ENTERPRISE ARCHITECT

Work with multiple project files

by SparxSystems Japan

複数のプロジェクトファイルを活用する設計開発 マニュアル

(2015/09/04 最終更新)

SPARX SYSTEMS

目次

1　はじめに

このドキュメントでは、Enterprise Architect を利用して複数人数が分散開発するような状況について役立つ機能について説明します。特に、バージョン管理の機能に重点を置き、説明します。

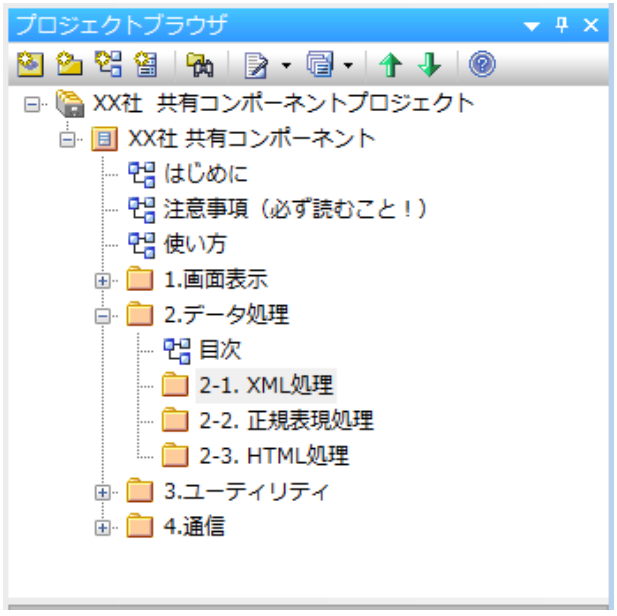
このドキュメントを読む前に、PDF ドキュメント「複数人数による設計開発での利用法 マニュアル」の内容をご覧下さい。「複数人数による設計開発での利用法 マニュアル」に記載されているさまざまな方法の中で、複数のプロジェクト(プロジェクトファイルまたはリポジトリ)を利用する場合について、このドキュメントでは詳細に説明します。

2　「コンポーネント単位の開発」とは？

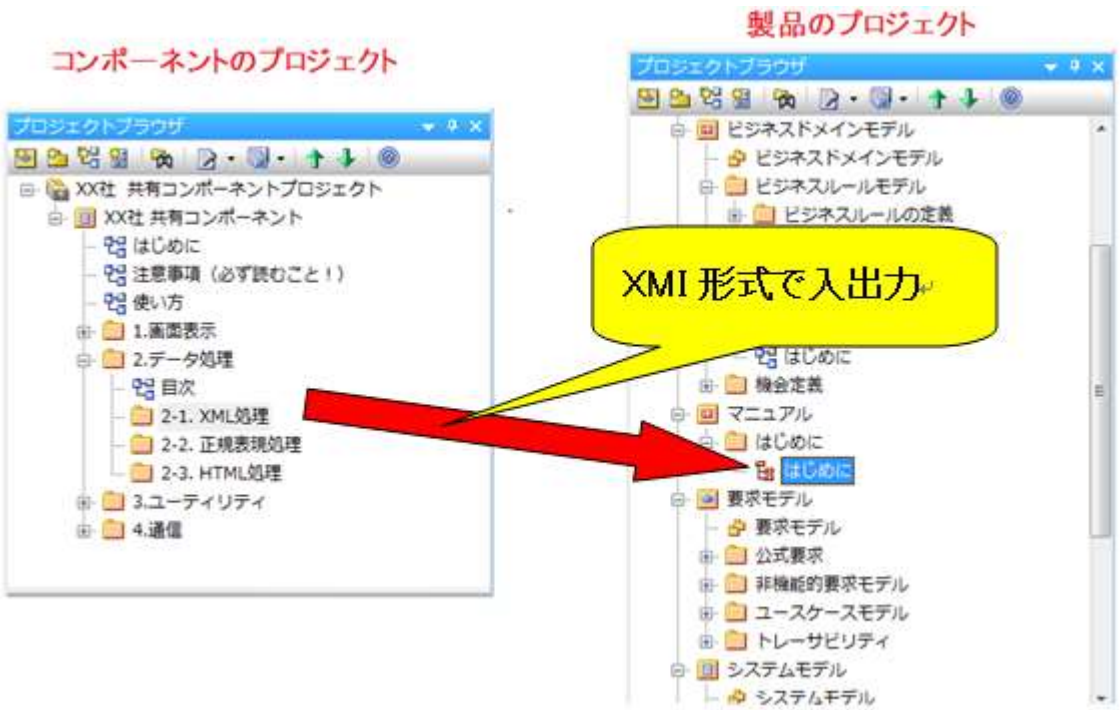
このドキュメントで説明する「コンポーネント単位の開発」とは、コンポーネント単位で開発し、そのコンポーネントを組み合わせて利用し製品を構築する開発手法を指します。具体的には、多くの人々が複数のプロジェクトファイルや DBMS リポジトリ(以下、まとめて「プロジェクト」と表現します)を利用して設計を行い、必要に応じてプロジェクト間で要素の情報を共有・利用するような設計開発形態です。コンポーネントは、「モジュール」や「サブシステム」と呼ばれることもあります。いずれにしても、ある製品やシステムを構成する要素を示します。

このドキュメントでは、コンポーネント用のプロジェクトを作成し、各パッケージにコンポーネントを格納する形式を仮定します。その上で、作成されたコンポーネントを別のプロジェクトに読み込んで利用する方法をご紹介します。

コンポーネント用のプロジェクトの例は、例えば次のような形です。



こうして作成されたコンポーネントの設計情報は、次のスライドのように XMI と呼ばれる XML 形式のテキストファイルで情報をやり取りします。



このときに、バージョン管理ツールを間に挟んで情報をやり取りすることもできます。この方法では、以下のようなメリットがあります。

- 履歴情報が保管され、必要な場合には以前の状態に戻すことができる
- Enterprise Architect のバージョン管理の機能が利用できるので、操作のミスの可能性を減らすことができる

このドキュメントでは、XMI ファイルを利用する方法と、バージョン管理ツールを利用する方法のそれぞれについて説明します。なお、いずれの方法も、「コンポーネントプロジェクト」でコンポーネント(製品の一部分)の設計開発を行い、「製品プロジェクト」でコンポーネントプロジェクトにおいて開発されたクラスを活用する形を想定しています。

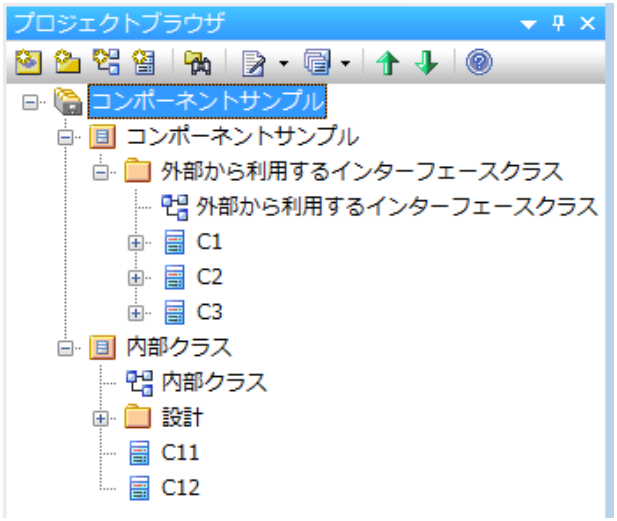
3 XMI ファイルを利用する方法

この方法は、Enterprise Architect が持つ XMI ファイルでの入出力機能を利用する方法です。バージョン管理ツールの準備の必要がなく、簡単に実行して確認することができます。

3.1. コンポーネントプロジェクトでの編集と出力

まず、コンポーネントプロジェクトで編集を行うには、通常の Enterprise Architect での編集とまったく同一になります。自由に編集してください。

XMI ファイルとして出力するのはパッケージの単位となりますので、特定のパッケージの下に作成します。また、コンポーネントを利用する製品側が特定のクラスのみを利用する場合には、その外部から参照されるクラス(インターフェース)と内部でのみ利用するクラスとを、パッケージを分けると良いです。



上の例では、「外部から利用するインターフェースクラス」のパッケージに、外部から利用するクラスを含めています。それ以外のクラスや、設計のための情報は別のパッケージに格納しています。

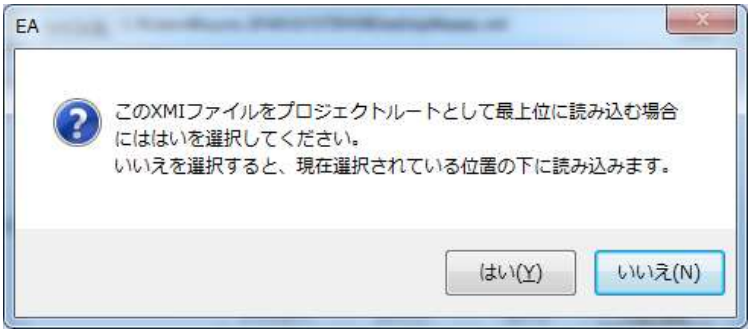
出力する場合は、対象のパッケージを右クリックして「モデルの読み込みと出力」→「XMI ファイルへ出力」を選択します。すると、出力する XMI に関する設定画面が表示されますので、ファイル名と出力位置を指定してください。

