le champ « Modifié » affiche toujours le mot « Oui ».
Sauvegarder	Cliquez sur ce bouton pour enregistrer le gabarit nouveau ou modifié. Vous ne pouvez pas passer à un autre gabarit sans enregistrer le gabarit actuel, ce qui enregistre également la transition.
Supprimer	Cliquez sur ce bouton pour supprimer le gabarit personnalisé actuel ou le remplacement de stéréotype, ou les modifications les plus récentes apportées à un gabarit intégré (ce qui revient effectivement au contenu de base par défaut). Vous ne pouvez pas supprimer un gabarit intégré. Vous êtes invité à confirmer la suppression.
Aide	Cliquez sur ce bouton pour afficher cette rubrique d'aide.

Notes

- L'édition gabarit de transformation est fortement basée sur l'édition gabarit de génération de code ; pour plus d'informations sur l'édition gabarits de transformation, consultez la section *Éditeur Gabarit de code* et la rubrique *Édition du code source*

Transformations d'écriture

Enterprise Architect fournit un facilité de créer vos propres transformations. Cela peut être utile pour automatiser le processus de génération de modèles plus spécifiques à partir de modèles plus généraux, en réutilisant la transformation et en évitant les erreurs qui pourraient survenir si les modèles étaient créés à la main. Les gabarits existants constitueront un guide et une référence utiles pour vous aider à créer de nouveaux gabarits .

gabarits de transformation sont basés sur le framework Code Generation Gabarit , et une compréhension de la manière dont ces gabarits fonctionnent est essentielle pour pouvoir ajuster gabarits de transformation existants ou en créer de nouveaux. Il est donc conseillé de lire et de comprendre les sujets traitant de Code Generation Gabarits avant d'utiliser le langage Transformation Gabarit .

Accéder

Ruban	Conception > Paquetage > Transformation > Transformation Gabarits
Raccourcis Clavier	Ctrl+Alt+H

Facteurs concernant la transformation Gabarits

Facteur	Détail
Gabarits de transformation par défaut	Enterprise Architect fournit un ensemble de gabarits de transformation par défaut que vous pouvez utiliser « tels quels » ou personnaliser selon vos besoins.
Syntaxe générale pour le langage intermédiaire	Les transformations dans Enterprise Architect génèrent une forme de code intermédiaire du modèle créé dans la transformation. Vous pouvez révision et éditer ce code.
Débogage de langage intermédiaire	Vous pouvez également déboguer les scripts de transformation en vérifiant le code intermédiaire généré à partir du script Transform.
Édition gabarits et du code de transformation	Lors de l'écriture des transformations, vous utilisez les facilités de l' Éditeur de Code commun.
Cadre de code Gabarit	Vous utilisez le framework Code Gabarit pour effectuer l'ingénierie avancée des modèles UML . Le framework Transformation Gabarit en est dérivé.
Syntaxe pour la création d'objets	Pour générer des objets ou des éléments dans une transformation, vous appliquez une syntaxe spécifique dans le script gabarit .
Syntaxe pour la création de connecteurs	Pour générer des connecteurs (relations) dans une transformation, vous appliquez également une syntaxe spécifique dans le script gabarit .
Transformer les informations en double	Dans de nombreuses transformations, il y a une quantité importante d'informations à copier. Plutôt que de placer ces informations dans le gabarit , vous pouvez utiliser des macros pour les lire depuis leur source.

Transformation des substitutions de paramètres Gabarit	Dans un gabarit de transformation, si vous transformez des substitutions de paramètres de liaison de connecteur Gabarit , vous pouvez utiliser les macros de substitution de paramètres Gabarit .
Conversion des types	Vous pouvez appliquer différentes méthodes pour convertir des types de données en différents types de plateformes cibles.
Conversion des noms	Vous pouvez appliquer différentes méthodes pour convertir les noms d'éléments en différentes conventions de dénomination de plateformes cibles.
Références croisées	Lors d'une transformation, vous pouvez effectuer une vérification croisée des éléments transformés.

