Utilisation des APIM

REFERENCE : APIM2DB
PROJET : VISHNU
CHEF DE PROJET : E.P. Capo-Chichi
CLIENT : EDF

INDEXATION DU DOCUMENT

	TITRE : Utilisation des APIM	

ACTION	NOM	DATE	SIGNATURE
RÉDIGÉ PAR	K. Coulomb	04 avril 2012	

SUIVI DU DOCUMENT

INDICE	DATE	MODIFICATIONS	NOM
1	02/04/2012	Premiere version	K. Coulomb

DIFFUSION DU DOCUMENT

ORGANISATION	DESCRIPTION
SysFera	L'équipe VISHNU

Table des matières

Introduction

Ce document a pour but de présenter l'intérêt des APIM [1] faites à l'aide d'Eclipse [2] et leur fonctionnement. La première partie sera consacrée à la présentation de ce qu'est une APIM. La seconde partie est liée à comment la construire. La dernière partie présentera comment utiliser le code générant la documentation se basant sur l'APIM en docbook.

Attention, ce document a été rédigé au début de l'utilisation des APIM, il est possible que certaines informations soient obsolètes.

APIM

Définition

Une APIM correspond à un modèle d'API, c'est à dire une abstraction de l'API qui doit être construite. Les APIM que nous avons construites se basent sur Eclipse et plus particulièrement sur EMF (Eclipse Modeling Framework) qui permet de gérer les modélisations.

Buts

Le but de l'APIM est de fournir une abstraction de telle sorte que l'on ne manipule plus qu'une abstraction de l'API. Ainsi une modification dans le modèle est automatiquement répercutée dans tous les documents se basant sur l'API. Cela permet d'assurer une cohérence entre les divers documents basés sur l'API mais également une justesse des documents issus du modèle. En effet les documents étant automatiquement extraits du modèle, cela supprime le risque d'erreur humaine lors du passage d'une API aux documents qui lui sont liés (la documentation, les manuels, etc...).
De plus, le modèle APIM étant simple (du XML), il est facile de parser la structure pour générer des documents qui vont bien.

Prérequis

Afin de pouvoir utiliser une APIM, nous avons eu besoin des éléments suivant :

* Eclipse version Helios (pour les futures versions, il ne devrait pas y avoir de problème de compatibilité)
* Le package EMF de Eclipse. Voici ci-dessous les manipulations à faire dans Eclipse pour avoir les bons packages bien configurés (par B. Isnard) :
  # La liste des packages faisant partie de la distribution Eclipse Helios (site *'http ://download.eclipse.org/releases/helios'*, normalement déjà configuré dans la liste) nécessaires sont les suivants :
    – Modeling/EMF Modeling Framework SDK
    – General Purpose Tools/Eclipse Plug-In Development Env
    – Programming languages/C/C++ Development Tools
  # Pour XPand [3]/XTend [4] :
    – Récupérer le .zip dans le SVN de Sysfera : SYSFERA/startup/devel/Eclipse/Plugins/m2t-xpand-Update-1.0.1.zip
    – Aller dans Help/Install New Software
    – Cliquer sur Add... puis ajouter un nouveau *site* correspondant au .zip en cliquant sur *Archive* et en indiquant le chemin vers le .zip
    – Le package XPand doit apparaître en-dessous afin de l'installer suivant la procédure habituelle
    – Après l'installation (et redémarrage), il faut modifier la configuration en allant dans le menu Window>Preferences>XPand/Xtend et cliquer sur 'JavaBeans Metamodel' puis Apply

1. API Modèle
2. Eclipse est un framework populaire pour développer en JAVA
3. XPand est un language spécialisé pour générer du code basé sur des modèles EMF
4. Language lié aux méta-modèles

Pour le plugin APIModel :
  – Récupérer le .zip dans le SVN de Sysfera : SYSFERA/startup/devel/Eclipse/Plugins/com.sysfera.codegen.api.zip
  – Unzipper le fichier dans votre repertoire d'install eclipse (cela va installer le .jar dans le sous-repertoire plugins/)
  – redémarrer Eclipse
Voilà, il ne reste plus qu'à récupérer les sources des plugins présents dans le depôt git (eclipse_1.git) sur graal, puis importer les projets dans votre workspace (dans le Package Explorer : click droit>Import...>Existing projects into workspace>Selectionner le directory de votre workspace> La liste des plugins à importer doit apparaître avec des cases à cocher>Les selectionner puis faire Import.) Les modèles existants UMS.apim, TMS.apim, IMS.apim, FMS.apim sont disponibles dans le projet EDF-API. Normalement en double-cliquant dessus dans l'explorateur d'Eclipse cela ouvre l'editeur EMF du modèle (affichage de l'arborescence)