3.2. 製品プロジェクトでの読み込み

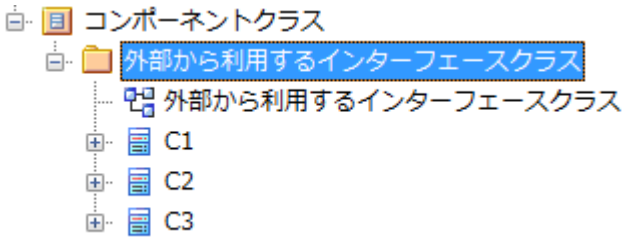
次に、製品プロジェクト側で XMI ファイルを読み込みます。製品プロジェクト側で、読み込む先となるパッケージを右クリックし、「モデルの読み込みと出力」→「XMI ファイルから読み込み」を選択してください。

XMI ファイルの指定画面になりますので、先ほど作成した XMI ファイルを指定してください。すると、コンポーネントプロジェクトで作成したクラスを読み込み、利用することができるようになります。なお、このとき「GUID の初期化」にはチェックを入れないでください。この項目にチェックを入れると、2 つのプロジェクト間の一貫性が失われてしまいます。

また、プロジェクトツリーの構成によっては、次のようなメッセージが表示される場合があります。今回は、指定したパッケージの下に読み込むので「いいえ」を選択してください。



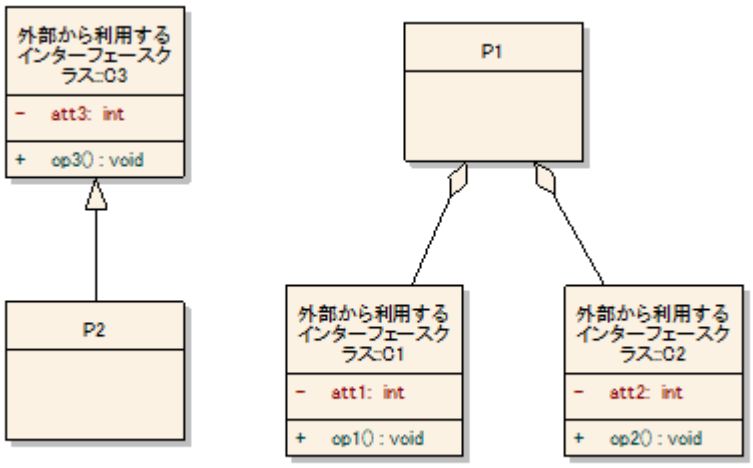
処理が完了すると、プロジェクトブラウザの構成が更新され、コンポーネントのクラスが読み込まれました。



以前に読み込んだパッケージの内容を更新する場合には、対象のパッケージ(この例であれば「外部から利用するインターフェースクラス」のパッケージ)を右クリックし、「モデルの読み込みと出力」→「XMI ファイルから読み込み」を選択してください。

あとは、製品側の設計において、読み込んだクラスを利用して設計をしてください。上記のような手順を踏むことで、コンポーネントプロジェクト側に変更があった場合でも製品プロジェクト側で利用している要素や関係を破壊することなく、同期更新を行うことができます。例えば、下記の例では、「外部から利用するインターフェースクラス」パッケージに含まれているクラスが更新された場合でも、クラス P2 や P1 が持つ、汎化や集約の接続はそのまま維持されます。

この際の一貫性を保持するための仕組みが、上記の設定項目に現れた「GUID」です。Enterprise Architect では要素や属性・操作などにこの GUID が割り当てられ、プロジェクトファイルを超えて情報をやり取りする場合に、それぞれの要素や属性が同じであるかどうかをこの GUID を利用して判定します。
(要素名で、同一の要素かどうかを判断しているわけではありません。)



この例では、もし、クラス C1 や C2・C3 について、要素をいったんモデルから完全削除し、再度同名の要素を作成し直してしまうと、名前が同じであっても GUID が異なりますので、XMI ファイルを読み込むと集約や汎化の接続は維持されません。同じかどうかの判定は、要素の名前ではなく、GUID であることに注意してください。
(逆に言えば、名前を変更した場合でも、関係は維持されます。)

4　バージョン管理機能を利用する方法

Visual SourceSafe・Subversion といったバージョン管理ツールを利用して分散環境での設計開発を進めることもできます。バージョン管理機能を利用することにより、編集の履歴を保存することができます。この履歴の保存には、内部的に XMI ファイルを生成して、その XMI ファイルをバージョン管理ツールで管理する形になっていますので、このバージョン管理機能を利用する方法も XMI ファイルを利用していることになります。

なお、バージョン管理機能の設定についてはここでは割愛します。バージョン管理機能の設定は、ヘルプファイルおよび PDF ドキュメント「バージョン管理機能　機能ガイド」(Web サイトよりダウンロード可能)をご覧ください。

http://www.sparxsystems.jp/products/EA/ea_documents.htm

4.1　モデルのバージョン管理の利点

バージョン管理機能の利点には、一般的に、並列/分散作業の可能性増加・経時的変更の追跡・修正履歴の自動管理などがあります。

モデリング環境にバージョン管理機能を適用する利点を以下に挙げました。特に、複数の利用者がモデルを共有している場合や、利用者が地理的に分散している場合に便利です。

・　簡単で効果的なモデルの複製方法を提供し、複数の異なる拠点に分散したモデルの管理と編集を実現す

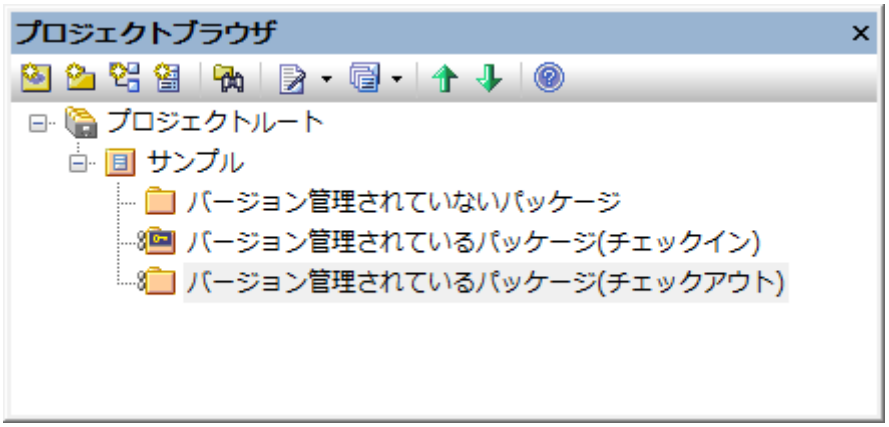
る

- モデルをローカルに配置し、データのやり取りは変更に関する内容だけにすることで、低速ネットワークでつながっている地理的に分散したチームでの設計開発を可能にする
- 「編集」のアクセスを管理情報として調整し、チーム内での排他制御を自動化し、偶発的な変更を防止することができる
- 不必要な変更を元に戻し、間違いを元に戻して最後に「良い状態」であったバージョンにロールバックすることができる
- 「誰が何をいつ変更したか」が分かるように、作業履歴とモデルの変更追跡記録を保持することができる

4.2 Enterprise Architect のバージョン管理機能とは

Enterprise Architect では、個々のパッケージに対してバージョン管理機能を適用できます。

パッケージとは、UML モデルの主要な組織構成です。バージョン管理はどのパッケージにも適用できます。ただし、プロジェクトルートに対してバージョン管理を適用することは推奨しません。ほとんどの場合には意味がないか、管理上の悪影響を及ぼします。



(図 1: プロジェクトブラウザ内のパッケージのアイコンの変化)

Enterprise Architect は、バージョン管理を実現する 2 種類の機能を提供しています。

1. ベースライン:

この機能は、Enterprise Architect コーポレート版で利用可能です。モデル内にあるパッケージについて、特定の時点でのスナップショットをプロジェクト内に格納します。ベースラインの概念は、Enterprise Architect の「比較とマージ」機能の基礎にもなっています。

バージョン管理を実現したい場合には、ベースライン機能の使用も考慮しなければなりません。バージョン管理

の主目的が、単純な構成での比較・マージ・ロールバックの変更履歴を維持することである場合には、ベースライン機能で代用できる場合があります。この意味では、ベースライン機能は、バージョン管理ツールのインストールや環境設定をせずに、すぐに利用できる機能と言えます。

2. バージョン管理ツールとの連携:

Enterprise Architect はサードパーティー製のバージョン管理ツールと連携して動作してバージョン管理機能を実現していますので、ユーザーが希望するツールにパッケージの変更履歴を格納できます。Enterprise Architect がサポートしているバージョン管理ツールには、Subversion・Visual Source Safe(および Microsoft の SCC 互換ツール)・TFS(Team Foundation Server)・CVS があります。推奨ツールは Subversion です。CVS の利用の場合には制限事項があり、推奨しません。

分散型のチーム設計では、このような専用のバージョン管理システムを使って、モデルデータの共有を管理する必要がある場合があります。なお、ベースライン機能を利用して分散設計に対応することはできませんので、分散設計の場合はバージョン管理機能の利用が必須です。