Notes

- D'autres conseils et astuces peuvent être obtenus en étudiant attentivement les Gabarits de transformation fournis avec Enterprise Architect
- L'éditeur Transformation Gabarit applique les facilités de l' Éditeur de Code commun

Gabarits de transformation par défaut

gabarits de transformation permettent de représenter les informations existantes dans un modèle de manière modifiée. Lors de la création d'une nouvelle transformation, Enterprise Architect fournit un ensemble de gabarits de transformation par défaut qui effectuent une copie directe du modèle source vers le modèle cible. Cela vous permet de réfléchir en termes de différences entre le modèle source et le modèle cible. Pour chaque gabarit vous pouvez empêcher la copie des propriétés et ajouter des informations supplémentaires jusqu'à ce que le modèle cible approprié soit créé.

Vous pouvez répertorier et examiner les gabarits par défaut dans l'éditeur de transformation. La combinaison des gabarits par défaut varie en fonction de la langue que vous transformez.

Accéder

Ruban	Conception > Paquetage > Transformation > Transformation Gabarits
Raccourcis Clavier	Ctrl+Alt+H

Notes

- Lors de la création d'une nouvelle transformation, vous devez modifier au moins un gabarit avant que la nouvelle transformation ne soit disponible

Langue intermédiaire

Toutes les transformations dans Enterprise Architect créent une forme de langage intermédiaire du modèle à générer. Vous pouvez accéder au fichier contenant ce code de langage intermédiaire et le modifier à l'aide d'un éditeur externe. Chaque object est représenté dans ce langage par le type object (par exemple, Classe, Action , Méthode, Généralisation ou Étiquette) suivi des propriétés object et des fonctionnalités dont il est composé ; la grammaire de la description object ressemble à ceci :

élément:

nomélément { (propriétéélément | élément)* }

élémentProperty:

nom du package

stéréotype

propertyName = " propertyValueSymbol* "

nomdupackage:

nom = " propertyValueSymbol* " (. " propertyValueSymbol* ")*

stéréotype:

stéréotype = " propertyValueSymbol* " (, " propertyValueSymbol* ")*

propertyValueSymbol :

\\

\ "

Tout caractère sauf " (U+0022), \ (U+005C)

- elementName est l'un des types d'éléments de l'ensemble
- propertyName est l'une des propriétés de l'ensemble

Les chaînes littérales peuvent être incluses dans les valeurs de propriété en « échappant » un caractère guillemet :

default = " \"Une valeur string .\ " "

Débogage de langage intermédiaire

Le script d'un gabarit MDA produit un texte en langage intermédiaire. Cependant, lors de la génération du modèle, ce script peut renvoyer des erreurs. Lorsqu'une erreur se produit, vous pouvez visualiser et déboguer le texte généré en externe, de préférence dans un éditeur qui prompts à mettre à jour les modifications du fichier.

Accéder

Ruban	Conception > Paquetage > Transformer > Transformer la sélection
Raccourcis Clavier	Ctrl+H (transformer les éléments sélectionnés) Ctrl+Maj+H (transformer Paquetage actuel)

Déboguer lorsque des erreurs sont renvoyées lors de la génération de code modifié

Étape	Description
1	Sélectionnez le Paquetage à transformer, puis l'option « Transformer Paquetage ». La dialogue ' Modèle Transformations' s'affiche.
2	Dans la colonne « Nom », cochez la case correspondant au type de transformation à modifier.
3	Dans le champ « Fichier intermédiaire », cliquez sur le bouton  et définissez l'emplacement du fichier dans lequel générer le code.
4	Cochez la case « Toujours écrire » et cliquez sur le bouton Écrire maintenant pour générer le script. Cela génère uniquement le script, pas le modèle.
5	Si une erreur est renvoyée en spécifiant le numéro de ligne du problème, ouvrez le fichier dans un Éditeur de Code externe (avec Numérotation de Ligne) et localisez le numéro de ligne du problème.
6	Modifiez le code gabarit pour corriger l'erreur.
7	Cliquez sur le bouton Faire la transformation pour vérifier que la modification a corrigé le problème.

