Services Web pour TWiki

Manuel de maintenance

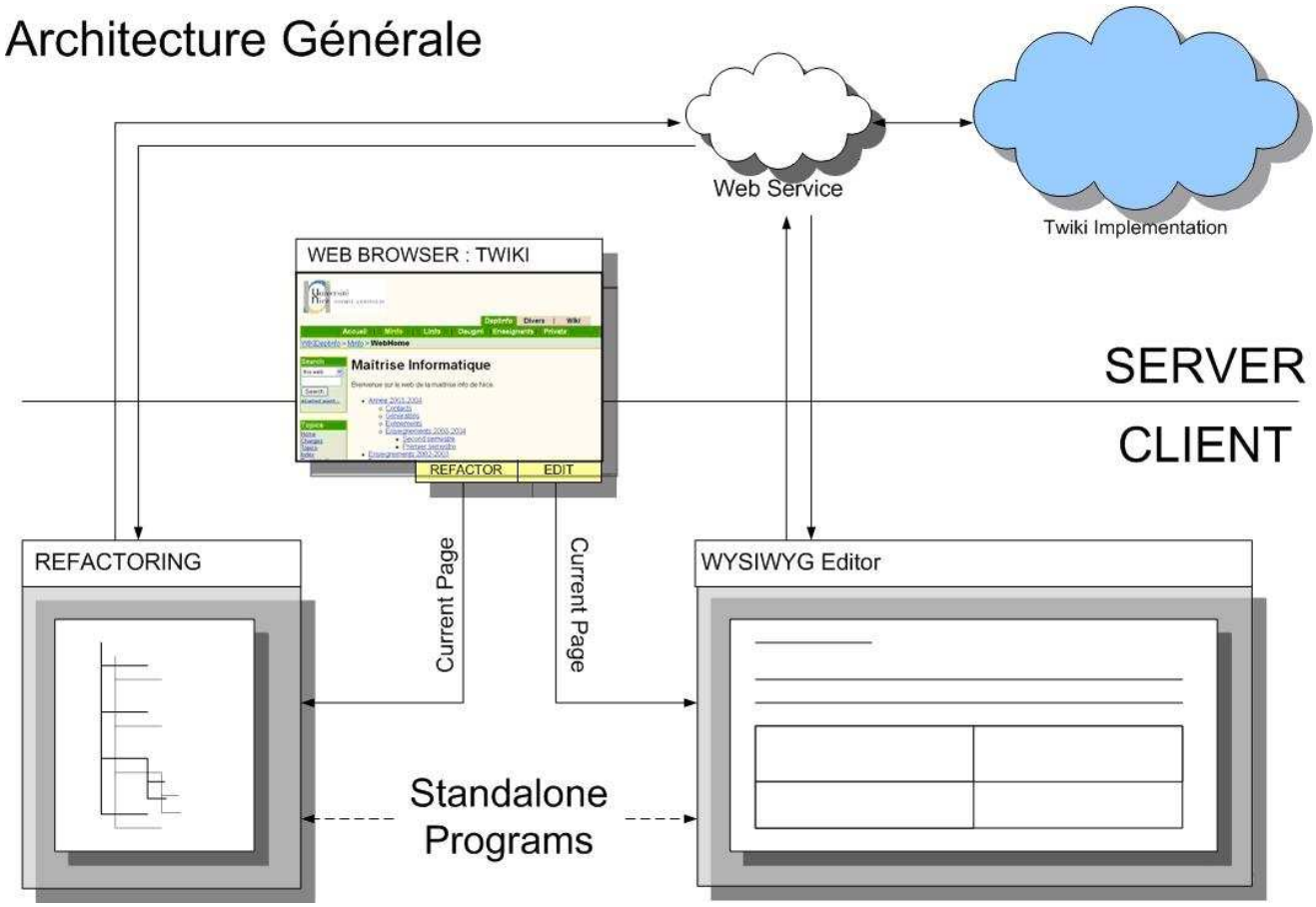
Maxime Lamure (maximelamure@hotmail.com)

Romain Raugi (romano38@wanadoo.fr)

Table des matières

Rôle des services Web

Les services Web permettent aux outils d'édition WYSIWYG et de refactoring développés dans le cadre de notre projet d'interagir avec TWiki. Sauvegarder ou rapatrier un topic, le déplacer ou le supprimer par exemple, se fait grâce à eux. TWiki est conçu pour être utilisable directement par un internaute. Les services étendent son utilisation aux applications en général. Voici un schéma illustrant cette idée:

