# Document de spécification des interfaces Web Service du suivi d'anomalie

| | |
|---|---|
| **Livrable du au titre du projet** | HELIOS |
| **Lot** | WP3 |
| **Tache** | 3.1 |
| **Livrable** | LI-15-RAP-2 |

| Rédacteur(s) | Vérificateur(s) | Approbateur(s) |
|---|---|---|
| O. Berger | | |

| Documents de références | |
|---|---|
| | Annexe technique du projet HELIOS |
| | |
| | |
| | |

| | |
|---|---|
| helios | Titre du document : Document de spécification des interfaces Web Service du suivi d'anomalie |
| | Référence : LI-15-RAP-2 |
| | Version 0.2 du 15/09/09 |

## Gestion des versions

| N° de version | Date | Auteurs | Modification apportées |
|---|---|---|---|
| 0.1 | 07/09/09 | O. Berger | Initialisation du document |
| 0.2 | 15/09/09 | O. Berger | Détail des besoins |
| | | | |
| | | | |
| | | | |

**Copyright : ©2009 Institut TELECOM, O. BERGER**

**Ce document est diffusé sous les termes de la licence XXX à définir XXX.**

**Ce document est le produit de travaux effectués dans le cadre du projet Helios, avec l'aide financière de la Région Ile-de-France.**

# Table des matières

1 Objectifs de ce document....3
1.1 Contexte : tâche 3.1 du projet Helios (Étude et modélisation)....3
1.2 Structure du document....3
2 Définition du besoin....5
2.1 Rappels des éléments définis dans l'annexe technique....5
2.2 Compléments à l'issue d'une première phase d'étude....7
2.3 Contextes d'application....7
2.3.1 Suivi générique de bugs saisis dans des bugtrackers externes....8
2.3.2 Suivi des liens entre bugs identifiés sur les mêmes paquetages logiciels....9
2.3.3 Miroir de bugtracker / duplications de bugs....10
2.3.4 Gestion de dépendances entre anomalies / composants logiciels....10
2.3.5 Gestion des branches de développement....11
3 Interfaces of a global bug monitoring system....13
3.1 Interfaces of the core generic system....13
3.2 First complementary applications....14
4 General architecture....16
5 Interfaces with bugtrackers....19
6 Integration with WP 2's documentation portal....20
7 Integration with the rest of the Helios platform....21

# 1 Objectifs de ce document

Ce document présente le deuxième résultat principal de la tâche 3.1 « Étude et modélisation » du sous-projet 3 (WP3) portant sur la « Traçabilité des anomalies entre plateformes de développement »

Le présent document (LI-15-RAP-2) fournit une **description des interfaces sous formes de Web Services du suivi d'anomalie**

**Ce document décrit un système générique de suivi de faits relatifs à des bugs saisis dans des bugtrackers externes qui sera développé dans le cadre du projet Helios. Ce système vise à aggréger dans une base de données unique et dans un format cohérent, dans une sémantique bien définie, un ensemble d'informations relatives au suivi et au traitement de bugs qui permette de fournir :**

- **des interfaces d'accès et de suivi des bugs selon différents critères propres aux utilisateurs (interfaces Web ou APIs)**
- **un support pour l'expertise collaborative autour des bugs facilitant un certain nombre de tâches relatives à la maintenance des paquetages logiciels (notamment via des fonctions d'annotation sémantique)**
- **une support pour la réalisation de premières applications utilisatrices, telles une version plus générique de l'outil bts-link, et des modules de suivi des bugs entre dépendances logicielles et au long d'un arbre de versions, intégrés dans la plate-forme Helios (WP4)**

**Une description de l'architecture générale et d'un certain nombre de contraintes de réalisation sont également fournies.**

Ce document fait suite au document LI-15-RAP-1 qui fournit une **spécification d'un modèle générique de suivi d'anomalie.**

## 1.1 Contexte : tâche 3.1 du projet Helios (Étude et modélisation)

La tâche 3.1 du projet Helios a pour objet une étude visant à définir une « *Spécification de modèle générique de suivi d'anomalie et une description des interfaces sous formes de WebServices du suivi d'anomalie* » (cf. 2.1 pour un rappel de l'ensemble du descriptif de cette tâche).