Exemple

Pour une base de données MySQL, le code gabarit pourrait ressembler à ceci :

$enumFieldName = « test »

Colonne

{

nom= %qt%% CONVERT_NAME ($enumFieldName, "Pascal Case", "Camel Case")%%qt%

type= %qt%% CONVERT_TYPE (genOptDefaultDatabase, "Enum")%%qt%

}

Cela renvoie la sortie dans le fichier texte généré comme :

Colonne

{

nom = "test"

type = "ENUM"

}

S'il y a une erreur dans la transformation d'origine, comme une faute d'orthographe - « Colum », cliquer sur le bouton Faire la transformation renvoie un message d'erreur faisant référence à la première ligne de code intermédiaire qui inclut l'erreur « Colum ».

Objets

Les objets sont générés lors d'une transformation sous forme de texte sous cette forme :
type d'objet
{
Propriétés de l'objet*
XRef{xref}*
Étiquette { étiquette }*
Attribut{attributes}*
Opération{opérations}*
Classificateur{classificateurs}*
Paramètre{paramètres}*
}

Par exemple:
Classe
{
nom = "Exemple"
langue = "C++"
Étiquette
{
nom = "defaultCollectionClass"
valeur = "Liste"
}
Attribut
{
nom = "compter"
type = " int "
}
}

Chaque object créé dans une transformation doit inclure un élément de syntaxe XRef (voir la fin de cette rubrique), car il aide le système à se synchroniser avec l' object et permet de créer un connecteur vers cette classe dans la transformation.

Éléments de syntaxe dans le code

Élément	Détail
type d'objet	objectType est l'un de ceux-ci : • Action • ActionPin • Activité

	• Paramètre d'activité • ActivitéPartition • ActivitéRégion • Acteur • Association • Changement • Classe • Collaboration • CollaborationUtilisation • Composant • Spécification de déploiement • DiagrammeCadre • Décision • Point d'entrée • Événement • Gestionnaire d'exceptions • Environnement d'exécution • Point de sortie • Nœud d'extension • Région d'expansion • Interface exposée • Élément GUI • Fragment d'interaction • InteractionOccurrence • État d'interaction • Interface • Région d'activité interruptible • Problème • Itération • Object • Noeud d'objet • Fusionner les nœuds • Point de terminaison du message • Nœud • Paquetage • Paramètre • Partie • Port • Interface fournie • Interface requise • Exigence • Séquence • State • Statemachine

	• Nœud d'état • Synchronisation • Tableau • Chronologie • Déclencheur • Diagramme UMLD • Cas d'utilisation
Propriétés de l'objet	objectProperties est égal à zéro ou à une instance d'un ou plusieurs de ces éléments : • Abstrait • Alias • Arguments • Auteur • Cardinalité • Classificateur • Complexité • Concurrence • Nom de fichier • En-tête • Importer • Est actif • Est-ce queLeaf • Est-Root • EstSpécification • Mot-clé • Langue • Multiplicité • Nom • Notes • ntype • Persistance • Phase • Portée • Statut • Stéréotype • Version • Visibilité
Attribut	L'attribut a la même structure que objectType et inclut les propriétés suivantes : • Alias • Classificateur • Collection • Récipient • Endiguement

	• Constante • Défaut • Dérivé • LowerBound • Nom • Notes • Ordonné • Portée • Statique • Stéréotype • Type • Limite supérieure • Volatil L'attribut comprend également ces éléments : • Classificateur • Étiquette • XRef
Opération	Operation a la même structure que objectType et inclut ces propriétés : • Abstrait • Alias • Comportement • Classificateur • Code • Constante • Est-ce que la requête • Nom • Notes • Pur • Tableau de retour • Portée • Statique • Stéréotype • Type L'opération comprend également ces éléments : • Classificateur • Paramètre • Étiquette • XRef
Paramètre	Le paramètre a la même structure que objectType et inclut l'élément Étiquette et ces propriétés : • Classificateur • Défaut • Fixé

	• Nom • Notes • Gentil • Stéréotype
Étiquette	Étiquette a ces propriétés : • Nom • Valeur

Cas particuliers

Certains types d' object présentent des variations dans la syntaxe de définition object .