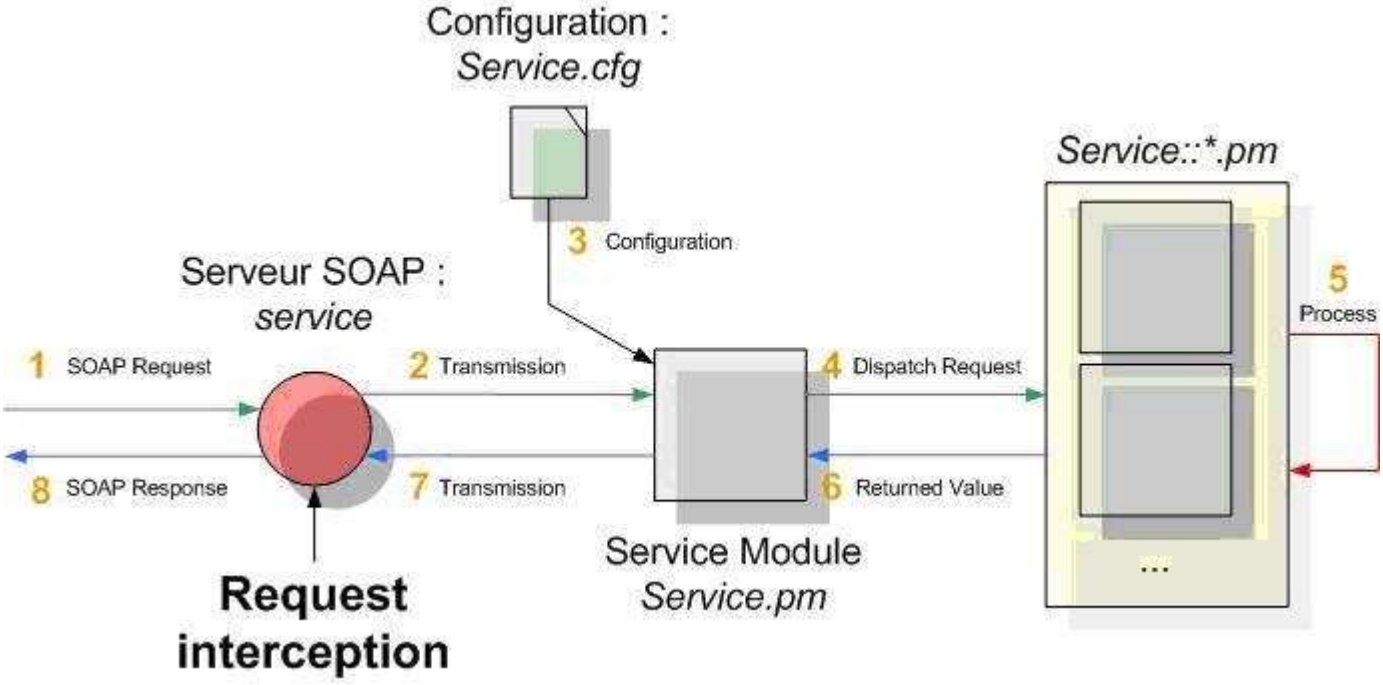


Les services Web font interface entre TWiki et nos applications. Dans le cadre de notre projet, les outils développés sont lancés via Java Web Start mais ce n'est pas du tout obligatoire pour les utiliser. Dans cette illustration, l'indication « Current Page » marquée sur certaines flèches indique des arguments Web Start précisant quel est le topic courant pour le lancement des applications. Le rapatriement de topics ou la récupération de hiérarchies se fait par service et n'a plus rien à voir avec Web Start.

Services

Application

Les services développés ne sont pas des fonctions réparties au sein du code source de TWiki mais font l'objet d'une application à part entière. Du point de vue du serveur, l'application s'initialise lors de l'interception d'une requête et se termine une fois le traitement effectué; les requêtes pouvant être une demande d'opération de refactoring, de récupération de topic, de sauvegarde etc. Voici un schéma illustrant ce qu'il se passe entre ces deux instants clefs:



Le serveur SOAP a pour objectif d'écouter sur un port donné afin d'intercepter des messages SOAP. Une fois le message intercepté, il le désérialise et transmet la requête à un module *Service*. Ce module se charge d'initialiser la configuration du service en paramétrant des variables globales, de la même manière que TWiki. Une fois la configuration effectuée, le module répartit les requêtes dans des sous modules et renvoie le résultat (au serveur SOAP). Il agit en quelque sorte comme un skeleton de serveur.

Le résultat retourné est converti en message SOAP par le serveur SOAP. Le contenu XML du message SOAP peut être écrit de manière automatique ou manuelle (dans le cas de booléens ou de structures complexes par exemple). Perl étant atypé et *SOAP::Lite* n'offrant pas d'équivalence objet/structure complexe XML, on ne peut avoir une sérialisation XML aussi automatique qu'en C# par exemple. Le module *Service::Conversions* a été développé afin de réaliser explicitement certaines sérialisations.

Aspects non fonctionnels

Une grande partie des implémentations réalisées du coté de TWiki concerne les aspects non fonctionnels des services. Parmi eux, la gestion de la concurrence et la gestion des connexions afin de suivre et authentifier les clients du coté du serveur, sont les plus importantes. Mais d'autres aspects ont aussi du être gérés ...

Persistance des données entre appels de services

Principe

Les services Web sont des méthodes accessibles à distance. Une fois exécutées, elles sont lancées dans des threads indépendants du côté du serveur. Il est nécessaire d'avoir un moyen de faire partager des données entre ces threads et de permettre de les sauvegarder une fois un appel terminé. La gestion des connexions a besoin de ce cela par exemple. Plusieurs moyens existent parmi lesquels les segments de mémoire partagée ou plus simplement la sauvegarde sur disque dur dans un fichier. Nous avons opté pour cette dernière technique, la plus simple mais aussi la moins risquée en ce qui concerne la compatibilité entre systèmes d'exploitation. Les services qui requièrent des données persistantes doivent donc dans un premier temps charger le contenu du fichier (dans une table de hachage la plupart du temps) et ensuite le mettre à jour si nécessaire.