## 1.2 Structure du document

Ce document est subdivisé en plusieurs chapitres :

1. Définition des besoins en matière de suivi d'anomalies et des contextes d'application étudiés

2. Définition des interfaces et des fonctions détaillées attendues d'un système de suivi global des bugs
3. Une description de l'architecture générale du système
4. Une ébauche de liste d'interfaces avec des bugtrackers existants
5. Une description des éléments relatifs à l'intégration avec les autres développements du projet HELIOS

**Avertissement :**

**Ce document technique est destiné à une audience de spécialistes supposés apte à la lecture de documentation en anglais.**

**Certaines sections du document sont donc rédigée *en anglais*, afin de faciliter sa diffusion directe auprès des communautés internationales concernées (dévelopeurs d'outils de gestion de bugs, communautés Open-Source, contributeurs tiers...).**

# 2 Définition du besoin

En introduction de ce chapitre, nous rappelons la définition de tâches définie dans l'annexe technique pour le WP3 « Traçabilité des anomalies entre plateformes de développement ».

Cette définition est ensuite complétée pour présenter l'état des besoins en matière de gestion de bugs et une description globale du périmètre des applications à développer dans le cadre du WP3.

## 2.1 Rappels des éléments définis dans l'annexe technique

L'annexe technique mentionne (nous soulignons en gras les éléments essentiels) :

> *« Une application construite à base de composants Open Source est le résultat du travail de plusieurs groupes qui utilisent chacun leurs propres environnements de développement de type forge (développeurs des briques de base, développeurs du logiciel et développeurs de la ou des distributions). Ceci est vrai pour chaque version de l'application. Dans ce contexte il est difficile de* ***savoir si un bug (anomalie) identifié par une version remonte suffisamment loin puis redescend dans toutes les versions****. De même il est difficile d'****assurer qu'une correction de bug dans une bibliothèque est reportée dans tous les logiciels qui l'utilisent****. Les processus de remontée de bugs sont, à ce jour, manuels. Ce WP s'attachera à proposer des mécanismes d'interaction associés à ces traitements et à* ***permettre la traçabilité de ces interactions pour faciliter le suivi du traitement de bug*** *par l'ensemble des groupes concernés. Les mécanismes proposés seront intégrés dans la plate-forme d'industrialisation pour faciliter les* ***interactions entre les groupes travaillant autour des mêmes logiciels****.*
> *Objectif :*
> *L'objectif de ce WP est de* ***développer un outil de synchronisation entre bug-trackers*** *(outils de suivi d'anomalies) en prenant en compte les bug-trackers les plus couramment utilisés et Open Source, à savoir Bugzilla, Mantis, Trac entre autres.*
>
> [...]
>
> *<u>Tâche 3.1 : Étude et modélisation</u>*
>
> *Cette étude proposera non seulement un modèle générique d'anomalie, mais aussi des interfaces sous forme de Web Services pour automatiser l'enchaînement des traitements associés aux anomalies. La description des inter-relations entre briques logicielles telle qu'on peut la trouver dans les gestionnaires de paquet logiciel utilisés dans les distributions libres (rpm, et deb) sera utilisée pour* ***faciliter la recherche des relations entre les différents niveaux auxquels un bug peut s'exprimer****. Cette étude proposera des scénarios pour analyser la* ***circulation de l'information relative aux traitements des erreurs*** *dans le cas de* ***circulation ascendante*** *(d'une brique utilisatrice vers une brique de base),* ***descendante*** *(d'une brique de base vers une brique utilisatrice), ou* ***horizontale*** *(entre briques de même niveau qui utilisent les mêmes briques de base). Ce modèle peut s'appuyer sur des études existantes comme Beatle ou les formats utilisés par des logiciels tels que*

*ScmBug.*
*L'un des livrables de cette tâche correspond à une proposition de norme de description d'un bug, ainsi que des interfaces d'interaction avec un gestionnaire de bug. Certains résultats du projet européen Edos pourront être réutilisés.*
*Livrable : LI-15-RAP (dû à T0+8) Spécification de modèle générique de suivi d'anomalie et une description des interfaces sous formes de WebServices du suivi d'anomalie*

*Tâche 3.2 : Développement d'un mécanisme de synchronisation entre bug-trackers.*

*Ce mécanisme devrait* ***permettre à un bug-tracker de se synchroniser avec un second bug-tracker sur certaines anomalies bien identifiées****. Les bug-trackers concernés par cet outil sont les suivants :*

- *Bugzilla*
- *Mantis*
- *Trac*
- *Bug-tracker LibreSource*

*Les interfaces définies dans les tâches seront réalisées dans au moins deux systèmes de suivi de bugs utilisés par les communautés de développement des logiciels libres en particulier ceux qui sont associés avec des plates-formes de développement de logiciels telles que LibreSource et Gforge. L'interconnexion entre ces deux systèmes sera la preuve de la faisabilité du mécanisme. Les connecteurs construits selon ce modèle et intégrés aux logiciels libres utilisés seront développés en collaboration avec la communauté correspondante afin d'en faire profiter le plus grand nombre rapidement. Les connecteurs associés à des systèmes de Wiki seront aussi réalisés et diffusés.*
*Livrable : LI-16-LOG (dû à T0+19) Logiciel correspondant à une implémentation du mécanisme de synchronisation dans deux bug-trackers. Le système sur lequel cette implémentation sera réalisée dépend des choix du WP1.*