Object	Détail
Paquetages	Paquetages diffèrent des autres objets de ces manières : • Ils ont un ensemble réduit de propriétés : alias, auteur, nom, namespaceRoot, notes , portée, stéréotype et version • La propriété namespaceRoot n'est donnée qu'à Paquetages • Un nom doit être spécifié pour chaque Paquetage • La propriété name peut être un nom qualifié ; lorsqu'un nom qualifié est spécifié, les propriétés données s'appliquent uniquement au Paquetage final • Seuls Paquetages peuvent contenir d'autres Paquetages • Paquetages ne peuvent pas contenir d'attributs ni d'opérations
XRef	Les références croisées sont définies à l'aide des instructions de transformation. Les propriétés incluent : • Namespace • Nom • Source • Notes
Tableaux	Tableaux sont un type d' object spécial, avec les différences suivantes par rapport aux autres types object : • Ils peuvent inclure des colonnes et primary keys • Ils ne peuvent pas inclure d'attributs
Colonnes	Les colonnes sont similaires aux attributs, mais possèdent un élément autonumber contenant Startnum et son incrément, ainsi que ces propriétés ajoutées : • Longueur • NonNull • Précision • Clé primaire • Échelle • Unique Dans la définition de colonne, vous ne pouvez pas attribuer de valeur aux propriétés

	NotNull, PrimaryKey ou Unique.

Connecteurs

Le processus de création de connecteurs dans une transformation a la même forme que pour la création d'éléments (objets). Il est un peu plus complexe, car vous définissez également chaque extrémité du connecteur : la source et la cible.

Les connecteurs sont représentés dans le langage intermédiaire comme :

Type de connecteur

{

Propriétés du connecteur*

AssociationClass {associationClassProperties*}

Source {sourceProperties*}

Cible {targetProperties*}

}

Par exemple:

Association

{

nom="uneAssociation"

stéréotype=" "

direction="Non spécifié"

Source

{

accès="Privé"

navigabilité="Non spécifié"

}

Cible

{

accès="Privé"

multiplicité="1..*"

}

}

Éléments de syntaxe dans le code

Élément	Détail
Type de connecteur	ConnectorType est l'un de ceux-ci : • Abstraction • Agrégation • Assemblée • Association • Collaboration

	• Contrôle du flux • Connecteur • Déléguer • Dépendance • Déploiement • Clé étrangère • Généralisation • Flux d'informations • Instanciation • Interface • Interruption du flux • Manifeste • Imbrication • NoteLink • Flux d'objets • Paquetage • Réalisation • Séquence • Substitution • Liaison de modèle • Transition • Usage • Cas d'utilisation • Utilisations
Propriétés du connecteur	connectorProperties est égal à zéro ou à une instance d'un ou plusieurs de ces éléments : • alias • direction • notes • nom • stéréotype • étiquette • XRef
associationClassProperties	associationClassProperties en est une instance : • Classificateur • XRef
sourcePropriétés Propriétés de la cible	sourceProperties et targetProperties sont chacune une référence à un élément et zéro, ou une instance d'un ou plusieurs d'entre eux : • agrégation • alias • autoriser les doublons • changeable • contrainte

	• endiguement • navigabilité • type de membre • multiplicité • Notes • ordonné • qualificatif • rôle • portée • stéréotype • étiquette
Référence d'élément	Une référence d'élément est soit un GUID qui référence un élément qui existe déjà avant la transformation, soit une XRef pour référencer un élément créé par une transformation. • guide • XRef

Notes

- Chaque connecteur est transformé aux deux extrémités des objets, par conséquent le connecteur peut apparaître deux fois dans la transformation ; ce n'est pas un problème, bien que vous deviez vérifier soigneusement que le connecteur est généré exactement de la même manière, quelle que soit l'extrémité de la classe actuelle.

