


| | | | |
|---|--|-----------------------------|-----------------------|
|  Compte rendu | Réunion 2020.07.04 (Dév-Inf.) <i>(clarification code Java, cahier des charges)</i> | Date création 29/08/2020 | Référence 32CR.026 |
| | | Dernière modif. 27/10/20 | 5 page (s) |

Rédaction : Papa Souleymane Ndiaye (PSN)

Révision : Jean Le Fur (JLF)(22.09.2020)

Mots-clés : [application CI](#), [application CI \(architecture\)](#), [cahier des charges](#), [compte-rendu](#), CI 3.0, Dév-Inf,

Résumé : présentation de la structure du logiciel Java et de son architecture data-business-presentation,

Présidents de Séance : JLF

Secrétaires de Séance : PSN

Présents : Moussa Sall, Birahime Fall, PSN, JLF

Excusé : Néant

Destinataires : équipe CI-SanarSoft

Réunion tenue sur Skype

La réunion du département Dév-Inf. s'est tenue le samedi 04 Juillet 2020 sur Skype. Elle a démarré à 8h 30 heure Sénégal et s'est terminée à 10h 07.

Ordre du jour

1. Clarification code Java / PHP : présentation
2. Premières actions à faire et planning
3. Accès restreint au site

Clarification code Java / PHP : présentation

Dans ce point Jean a commencé par expliquer à nouveau le principe général du projet CI-SanarSoft. Il a rappelé dans sa présentation que :

Primo, le projet CI-SanarSoft en tant que tel repose sur un logiciel qui est open source destiné à des gens qui n'ont pas besoin de prestation de service sur ce logiciel et par conséquent il faut leur proposer un logiciel garanti et de qualité.

Secundo, SanarSoft se propose d'offrir une prestation de service sur ce logiciel qui doit être fiable et costaud et donc l'objectif est de rendre l'application costaud et fiable justement.

Après la présentation du principe général du projet il a ensuite poursuivi en expliquant [les cinq composantes de l'application CI](#). Ainsi il est revenu sur chaque composante de l'application. Cependant les composantes sur lesquelles il s'est le plus appesantit sont celles que le département Dév-Inf. dirigé par Birahime doit travailler pour les mettre de qualité. Il s'agit de :

La composante masque de saisie : qui est à sa version 3.6 car étant déjà débogué il ne reste qu'à rendre propre son code source pour aboutir à sa version 7.0 à mettre open source (clarification à faire après, une fois qu'on sera au point sur le code Java),

La composante base de données : qui est terminée on n'y touche pas

Le logiciel Java 2.2 : qui assure la mise en relation et la restitution internet.

La dernière composante présentée qui est **la composante logiciel Java 2.2** est celle définie comme priorité concernant le travail de clarification du code source. On note toujours dans la présentation de Jean que la composante logiciel Java 2.2 comprend quatre servlets gérés par **apache** qui peuvent fournir toutes les réponses demandées.

1. ● **informationList?type=xxx** (keyword, fullName, etc.) **&contents=yyy**
 ● (option) **informationList?type=all**
2. ● **descriptorInstances?type=xxx** (keywordName, etc.) **&contents=yyy**
3. ● **metaDescriptorInstances ?type=xxx** (MetaKeyword, etc.) **&contents=yyy**
4. **information?idInformation=nn**

Tableau 1 les quatre servlets utilisés dans le CI (les codes couleurs sont illustrés sur le site <http://sanarsoftQualite.biz>).

L'application Java utilise MySQL pour récupérer les données de la base de données ensuite elle va construire une information. Tout le programme est partagé en trois paquets : le **paquet data** qui collecte les données et construit les informations, le **paquet business** qui va faire les mises en relation et le **paquet présentation** qui va configurer l'information pour la mettre à disposition aux quatre servlets.

Pour ce qui est du paquet **data**, il récupère à la demande les données dans la base SQL. Il contient une fabrique d'objet (pattern « fabrique » de codage) pour fabriquer (réifier) les informations à partir de la base MySQL.

Le paquet **business** quant à lui est centré sur une information qui reprend le schéma de la base de données SQL. Il gère des listes d'informations, pour les mot-clefs et autres, les listes de descripteurs. Cependant concernant les descripteurs il y en a deux sortes qui sont liés à la structure de la base de données (Figure 1).

