Versioning problem

Dans ce document, nous allons expliquer le versioning des plug-in Eclipse et comment changer la version des plug-ins pour faire en sorte de refléter la nature des changements effectués.

La pratique courante est la suivante : par exemple, pendant le cycle de développement de Eclipse 3.1, les numéros de version des plug-ins sont passés de 3.0.0 à 3.1.0. C'est pratique car, en regardant le numéro de version des plug-ins, on peut directement savoir de quelle version d'Eclipse ces plug-ins proviennent. Mais cette approche a plusieurs inconvénients :

- Le changement arbitraire d'un numéro de version crée des incompatibilités artificielles pour des plug-ins exprimant correctement leurs prérequis. Par exemple, si un plug-in a reçu des corrections de bugs mineurs, sa version sera passée à la suivante, forçant ainsi inutilement les utilisateurs qui possédaient une version inférieure à lancer une nouvelle version de leur plug-in.
- Si on changeait la version à chaque nouvelle version (chaque release), les plug-ins comme EMF et GEF auraient incrémenté de numéro de version pour correspondre à celui de la version d'Eclipse, indépendamment du fait qu'ils aient été modifiés ou non. Un petit changement se répercuterait ainsi sur toute la hauteur.
- Comme la succession des versions de plug-in n'est pas basée sur le build, le gestionnaire de mise à jour ne peut pas être utilisé pour distribuer des plug-ins et n'est par conséquent pas manœuvré par l'équipe de développement.

Dans Eclipse, les numéros de version sont composés de quatre segments : 3 entiers et une chaîne de caractères → major.minor.service.qualifier
Chaque segment représente un but différent :

- le **segment major** indique une rupture dans l'API Quand ce segment est changé, le minor et le service sont remis à 0. En bref, un breaking change rompt la compatibilité et l'on modifie le major segment seulement dans ces cas-là.
- le s**egment minor** indique les changements « visibles de l'extérieur ». On le change par exemple lors de code retravaillé, de changements significatif de performance... Un autre moyen de savoir quand changer ce segment : par exclusion. Il doit indiquer les changements qui ne sont ni des corrections de bugs (indiqués par le service segment) ni modifications de rupture de l'API (indiqué par le segment majeur) . Lorsque le segment mineur est modifié, le segment de service est remis à 0 .
- le s**egment service** indique des corrections de bugs et le changement de flux de développement. Il doit être incrémenté entre deux versions chaque fois qu'on a pas de changement visible dans son API (par exemple correction de bugs, le manifeste a changé, la doc a changé, les paramètres de compilation ont changé). Si le changement arrive dans une version de maintenance, on rajoute 1. S'il arrive pour la prochaine version officielle, 100 doit être ajouté. Ainsi, on a toujours un segment service qui finit par 0 et on est sûr que les plug-ins des prochaines versions auront un numéro de version plus grand que celui des versions de maintenance.
- le **segment qualifier** indique un build particulier. Comme changer le numéro de version à chaque commit est lourd pour l'équipe de développement, il est recommandé de faire cela une fois par cycle de publication. Cependant, puisque nous voulons permettre l'utilisation du gestionnaire de mise à jour par les équipes de développement , nous allons utiliser le segment de qualification pour indiquer les changements entre deux compilations.

Plug-ins sans API

Selon les règles précédentes, ces plug-ins ne vont jamais évoluer plus que dans leur segment de service. Le numéro de version de ces plug-ins doit donc évoluer en synchronisation avec un plug-in auquel ils sont associés. Comme ils ne contiennent pas d'API, ils sont généralement explicitement requis par les plug-ins auxquels ils sont associés.
Exemple : un plug-in de test/source/doc doit évoluer de version en synchro avec le plug-in dont il fournit la source/test/doc.

Versioning de plug-ins qui enveloppent des bibliothèques externes

Exemple : Require-Bundle: com.xyz.widgets;bundle-version="3.8.1"
On requiert une librairie tierce enveloppée dans com.xyz.widgets et qui est compilée avec la version 3.8.1 du bundle.

Spécifier les dépendances entres plug-ins

Un plug-in qui requiert d'autres plug-ins doit donner un intervalle de version pour ces dépendances car l'absence d'intervalle signifie que n'importe quelle version satisfait la dépendance.

Versioning features

Les features sont un mécanisme qui permet le raisonnement en terme d'ensembles de plug-ins.
Les features font comme si leur API était l'ensemble des API des plug-ins les constituant. Ainsi, la version d'un feature doit indiquer le plus signifiant changement des plug-ins et features contenus :

- Incrémenter le nombre major du feature si un plug-in ou feature contenu augmente son nombre major.
- Sinon, incrémenter le minor du feature si un plug-in ou feature contenu augmente son nombre minor.
- Sinon, incrémenter le service du feature si un plug-in ou feature contenu augmente son nombre service.

Liens :

http://wiki.eclipse.org/index.php/Version_Numbering

http://www.eclipse.org/equinox/documents/plugin-versioning.html

http://wiki.eclipse.org/Evolving_Java-based_APIs (explications breaking changes)