Transformer les connecteurs

Lorsque vous transformez un connecteur, vous pouvez utiliser deux types de classe différents comme extrémités de connecteur : soit une classe créée par une transformation, soit une classe existante dont vous connaissez déjà le GUID.

Se connecter à une classe créée par une transformation

La connexion la plus courante est celle avec une classe créée par une transformation ; pour créer cette connexion, vous utilisez trois éléments d'information :

- Le GUID de classe d'origine
- Le nom de la transformation
- Le nom de la classe transformée

Ce type de connecteur est créé à l'aide de la macro de fonction TRANSFORM_REFERENCE ; lorsque l'élément se trouve dans la transformation en cours, il peut être omis de la transformation en toute sécurité. L'exemple le plus simple est celui où vous avez créé plusieurs classes à partir d'une seule classe dans une transformation et que vous souhaitez un connecteur entre elles ; considérez ce script de la transformation d'entité EJB :

```
Dépendance
{
%TRANSFORM_REFERENCE("EJBRealizeHome",classGUID)% stéréotype="EJBRealizeHome"
Source
{
%TRANSFORM_REFERENCE("EJBEntityBean",classGUID)%
}
Cible
{
%TRANSFORM_REFERENCE("EJBHomeInterface",classGUID)%
}
}
```

Dans ce script, la macro TRANSFORM_REFERENCE est utilisée à trois fins : une pour identifier le connecteur à des fins de synchronisation et les deux autres pour identifier les extrémités ; toutes trois utilisent le même GUID source, car elles proviennent toutes de la même classe d'origine. Aucune des trois ne doit spécifier la transformation, car les deux références renvoient à quelque chose dans la transformation actuelle ; chacune d'elles doit alors uniquement identifier le nom de la transformation.

Il est également possible de créer un connecteur à partir d'un autre connecteur. Vous pouvez créer un gabarit de connecteur et lister tous les connecteurs connectés à une classe à partir des gabarits de niveau classe ; vous n'avez pas à vous soucier de ne générer le connecteur qu'une seule fois, car si vous avez créé une TRANSFORM_REFERENCE pour le connecteur, le système les synchronise automatiquement.

Ce script copie le connecteur source :

```
%connectorType%
{
%TRANSFORM_CURRENT()%
%TRANSFORM_REFERENCE("Connecteur",connectorGUID)%
Source
{
%TRANSFORM_REFERENCE("Classe",connectorSourceGUID)%
```

```
%TRANSFORM_CURRENT("Source")%
}
Cible
{
%TRANSFORM_REFERENCE("Classe",connecteurDestGUID)%
%TRANSFORM_CURRENT("Cible")%
}
}
```

Connexion à une classe dont vous connaissez le GUID

Le deuxième type de classe que vous pouvez utiliser comme extrémité de connecteur est un élément existant dont vous connaissez le GUID actuel. Pour créer cette connexion, spécifiez le GUID de la classe cible dans l'extrémité source ou cible ; ce script crée une dépendance à partir d'une classe créée dans une transformation, vers la classe à partir de laquelle elle a été transformée :

```
Dépendance
{
%TRANSFORM_REFERENCE("Dépendance de la source",GUID de la classe)%
stéréotype="transforméDe"
Source
{
%TRANSFORM_REFERENCE("Classe",classGUID)%
}
Cible
{
GUID=%qt%%classGUID%%qt%
}
}
```

Notes

- Chaque connecteur est transformé aux deux extrémités des objets, par conséquent le connecteur peut apparaître deux fois dans la transformation ; ce n'est pas un problème, bien que vous deviez vérifier soigneusement que le connecteur est généré exactement de la même manière, quelle que soit l'extrémité de la classe actuelle.

Transformer Foreign Keys

Enterprise Architect supporte la transformation en Foreign Keys de nombreux types de relations différents définis entre des entités dans un modèle logique.