Gestion de la concurrence

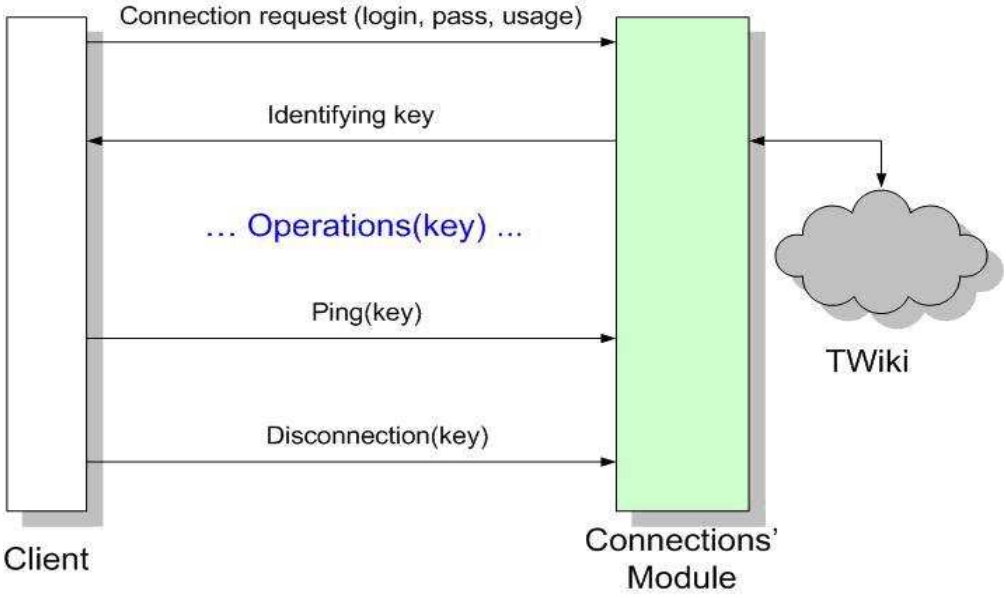
Sauvegarder et lire des données dans et depuis un même emplacement implique gérer la concurrence. Dans le monde UNIX, la commande *flock* aurait été parfaite pour verrouiller l'intégralité du fichier pendant un traitement et le déverrouiller ensuite. De plus, Perl nous permet d'y faire appel, que ce soit sous UNIX ou sous Windows. Mais *flock* a beau être disponible sous Windows, elle n'a aucun effet.

Nous avons donc développé un système plus ou moins similaire, inspiré des verrous utilisés dans TWiki. Le verrou utilisé est un fichier nommé du même nom que le fichier à verrouiller, avec comme extension *.lock*. Il est créé dans le même répertoire et contient sa date de création. Le module *Service::FileLock* implémente ces fonctions de verrouillage et de déverrouillage.

Connexion et authentification

Concept

Pour gérer les communications entre TWiki et nos applications, nous avons imaginé un système de connexion. Il permet notamment d'authentifier une seule fois un utilisateur. Plutôt qu'une longue description, voici un schéma montrant une séquence d'opérations :



Un client envoie une requête de connexion, en donnant son login (*WikiName*), son mot de passe (excepté si l'utilisateur est invité « guest »), et l'usage qu'il veut faire du service. La procédure de connexion va alors faire certains tests parmi lesquels authentifier le client à partir du login et du mot de passe fourni. Si tout se déroule correctement, une clef identifiante sera envoyée au client. Cette clef sera requise par chaque service nécessitant des droits d'accès. Du coté du service, certaines informations seront sauvegardées. Ceci est décrit dans la section suivante. La requête ping mentionnée plus haut permet de savoir si un client est toujours présent. C'est un « acte de présence » que le client devra faire au bout d'un certain temps s'il ne veut pas être déconnecté. Ceci sera décrit plus loin.

Données sauvegardées

Les informations concernant les clients sont sauvegardées à l'issue de la phase de connexion dans un fichier texte (comme décrit dans la section « Persistance des données entre appels de services »). Ce fichier est nommé par défaut *clients*. Retrouver les clients connectés consiste à lire ce fichier, mettre à jour des informations les concernant consiste à y écrire. Voici comment les informations sont sauvegardées :

Key	Usage	User	Connection's datetime	Echo's datetime
163208008	Refactoring	Mario	1082562176	1082562176
16479493	WYSIWYG	Damien	1082562213	1082562213

Le séparateur horizontal est le caractère espace. Voici un exemple :

```
163208008 Refactoring Mario 1082562176 1082562176
16479493 WYSIWYG Damien 1082562213 1082562213
```

La date d' « echo » mentionnée est le dernier moment où un utilisateur a fait un *ping*. Si cela fait trop longtemps, l'utilisateur est déconnecté au prochain chargement du fichier. Dire si cela fait trop longtemps ou pas se base sur un délai de garde (configurable).

Ping

Il s'agit d'un service qu'un client doit appeler au bout d'un certain temps pour indiquer qu'il est toujours là (et qu'il n'a pas planté par exemple). Ce ping, outre le fait qu'il évite à l'utilisateur de se faire déconnecter, propose aussi d'autres services :

- Actualisation des verrous déposés
- Actualisation du verrou administratif
- Mise à jour de l'adresse IP de l'utilisateur pour le service de notifications

Actualiser les verrous lors d'un ping est proposé mais n'est pas imposé, au même titre que le service de notifications. Tout ceci est configurable.

Déconnexion

A noter que la déconnexion, outre le fait qu'elle supprime un client des utilisateurs, supprime automatiquement les verrous posés, verrou administratif du client, si posé, compris.