*Tâche 3.3 : Interaction avec les autres WPs*

*Réalisation d'une interface de* ***gestion des bugs au sein du portail d'industrialisation*** *(voir WP4). Cette tâche doit notamment permettre de sélectionner un composant pour t****ransférer le bug à la communauté concernée et marquer le bug comme transféré****. Cette anomalie sera bien évidemment* ***transférée puis synchronisée*** *grâce à l'outil développé dans la tâche 3.2.*
*Cette tâche s'intéresse également aux possibles déclinaisons du modèle de suivi de bug dans le cadre de la gestion de la qualité documentaire. En particulier, elle investiguera les possibles* ***interactions entre des systèmes de type Wiki et des systèmes de type suivi des erreurs****.*
*Livrable :*

- *LI-17-LOG (dû à T0+21) Logiciel correspondant à l'intégration des résultats de la tâche 3.3 dans le WP4*
-

*Livrables finaux :*

- *LI-18-RAP (dû à T0+21) Spécification de modèle générique d'anomalie reprenant le livrable de la tâche 3.1 définie ci-après,*

- *LI-19-RAP (dû à T0+21) Documentation des interfaces d'un service de suivi des anomalies reprenant le livrable de la tâche 3.1 définie ci-après,*
- *LI-20-LOG (dû à T0+21) Logicielle sous forme de code source correspondant à l'implémentation d'un mécanisme de synchronisation entre bug-trackers prenant en compte deux bug-trackers reprenant le livrable de la tâche 3.2 définie ci-après,*
- *LI-21-LOG (dû à T0+21) Logiciel sous forme de code source correspondant à l'intégration d'un service de suivi des anomalies dans le WP4 reprenant le livrable de la tâche 3.3 définie ci-après, »*

## 2.2 Compléments à l'issue d'une première phase d'étude

A l'issue de la tâche 3.1 de spécification, il nous apparaît nécessaire de clarifier certains points et de reformuler les objectifs de l'étude, afin de mieux répondre aux problématiques rencontrées. Cela permettra de définir précisément le périmètre des développements à réaliser dans le cadre du WP3, ainsi que de développements complémentaires envisageables.

La notion de synchronisation entre bugtrackers nécessite d'être explicitée. Nous l'entendons comme étant l'ensemble des mécanismes qui peuvent être mis à disposition des « utilisateurs » (utilisateurs de logiciels ou développeurs/mainteneurs), dans le cadre des processus de gestion (de rapports) d'anomalies, afin de leur faciliter la tâche de suivi des liens existants entre différents rapports de bugs, et des mises-à-jour éventuelles nécessaires de l'état de ces rapports dans les bugtrackers.

Ce type de besoin peut exister par rapport à différents processus, et nous ne prétendons pas couvrir tous les cas possibles dans le cadre du projet HELIOS.
Nous nous attacherons donc à répondre à certains besoins précis identifiés plus bas, et à fournir une technologie permettant une extension facilitée pour répondre à d'autres besoin, au-delà du projet ou des fonctions envisagées ici.

Le cœur de cette technologie consistera en **un système de suivi de bugs global** permettant de modéliser, au sein d'un même référentiel, un ensemble de propriétés relatives à des rapports de bugs physiquement présents dans leurs propres bugtrackers, et permettant de faire fonctionner **différentes applications** de suivi des évolutions de ces propriétés, relativement aux liens existants entre ces bugs, spécialisées, et répondant chacune à des objectifs particuliers.

Nous décrivons ci-après un ensemble de telles applications, mais seulement certaines d'entre-elles seront réalisées dans le cadre du projet.

## 2.3 Contextes d'application

Nous listons différents contextes d'utilisation envisagés pour les technologies développées dans le contexte défini ci-dessus.

Certaines applications seront développées dans le cadre du projet car s'articulant particulièrement

bien avec les autres outils développés dans le projet, notamment vis-à-vis d'un contexte d'Application Lifecycle Management, et relativement aux processus de maintenance de logiciels opensource. D'autres sont listées ci-dessous à seule fin d'illustration des applications potentielles des technologies développées.