Formatage pour docbook

Pour générer du docbook en utilisant notre générateur, il faut respecter plusieurs règles. Une règle générale concerne le nommage des services/ports, il faut suivre les conventions définies dans le document codingStandard.pdf défini pour le projet EDF. Pour rappel ces conventions sont :

– Pour un service, la première lettre doit être une minucule. Si plusieurs mots sont attachés uniquement la première lettre de chaque mot (hormis le premier) est en majuscule. Le nom du service doit comporter un verbe d'action, exemple 'run, startProcess, fillData'.
– Pour les ports, les noms doivent être en minuscule, si plusieurs mots sont accolés, la première lettre de chaque mot (hormis le premier) doit être en majuscule.

Le fichier data.ecore

Ce fichier ecore correspond à un fichier EMF qui sert à définir les classes/types qui seront par la suite utilisés dans l'API. Bien entendu, le nom du fichier peut être quelconque (ici data sert d'exemple juste).

Pour créer une classe :

* Clic droit sur data > new Child > EClass
* Un nouvel élément doit être accessible sous data
* Remplir les champs correspondants à la classe :
  # Abstract : Si c'est une classe abstraite
  # DefaultValue : Valeur par défaut s'il y en a une
  # ESuperType : Type de la super classe s'il y en a une
  # InstanceTypeName : Le nom de l'instance crée (apparait entre crochet à coté du nom)
  # Interface : Si c'est une interface
  # Name : Nom de la classe
* Maintenant la classe est prête, on peut lui ajouter un attribut
  # Clic droit sur la classe > New child > EAttribute si c'est un type simple, EReference si c'est un type déjà créé
  # Remplir les champs correspondants
    – Default Value Literal : La valeur literale par défaut
    – Derived : Si c'est un type dérivé
    – EAttributeType : Remplit automatiquement à la valeur de EType
    – EType : Le type de l'attribut
    – ID : Si c'est un ID
    – LowerBound : Pour la cardinalité (0 ou 1 au minimum)
    – Name : Le nom de l'attribut
    – Ordered : Si l'attribut est ordonné

  - Transient : Si l'attribut est transient
  - Unique : Si l'attribut est unique
  - Unsettable : Si l'attribut admet un setter
  - UpperBound : 1 par défaut, mettre -1 pour dire nombre indéterminé
  - Volatile : Si l'attibut est volatile
- # Ajout de la description de l'attribut
  - Clic droit sur l'attribut > New Child > EAnnotation
  - Remplir le champs source avec la chaîne de caractère *Description*
  - Clic droit sur l'élément EAnnotation > New Child > Details entry
  - Remplir le champs key avec la chaîne de caractère *content* et le champs value avec la description de l'attribut
  - Si l'attribut est une option d'un type non basique, ajouter un autre *Details entry* avec comme champs *shortOption* dans key et dans value l'option (par exemple **p** si l'option est *-p*)
* Ajouter autant d'attributs et de classes que nécessaire

Pour créer une énumération :

* Clic droit sur data > new Child > EEnum
* Un nouvel élément doit être accessible sous data
* Remplir les champs correspondants à l'énumération :
  - # DefaultValue : Valeur par défaut, mise au premier attribut ajouté
  - # InstanceTypeName : Le nom de l'instance crée, automatiquement mis à la valeur de Name (sauf si l'on modifie directement ce champs)
  - # Interface : Si c'est une interface
  - # Serializable : Si l'objet est sérialisable
* Ajouter un champs à l'énumération
  - # Clic droit sur l'énumération > New Child > EEnum literal
  - # Remplir le champs Name avec le nom de l'élément dans l'énumération et le champs value par sa valeur
* Ajouter autant de champs et d'énumérations que nécessaire

Maintenant que l'on a tout les types de données dans notre data.ecore, on peut modéliser notre APIM.