Module

L'implémentation du mécanisme de connexions se trouve dans le module Service::Connection. Cinq services accessibles à distance y sont proposés:

- *connect* pour se connecter
- *disconnect* pour se déconnecter
- *ping* pour faire acte de présence
- *getUsers* pour obtenir la liste des utilisateurs connectés
- *setAdminLock* pour poser un verrou administratif (cf. section consacrée)

Gestion des verrous

Besoins

Les verrous TWiki sont indispensables pour gérer la concurrence lors de l'édition de topics et lors de la réalisation d'opérations de déplacement, renommage ou de suppression. Ils se présentent sous la forme de fichiers portant le même nom que le topic et ayant comme extension *.lock*. Les verrous sont gérés dans TWiki à partir du *WikiName* du client, qui est sauvegardé dans le verrou.

Un problème posé est qu'un même *WikiName* peut être partagé par plusieurs utilisateurs, par exemple guest si l'utilisateur ne s'est pas connecté. Deux utilisateurs invités peuvent mutuellement écraser leurs verrous par exemple. Autant cela n'est pas vraiment problématique dans le cadre de l'usage normal de TWiki, autant cela l'est d'avantage pour nos services. Nous ne devons pas permettre à un administrateur de déplacer un topic alors qu'il est cours d'édition WYSIWYG par exemple. Nos services doivent marcher sur des TWiki configurés sans authentification (clients tous guest), et doivent permettre plusieurs connexions avec un même login pour l'usage simultané de plusieurs logiciels clients du service.

Nous avons à notre disposition avec le système de connexions d'un identifiant plus « fin » que le login: la clef de connexion. L'objectif de cette section est de montrer comment nous gérons les verrous TWiki à partir de cette clef.

Principe

Nous avons pensé à deux systèmes possibles afin de répondre à ce besoin:

- Surcharger les verrous de TWiki en y mettant la clef comme information supplémentaire
- Maintenir une table d'associations verrou/clef

La première idée a été écartée à cause d'incompatibilités que cela aurait engendré dans TWiki. Après analyses, cela aurait posé des problèmes et il aurait été nécessaire de proposer un patch. La deuxième solution a été choisie, avec une différence:

Un chemin identifiant un verrou ne nous parait pas suffisant pour dire qu'un verrou appartient bien à un utilisateur. Ce verrou a très bien pu être posé par un autre utilisateur TWiki ayant le même *WikiName* (fortement possible avec *guest*), et si l'association est toujours présente, ce verrou appartiendra à l'utilisateur enregistré alors que ce n'est pas le cas. Nous avons ajouté la date de modification du verrou au chemin comme identifiant.

Représentation des données

Les associations sont sauvegardées dans un fichier texte comme pour les connexions. Ce fichier est nommé *locks* par défaut. Retrouver les associations consiste à lire ce fichier et en mettre à jour à l'écrire. Voici comment ces associations sont sauvegardées:

Key	Modification date	Filename
163208008	1100848409	/twiki/data/Sandbox/PageDeTest1.lock
16479493	1100848410	/twiki/data/Sandbox/PageDeTest2.lock

Voici un exemple de contenu de fichier:

```
163208008 1100848409 /twiki/data/Sandbox/PageDeTest1.lock
16479493 1100848410 /twiki/data/Sandbox/PageDeTest2.lock
```

Module

L'implémentation de ce système se trouve dans le module *Service::Locks*.

Verrou administratif ou comment gérer la concurrence entre applications de refactoring

Besoins

L'outil de refactoring a été prévu pour fonctionner en mode « déconnecté ». Les opérations peuvent être faites localement avant de les valider sur TWiki par service Web. Un utilisateur peut donc avoir une liste de modifications à valider ... en même temps qu'un autre. Si les modifications destinées à être apportés par chacun n'interfèrent pas, il n'y a aucun problème. Mais si un utilisateur veut par exemple déplacer un topic qu'un autre veut supprimer, il y a conflit. Si le premier utilisateur valide en premier, le deuxième utilisateur va voir sa liste de modifications en attente irréalisable. Il nous faut donc un moyen d'empêcher cela.

Principe

Nous avons choisi de proposer un mécanisme de prévention (il n'est pas imposé, le service peut fonctionner sans). Avec ceci, un administrateur doit prévenir les autres qu'il va réaliser des opérations de refactoring s'il veut les effectuer. Ceci se fait par le biais d'un « verrou administratif ». Il s'agit d'un verrou - fichier (comme les autres verrous dont nous avons parlé), qui contient la clef de l'utilisateur et sa date de création. Sa gestion est similaire à celle des verrous TWiki. Si le service est configuré pour le gérer, les opérations de refactoring échoueront s'il n'est pas posé.

Module

Le service *setAdminLock* permettant de poser un verrou administratif est implémenté dans le module *Service::Connection*. Tout ce qui nécessaire à sa gestion est implémenté dans *Service::AdminLock*.

Notifications

Il s'agit d'une amélioration qui n'était pas envisagée lors de la rédaction de notre cahier des charges. Les notifications sont des messages émis lorsqu'un événement se produit

afin d'informer les autres utilisateurs. Elles sont gérées uniquement au sein du service, pas dans TWiki en général car cela aurait nécessité de modifier son code (donc de faire un patch). Dans ce sens, ce système n'est pas terminé mais s'avère être un projet ouvert à d'autres développeurs.