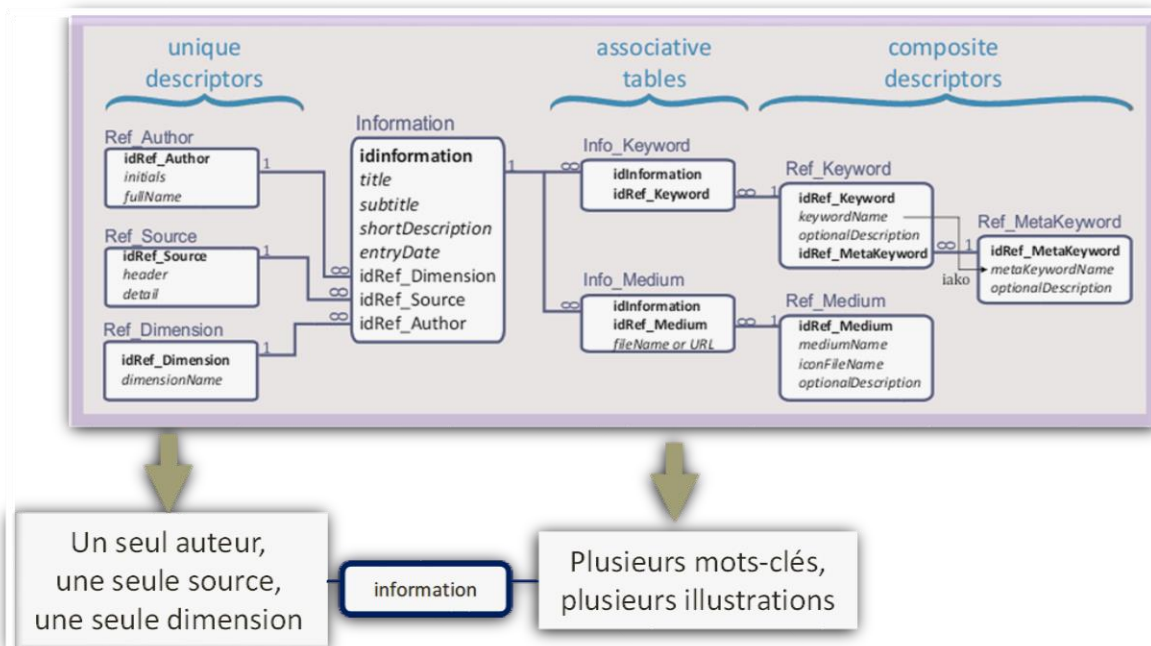


Figure 1 Architecture de la base de données

Il y a d'un côté les descripteurs simples (les relations directes à l'information) et de l'autre les descripteurs composés qui passent par une table intermédiaire. Cela se traduit de manière logique par le fait que l'on peut avoir plusieurs mot-clefs pour une information mais on ne peut avoir qu'une seule dimension, qu'une seule source, qu'un seul auteur pour une information. En périphérie il y'a deux modules qui ne sont pas les cœurs du programme. C'est deux modules IA qui permettent pour n'importe quelle information de la lier par une autre c'est-à-dire d'avoir une suggestion pour n'importe quelle information.

Dans le même paquet il y a une option qui été développée dans l'application qui permet quand on coche le nom d'un rédacteur/trice par exemple de faire des recherches sur les informations

écrite par le rédacteur/trice qu'on a coché est l'auteur. Mais comme ce n'était pas au point la fonctionnalité a été suspendue mais existe toujours et devra faire partie de la V3.0 a ajouté Jean.

Après sa présentation Jean a conclu par rappeler **l'objectif de la restitution du logiciel open source qui est**

1. **dans un premier temps de fournir un code source qui soit clean selon les normes qualité de SanarSoft et de l'IRD,**
2. **et dans un deuxième temps avoir une documentation claire du projet qui explique cette restructuration.**

Ainsi à la fin on devrait avoir un code source propre et fonctionnel, une procédure d'installation et une documentation associée à cet ensemble (documentation pour le CI, pour le masque PHP, sur la base, sur comment fonctionne l'application Java pour que des gens qui veulent développer dessus puissent travailler correctement).