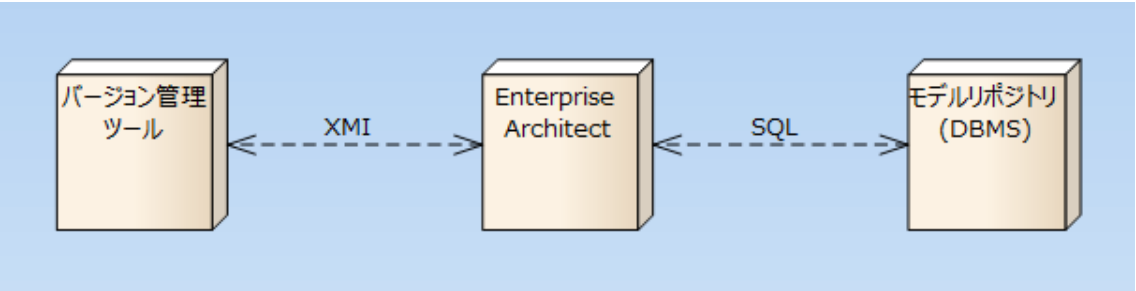
このドキュメントでは、このバージョン管理機能を利用したアプローチに焦点を当てます。変更管理機能や別の選択肢に興味のある方は、付録 B をご覧ください。バージョン管理機能の設定方法や利用方法・操作方法は、ドキュメント「バージョン管理機能 機能ガイド」をご覧ください。このドキュメントでは割愛します。

なお、バージョン管理機能は、全てのエディションで利用できます。

5 バージョン管理ツールと Enterprise Architect

図2 では、Enterprise Architect とバージョン管理ツール間にある関係の概要を表しています。それぞれのモデルの情報は XMI ファイルで格納されます。XMI は、特定の時点でのスナップショットを作成する時に、パッケージ内のモデル情報をテキスト化するために使われています。

Enterprise Architect では、バージョン管理されているパッケージに対する編集を一人のユーザーに限定しています。これは、「ロック － 修正 － ロック解除」 ソリューションという、変更内容の一貫性確保に使われるアプローチです。XMI ファイルはテキストファイルですが、それぞれの行ごとに独立した内容ではないため、ソースコードのように行単位でマージすることはできません。そのため、この方法での管理が必要になります。



(図 2: バージョン管理ツール・Enterprise Architect・DBMS の関係)

ここからは、Enterprise Architect を展開するシナリオ決定のサポート、そしてそれに合ったバージョン管理アプローチを紹介します。各アプローチでは、取り組むべき課題も取り上げています。

なお、バージョン管理機能を利用する場合でプロジェクトファイルを利用する場合、プロジェクトファイル自身はバージョン管理の対象にはしません。バージョン管理は、モデル内のパッケージの単位で行います。

チームの所在: 集中か分散か

集中型/分散型モデルのバージョン管理にどのアプローチを採用するかの判断には、モデルの利用者の分布状況が鍵になります。 以下のようなさまざまな状況が考えられます。

- チームメンバーは地理的に集中し、高速ネットワークでつながっている。
- 利用者は遠隔地、もしくは単独で作業をすることが多く、時には共有ネットワークから長時間離れることがある。
- モデルを共有・編集する主要地点が世界中に存在する。

このような状況の違いが Enterprise Architect のモデルの共有方法を決定し、それによってバージョン管理の適用方法が決まります。

次の章以降では、いくつかのよくあるシナリオにバージョン管理を適用する方法を紹介します。

このドキュメントで想定するのは、以下のシナリオです。

1. **集中型チーム:** 全ての利用者のマシンは高速ネットワークでつながっていて、データベース管理システム(DBMS)にある同じモデルを参照・編集可能。

2. **分散型チーム:** 大多数の利用者は同じネットワークを使っていない。オフラインで作業する場合もあるので、自分のコンピューターにモデルのローカルコピーが必要。

3. **分散作業環境:** 地理的に離れた場所から同じモデルを編集。作業拠点をつなぐ高速ネットワークがない。各

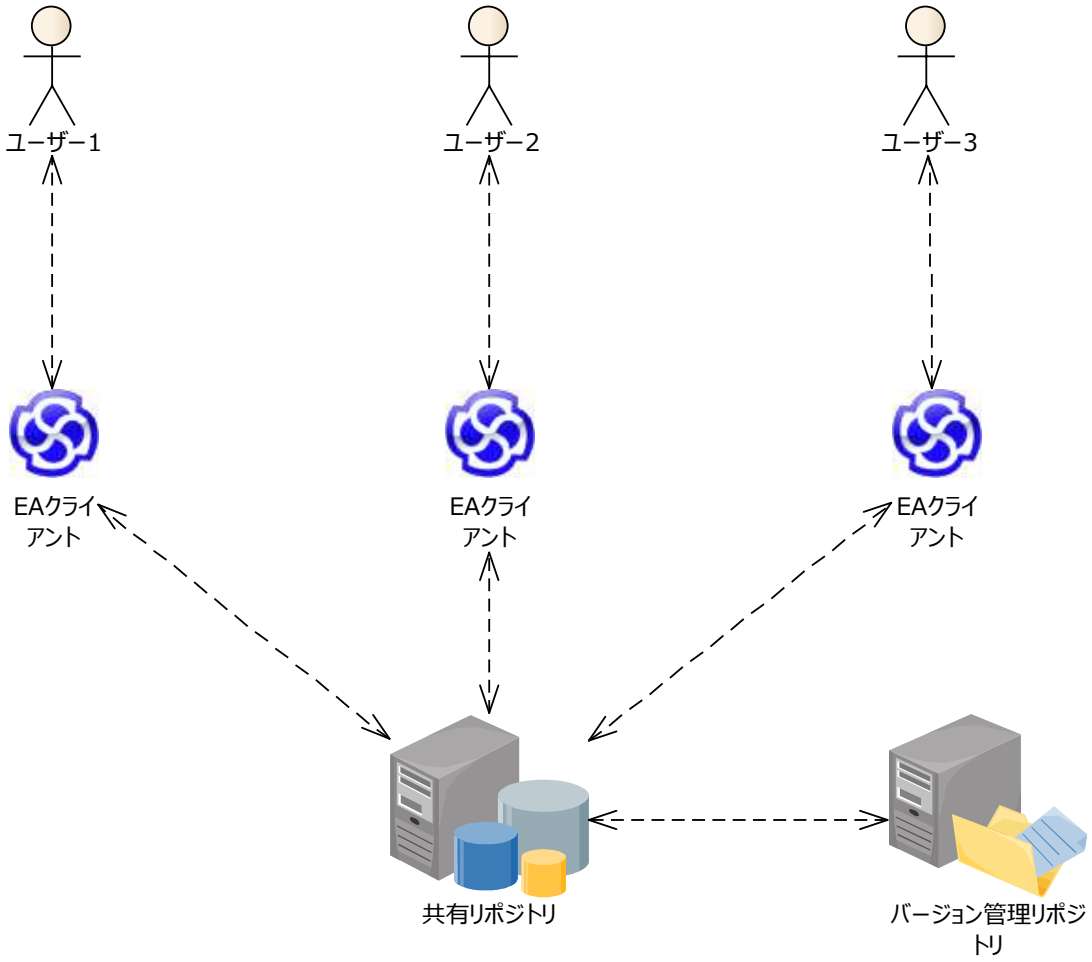
拠点のチームは、拠点ごとのチームで 1 つのモデル(のローカルコピー)を共有。

6　シナリオ 1: 集中型チーム

全てのチームメンバーが高速ネットワークインフラでつながっている、というのがこのシナリオの特徴です。5 人以上のメンバーがモデルに同時アクセスする場合、共有ネットワーク上にある同じプロジェクトファイルにアクセスする場合には、性能面・品質面で予期しない問題を引き起こす可能性があります。この点は拡張子 EAP のプロジェクトファイルの形式である、MS-JET データベースエンジンの制限です。MS-JET データベースエンジンは MS-Access のデータベースエンジンで、個人での利用を前提としているため、トランザクションの不備や問題発生時の復旧力がないなどの問題があります。
(環境などのさまざまな条件により、同時アクセスする場合に問題となる人数は変わります。5 名以上だと必ず問題が発生するという意味ではありません。)

そこで、5 名以上が同時にモデルを参照・編集する場合には、モデルを保管する専用 DBMS リポジトリの利用を推奨します。

この状況で共有 DBMS リポジトリ(あるいはプロジェクトファイル・以下の説明ではまとめて「DBMS リポジトリ」と表現します)を利用する利点は、全チームメンバーが最新状態のモデルを表示・編集できることです。この構成の場合、最新の情報を得るために同期処理を実行する必要はありません。



(図 3: 集中型の構成)

この状況でのバージョン管理リポジトリの役割は、分散モデル環境の仕組みの提供ではありません。このシナリオでは、排他制御・履歴の管理や変更のロールバックのために利用します。

このようなシナリオでバージョン管理を行う際には、以下のような質問がよくあります。

- どの粒度レベルでバージョン管理を適用するか - 最上位のパッケージ・子パッケージなど。
- 別のメンバーの作業を上書きしない方法。
- 誰かが間違えた場合、どのようにしてパッケージを間違える前の状態に復元できるか。

その答えは、次に紹介する推奨手順とベストプラクティスにあります。

6.1 モデルのバージョン管理推奨手順

1. **DBMS リポジトリを設定する:**

(a) Enterprise Architect がサポートする専用 DBMS サーバーを構築し、全チームメンバーからアクセスできるようにする

(b) Enterprise Architect 用に作成したデータベースに対して、テーブル作成用の SQL スクリプト(スパークスシステムズジャパンの Web サイトからダウンロードできます)を実行し、さらに初期モデル(通常は EABase.eap ファイル)を転送する。

(c) 必要に応じて、Enterprise Architect のモデルのセキュリティ機能(アクセス権機能)を有効にする。セキュリティ機能により、Enterprise Architect の機能をユーザーごとに制限・管理できます。