Principe

Il faut savoir qu'à la date de réalisation de ce projet, les callbacks ne sont pas supportés par les services Web en standard. Seules quelques implémentations sont allées au-delà du standard afin de les proposer, comme .NET ou Java par le biais de JMS. Nous avons opté pour le concept d'abonnement/notifications, comme pour la gestion des événements en Java. L'abonnement est réalisé par service Web « habituel » par *SOAP::Lite*. Les notifications sont gérées aussi par service Web mais ce ne sont plus des messages SOAP gérés par *SOAP::Lite* mais des envois de données XML. Elles sont programmées à l'échelle des sockets et des messages.

Services d'abonnement

Deux services Web *subscribe* et *removeSubscription* permettent de s'abonner et de résilier un abonnement à des notifications. On demande en outre lors de la souscription le port vers lequel envoyer le message au client. Ces services consistent à ajouter ou enlever un utilisateur d'une liste de contacts, sauvegardés dans un fichier comme pour les connexions. Chaque entrée de ce fichier indique une clef de connexion, une adresse IP et un port sur lequel on doit établir une connexion pour envoyer le message.

Envoi de messages

Si le service est configuré pour, un message est envoyé à chaque opération de :

- refactoring
- (dé)verrouillage de verrou administratif
- modification de topic
- mise à jour des liens d'un topic
- (dé)connexion

Ces envois consistent à produire un événement, à se connecter chez chaque client abonné (en dehors de l'émetteur du message), et à envoyer un message XML contenant l'événement sérialisé.

Message

Un message contient quatre « éléments » :

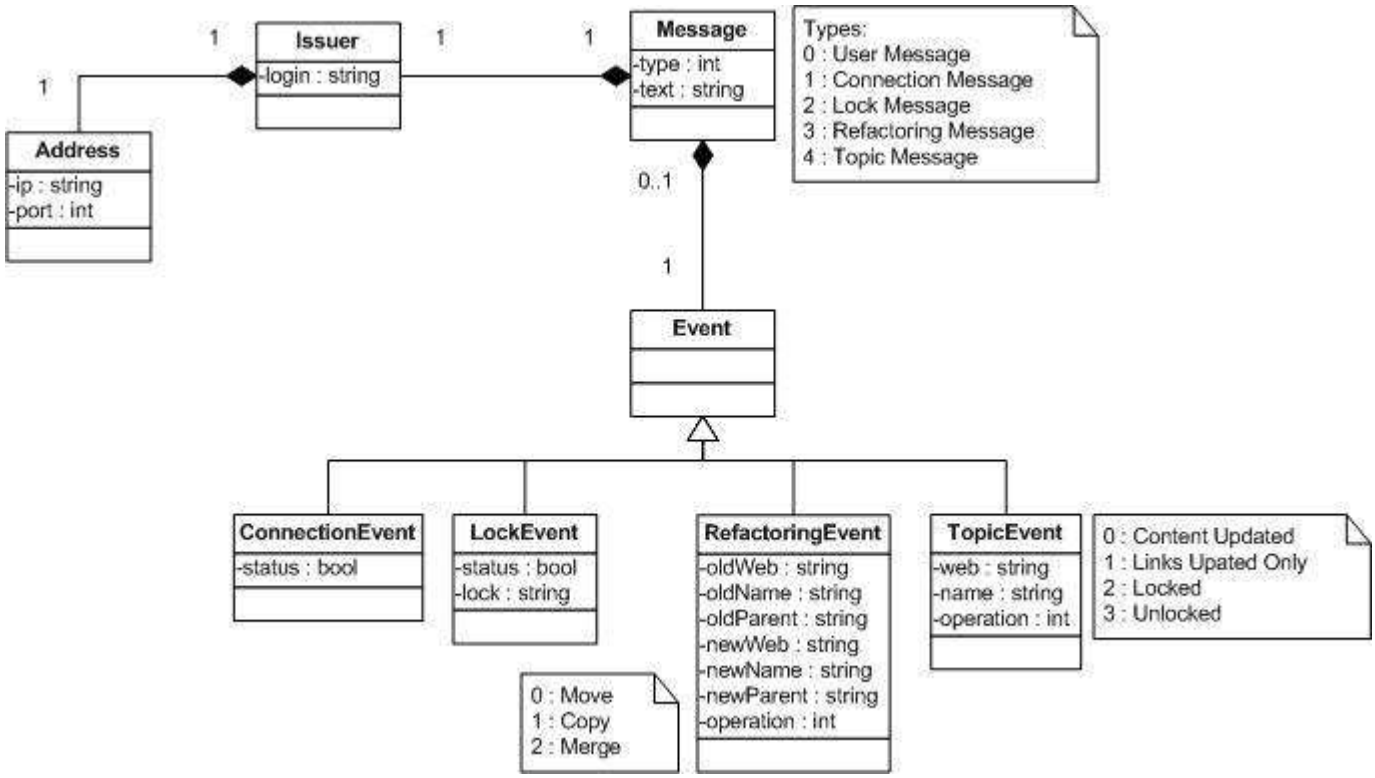
- un type
- une origine
- un événement (facultatif)
- un texte

Le type indique quel est le type du message, s'il concerne par exemple les connexions ou les opérations de refactoring. L'origine indique l'utilisateur émetteur du message, c'est-à-dire son login et s'il est inscrit au service, son adresse IP et le port sur lequel il écoute (afin éventuellement de permettre à quelqu'un qui reçoit le message de le contacter). Le texte est le contenu même du message. Un événement peut aussi y être

attaché afin de préciser ce qu'il s'est passé. Nous avons développé quatre objets Perl différents pour les modéliser :