### 2.3.1 Suivi générique de bugs saisis dans des bugtrackers externes

Au sein d'un projet de développement particulier, doté d'un bugtracker configuré pour ses besoins, le cycle de vie des rapports de bugs est imposé par ce couple bugtrackers / projets. Seuls des utilisateurs autorisés ont les prérogatives pour gérer les évolutions de certaines propriétés des rapports de bugs (statut notamment : ouvert / fermé, etc.).
Par contre les soumissionnaires des rapports de bugs extérieurs au projet peuvent avoir leurs propres processus de gestion des rapports d'anomalies constatées, sans qu'ils puissent nécessairement décider des étapes suivantes du cycle de vie des bugs qu'ils ont soumis.
Les délais dans lesquels la prise en charge des bugs est assurée n'est en général pas garantie une fois les bugs saisis.

Les personnes extérieures aux groupes d'utilisateurs autorisés des bugtrackers ont ainsi la nécessité de « resynchroniser » leurs processus propres avec les processus en cours dans des projets externes.

Aujourd'hui un tel suivi est essentiellement géré de façon manuelle, reposant essentiellement sur les fonctions de notification (par mail) envoyées par les bugtrackers aux personnes s'étant abonnées à certains bugs (dont les soumissionnaires).

Des applications de *recherche*, *d'annotation* et de *notification* de changement de certaines propriétés (notamment le statut représentant les différents états du cycle de vie des bugs) seraient ainsi nécessaires, afin de permettre aux soumissionnaires de surveiller des listes de bugs qui les intéressent, selon leurs critères personnels (ou par extension pour des groupes d'utilisateurs partageant les mêmes critères : processus de projets).

Une application sera réalisée afin de permettre la **collection et la restitution de listes de bugs à partir de recherches multi-critères sur leurs propriétés[1].** Cette application gérera :

- les *propriétés intrinsèques*, telles que définies dans les bugtrackers,
- ainsi que des propriétés définies par l'utilisateur lors d'*annotations sémantiques* de rapports de bugs préalablement identifiés.

Une application d'**annotation sémantique de rapports de bugs externes[2]** sera ainsi développée.

Une **application de notification** sur la base des **changements constatés dans les résultats de recherche multi-critères[3]** sera réalisée. En complément de l'application de restitution décrite ci-dessus (pull), des alertes (push) pourront être transmises (par mail / flux) par un système de

1 Feature 1 : retrieval of aggregated lists of bugs based on multi-criteria queries
2 Feature 2 : semantic annotation of external bug reports
3 Feature 3 : notification of changes

vérification périodique (selon des critères programmés à l'avance). L'utilisateur pourra ainsi être informé automatiquement quand certains critères de recherche renverront des résultats différents de lors de la dernière exécution.

Ces applications sont génériques, et permettent de répondre à de nombreux cas d'utilisation plus particulier, sur la base de la recherche de certains critères sémantiques précis liés aux rapports de bugs.

Nous détaillons ci-après différentes cas d'utilisation ou de spécialisation de ces applications.

### 2.3.2 Suivi des liens entre bugs identifiés sur les mêmes paquetages logiciels

Les applications présentées ci-dessus peuvent être mises en pratique dans le contexte de la diffusion et de l'utilisation de paquetages logiciels.
Différents groupes peuvent s'intéresser au suivi des rapports de bugs soumis sur un même paquetage, mais dans différents bugtrackers. C'est utile aux utilisateurs des paquetages en question, aux mainteneurs/packageurs des mêmes paquetages pour différentes distributions, aux développeurs amont qui veulent un retour depuis les distributions.

Les recherches peuvent ainsi porter sur :
- l'apparition de nouveaux bugs saisis,
- ou fermés,
- des patches soumis pour corrections,
- l'identification des doublons,
- l'identification de liens entre bugs dépendants les uns des autres (correction dépendant d'une autre correction),
- analyse des traces identiques, etc.

Ces différents éléments peuvent s'exprimer par différentes propriétés ou liens (sémantiques) extraits des bugtrackers ou définis par les utilisateurs.

Les entités principalement concernées dans ce type d'utilisation sont les paquetages logiciels et les bugs (cf. « spécification de modèle générique de suivi d'anomalie »)

Nous n'envisageons pas, dans le cadre du projet, de mécanismes d'automatisation ayant un impact sur les bugtrackers pour changer l'état des bugs concernés : le système se contente d'assister les experts concernés (par la facilitation des recherches ou par des alertes).
Les utilisateurs restent experts et souverains dans la modification des bugs dans les différents bugtrackers concernés.

Dans ce contexte, une application dérivée du système de recherche et de notification générique sera réalisée, dans le cadre du projet, afin de **supporter le fonctionnement de l'outil bts-link et pour le rendre plus largement utilisable dans n'importe quel projet d'empaquetage**[4]. bts-link notifie les mainteneurs des paquetages de distributions (actuellement uniquement pour Debian) des évolutions survenant sur les bugs des projets amont correspondants (tels qu'ils ont été mis en relation par ces

4 Feature 4 : more generic bts-link tool

mêmes mainteneurs). L'objectif est de le rendre encore plus indépendant du bugtracker sous-jacent afin qu'il puisse également être utilisé par d'autres projets en dehors de Debian.