Modélisation APIM

* Charger le(s) fichier(s) ecore nécessaire(s)
  - # Clic droit > Load Ressource ... > Fournir le chemin vers le(s) ecore(s)
* Ajouter d'un module
  - # Clic droit sur API > New Child > Module
  - # Renseigner le champs Name du module
* Ajouter les types disponibles
  - # Ajouter Type List (clic droit > New Child > Type List) au niveau de l'API (on a droit à un Type List par niveau, cela défini la visibilité du type)
  - # Pour lui ajouter un type ensuite
  - # Clic droit > New Child > Type
  - # Renseigner les champs
    - Description : La description de ce type
    - EcoreType : Le type Ecore de ce type, soit un type de base (E*), soit un type défini dans un ecore de l'utilisateur (si le fichier n'a pas été chargé avec le Load Ressource, les types utilisateurs ne seront pas visibles)
    - Multiple : Si le type est multiple
    - Name : Le nom du type

* Ajouter les types de retour des fonctions
  # Ajouter au module les types de retour
  # Clic droit > New child > Result code list
  # Ajouter un type de retour à la result code list
    – Clic droit > New Child > Result code
    – Remplir les champs
      $ Description : Description du code de retour
      $ Name : Nom du code de retour
      $ Value : Valeur du code de retour
* Ajouter un service (une fonction de l'API)
  # Clic droit sur le module > New Child > Service
  # Remplir les champs
    – Admin Only : Si la fonction est pour tout les utilisateurs ou les admins uniquement
    – Description : Une description de ce que fait la fonction, toujours la commencer par un verbe conjugué à la 3ème personne du singulier car un préfixe lui est automatiquement ajouté lors de la génération
    – Name : Le nom de la fonction
    – ResultCode : Ajouter les codes d'erreurs que peut renvoyer la fonction (ces codes proviennent de la liste Result Code précédemment citée)
  # Ajouter un port (=paramètre de la fonction)
    – Sur le service, clic droit > New Child > Port
    – Remplir les champs utiles du port
      $ Data Type : Le type du port
      $ Description : La description du port
      $ Direction : Si le port est en entrée, en sortie, ou les deux
      $ Name : Le nom du port
      $ Optionnal : Si le paramètre est optionnel (il a une valeur par défaut qui lui est attribué)
      $ Short Option Letter : Si ce port est une option pour la ligne de commande est qu'il est de type simple, alors on spécifie ici la lettre à utiliser pour l'option de ce paramètre
* Faire un service pour chaque fonction de l'API

Générer du docbook

Le code du main de génération du docbook se trouve dans le dépot git concernant eclipse dans le package : com.sysfera.codegen.docbook.apimodel.SpecApiCreate.
Il faut lancer de package avec 3 arguments principaux :

* *-I* Pour indiquer que l'argument suivant contient le chemin vers la racine du dépot git
* Le chemin (absolu) vers le dépot git
* Le chemin (absolu) vers le template docbook. Ce fichier doit etre nommé **-template.docbook*. Ce template utilisant les chemins relatifs à partir du dépot git, c'est pour cela qu'on l'a passé précédemment avec *-I*.

Le fichier de configuration **-template.docbook* doit être écrit par l'utilisateur à la main, au format docbook (=xml particulier). Pour spécifier les éléments à rechercher automatiquement dans l'APIM, il faut les balises suivantes :

* *chapter*, qui permet de dire que l'on a un nouveau chapitre, avec le tag *annotations*, ce tag a pour valeur le fichier APIM avec son module. Le chemin donné est relatif par rapport au chemin indiqué précédemment. Il peut être absolu et à ce moment là les 2 paramètres précédents du programme sont inutiles. Pour une meilleure portabilité, il est recommandé d'utiliser des chemins relatifs. Ici on démarre systématiquement les chemins à la racine du dépot eclipse contenant les APIM. Exemple d'annotation :***annotations="EDF-API/UMS.apim#UMS"***.
* Dans les balises *chapter*, on doit mettre la balise *title* qui correspond au titre du chapitre.

Et cela suffit pour faire fonctionner a génération de code dans un fichier docbook à partir de l'APIM (ou des APIM, rien n'empêche de mettre plusieurs chapter avec des APIM différentes dans le template).