2. **バージョン管理リポジトリを設定する:**

(a) 利用するバージョン管理ツールのサーバーをインストールする。また、対応するクライアントソフトウェアをチームメンバーがインストールしているか確認する。

(b) Enterprise Architect で使用するバージョン管理ツール用のリポジトリを新規に作成する。

3. **バージョン管理クライアントを設定する:**

(a) 初期設定を行うクライアントマシンで、共有プロジェクトを開き、バージョン管理に関する基本的な設定を行う。なお、バージョン管理設定の定義方法は、使用しているバージョン管理ツールによって異なります。

(b) 初期設定を行ったメンバー以外のメンバーが DBMS リポジトリに設定後はじめてアクセスした時には、ユーザーごとのローカルのバージョン管理設定を行う必要があるメッセージが表示され、各自の設定を行う必要があります。

4. **対象パッケージの指定:**

(a) Enterprise Architect で、バージョン管理を個々のパッケージに適用します。詳しくは、付録 C をご覧ください。

セットアップを終えたら、ユーザーは編集したい場合に、パッケージをチェックアウトします。チェックアウトを実行すると、そのユーザーのみが編集可能になります。

このシナリオでは、「最新バージョンを全て取得」や「最新バージョンを取得」の機能を実行する必要はありません。最新のモデル情報は常にモデルリポジトリに含まれています。不要な混乱を防ぐため、この構成の場合には、これらの機能は利用しないことを強く推奨します。

> ベストプラクティス 1:
>
> **集中型チームのモデルにおいて、モデル階層にあるビューやパッケージなど、全パッケージにバージョン管理を適用することで、並列作業の可能性を最大限にします。一方で、パッケージのチェックアウト・チェックインの操作の回数が増えてしまいますので、その点は注意が必要です。(「一括チェックイン」「一括チェックアウト」の機能を効果的に活用してください。)**

ベストプラクティス 2:

ユーザーまたはグループ権限で機能を制限したり、ワークフローのスクリプトを有効化したりする場合には、Enterprise Architect のセキュリティ機能(アクセス権機能)を有効にします。例えば、バージョン管理下にあるパッケージを扱える権限をもつセキュリティグループを作った場合、そのグループに所属するユーザーは「バージョン管理の設定」や「パッケージの設定」の操作を行うことができます。

管理するユーザーを選んでそのグループに追加することで、どのパッケージをバージョン管理に追加(もしくは削除)するかを、自分たちのガイドラインに沿ってよりよく管理することができます。

その上で、「バージョン管理機能の利用」の権限を外すことで、ユーザー毎にパッケージのチェックアウトを制限させることもできます。つまり、バージョン管理パッケージのアクセスを読み取り専用になるように、特定のユーザーを制限することができます。また、バージョン管理システムのアクセス権を使用して、バージョン管理リポジトリの特定フォルダに対して制限を追加することで、該当フォルダに格納されているパッケージのアクセスを読み込み専用に制限する方法もあります。

注意: 役割ベースのセキュリティ機能を、バージョン管理パッケージのロックのために利用することは推奨しません。パッケージ単位でのロックは、バージョン管理システムの利用ですでに実現できています。

ベストプラクティス 3:

全ての共有モデルを定期的にアーカイブすると、バックアップの際やオフライン時のモデル表示に便利です。
DBMS プロジェクトを プロジェクトファイルに転送する機能を利用することで、アーカイブできます。
ただし、このモデル転送は、バックアップの手段は別に利用してください。つまり、通常の DBMS でのバックアップ手順に追加して実行してください。

ベストプラクティス 4:

パッケージにチェックインするときには、毎回意味のあるコメントを入れてください。(Enterprise Architect では、チェックインの際にコメントを入力する画面が表示されます。)
後に変更内容を確認する際に、優れたコメントがあれば、その変更の本質の理解に大変役立ちます。
エラーなどでパッケージを以前のリビジョンに復元する必要がある場合にも、最後に「良かった」状態を見る指標としてもこのコメントは有用です。

6.2 ロールバック (元に戻す) 変更の推奨手順

ロールバックする場合、バージョン管理ツールの機能でロールバックすることもできますが、以下の方法で対応することを推奨します。(バージョン管理ツール側でのロールバックなどの操作は、Enterprise Architect から見た

場合の一貫性の維持ができなくなるなどの問題もあり、サポートの対象外の操作となります。）

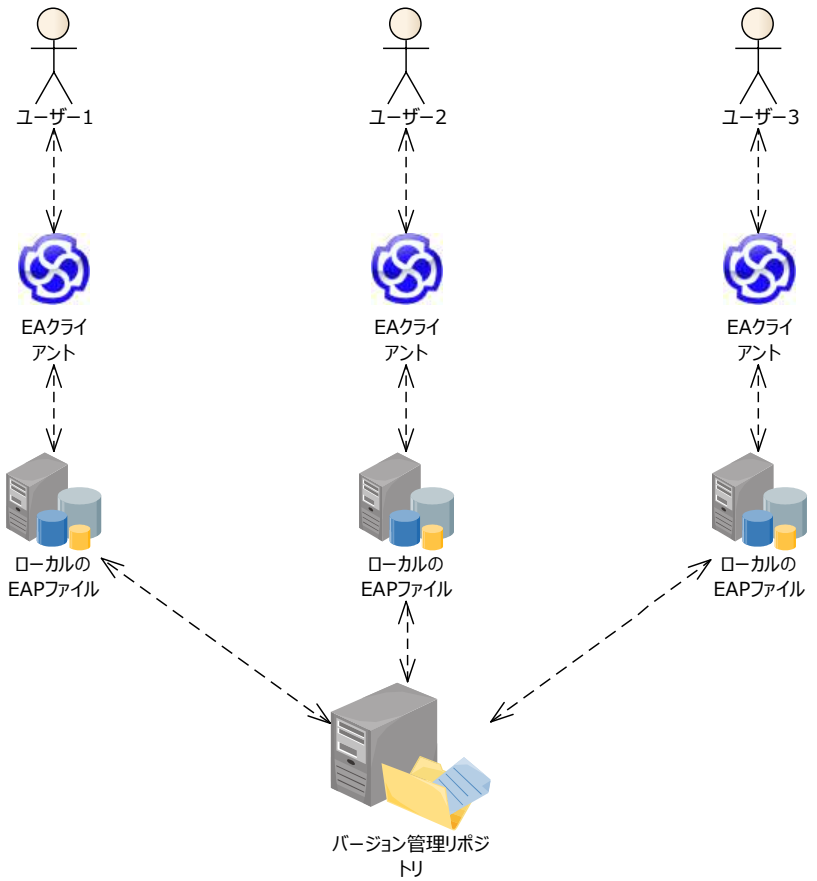
1. 対象のパッケージを右クリックし、「パッケージの管理」→「パッケージの履歴」を実行し、該当するリビジョンを選択し、「チェックアウト」ボタンを押して編集可能な状態で読み込む
2. （復元した）パッケージをチェックインする。
   以上で、復元処理が完了します。

なお、特定のリビジョンの内容に完全に戻すのではなく、特定の要素など一部の内容のみを戻すこともできます。この場合の手順については、PDF ドキュメント「バージョン管理機能 機能ガイド」をご覧下さい。

7 シナリオ 2: ローカルモデルを使用する分散型チーム

このシナリオでは複数の利用者がモデルを使用しますが、共有のモデルリポジトリを使用することはありません。代わりに、各利用者はプロジェクトファイルでモデルのローカルコピーを持ち、共有バージョン管理リポジトリから定期的に自分のコピーを取得・更新します。

このアプローチでは、DBMS リポジトリを管理することなく、広い範囲でモデルを複製することができます。



(図 4: 分散型の構成)

7.1　モデルのバージョン管理における推奨手順

1. **バージョン管理された Enterprise Architect のモデルを設定:**

   a) バージョン管理ツールのリポジトリを作成します。

   b) 設定者のマシンのバージョン管理システムから、バージョン管理ツールのリポジトリを参照できるように設定します。

   c)設定者が Enterprise Architect のプロジェクトファイルを作成し、バージョン管理設定を定義します。具体的には、6.1 章の手順 3 と同じ操作になります。

   d) その設定者が、Enterprise Architect で、バージョン管理が必要な(=共有する必要のある)個々のパッケージにバージョン管理を適用します。具体的には、6.1 章の手順 4 と同じ操作になります。(詳しくは、付録 C を参照。)

2. **チームメンバーにモデルを配布:**

   a) バージョン管理の設定が完了したプロジェクトファイルを作成したら、チーム全体にそのプロジェクトファイルを配布します。

   b) Enterprise Architect を利用するメンバーは、ステップ 1b の作業を行い、バージョン管理ツールが利用できるようにします。

   c) モデルを初めて開いた時に、モデルで使用されるバージョン管理設定の定義を設定する必要のあるメッセージが表示されますので、すべて設定します。この作業の中で、ローカル作業コピーファイルへのパスを指定し、ローカルマシンに定義を保存します。

   d) これでプロジェクトファイルがバージョン管理ツールにつながり、利用可能になります。

ベストプラクティス 5:

集中型チームと同様、モデル階層にあるビューやパッケージなどのパッケージ全てにバージョン管理を適用することで、並列作業を最大限にします。これは、パッケージ間の依存が慎重に管理(次のセクションで説明)されていれば、適切な方法です。

分散型環境で使用する場合、このアプローチには更なる利点があります。それは、最新の変更を確定・復元する時に、ファイルサイズの大きいプロジェクトファイルの全体ではなく、ファイルサイズの小さい XMI ファイルをバージョン管理リポジトリと Enterprise Architect 間で転送することになる点です。