Il existe de nombreuses distributions réalisant l'empaquetage des mêmes logiciels, donc une masse importante de bugs potentiellement dupliqués (en plus de ceux du projet amont), rapportés individuellement dans chaque bugtracker de distribution par sa communauté d'utilisateur, et qui gagneraient à être correllés entre distributions pour maximiser les chances d'amélioration de l'expertise collaborative (triage, expertise) autour de leur résolution.
La capacité d'annotation sémantique par l'ensemble des utilisateurs pour indiquer des liens entre de tels bugs devrait permettre d'envisager le signalement de ces bugs aux mainteneurs (seuls utilisateurs « autorisés » des bugtrackers), et d'élargir ainsi la capacité d'expertise à un plus grand nombre, afin de leur faciliter la résolution des problèmes. Mais nous ne développerons pas, dans le cadre du WP3, une exploitation de tels liens, réservant de tels développement applicatifs dans le futur (ou via des contributions externes).

### 2.3.3 Miroir de bugtracker / duplications de bugs

Dans le contexte particulier d'un projet, il peut s'avérer nécessaire d'effectuer une distribution d'une même ensemble de rapports de bugs dans plusieurs bugtrackers différents, afin de maintenir des miroirs de bugs, par exemple pour diminuer la dépendance vis-à-vis des liens réseau, ou pour des questions de gestion de projet globale (fork), ou pour certains besoins de confidentialité (bugs de sécurité, etc.).

Il nous apparaît cependant que de tels besoins mettent essentiellement en évidence la carence de certaines fonctions dans les bugtrackers utilisés (bugtrackers distribués, confidentialité de certains attributs, fonctions d'import/export, etc.).

Bien qu'on puisse envisager d'utiliser les technologies développées dans le cadre du projet pour répondre à ce type de besoin (notamment le format d'échange standard basé sur une ontologie générique et RDF), nous ne nous y consacrerons pas dans le cadre du WP3.

### 2.3.4 Gestion de dépendances entre anomalies / composants logiciels

Le graphe de dépendance entre composants logiciels (du point de vue de l'architecture applicative) implique un graphe de dépendance de certains rapports de bugs portant sur différents composants de la même application :

- code dupliqué (certaines applications embarquent des éléments de code identique),
- utilisations de bibliothèques (différentes applications nécessitant une nouvelle fabrication pour prendre en compte la dépendance statique ou dynamique par rapport aux évolutions de la bibliothèque), etc.

Ce type de dépendance est particulièrement courant quand un bug est signalé pour une application finale, mais correspond en réalité à un problème présent dans une des bibliothèques utilisées par l'application, et doit donc être résolu par une autre équipe de maintenance. Une stratégie pour la gestion de tels bugs peut consister à réaffecter le bug vers le composant « fautif », ou à saisir un

nouveau bug pour ce composant, et à garder un lien de dépendance entre celui saisi originellement sur l'application et le nouveau bug ainsi créé sur le composant.

Autre cas où ces liens entre bugs sont souvent mis en évidence : le cas de failles de sécurité présentes du fait de la mauvaise utilisation d'une API, et nécessitant un correctif dans toutes les applications utilisatrices de la bibliothèque fautive. Les alertes de sécurité doivent tracer explicitement ce type de lien afin de gérer correctement la gravité des anomalies et de rendre prioritaire la diffusion de versions corrigées de tous les composants impactés.

Le couplage entre les modèles de bugs et les modèles de définition des dépendances applicatives sera utilisé pour construire **une aggrégation des listes de bugs identifiés sur l'ensemble des dépendances d'un composant logiciel**[5], dans le cadre de l'intégration des technologies développées avec le reste de la plate-forme d'intégration du WP4.

### 2.3.5 Gestion des branches de développement

Le suivi des anomalies au cours de la fabrication et de la publication des versions successives d'une même application ou d'un même composant logiciel peut obéir à différentes stratégies de gestion de versions et de publication.

Différents modèles de branches, de versions (stables, de qualification, mineures, etc.) peuvent être utilisés pour l'application d'une méthodologie d'assurance qualité. Il n'existe pas un seul modèle général pour ce faire, ni une seule façon de gérer le couplage entre les anomalies constatées et les versions dans lesquelles elles sont présentes ou corrigées ( et donc les tests de vérification de correction, de non-regression, etc.).