- *ConnectionEvent* : événements de connexion
- *LockEvent* : événement lié à un (dé)verrouillage « général » (pas sur topic). Seul les opérations sur verrou administratif produisent ce type d'événement actuellement.
- *RefactoringEvent* : événement de refactoring sur topic
- *TopicEvent* : événement sur topic, c'est-à-dire modification, (dé)verrouillage ou mise à jour des liens

Voici un diagramme de classes de type UML spécifiant ce qu'est un message:



Sérialisation XML

Un message (ainsi que l'événement attaché) est sérialisé en XML afin d'être transmit au client. La sérialisation se fait par le biais de la librairie Perl *XML::TreeBuilder*. Le message suit un XML Schema précis qui est disponible dans les ressources du projet: *messages.xsd*.

Modules

Le module *Service::Notification* propose les services d'abonnement (*subscribe*) et de résiliation d'abonnement (*removeSubscription*), ainsi que la sérialisation du message même. Un répertoire *Notification* contient les objets (modules particuliers) Perl modélisant les événements.

Package client TWikiListener

Un package Java a été développé afin de faciliter l'interception de messages côté client. Ce package contient deux types de classes:

Messages

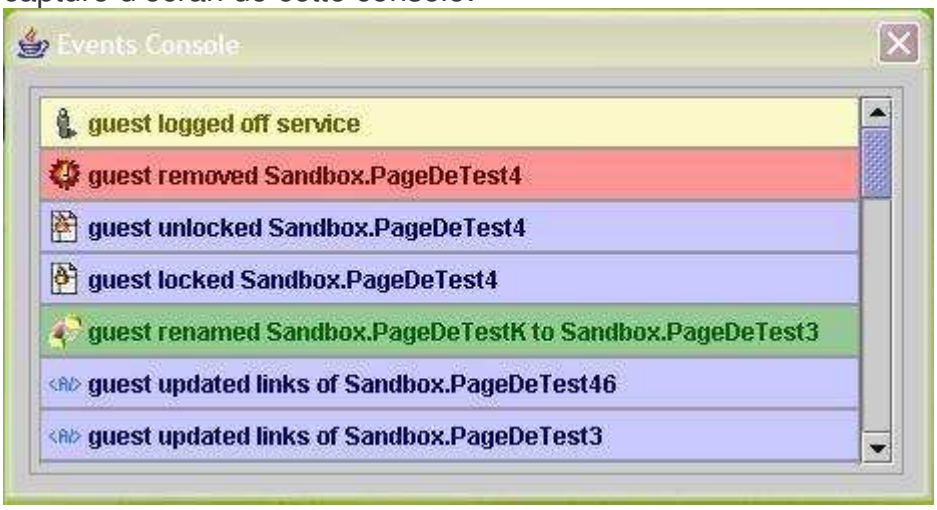
Des classes « mappées » sur les structures XML des messages attendus. Elles ont été générées à l'aide d'un « binding compiler » du projet Open Source *Castor*, à partir du XML Schema *messages.xsd*. Ce sont des instances de ces classes qui sont renvoyées au client.

Ecouteur : TWikiListener

Une classe d'objet qu'il faut instancier afin d'écouter le port que l'on a donné à l'abonnement. L'objet instancié fonctionnera selon le design-pattern observers/observable. A chaque interception de message, il enverra à ses observers le message reçu. La désérialisation (unmarshalling) se fait avec l'aide des bibliothèques de *Castor* et de *Xerces*. Ces librairies sont requises pour pouvoir utiliser *TWikiListener*.

Console Java

Une « console » Java a été développée afin de montrer l'utilisation du package *TWikiListener* et du service de notification développé. L'abonnement n'est pas géré par ce programme. Si cette console doit être utilisée dans une application, c'est cette dernière qui doit faire l'appel au service. Pour tester la console indépendamment, il faut passer par un programme tel que *Capescience NetTool* pour envoyer manuellement les requêtes SOAP. Certaines sont disponibles dans les ressources du projet. Voici une capture d'écran de cette console:



Seules les sources sont disponibles dans les ressources du projet. Cette console n'est pas destinée à être une application standalone mais plutôt à être incluse dans d'autres programmes telles que l'outil de refactoring.

Aspects fonctionnels

Hiérarchie des topics

Récupération des propriétés d'un topic

Ce service permet de récupérer les propriétés d'un topic à partir de son chemin. Ses propriétés sont récupérées directement dans les données meta du topic et sont envoyées par l'intermédiaire d'un tableau de chaînes de caractères. Les valeurs représentent respectivement dans l'ordre la taille, le nom du Web, le chemin, le topic parent, l'auteur, la date de création, la date de modification, la version, le format et le

droit. Pour ce dernier, 0 traduit le fait que l'utilisateur n'a aucun droit d'écriture, 1 qu'il a le droit de modification, 2 pour le droit de renomage et 3 pour le renomage et la modification. Ces droits sont calculés à partir de la clé donnée en paramètre, représentant l'utilisateur au niveau serveur.

Récupération des propriétés d'un Web

Ce service permet de récupérer les propriétés d'un Web, c'est-à-dire sa taille. Cette dernière n'est pas calculée récursivement, c'est-à-dire qu'elle fait la somme de toutes les tailles des topics et ajoute une constante pour chaque Web (elle ne calcule pas la taille des sous webs). Cette fonction utilise la commande *stat* d'Unix.