Chaque Foreign Key d'un modèle physique est représentée par la combinaison d'un connecteur stéréotypé et d'une opération dans chacun des Tableaux concernés. Les transformations Foreign Key sont réalisées avec le gabarit « Connecteur » dans le langage DDL. Ce gabarit génère un ensemble de données intermédiaire qui est ensuite interprété par le moteur de transformation d' Enterprise Architect pour créer toutes les entités physiques et tous les connecteurs requis.

Par défaut, Enterprise Architect supporte les transformations de ces types de connecteurs :

- Généralisation - ce type de connecteur créera une Foreign Key avec une multiplicité de 0..1 dans la source et 1 dans la destination
- Classe d'association - ce type de connecteur créera un tableau « joint » reliant les Tableaux source et de destination
- Association/Agrégation - ces types de connecteurs utilisent la multiplicité définie dans la relation du modèle logique pour joindre les Tableaux source et de destination

Toutes les définitions Foreign Key entraîneront l'ajout d'une nouvelle colonne integer (ou équivalent) dans Tableaux source et de destination, qui agira comme Primary Key dans le Tableau source et comme colonne Foreign Key dans le Tableau de destination. Les noms par défaut des nouvelles colonnes seront le nom Tableau avec le suffixe « ID » ajouté, tandis que les noms des Foreign Keys seront automatiquement générés à l'aide du gabarit FK DDL.

Copier les informations

Dans de nombreuses transformations, une quantité importante d'informations doit être copiée.

Il serait fastidieux de saisir toutes les informations communes dans un gabarit afin qu'elles soient copiées dans la classe transformée ; l'alternative est d'utiliser les macros de fonctions TRANSFORM_CURRENT et TRANSFORM_TAGS.

Utilisation des macros

Objectif	Détail
Copier Object	TRANSFORM_CURRENT (<listOfExcludedItems>) La fonction génère une copie exacte de toutes les propriétés de l'élément actuel, à l'exception des éléments nommés dans <listOfExcludedItems>.
Copier le connecteur	Une autre forme de la fonction est disponible lors de la transformation des connecteurs pour copier l'une ou l'autre extrémité du connecteur : TRANSFORM_CURRENT (<connectorEnd>, <listOfExcludedItems>) Cela génère une copie exacte de l'extrémité du connecteur spécifiée par <connectorEnd> (Source ou Cible) à l'exception des éléments nommés dans <listOfExcludedItems>.
Copier Étiquettes	TRANSFORM_TAGS (<listeDesÉlémentsExclus>) La fonction génère une copie exacte de toutes les Valeur Étiquetés de l'élément actuel, à l'exception des éléments nommés dans <listOfExcludedItems>.

Convertir les types

Les différentes plateformes cibles nécessitent presque certainement des types de données différents, vous avez donc généralement besoin d'une méthode de conversion entre les types. Ceci est proposé par la macro :

CONVERT_TYPE (<langue de destination>, <type d'origine>)

Cette fonction convertit <originalType> en type correspondant dans <destinationLanguage> en utilisant les types de données et les types communs définis dans le modèle, où <originalType> est supposé être un type commun indépendant de la plateforme.

Une macro similaire est disponible lors de la transformation de types de données communs en types de données pour une base de données spécifiée :

CONVERT_DB_TYPE (<base de données de destination>, <type d'origine>)

Cette fonction convertit <originalType> en types de données correspondants dans <destinationDatabase>, qui est défini dans le modèle ; <originalType> fait référence à un type de données commun indépendant de la plateforme.

Convertir les noms

Les différentes plateformes cibles utilisent des conventions de dénomination différentes. Il est donc possible que vous ne souhaitiez pas copier les noms de vos éléments directement dans les modèles transformés. Pour faciliter cette exigence, les gabarits de transformation fournissent une macro de fonction CONVERT_NAME.

Une autre façon de transformer un nom est de supprimer un préfixe du nom d'origine, avec la macro REMOVE_PREFIX.