Ce couplage entre versions et anomalies peut être géré différemment dans les bugtrackers, avec, par exemple, l'une des deux stratégies suivantes :

- les rapports de bugs sont saisis une seule fois dans le bugtracker, pour une même anomalie, et des liens entre versions présentant les dysfonctionnements, ou comportant un correctif sont gérés par le bugtracker (ainsi qu'une gestion des suivis des arbres de versions successives), par exemple dans debbugs, dans le cas de la distribution Debian.
- ou, à l'opposé, l'ensemble des bugs toujours présents dans une version donnée est dupliqué lorsqu'est créée la version suivante, donnant lieu à l'existence d'ensembles de bugs distincts pour chaque version. Chaque test effectué permet de clôre un bug et un seul. Des bugs correspondant à un correctif similaire, appliqué sur deux branches distinctes pour corriger une même anomalie constatées dans ces deux branches, seront gérés de façon individuelle.

Le **couplage entre bugs et versions sera géré dans le cadre du projet, aux besoins de l'intégration dans le cadre de la plate-forme d'intégration du WP4**[6], mais sans envisager des fonctionnalités applicatives génériques permettant d'effectuer du suivi entre versions dans tous les cas possibles, pour tout modèle de suivi de version.

5 Feature 5 : Component dependencies based bug monitoring
6 Feature 6 : versions – bugs links monitoring

On peut noter que les bugtrackers distribués à l'étude actuellement, qui visent à gérer le code et les bugs dans les mêmes référentiels distribués (type bzr ou git), comme bugseverywhere, ou dist-bugs, ne font pas l'objet de notre étude, n'étant pas suffisamment développés actuellement pour avoir fait l'objet d'une évaluation dans le cadre de la définition d'un modèle générique d'anomalie.

# 3 Interfaces of a global bug monitoring system

As explained in the previous section, we intend to develop, as part of WP3 a generic-purpose core system that will help to access and monitor bugs that are managed in external bugtrackers, in order to offer a services in the form of Web Services accessible by applications, or directly as a Web interface.

We'll develop also a set of first applications using this system in order to fulfill some particular needs. Some of them will be integrated together in the default Web interface, and some may be provided as other distinct Web applications.

## 3.1 Interfaces of the core generic system

Many of the system's expected features may be better understood with an analogy to RSS feeds: the system would be a collaborative web-based RSS aggregator like Google's reader or FriendFeed.

The following features are identified for such a system (most of them will be identified as relating to *feature 1* “retrieval of aggregated lists of bugs based on multi-criteria queries” described in the previous section) :

- collect facts about bugs managed by remote bugtrackers, to add them to a common bug-related facts/properties database:
  - either using custom retrievers for each tool's interfaces,
  - or using standard interchange formats (like the proposed standard based on RDF and EvoOnt BOM ontology);
- handle the refreshing / purging of these bug facts so that they're reasonably new and thus useful for users of the system
- offer a semantic query interface on the database of bug facts :
  - to local users in Web UI,
  - or to tools through an API, for instance through SPARQL language queries
  - about :
    - facts retrieved from the cache of bugs retrieved from the remote bugtrackers
    - and annotations provided by users,
    - but also on other metadata of the system;
- provide an API for submitting semantic annotations on bugs (by users of the system)
- provide feeds (for instance in the form of RDF feeds containing RSS 1.0 plus other ontologies) corresponding to the queries interface, to be consumed by other applications
- provide AJAX compatible server-side representation of facts from the database (lists and individual resources properties) to be consumed by other client-side Web applications

- provide a Web interface for users of the system in order to render contents of the system's database for selected registered users :
  - displaying contents of the database of bugs facts
  - displaying users related informations (“my bugs” like view)
  - displaying informations on bugs related to software packages (“software package” like view)
  - displaying additional community-oriented features (reputation of users, etc.)
- provide a Web interface to create semantic annotations on entities of the database (bugs, packages, etc.) (corresponding to feature 2 “semantic annotation of external bug reports” of the previous section);
- provide a notification system (corresponding to feature 3 “notification of changes” of the previous section) :
  - directly in the web interface or through other communication mechanisms (feeds, email, etc.)
  - concerning linked bugs (status monitoring ala bts-link)
  - concerning certain alerts configured by users as queries, with corresponding thresholds verifications

The Web interfaces related to user's preferences views (“my bugs”) will help provide a dedicated web application acting as a central proxy to the various bugtrackers for many users, helping them retrieve and monitor the bugs they care for, independently of the bugtrackers where they were submitted and are managed. This dedicated web application will be deployable on a specific domain on the Web, as part of WP3.

The APIs and/or the AJAX compatible interfaces will help provide the necessary data to be rendered in the bugs facets of the documentation portal of the Helios platform implemented as part of WP 2 (2.2 sub-part).

The annotations may be provided by the WP2's *doc4* components of the Helios platform, or by client tools in relation to the Nepomuk project.