編集手順の単純化は難しいことではありません。しかし、それにはバージョン管理されているパッケージ間の依存削減が必須になります。そのためには、たとえばモデル階層の低位レベル(深い階層)にはバージョン管理を適用しない、という方法もあります。

パッケージ間の依存関係を失う可能性を低くすると、作業効率の低下や並列作業ができなくなる箇所が発生するデメリットがあります。

ベストプラクティス 6:

「マスター」となるプロジェクトファイルを保守する「モデル管理者」を、チームメンバーから一人選びます。このプロジェクトファイルは日々の作業には使用しません。このファイルの目的は、新しいチームメンバーが作業に参加する場合の、「出発点」の提供にあります。

保守を行う場合には、バージョン管理リポジトリから、Enterprise Architect の「最新バージョンを全て取得」コマンドでマスタープロジェクトファイルを更新します。さらに、必要に応じて、新規に追加されたパッケージを「パッケージを指定して追加」機能でマスターに追加します。

このように新規のパッケージを管理者が忘れずに追加するためには、新しいパッケージがバージョン管理に追加されたことをモデル管理者に通知する手順を定義する必要があります。

「マスター」プロジェクトファイルのコピーを使用することで、同じ設定識別 ID を使うようにすることができ、バージョン管理の一貫性を維持できます。

ベストプラクティス 7:

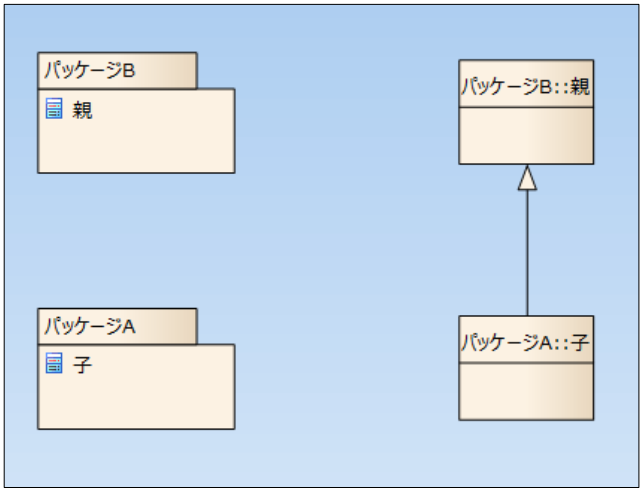
ローカルのプロジェクトファイルを編集する分散型チーム環境では、バージョン管理リポジトリに Enterprise Architect のセキュリティ機能(アクセス権)の情報が格納されないので、自動更新はされません。そのため、このシナリオではセキュリティ機能は利用できません。

この方式の場合には、それぞれの利用者が保持するローカルのプロジェクトの内容は最新とは限りません。そのため、必要に応じて「最新バージョンの取得」あるいは「最新バージョンを全て取得」機能を実行し、最新の内容に更新する必要があります。

7.2 パッケージ間の依存管理

パッケージは単独で管理できるので、編集しているモデルコピーに関係線の片方の要素がない場合でも、モデルを(意図的、または偶発的に)分割することができます。

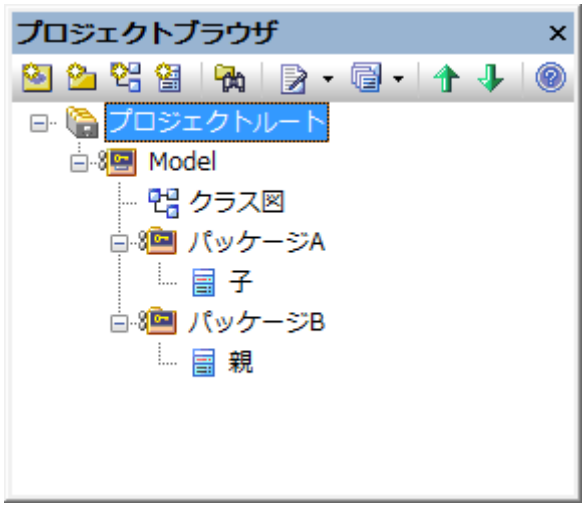
そのモデルの利用者が必要とするモデル情報のスコープを制限するために利用することがありますが、この場合、モデルの情報が欠落する可能性があります。図 5 にあるモデルを例にしてみましょう。



(図 5: 親子の要素が異なるパッケージに格納)

この例で、クラスの親と子は、別々にバージョン管理されているパッケージに定義され、その間に継承関係が存在します。子クラスはパッケージ A、親クラスはパッケージ B で定義されています。これはバージョン管理パッケージ間に依存関係が存在するシナリオのよくある例です。

Enterprise Architect のプロジェクトブラウザでは、図 6 のように表示されます。



(図 6: プロジェクトブラウザの構成)

両方のパッケージが同じモデルにある場合、パッケージＡとＢの中にある要素間の関係情報は維持されています。一方で、ローカルモデルに片方のパッケージしかない場合には関係情報が維持できません。なぜなら必要な要素のひとつが失われているからです。その結果、ダイアグラムで表示される内容やモデルの内容・要素間の関係が欠落します。

さらに、不完全なモデルにパッケージをチェックアウトし、その後新しいリビジョンをチェックインした場合には、そこから生成される XMI には欠落した関係は反映されないので、結果的にモデルを変更してしまう可能性があります。このような意図しない変更を防ぐために、オプションの「XMI1.1/2.1 の読み込みで、存在しない要素のプレースホルダを配置」あるいは「パッケージ間の参照情報を常に保持」の設定が役に立ちます。
(このオプションについては後述します。)

多くのプロジェクトでは、バージョン管理されているパッケージ間に、このような関係が存在します。次の章では、このようにパッケージ間に依存関係があるモデルにおいて、推奨する安全な編集手順について考えていきます。

ベストプラクティス 8:

パッケージの依存関係を事前に計画したものとし、論理的に適切な形で依存関係を維持します。そのためには、UML のパッケージ図などを使用して、その間にある依存関係を確実に把握・定義する必要があります。

一つの方法として、全てのパッケージが利用可能な状態になっている「マスター」プロジェクトファイルにその依存関係図(パッケージ図)を保存します。

このように、モデル化の規則やガイドラインを定義することで、不必要・不適切な依存関係が作成されることを防ぎます。

特定の要素がモデル内の他の要素にどのように関係しているかを確認したい場合には、トレーサビリティサブウィンドウや接続ブラウザ・関係マトリックスなどの Enterprise Architect の機能が役に立ちます。また、スパークスシステムズ ジャパンの Web サイトからダウンロードできる、パッケージ間に依存関係などの接続の情報を表示するアドインなどを利用して、依存関係図を自動的に作成することも可能です。
参考: http://www.sparxsystems.jp/products/EA/tech/Addins.htm#4

ベストプラクティス 9:

作業はできる限り完全なモデルで行います。一部ではなく、全てのモデルをローカルのプロジェクトファイルにコピーすると、パッケージ間にある依存の制御が容易になります。

作業を「マスター」のプロジェクトファイルのコピーから始めることや、「最新バージョンを全て取得」コマンドを定期的に実行することで、依存関係を無くす恐れのある変更を実行する危険性を最小限にとどめることができます。

7.3 変更を安全に行うための推奨手順

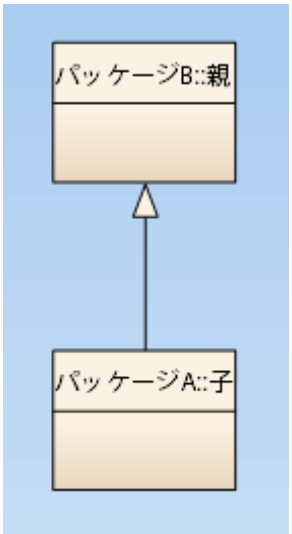
ここで取り上げるのは、パッケージ間の依存が原因で、モデル更新が複数のバージョン管理パッケージに影響する、というシナリオです。

その依存関係は UML の汎化の関係のように明示的な場合もあれば、属性の型として利用されている分類子を参照するような暗黙的な場合もあります。

バージョン管理されている 2 つのパッケージ間の接続をモデル化する

図 5 と図 6 でモデル化されているように、2 つの独立したバージョン管理パッケージ A と B があると仮定します。

この 2 つのパッケージにある要素間に次の 4 つの編集を行います。



(図 7: 汎化の作成)

1. パッケージ A 内の要素からパッケージ B 内の要素に接続を新規に作成する
2. 接続先を別の要素に変更する
3. 接続の方向を反対に向け、接続のタイプとモデルを双方向に変更する
4. 接続を削除する

このそれぞれの編集について、関係のあるパッケージを更新する場合の推奨手順を定義します。この推奨手順によって、モデルの利用者全員が確実に更新を反映させることができます。

1. パッケージ A の要素からパッケージ B の要素に接続を作成

   図 7 にあるような状況をモデル化します。

   汎化の関係の追加は親クラスには影響を与えないので、子クラスはこの関係を一方的に「所有」していることになります。

   そのため、パッケージ A をチェックアウトするだけでこの変更は実行できます。

このような一方向の接続を作る手順は以下の通りです。

i. 「最新バージョンを全て取得」で、モデル全体を確認 (ベストプラクティス 9)
ii. パッケージ A をチェックアウト
iii. 汎化の関係を作成
iv. パッケージ A をチェックイン