CONVERT_NAME (<nom d'origine>, <format d'origine>, <format cible>)

Cette macro convertit <originalName>, qui est supposé être dans <originalFormat>, en <targetFormat>.

Les formats pris en charge sont :

- Camel Case : le premier mot commence par une lettre minuscule, mais les mots suivants commencent par une lettre majuscule ; par exemple, myVariableTable
- Pascal Case : la première lettre de chaque mot est en majuscule ; par exemple, MyVariableTable
- Espacé : les mots sont séparés par des espaces ; la casse des lettres est ignorée
- Souligné : les mots sont séparés par des traits de soulignement ; la casse des lettres est ignorée

Le format d'origine peut également spécifier une liste de délimiteurs à utiliser. Par exemple, une valeur de '_' coupe les mots chaque fois qu'un espace ou un trait de soulignement est trouvé. Le format cible peut également utiliser une string de format qui spécifie la casse de chaque mot et un délimiteur entre eux. Cela prend la forme suivante :

<premierMot> (<délimiteur>) <autresMots>

- <firstWord> contrôle la casse du premier mot
- <delimiter> est la string générée entre les mots
- <otherWords> s'applique à tous les mots après le premier mot

<firstWord> et <otherWords> sont tous deux une séquence de deux caractères. Le premier caractère représente la casse de la première lettre de ce mot, et le deuxième caractère représente la casse de toutes les lettres suivantes. Une lettre majuscule force la sortie en majuscules, une lettre minuscule force la sortie en minuscules, et tout autre caractère préserve la casse d'origine.

Exemple 1 : Pour mettre en majuscule la première lettre de chaque mot et séparer plusieurs mots par un espace :

"Ht()Ht" pour afficher "Mon Tableau de Variables"

Exemple 2 : Pour générer l'équivalent de Camel Case, mais en inversant les rôles des majuscules et des minuscules ; c'est-à-dire que tous les caractères sont en majuscules, à l'exception du premier caractère de chaque mot après le premier mot :

"HT()hT" pour afficher "MA table de variables"

REMOVE_PREFIX(<nom d'origine>,<prefixes>)

Cette macro supprime tout préfixe trouvé dans <prefixes> de <originalName>. Les préfixes sont spécifiés dans une liste séparée par des points-virgules.

La macro est souvent utilisée en conjonction avec la macro CONVERT_NAME. Par exemple, ce code crée un nom de propriété get en fonction des options pour Java :

$propertyName=%REMOVE_PREFIX(attName,genOptPropertyPrefix)%

%si genOptGenCapitalisedProperties=="T"%

$propertyName=%CONVERT_NAME($propertyName, "cas chameau", "cas pascal")%

%finSi%

Notes

- Les acronymes ne sont pas pris en charge lors de la conversion depuis Camel Case ou Pascal Case

Références croisées

Les références croisées sont une partie importante des transformations. Vous pouvez les utiliser pour :

- Trouver la classe transformée avec laquelle se synchroniser
- Créer des connecteurs entre les classes transformées
- Spécifier un classificateur d'un type
- Déterminer où se transformer pour les transformations futures

Chaque référence croisée comporte trois parties différentes :

- Un Namespace , correspondant à la transformation qui a généré l'élément
- Un nom, qui est une référence unique à quelque chose qui peut être généré dans la transformation, et
- Une source, qui est le GUID de l'élément à partir duquel cet élément a été créé

Lors de l'écriture des gabarits d'une transformation, le plus simple est de générer les références croisées à l'aide de la macro définie à cet effet :

TRANSFORM_REFERENCE (<nom>, <sourceGuid>, <espace de noms>)

Les trois paramètres sont facultatifs. La macro génère une référence qui ressemble à ceci :

XRef{namespace="<namespace> " name="<name> " source="<sourceGuid> "

- Si <name> n'est pas spécifié, la macro obtient le nom du gabarit actuel
- Si <sourceGUID> n'est pas spécifié, la macro obtient le GUID de la classe actuelle
- Si <namespace> n'est pas spécifié, la macro obtient le nom de la transformation actuelle

Le seul moment où une référence croisée doit être spécifiée est lors de la création d'un connecteur vers une classe créée dans une transformation différente.