The RDF feeds of the query APIs will be available to be used by other applications, either Web applications or desktop clients.

## 3.2 First complementary applications

A complementary notification application will be developped, which will use the interfaces of the core system to provide a *bug status* monitoring system, for the specific case of interlinked bugs that live along the release/packaging streams (Cf. Feature 4 “more generic bts-link tool” in the previous section).

Two other particular applications will be developped in order to integrate with the rest of the Helios

integration platform (Cf. WP4), to provide :

- dependent bug monitoring features based on component dependencies (see feature 5 in previous section),
- bug tracking along version trees for the same components (see feature 6 in the previous section)

Other applications may be developped by third parties, exploiting the interfaces of the core system, in order to fit their particular needs.

# 4 General architecture

The following diagram (Figure 1) illustrates the architecture of the system.

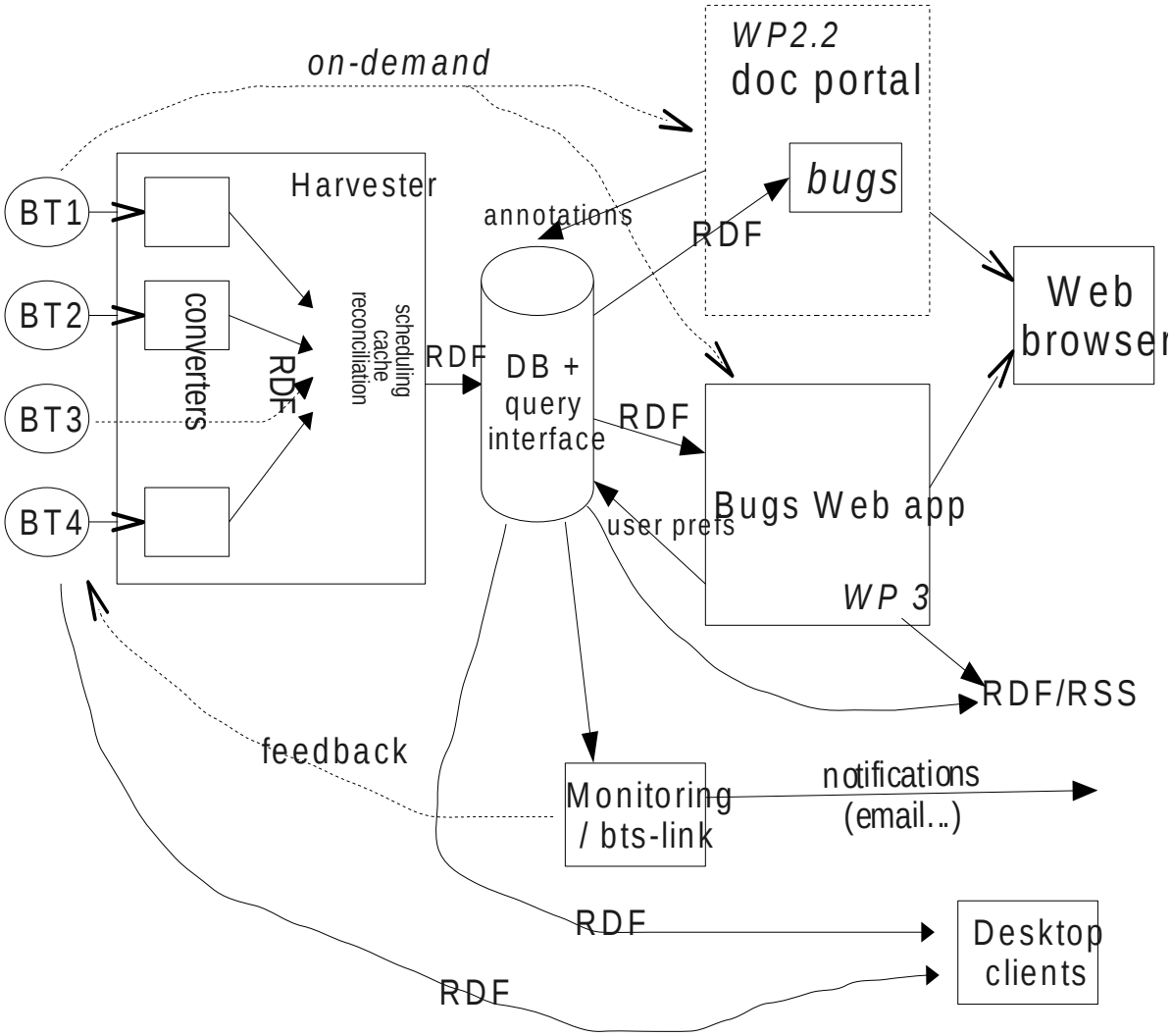


*Figure 1: General architecture diagram*

The system contains the following components :