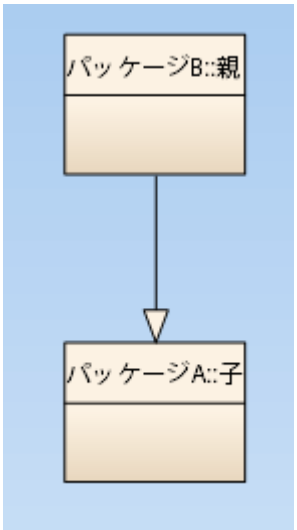
注意:別のパッケージにあるダイアグラム内で関係を定義する場合には、そのダイアグラムを含むパッケージを、パッケージ A と同様にチェックアウトし、チェックインすることが必要です。

2. 接続先を別の要素に変更

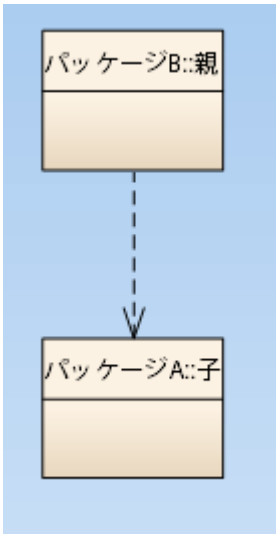
   前回の編集と今回の編集は、本質的に同じものです。なぜなら、接続の端の要素の変更は、その変更が可能な場合には、元々の対象要素や新規対象要素のどちらも変更しないからです。そのため、同じ工程を利用できますが、ステップ 3 を対象要素の変更に変えてください。

3. 接続の方向を反対にし、接続のタイプとモデルを双方向に変更

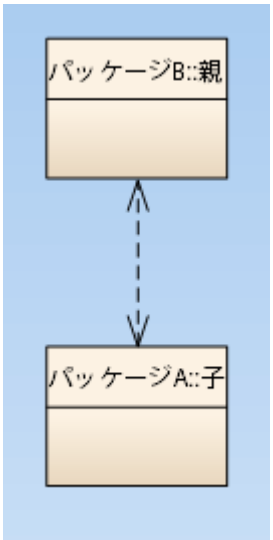
   図 8～10 にある一連の変更を同じモデルに行うとしたら、パッケージ A とパッケージ B の両方の要素を変更することになります。 したがって、変更を行うために両方のパッケージをチェックアウトしてから編集作業を行い、編集が終わったら両方のパッケージを同時にチェックインしなりません。



（図 8: 向きを反対に）



（図 9: 種類の変更）



（図 10: 双方向に変更）

これらの変更には、以下の手順を推奨します。

i. 「最新バージョンを全て取得」で、モデル全体を確認（ベストプラクティス 9）
ii. パッケージ A とパッケージ B をチェックアウト
iii. 関係を編集
iv. 「一括チェックイン」の機能を利用してパッケージ A とパッケージ B をチェックイン（ベストプラクティス 10）

4. 接続を削除

ここで、図 7 の内容で、その接続を削除したいと仮定します。現時点での Enterprise Architect は、この操作では、両方のパッケージをチェックアウトする手順が必要になる場合があります。

図 7 の状況で、もしこの章で説明している方法、つまり片方のパッケージのみをチェックアウトして汎化の関係を追加した場合には、そのチェックアウトしたパッケージにのみ汎化の情報が記録されています。この場合には、追加時にチェックアウトしたパッケージのみをチェックアウトし、削除の操作を実行すれば問題ありません。

一方で、汎化の関係を結ぶ要素が含まれるパッケージの両方をチェックアウトして追加し、チェックインした場合には、その接続の情報は両パッケージの XMI に保存されます。この場合には、削除する際には両方のパッケージを更新しなければなりません。片方のパッケージのみをチェックアウトして削除した場合には、オプションの設定によっては、例えば「最新バージョンを取得」を実行し、強制的に再読み込みする場合などに、削除した関係が復活する場合があります。接続の情報が XMI ファイルに含まれているかどうかは、XMI ファイルの内部を直接確認する以外の方法ではわかりませんので、確認することは現実的ではありません。そのため、編集するために必要な側のパッケージのみをチェックアウト・チェックインするようにするか、あるいは(削除の場合には)両側のパッケージをチェックアウトし、編集後チェックインするというような対応が必要になります。

ベストプラクティス 10: 一貫性を保つチェックイン

複数のパッケージに影響を与えるような変更をチェックインするには、Enterprise Architect の「一括チェックイン」コマンドを使います。
このコマンドは、影響するパッケージを全て同時にチェックインできるので、他の利用者が更新したものの一部だけをチェックインすることや関連性のある変更を失うことを防ぎます。
また、同じチェックインコメントを全てのパッケージに使うことで、同時期の変更を論理的にグループ化することもできます。

ベストプラクティス 11: 変更は小さく、単独変更は定期的に

複数のパッケージを一定期間チェックアウトする場合には無関係な変更が多くなり、それによってパッケージ間の依存関係の数と、ロールバックを実施する場合の複雑度が増加する傾向にあります。

なお、オプションの「XMI1.1/2.1 の読み込みで、存在しない要素のプレースホルダを配置」および「パッケージ間の参照情報を常に保持」を有効にすることで、パッケージ間にまたがる関係がある場合に、その片方のパッケージを別のプロジェクトで利用する場合の整合性を適切に維持することができます。「XMI1.1/2.1 の読み込みで、存在しない要素のプレースホルダを配置」のオプションはマシンに保存され、そのマシンでの操作結果に影響します。一方、「パッケージ間の参照情報を常に保持」の設定はプロジェクトに保存され、そのプロジェクトに対して操作する人の共通の設定として動作します。パッケージ間にまたがる関係がある場合には、「パッケージ間の参照情報を常に保持」の設定を有効にすることをお勧めします。

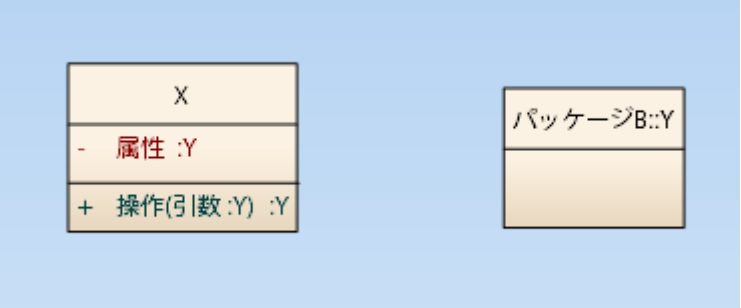
全ての利用者が 1 つの同じプロジェクトを参照している場合には、これらのオプションの設定は不要です。パッケージ間にまたがる関係がある場合でも、その片方のパッケージを別のプロジェクトで利用する場合が存在しないからです。また、この状況の場合には、「最新バージョンの取得」機能は利用しませんので、片方のパッケージのみに関係の情報が含まれるような場合でも、適切に利用している限り削除した関係が復活することはありません。

分類子が別のパッケージに含まれる場合

ここで再び、個別にバージョン管理されているパッケージ A とパッケージ B があると仮定します。パッケージ A には要素 X、パッケージ B には要素 Y があります。ここで、要素 X について、要素 Y を分類子として利用する場合を考えます。

具体的には、このようなモデリング状況は、以下の場合に発生します。

- 要素 X の属性の型として要素 Y を利用
- 要素 X の操作の戻り値の型、または引数の型として要素 Y を利用
- X がインスタンス(オブジェクト)要素の場合、要素 X に対する分類子として要素 Y を利用



(図 11: 別のパッケージの分類子を、属性や操作などで型として利用)

要素間に明示的な接続がなくても、このような場合にはパッケージ間の依存関係が発生していることになります。

この暗黙的な依存の作成・更新は、以下の手順になります。

i. 「最新バージョンを全て取得」で、全てのモデルを確認 (ベストプラクティス 9)
ii. パッケージ A をチェックアウト
iii. 分類子への参照を追加・更新または削除
iv. パッケージ A チェックイン

つまり、型や分類子として指定される要素を含むパッケージは、チェックアウトを行う必要はありません。また、もし型や分類子の定義側を更新する場合には、型や分類子として利用される側のチェックアウトは不要です。

2 つのバージョン管理されているパッケージ間に要素を移動

ここで、パッケージ A の要素 X をパッケージ B に移動すると仮定します。この移動は、両パッケージのモデルに明確な影響を与えますので、両方のパッケージのチェックアウトが必要です。

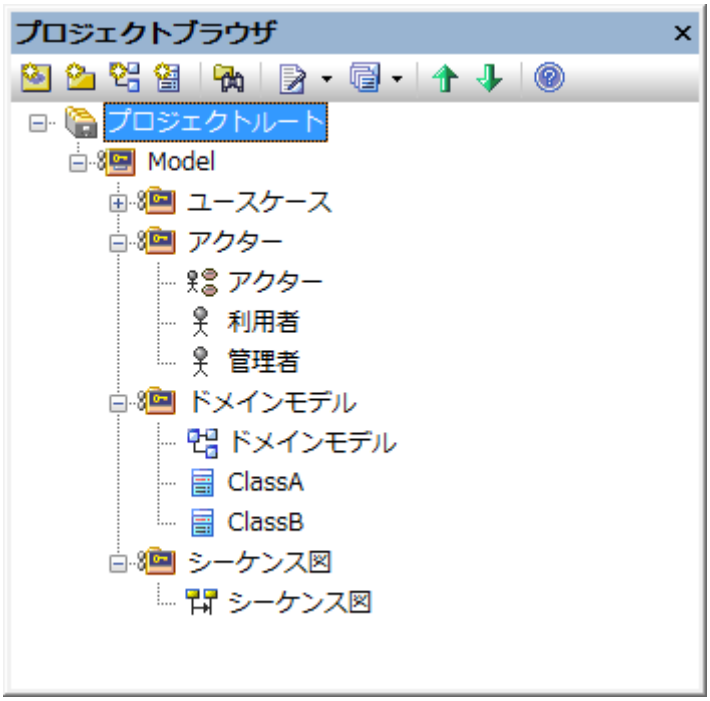
更新手順は以下の通りです。

i. 「最新バージョンを全て取得」で、全てのモデルを確認
(ベストプラクティス 9)
ii. パッケージ A とパッケージ B をチェックアウト
(Enterprise Architect が移動を許可するために必要)
iii. 要素 X をパッケージ A からパッケージ B に移動
iv. 「一括チェックイン」でパッケージ A とパッケージ B をチェックイン
(ベストプラクティス 10)