Un bon exemple d'utilisation des références croisées est la transformation DDL fournie avec Enterprise Architect . Dans le gabarit de classe, une référence croisée est créée avec le nom « Tableau ». Ensuite, jusqu'à deux connecteurs différents peuvent être créés, chacun devant identifier les deux classes qu'il connecte à l'aide de références croisées, tout en ayant sa propre référence croisée unique.

Spécifier les classificateurs

Les objets, les attributs, les opérations et les paramètres peuvent tous faire référence à un autre élément du modèle en tant que type. Lorsque ce type est créé à partir d'une transformation, vous devez utiliser une référence croisée pour le spécifier, à l'aide de la macro :

TRANSFORM_CLASSIFIER (<nom>, <sourceGuid>, <espace de noms>)

Cette macro génère une référence croisée dans un élément classificateur, où les paramètres sont identiques à la macro TRANSFORM_REFERENCE mais le nom Classifier est généré à la place de XRef.

Si le classificateur cible existe déjà dans le modèle avant la transformation, TRANSFORM_CLASSIFIER est inapproprié, donc le GUID peut être donné directement à un attribut de classificateur.

Si un classificateur est spécifié pour un type, il remplace ce type.

Transformation Gabarit Substitution de paramètres

Si vous souhaitez fournir un accès dans un gabarit de transformation aux données concernant la transformation de la substitution de paramètre de liaison d'un connecteur de liaison Gabarit dans le modèle, vous pouvez utiliser les macros de substitution de paramètre Gabarit .

Facteurs de la transformation

Facteur	Détail

Langue intermédiaire	Les substitutions de paramètres Gabarit sont représentées dans le langage intermédiaire comme suit :
	Substitution de paramètres de modèle
	{
	Formel { FormalProperties }
	Actuel { ActualProperties }
	}
	
	Par exemple:
	Substitution de paramètres de modèle
	{
	Officiel
	{
	nom=%qt%%parameterSubstitutionFormal%%qt%
	}
	Réel
	{
	nom=%qt%%paramètreSubstitutionActual%%qt%
	%TRANSFORM_CLASSIFIER("Classe", paramètreSubstitutionActualClassifier)%
	}
	}

Propriétés formelles ou Propriétés réelles	FormalProperties et ActualProperties sont nuls ou une instance de l'une de ces propriétés : • nom • classificateur
Transformation de la substitution de paramètre Paramètre réel	Si une expression String est attribuée au paramètre Actual, il sera transformé en nom Actual. Vous pouvez attribuer le classificateur Actual si vous connaissez le GUID : Substitution de paramètres de modèle (Officiel { nom=%qt%%parameterSubstitutionFormal%%qt%

	}
	Réel
	{
	nom=%qt%%paramètreSubstitutionActual%%qt%
	classificateur=%qt%%paramètreSubstitutionActualClassifier%%qt%
	}
	}
	
	Si vous souhaitez que le paramètre Actual soit transformé de sorte que son classificateur soit affecté à un élément transformé, utilisez TRANSFORM_CLASSIFIER ou TRANSFORM_REFERENCE, comme indiqué :
	
	Substitution de paramètres de modèle
	{
	Officiel
	{
	nom=%qt%%parameterSubstitutionFormal%%qt%
	}
	Réel
	{
	nom=%qt%%paramètreSubstitutionActual%%qt%
	%TRANSFORM_CLASSIFIER("Classe", paramètreSubstitutionActualClassifier)%
	}
	}
	
	Ou
	
	Substitution de paramètres de modèle
	{
	Officiel
	{
	nom=%qt%%parameterSubstitutionFormal%%qt%
	}
	Réel
	{
	nom=%qt%%paramètreSubstitutionActual%%qt%
	%TRANSFORM_REFERENCE("Classe", paramètreSubstitutionActualClassifier)%
	}
	}