- a harvester system that can :
  - launch harvesters and converters that can periodically collect data on the bugtracker interfaces, and save it to a sandard format (RDF)

  - schedule the harvesting by the harvesters depending on data freshness in the DB's cache
  - reconciles parts of data between the different bugtrackers that refer to same entities (packages, users, bugs, etc.)
- a facts database, storing semantic data for :
  - bug facts
  - people facts
  - software package facts
  - annotations on these
  - administrative meta-data for the system (dates of collection, source of data, etc.)
- a query interface for the database :
  - supporting the SPARQL language for semantic queries,
  - and outputting RDF results to be consumed by :
    - other components of the system containing
    - other applications (for public data)
- a monitoring system that can :
  - perform monitoring of facts in the database in order to notify interested parties of some detected situations,
  - prepare feedback (for instance based on analysis of data or annotations) to be provided back to bugtrackers
- A Web application available for users on the Web (standalone) which can :
  - allow users to access the whole database to retrieve bugs they are interested in
  - perform additional custom queries to bugtrackers
  - store user preferences and User Generated Content (annotations, co-marking, etc.) to the database
  - provide RDF/RSS feeds corresponding to their queries/preferences for export to user's applications
- Web components that can :
  - render data retrieved on-demand from bugtrackers or from the database cache
  - be embedded into the documentation portal for bugs visualisation
  - provide annotations back to the facts database

Additional considerations :

- Some of the components developped for the system may also be reused in desktop systems. For instance, the storage, caching, query and annotation mechanisms may be also useful on

client desktops for offline processing and advanced bug management. Such interfaces may benefit from compatibility with the Nepomuk technology for desktop integration, or with Mylyn for integration with the Eclipse IDE.

- Some Web interfaces could be developped as AJAX technology, allowing the embedding of bug representation widgets in other applications (Web or desktop/client).

# 5 Interfaces with bugtrackers

Some bugtrackers offer APIs for interfacing with them, like Mantis' SOAP interface.

Some bugtrackers offer native export in structured formats like Bugzilla's XML (and RDF, although not really satisfying as of bugzilla 3.4) feeds, to retrieve lists of bugs or individual bugs descriptions.

They may not always offer full access / export of all the bugs properties, and no unified format exists for doing so.

Some bugtrackers need to be accessed with “web-scraping” harvesters that parse the content of the Web interfaces of the bugtrackers (as is done in some Mylyn connectors, or in bts-link).

There are lots of libraries or tools interfacing with different bugtrackers, which may be reused to implement the bug harvesters :

| Tool / library | Programming language | Supported bugtrackers | | | | | |
|---|---|---|---|---|---|---|---|
| | | Bugzilla | Mantis | Debbugs | Launchpad | Fusionforge | Trac |
| bts-link | Python | | | | | | |
| | | | | | | | |
| | | | | | | | |

XXX TODO : to be completed XXX

The individual bugtracker converters may all output complete or partial RDF resource descriptions for the retrieved bugs, which could then all be aggregated into the same database.

Note that not all attributes may need to be converted for the system to still be able to operate. Additional attributes may be retrieved on-demand when necessary, as long as key core attributes about bugs, people and software package have been collected.

Still, some complete converters may be necessary, for instance to offer additional migration / replication services, that may also be needed for integration with the rest of the Helios platform.

# 6 Integration with WP 2's documentation portal

Bugs are just one of the many entities that participate to the documentation of software components/packages. Some facts known by the generic bug monitoring system (mainly links between bugs and package versions) will be particularly interesting to WP2's doc4 heterogenous documentation portal.

Any view / query result about bug facts made on the bugs facts database (through the interfaces presented in 3.1 ) may be integrated/replicated into the WP2 portal in this respect.

Query interfaces and annotation interfaces will be developped jointly between WP2 and WP3.

The WP2 portal and some of the WP3 Web Uis will be developped using the same Xwiki-based technology.

# 7 Integration with the rest of the Helios platform

The design of the system proposed above is targetting a general case where such a system may be deployed as a standalone application (or, for some components as part of the WP2's documentation portal).

It may also be deployed as a particular instance running in the frame of the Helios platform, in particular in order to monitor bugs of the platform's bugtracker (Mantis), for users of the platform (shared user directory), and concerning software components of projects instanciated in Helios (to track component dependency and version trees in the bugs database).

As a consequence, the Mantis harvester components may be covering the most complete set of attributes of Mantis bugs.

The sharing of the same semantic database with other tools of the platform may help integration (see WP4).

The sharing of the system's metadata model may also help integration (see WP4).