シーケンスとコミュニケーション図の注意

シーケンスモデル作成時に、ユースケース図のアクター要素やドメインモデルのクラス要素などの分類子をシーケンス図とは別のパッケージで定義・管理するのは、特別な操作ではありません。むしろ、モデルをよりよく整理できるので合理的です。

以下の図 12 はモデル階層の例です。

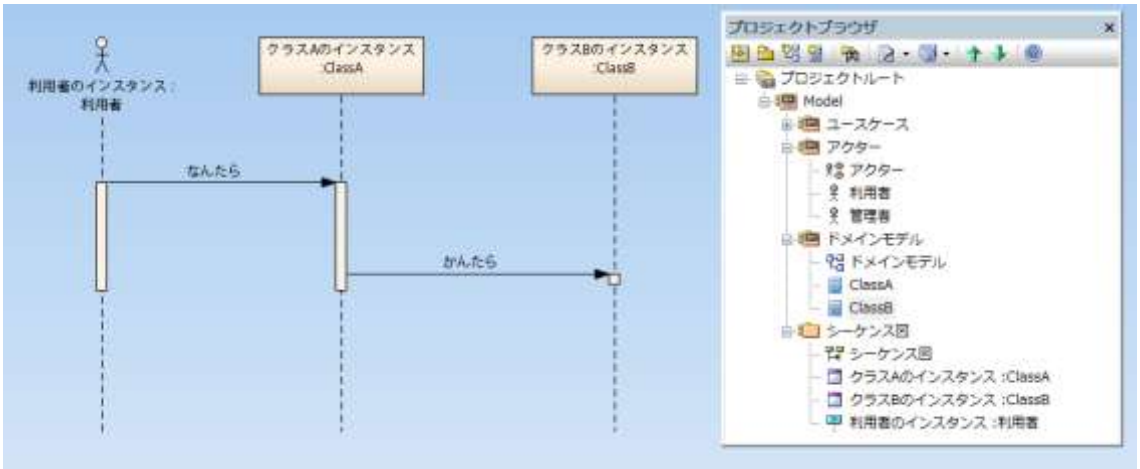


(図 12:「シーケンス図」パッケージの利用)

これらの外部パッケージから分類子を使用する時には、シーケンス図上に、その分類子のインスタンスを作成することを強く推奨します。これは、UML モデリングの観点からも、またバージョン管理のチェックインとチェックアウトでパッケージを往復する際にダイアグラムの接続情報を失う可能性を防ぐ点からも正しい方法です。

シーケンスとコミュニケーション図のメッセージは、そのダイアグラムを持つパッケージの XMI に所属し、保存されます。インスタンス要素とダイアグラムが同じパッケージにある場合、全ての接続情報は次に行われる読み込み作業中に保存されます。

次の図 13 は、上記モデルからシーケンス図を作成する推奨モデリング方法です。



(図 13: インスタンスとして配置した状況)

ベストプラクティス 12:シーケンスとコミュニケーション図の作成時には、分類子を参照するインスタンスを使う

これは絶対守る必要のあるプラクティスです。インスタンス要素とダイアグラムを同じパッケージで保持することで、チェックインとチェックアウト中にメッセージが確実に保存されます。
(分類子要素をシーケンス図やコミュニケーション図で「そのまま配置」し、直接使用しないでください。)

7.4 ロールバック(元に戻す)変更の推奨手順

ロールバック変更(モデルを前のリビジョンに戻す)は、集中型チームで紹介した手順と同じ手順を使用します。

適用できるベストプラクティスは以下の通りです。

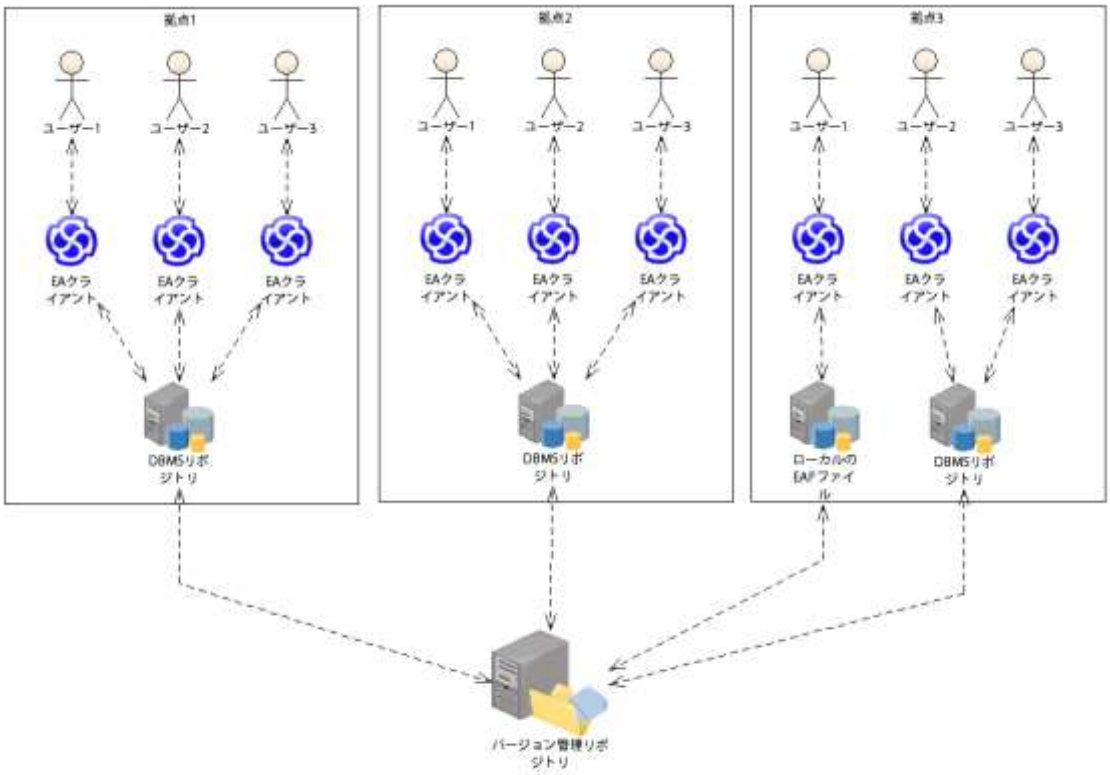
- ベストプラクティス 5: バージョン管理を下位パッケージに適用し、並列作業の可能性を大きくする。バージョン管理パッケージ間の依存に対応する潜在的増加とバランスをとる。
- ベストプラクティス 6: 「マスター」プロジェクトファイルを責任もって管理する、モデル管理者を決める。
- ベストプラクティス 7: ローカルプロジェクトファイルを使用する場合には、役割ベースのセキュリティを適用しない。
- ベストプラクティス 8: パッケージの依存関係を事前に計画し、認識されている依存関係を維持。
- ベストプラクティス 9: 作業はいつも完全なモデルで行い、パッケージをチェックアウトする前に「最新バージョンを全て取得」を使用
- ベストプラクティス 10:複数パッケージに影響する変更をチェックインする場合には、「一括チェックイン」コマンドを使用。
- ベストプラクティス 11: 変更は小さく、単独変更は定期的に
- ベストプラクティス 12: シーケンスやコミュニケーション図に分類子を参照するインスタンスを使用し、インスタンスとダイアグラムを同じパッケージで保持。

8　シナリオ 3: 分散作業環境

大規模企業では、地理的に離れた複数の開発拠点でモデル情報を共有することが一般的になりつつあります。この場合には、各拠点での最新モデル情報の維持が課題になっています。同期ツールを使用して拠点の DBMS レベルを同一化できるかもしれませんが、バージョン管理されているパッケージを活用することでシンプルかつ効果的な手段が可能になります。

シナリオ 1 と 2 で定義した展開を組み合わせたのがこの状況です。

各拠点では、ローカル DBMS リポジトリに格納されたモデルで作業しているかもしれません。また、次の図 14 にあるように、プロジェクトファイルが利用者個人によって使われていることも考えられます。



(図 14: 複数の挙体がある場合の構成例)

複数拠点で展開するには、各共有 DBMS リポジトリ(シナリオ 1 で説明) がローカルの DBMS リポジトリのように扱われることを前提とします(シナリオ 2)。DBMS リポジトリを設定する手順は、シナリオ 1 とシナリオ 2 で説明されている共有 (DBMS)モデルとローカルプロジェクトファイルにそれぞれ適用されます。パッケージ間依存の管理は、シナリオ 2 のベストプラクティスにも関係します。

共通プロジェクトの定義が拠点間で共有できるように、Enterprise Architect のリファレンス情報の管理についても検討が必要です。なお、リファレンス情報については、付録 A で記述しています。