Premières actions à faire et planning

Il va falloir structurer tout cela et faire le travail progressivement. Pour l'instant il a été décidé que nous allons faire la clarification du code Java. On fera deux documents :

1. un qui explique comment fonctionne le code source
2. et le **javadoc**.

Toujours dans la discussion pour élaborer le planning Jean qui partageait son écran a ouvert son Eclipse pour essayer de montrer que dans le code source du CI il y a des parties ou c'est déjà vu et revu. Comme par exemple dans le **C_DataFactory** tout a été fait dans le **java doc**.

Au vu de l'évaluation du travail à faire, nous allons dans un premier temps faire un cahier des charges sur l'écriture du code source pour que le code soit tout le temps écrit de la même façon suivant le même principe.

Sur proposition de Jean on va se baser sur le code source utilisé pour écrire les codes sources de SimMasto qui utilise la méthode de codage suivante :

- ☒ Pour chaque classe comprendre ce qu'elle fait.
- ☒ Ensuite, construire un javadoc complet en décrivant au niveau de la classe à quoi elle sert, au niveau des méthodes ce qu'elles font (instructions que l'on peut leur donner¹), au niveau des champs ce qu'elles sont etc.
- ☒ Puis mettre le commentaire sur le code source avec deux types de commentaires :
 - Commentaire interne au code source : à l'attention des développeurs informatiques qui veulent modifier le code afin qu'ils puissent savoir ce que fait le code.
 - Et un commentaire type javadoc généré sans erreur pour que quelqu'un qui veut juste comprendre comment fonctionne le logiciel puisse le faire sans aller dans le code source.

Nous allons dans un premier temps rendre le code clair le plus propre possible mais en français sans se soucier de l'anglais. Cependant **il faut noter que tous les noms de classes de champs, de propriétaire doivent être tous en anglais.**

On va faire un formulaire qu'il faudra vérifier et cocher pour chacun des fichiers du code source.

¹ La logique d'écriture des commentaires javadoc pour les méthodes consiste à écrire (i) l'instruction que l'on peut lui donner, (ii) les paramètres requis pour cette instruction et (iii) le retour que l'on obtient.

Premier cahier des charges

Pour la restructuration du code source on se base sur la charte SimMasto (exemple Figure 2)

```
1 /* This source code is licensed under a BSD license as detailed in file SIMmasto_0_license.txt */
2 package thing;
3
4 import java.util.TreeSet;
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 /** This class accounts for animals as moving agents.
22  * @author Q.Baduel, J-E Longueville, J. Le Fur 2011-02, 2013.01, 04.2015 */
23 public abstract class A_Animal extends A_Organism {
24     //
25     // FIELDS
26     //
27     private double maxDispersalDistance_Umeter = 0.0;
28     private String desire = ""; // Should never be null or else desire.equals will not work JLF 01.2018
29     protected boolean trappedOnBoard = false;
30     protected I_SituatedThing target = null; // store the container or agent selected as a destination
31     protected double speed_UmeterByTick;
32     protected Coordinate nextMove_Umeter = new Coordinate();
33     protected boolean hasTolaveFullContainer = false; // used to avoid return to containers left for fullness
34     protected I_Container lastContainerLeft; // Used to compute dispersal
35     //
36     // CONSTRUCTOR
37     //
38     public A_Animal(I_DiploidGenome genome) {
39         super(genome);
40         initParameters();
41         this.male = ((C_GenomeEucaryote) genome).getGonosome().isMale();
42         String sex = "+F:";
43         if (testMale()) sex = "-M:";
44         this.setName(sex + this.retrieveName());
45     }
46     //
47     // OVERRIDDEN METHODS
48     //
49     /** Remove references to last container left, targeted container and eggs */
50     @Override
51     public void discardThis() {
52         this.lastContainerLeft = null;
53         this.discardTarget();
54         super.discardThis();
55     }
56     /** The heart of SimMasto agents<br>
57     * Realize actions bound to foraging desire, then super<br>
58     * @version jlf 2017.08 2017.12 2018.01 */
59     @Override
60     public void step_Utick() {
61         if (!this.trappedOnBoard) { // if trapped, do nothing - JLF 07.2013
```