Représentation de la hiérarchie des topics

Le service *sameTopicParent* est le cœur de l'application de refactoring pour la représentation arborescente des topics. Il prend trois paramètres : le nom du topic parent, le nom du web pour définir d'où commence la recherche et la clé pour identifier l'utilisateur au niveau serveur. Suivant les paramètres, ce service réagit de façons différentes. Je vais énumérer les cas les plus intéressants :

- Si le nom du topic parent est «» , cette fonction renverra la liste des topics parent présents dans le Web donné en paramètre mais aussi dans ses sous webs. De plus il renverra la liste des topics qui ne sont ni des enfants, ni des parents. Cela à pour but de ne pas tout envoyer au client. Si un topic parent est lui-même enfant, il ne sera visible que lorsqu'on aura déployé son père.
- Si le nom du web est « » alors la recherche se fait dans toute la hiérarchie data de TWiki. En effet, un topic peu avoir son père dans un web qui ne se trouve pas dans sa même hiérarchie. Dans l'exemple ci-dessous, titi peut être le père de toto.
  Web A
  |____Web B___toto.txt
  |____Web C___ titi.txt

- Si le nom du web est spécifié, la recherché se faire suivant Nomtopic et Web.Nomtopic

Cette fonction est très gourmande en ressource, par le simple fait qu'elle doit constamment parser tous les topics du serveur pour récupérer les infos dont elle a besoin.

Récupération d'un contenu d'un Web

Ce service permet de connaître la liste des sous webs, topics présent dans le Web donné en paramètre. Il correspond à la commande « ls » du système Unix.

Opérations de refactoring

Toutes les fonctions dédiées aux opérations de refactoring suivantes ont été implémentées dans le module *Service::Refactoring*.

Renommage

Le renommage se fait au sein d'un même web. Le service *renameTopic* demande en paramètre quel topic renommer et quel nouveau nom lui donner. Après différents tests

(existence, verrouillage, autorisations, verrou administratif, nouveau nom WikiWord etc.), il fait appel à la fonction générale *TWiki::Store::rename* de TWiki, qui lui sert au renommage mais aussi aux déplacements. La mise à jour des liens est proposée en option.

Déplacement

On peut distinguer deux types de déplacements dans TWiki. Le premier est un déplacement réel qui consiste à changer les fichiers du topic d'emplacement dans le file system. Par « les fichiers », nous voulons dire le fichier texte qui constitue le topic, sa pile RCS pour gérer les versions et éventuellement ses attachements. Le deuxième déplacement, qui n'en est pas un, consiste simplement à changer une référence dans les « meta-informations » du topic. Le service de déplacement moveTopic demande en arguments le web et le nom du topic à déplacer, le web de destination et le topic parent auquel le rattacher. Le déplacement inter-webs sera lancé si le web source est différent du web destination. Les deux types de déplacement peuvent être combinées en fonction des arguments. La mise à jour des liens est proposée en option.

Déplacement inter-webs

Le déplacement inter-webs s'apparente à un renommage étant donné que la fonction « interne » à TWiki pour faire cela est la même (*TWiki::Store::rename*). Le nouveau nom est identique au précédent, mais les web source et destination sont différents.

Changement de parent

Le changement de parent consiste à changer un lien placé dans l'entête du topic. TWiki nous offre la possibilité d'ouvrir un topic et de récupérer de manière distincte ses meta-informations (modélisées par une table de hachage) et son contenu. Mettre à jour une meta-information consiste à changer une entrée de la table et à sauvegarder le topic.

Suppression

Plusieurs types de suppression sont proposés par le service *removeTopic*. La suppression « classique » de TWiki consiste à déplacer un topic dans le web « Trash ». Si un topic dans ce web a le même nom, la suppression échoue. Mais TWiki propose de donner un nouveau nom pour éviter cela. Quatre options de suppression sont proposées, certaines peuvent être désactivées depuis le fichier de configuration:

- Déplacer un topic dans le web « Trash » en renommant avec un nom demandé en paramètre
- Déplacer un topic dans le web « Trash » et générer automatiquement un nouveau nom si le topic existe déjà.
- Déplacer un topic dans le web « Trash » et supprimer celui qui y est éventuellement présent.
- Supprimer réellement un topic.

Les attachements sont gérés de manière « similaire », c'est-à-dire qu'une suppression « réelle » impliquera la suppression des attachements et un déplacement, un déplacement des attachements.

Fusion

La fusion est une concaténation de topics et/ou des attachements. Le service *mergeTopics* demande en arguments le topic cible et le second topic à partir duquel faire la fusion. Géré les attachements est proposé en option. Au cas où ce serait demandé et dans le cas peu probable ou deux attachements auraient le même nom sur les topics, un renommage est automatiquement fait. Ce service propose en outre de faire de la fusion un rapport indiquant explicitement à quel topic appartient un contenu.

Copie

La copie de topics peut s'apparenter à un déplacement dans l'idée. Concernant l'implémentation, ils sont très différents car les fonctions proposées par TWiki ne concernent que les déplacements, autant en ce qui concerne les topics eux-mêmes que les attachements. Là où moveTopic réutilisait des fonctions de haut niveau de TWiki, *copyTopic* gère les topics à un niveau plus bas (à l'échelle de la création, de la recopie de contenu et de la sauvegarde) et les attachements au niveau quasiment le plus bas (cf. fonction *Service::Topics::copyAttachmentFile*).