Enterprise Architect の「プロジェクトの転送」機能は、遠隔作業環境間でモデルリポジトリ全体(リファレンス情報を含む) を転送することができます。例えば、拠点 1 で DBMS に作成したモデルをプロジェクトファイルに転送し、他の拠点に配布することが可能です。

ファイルを受け取る側は、プロジェクトの転送機能を使ってプロジェクトファイルから空の DBMS リポジトリにモデルを転送することができます。詳しくは、Enterprise Architect のヘルプファイルの「プロジェクトデータの転送」の内容を参照してください。

付録 A: バージョン管理リポジトリに格納されていない設定情報

この付録 A では、モデルと、モデルの付加情報のバージョン管理を取り上げます。

Enterprise Architect のプロジェクト(プロジェクトファイルまたは DBMS リポジトリ)には、リファレンス情報と呼ばれる付加的な設定情報が含まれていることに注意してください。この情報は、モデル(プロジェクトブラウザに表示されるパッケージや要素など)の中で直接定義することはありません。Enterprise Architect のリファレンス情報には、コード生成やドキュメント生成のテンプレート・要素の状態の種類やステレオタイプの定義などが含まれます。

リファレンス情報のファイルを、手作業でバージョン管理の対象に加えることで、他のプロジェクトから参照・読み込みする場合に、元の情報を一元化できます。それによって、関連する Enterprise Architect のモデル間で、同じ定義やテンプレートを活用することができます。

リファレンス情報を出力するには、Enterprise Architect の機能を利用してください。リファレンス情報の出力と読込の手順は、Enterprise Architect のヘルプファイルの「リファレンス情報」のページを参照してください。

なお、DBMS リポジトリのみを利用している場合で、ネットワークが常時接続されている環境の場合には、共有リポジトリ機能が利用できる場合があります。共有リポジトリ機能は、リファレンス情報を格納しているDBMSのテーブルを、他のデータベーステーブルと共有する方法で実現しています。データベースに対して、共有リポジトリを設定するためのスクリプトを実行する必要があります。詳細は、ヘルプファイルの「リファレンス情報をリポジトリ間で共有する」のページをご覧下さい。

付録 B: Enterprise Architect のさまざまな機能

ここまでは主に、サードパーティー製のバージョン管理ツールを利用して分散型チームでモデル情報を複製する方法や、モデルのリビジョンの管理方法について焦点を当ててきました。

Enterprise Architect には数々の機能があります。これらの機能によって大規模なチームモデリングの支援も可能であるため、バージョン管理機能を使用する必要がないかもしれません。

いくつかの機能の概要を説明します。それぞれの機能の詳細につきましては、Enterprise Architect のヘルプファイルをご覧ください。

監査

この機能は、複数のチームメンバーが同一のモデルリポジトリを共有する際に最適で、モデルのどの部分を誰が変更したか、いつ変更されたか、変更前の状態は何か、といった情報を提供します。監査情報は、バージョン

管理ツールのリポジトリではなく、Enterprise Architect のモデルリポジトリに直接格納されます。

監査機能は特定の時点でのスナップショットではなく、継続的な変更ログを提供します。

監査ログはファイルに出力可能です。ログに記録された要素の変化は、Enterprise Architect 内部で直接比較することができます。

なお、監査機能は、完成した設計モデルの保守フェーズでの利用を想定しています。設計中に利用すると、ログのサイズが膨大になりますので、ご注意下さい。

ベースラインの比較とマージ

Enterprise Architect は、パッケージのバージョンをベースラインとしてモデルリポジトリに直接格納することができます。このベースラインの機能を利用することで、パッケージを以前の状況と比較したり、不要な変更が発生した場合にロールバックしたりすることができます。

ベースラインは XMI 形式で保存されていますので、既にパッケージから出力済みの XMI ファイルとパッケージを比較することも可能です。

このベースラインの機能により、バージョン管理システムを使うことなく、モデルのローカルコピーに行われた変更を抜粋してマージすることもできます。

クラウドサーバ

Enterprise Architect バージョン 11.0 では、「クラウドサーバ」機能が追加されました。この機能は、分散環境でのモデルの共有を支援する機能です。DBMS リポジトリを利用したモデル共有では、TCP ポートを開けて通信可能にする必要があり、また通信内容は暗号化されません。また、それぞれのマシンには DBMS に対応する ODBC ドライバをインストールする必要があります。
「クラウドサーバ」機能は、HTTP や HTTPS でのモデル情報の参照や更新が可能です。HTTPS を利用することで、通信内容を適切に暗号化することができます。また、それぞれのマシンには DBMS に対応する ODBC ドライバをインストールする必要がなく、クラウドサーバ側にのみインストールが必要です。

クラウドサーバを利用することで、モデル情報の分散が不要になる場合があります。この場合には、このドキュメントで説明する、バージョン管理機能を利用した制御が不要になります。

役割ベースのセキュリティ(アクセス権)

Enterprise Architect には、2 つの重要な機能を果たすセキュリティ(アクセス権)機能があります。ひとつは、ユー

ザーにどの編集機能を与えるか制限することのできる機能です。そしてもうひとつは、ユーザー毎、もしくはグループ毎にパッケージと要素をロックする機能です。

セキュリティ機能が適用されているモデルがバージョン管理されている場合には、このセキュリティ機能により、編集などの機能を制限できます。

一方で、共有しているモデルがバージョン管理されていない場合には、チームでのモデリングを効率的に行う上で、このセキュリティ機能が重要な役割を果たします。

ロックを適用すると、互いの変更の上書きや、許可のないユーザーによる不注意なモデル変更を防ぐことができます。

付録 C: バージョン管理をパッケージに適用

この付録では、Enterprise Architect のバージョン管理機能を利用し、これから運用されるシナリオや展開シナリオのモデル構成を基に、パッケージにバージョン管理を適用する推奨アプローチを紹介します。より詳しい説明は、ヘルプファイルに記載されている、バージョン管理されているパッケージの設定をご覧ください。

モデルにある全てのパッケージをバージョン管理する方法:

各パッケージにバージョン管理を追加して並行編集の可能性を最大限にしたい、という時には、モデルのビュー(プロジェクトルート直下のパッケージ)で、Enterprise Architect の「一括してバージョン管理に追加」コマンドを使用してください。

この機能の対象には、選択したパッケージの子パッケージをすべて含みますので、「一括してバージョン管理に追加」をビューに対して適用すると、全ての子パッケージもバージョン管理の対象になります。その結果、バージョン管理が全てのパッケージとその子パッケージに再帰的に適用されます。

この機能を利用する場合、対応する XMI ファイルはパッケージの GUID に基づいて名前がつけられます。この GUID はパッケージの名前が変更されたとしても変わりません。

また、親パッケージの XMI ファイルには子パッケージの「スタブ」情報しか入っていないので、個々のファイルサイズは小さく抑えることが可能です。

なお、「一括してバージョン管理に追加」コマンドで、「階層管理ファイルも出力」オプションが選択できます。この階層管理ファイル(*.EAB)を利用すると、バージョン管理に登録するパッケージ構成全体を簡単に他のプロジェクトに追加できます。今回の構成を別のプロジェクトに読み込む必要がある場合に、この階層管理ファイルを利用すると、関係するパッケージをまとめて 1 回の操作で追加することができます。

バージョン管理を選択して適用する方法:

分散型チームで仕事をしている場合、バージョン管理されたパッケージ間の依存を減らしたい場合があります。その場合には、下位レベルの(深い位置の)パッケージを単独でバージョン管理しない方法があります。この場合には、下位レベルのパッケージは、バージョン管理リポジトリ内にある親パッケージとまとめて管理されます。

新規にバージョン管理対象として追加する手順:

1. 対象となる(上位)パッケージを指定してください。
2. 各パッケージに行う処理:
   i. 右クリックメニューから「パッケージの管理」→「設定」を選択、もしくはキーボードのショートカットキーCtrl+Alt+P を実行します。
   ii. 設定済みのバージョン管理の設定を選択し、必要であれば XMI ファイル名を変更します。他のオプションは既定値のままにしておきます。

空のモデルにバージョン管理を適用する手順:

1. モデルに、利用するパッケージ構成の骨組みを作成します。
2. 下記の中からひとつの方法を選択し、パッケージをバージョン管理に追加します。
   a) 全てのパッケージにバージョン管理を適用するため、「一括してバージョン管理に追加」コマンドを実行します。
   b) 個々のパッケージにバージョン管理を適用します。
      i. 右クリックメニューから「パッケージの管理」→「設定」を選択、もしくはキーボードのショートカットキーCtrl+Alt+P を実行します。
      ii. 設定済みのバージョン管理の設定を選択し、必要であれば XMI ファイル名を変更します。他のオプションは既定値のままにしておきます。
3. 設定完了後、モデルに新規パッケージを追加する場合には、「新規パッケージを作成」画面にバージョン管理に追加するかどうかのオプションが表示されますので、簡単に追加することができます。

○改版履歴

2008/03/07 バージョン 7.1 の情報を反映。ARCSeeker についての記述を追加。
2009/08/31 ドキュメントのタイトルを変更。
2011/05/18 バージョン 9.0 のリリースに伴い、内容を更新。
2011/12/06 バージョン 9.2 のリリースに伴い、内容を更新。
2012/04/11 4 章以下の内容を変更。バージョン管理機能についての説明を追加。
2013/08/30 全体的に内容・文章について見直し、最新の情報に更新。また、文章を読みやすく修正。
2014/04/22 バージョン 11.0 のリリースに伴い、内容を更新。
2014/05/07 全体的に内容・文章について見直し、最新の情報に更新。また、文章を読みやすく修正。
2014/09/04 最新の情報に更新。