Figure 2 exemple de structuration de code à utiliser (exemple tiré de SimMasto)

- ☒ On ne laisse pas de ligne vierge, tout est collé,
- ☒ Le code source est structuré en sections qui peuvent être, dans l'ordre :
 1. DATA,
 2. FIELD(S),
 3. CONSTRUCTOR(S),
 4. OVERRIDDEN METHODS,
 5. OTHER METHODS,
 6. SETTER(S) AND GETTER(S)
 7. MAIN().
- ☒ Pour séparer les sections, on rajoute trois lignes de commentaire : une ligne vierge avec slash slash (//), une ligne avec slash slash (//) et le nom de section, une troisième ligne vierge avec //.
- ☒ Section **FIELD(S)** : on met les noms des champs de la classe avec un commentaire type code source et/ou un commentaire type javadoc selon le contexte et l'importance.
- ☒ Section **CONSTRUCTOR(S)** : La partie constructeur qui contient en général une méthode ou deux s'il y a du polymorphisme mais qui contient tout ce qui est constructeur de la classe.
- ☒ Section **OVERRIDDEN METHOD(S)** : toutes les méthodes qui contiennent un override qui écrase les méthodes supérieures (donc dans toutes ces méthodes s'il n'y a pas l'annotation *override* c'est qu'il y a un problème)
- ☒ puis dans **OTHER METHOD(S)**, les méthodes qui ne sont pas override qui sont spécifiques à la classe (souvent private mais pas obligatoire)
- ☒ A la fin du code il y a la section **SETTER(S) AND GETTER(S)**. C'est toujours setters d'abord et les getters ensuite. S'il y a un seul setter on met setter et on fait attention de ne pas lettre le « s » de même pour getter.

- ☒ Enfin, la section **MAIN** est utilisée pour effectuer les tests réutilisables par un développeur.
- ☒ Utiliser le formateur qui est utilisé pour simMasto (*FormatageDuCode.xml*)
- ☒ les noms des champs sont en semi-camelCase avec `_Uxxx` s'il y a une unité à spécifier (voir Figure 2)
- ☒ Faire sauter les anciens codes commentés si les lignes n'ont pas d'effet dans le code
- ☒ Conserver les auteurs
- ☒ Toutes les interfaces ont des noms qui commencent par I majuscule underscore (*I_*), tout ce qui est abstract : A majuscule underscore (*A_*) et tout ce qui est classe : C majuscule underscore (*C_*)

Procédure (circuit qualité)

Pour s'assurer que toutes ces exigences établies seront satisfaites, il a été proposé d'établir un circuit qualité pour les fichiers du code source. Ainsi on pourra faire passer chaque code source à corriger dans ce circuit qualité comme on fait avec les documents du CI. Une fois que le code source aura passé par ce circuit on pourra y mettre l'indication de la licence plus la date qui montrera que le code a été vérifié et à quelle date.

Le circuit qualité sera sous la forme suivante : Birahime va premièrement analyser le code source, vérifier le nom des variables et faire un refactoring s'il y a besoin, commenter... bref tous les aspects du cahier des charges. Puis il transmet ensuite à Jules qui s'occupe de la qualité qui transmet ensuite à Moussa à un niveau T1 après des allers retours jusqu'à ce que le code source soit correct au vu du cahier des charges. Enfin on l'envoie à Jean pour y apposer l'indication de la licence et la date d'approbation.

On est convenu que pour démarrer on commencera avec les petits fichiers afin de bien asseoir la procédure avant d'attaquer les gros fichiers du code source.

Birahime fera une première proposition puis on établira le circuit qui sera amélioré au fur et à mesure. Au préalable on fera le cahier des charges qui sera accompagné d'un formulaire type check list.

Divers

En réponse à Birahime qui a demandé la version Java du code source, Jean a dit que c'est la version 1.8.

Jean a mentionné à Birahime que pour la connexion à la base de données SQL il faut dans data puis dans le fichier parameters.xml

Concernant l'accès restreint au site <http://SanarSoftQualite.biz> Jean a présenté la piste qu'il a trouvée qui pourrait permettre l'accès restreint aux documents confidentiels du site. Birahime s'est automatiquement imprégné de la piste et doit plus tard approfondir les recherches.

Frais de la réunion

Néant.