Mise à jour des liens

La mise à jour des liens se déroule en deux temps: rechercher les topics concernés par la mise à jour et la réaliser. La recherche se fait via la commande *egrep*, à qui on passe une expression régulière formatée selon le web et le nom du topic qui a changé. Tous les topics ayant un lien, que ce soit dans le contenu ou dans les « meta-informations » (donc les topics enfants) seront retournés. La recherche va se faire dans des emplacements particuliers, en fonction des paramètres passés. Cette première phase retournera un tableau de locations de topics (syntaxe *web.topic*). La procédure de mise à jour va prendre ce tableau en paramètre. Elle va tenter d'ouvrir chaque topic (s'il est verrouillé: échec et on passe au suivant), récupérer les « meta-informations » et mettre à jour le lien vers le topic parent si c'est nécessaire, effectuer du pattern-matching via expression régulière sur le contenu et le sauvegarder.

Rapatriement de topics vers l'éditeur

Création de topic

Le service *getTopic* permet de générer un objet *topic* à partir du topic donné en paramètre (par l'intermédiaire de son chemin). Il va directement chercher les informations écrites en dur dans les données meta du topic. Avant de renvoyer le résultat, les données sont formatées pour traduire les caractères spéciaux en Unicode afin d'être validées par tout parser XML. La clé donnée en deuxième paramètre permet de déterminer si l'utilisateur possède les droits nécessaire et si le topic n'a pas déjà de verrou administratif. (Un autre utilisateur fait des modifications dessus).

Création d'attachement

Le service *getAttach* permet de générer un objet *Attachment* à partir du topic et du nom de l'attachement donné en paramètre. Les propriétés de cet objet sont récupérées grâce aux attributs meta de l'attachement écrit en dur dans le topic. Il n'y a pas de vérification des droits, car on suppose que pour qu'un client possède le topic et le nom de l'attachement, il a du forcement appelé un service lui demandant sa clé, et donc vérifié ses droits. Il n'est donc pas nécessaire de vérifier de nouveau.

Récupération des attachements

Le service *getListAttach* permet de récuperer la liste des atatchement pour un topic donné. Pour ce faire, il recupere la liste des noms de fichiers directement dans les données meta ecrite en dur dans le topic. Le clé donnée en deuxieme paramètre permet de verifier si l'utilisateur possède bien le droit de lecture sur ce topic.

Listing

Le service *getAllTopic* permet de renvoyer tous les topics précèdent par leur web (web.topic) visible par l'utilisateur (identifié par la clé donnée en paramètre).

Sauvegarde de topics

Le service *setTopic* prend en argument un objet *topic*. Celui-ci dispose d'un nom, d'une location (le web), de propriétés, d'un contenu textuel et d'un tableau d'objet *attachment*; un objet *attachment* disposant lui-même de propriétés et d'un contenu, binaire celui-là. Côté client Java, les attachements et les topics sont réellement modélisés par des instances de classes. En XML, ce sont des structures complexes. Côté Perl, ils sont traduis en tables de hachage automatiquement. La sauvegarde de topic en elle-même est très simple étant donné que des fonctions de haut niveau sont disponibles dans TWiki. La difficulté réside dans l'upload d'attachements, gérés dans TWiki par le serveur Apache. Avant de parler de ceci, voici le « protocole » sur lequel se base la sauvegarde de topics à la réception :

- Un attachement du topic contient un contenu binaire : on ajoute/mets à jour l'attachement.
- Un attachement du topic est retourné mais ne contient pas de données binaires : cela ne sert que s'il existe déjà dans le topic sinon on ne fait rien. Si l'option *doKeep* est à false, on le garde. Sinon, si *doKeep* est à *vrai*, on ne fait rien.
- Le topic ne contient pas d'attachements : si *doKeep* est à *vrai*, on ne change rien, sinon on les supprime tous.

L'option *doKeep* à *false* permet de faire un comparaison sur les attachements présents et ceux retournés. Cela permet de les gérer. A l'heure où ces lignes sont écrites, *doKeep* est automatiquement mis à vrai car le rapatriement de topics ne gère pas les attachements. Concernant l'upload, les données binaires sont stockées en base64 dans les messages SOAP. A la réception, on récupère ces données et les écrits dans un fichier temporaire portant un nom lié à la clef de connexion (upload_*clef*). Ceci permet d'éviter des problèmes de concurrence entre uploads simultanés. Une fois ce fichier temporaire écrit, une fonction de haut niveau de TWiki permet de faire la sauvegarde à partir de celui-ci.

Génération de stubs client

Comme toute technologie liée aux applications réparties, les services Web requièrent des stubs afin que les clients les invoquent. Nous utilisons une procédure de génération automatique de stubs à partir d'un fichier de description (comme RPCGEN avec un fichier .x). Le fichier de description est une WSDL qui décrit les services Web en XML. Le générateur est *WSDL2Java* du framework *Apache WSIF*. A l'issu de cette génération, nous obtenons des stubs cablés vers l'adresse du serveur SOAP décrit dans la WSDL. Afin d'avoir des stubs dont l'adresse est paramétrable, une petite modification doit être

faite dans le fichier *ServiceLocator.java*. Au lieu d'avoir un attribut adresse *final*, nous devons faire un constructeur qui la prenne en paramètre. L'application, au lancement, devra la lui passer. Dans notre cas, nous la lui passons via un argument Java Web Start.