Deuxième livrable du projet Savonet: un ensemble d'outils d'aide à la mise en place d'une radio streamée $\boxed{\textbf{sens ?}}$

Équipe de développement du futur $\boxed{?}$ projet Savonet

16 décembre 2003

Table des matières

1 Présentation du projet **2**

2 Description **3**
- 2.1 Contexte 3
- 2.2 Choix du $\boxed{\textbf{langagage}}$ de programmation 4
- 2.3 Objectifs 4
- 2.4 Axes de réflexions 5

3 Organisation du travail **7**
- 3.1 Vue d'ensemble 7
- 3.2 LiquidSoap (SP1) 8
  - 3.2.1 Équipe de développement 8
  - 3.2.2 Description 8
  - 3.2.3 État actuel 8
  - 3.2.4 Rétractions 11
  - 3.2.5 Priorités 12
  - 3.2.6 Estimation de la durée de développement 12
- 3.3 LibSavonet (SP2) 12
  - 3.3.1 Équipe de développement 12
  - 3.3.2 Description 12
  - 3.3.3 Choix de programmation 13
  - 3.3.4 Structure du sous-projet 14
  - 3.3.5 Détail des réalisations et échéances. 15
- 3.4 Strider (SP3) 16
  - 3.4.1 Équipe de développement 16
  - 3.4.2 Description 16
  - 3.4.3 Dépendances 18
  - 3.4.4 État actuel 19
  - 3.4.5 tâches$\boxed{\textbf{Tâches}}$ futures 21
- 3.5 Soapdish (SP4) 22
  - 3.5.1 Équipe de développement 22
  - 3.5.2 Objectifs 22

3.5.3 Etat $\boxed{\textbf{État}}$ actuel . . . 22
3.5.4 Travail à venir . . . 24
3.6 Botanick (SP5) . . . 26
3.6.1 Équipe de développement . . . 26
3.6.2 Description . . . 26
3.6.3 Dépendances . . . 27
3.6.4 Estimation de la durée de développement . . . 27
3.7 Mixer (SP6) . . . 27
3.7.1 Équipe de développement . . . 27
3.7.2 Description . . . 27
3.7.3 Réalisation effective . . . 28
3.8 Administration du projet (SP7) . . . 29
3.8.1 Équipe de développement . . . 29
3.8.2 SP 7.1 : Affaires juridiques . . . 29
3.8.3 Questions juridiques : objectifs et état d'avancement . . . 29
3.8.4 SP 7.2 : Documentation et supervision du projet . . . 30
3.8.5 Documentation générale : objectifs et état d'avancement . 31

1 Présentation du projet

Nous avons proposé, dans le cadre du premier semestre de la maîtrise d'informatique de l'ENS de Lyon, un projet logiciel dénommé Savonet. Ce projet débouchera sur un produit fini, distribué sous licence GPL, et largement accessible au grand public.

Les membres du projet sont au nombre de 18 :

- Baelde David (100 % SP1 (responsable))
- Becker Florent (90 % SP3, 10 % SP5)
- Beaudou Laurent (90 % SP4, 10 % SP7.1)
- Bouchez Florent (100 % SP1)
- Burel Guillaume (100 % SP3 (responsable))
- Chevillard Sylvain (50 % SP1, 50 % SP 7.2)
- Cristau Julien (100 % SP6)
- Devautour Martin (90 % SP3, 10 % SP7.1)
- Duchesne Étienne (100 % SP1)
- Esperet Louis (90 % SP4, 10 % SP7.1)
- Gimenez Stéphane (100 % SP2)
- Mimram Samuel (50 % SP3, 50 % SP6)
- Pineau Jean-François (100 % SP2)
- Péron Mathias (100 % SP4 (responsable))
- Régnault Damien (100 % SP4)
- Richard Gaétan (30 % SP3, 70 % SP6)
- Rouquier Jean-Baptiste (100 % SP3)
- Tandabany Sattisvar (100 % SP5)

Le projet, d'assez large envergure, sera distribué sous forme de modules séparés, dont un certain nombre de bibliothèques OCaml, d'utilité publique. Ces bibliothèques, qui existent pour le langage C mais pas pour le langage OCaml, sont un apport appréciable pour la communauté de développement en OCaml.

Le logiciel Savonet en lui-même est un ensemble d'outils qui permettent à tout un chacun de créer facilement sa propre radio diffusée sur internet. Outre la mise en place facile d'une telle radio, Savonet comprend des outils puissants dont l'objectif est de rendre la radio la plus conviviale possible pour ses auditeurs, et la plus souple possible pour ses créateurs.

2 Description

2.1 Contexte

Il n'existe pas à l'heure actuelle de moyen simple pour mettre en place une radio diffusée en streaming sur internet. Il est possible d'installer sur une machine un serveur particulier appelé *Icecast* (`http://www.icecast.org/`) qui se charge de transformer un flux audio au format mp3 ou ogg en un contenu diffusable à partir d'une URL et accessible à n'importe quel lecteur multimédia tel que winamp [1], windows media player [2], iTunes [3], xmms [4]...

Cependant les outils qui sont développés autour de ce serveur ne permettent pas de mettre en place facilement une radio conviviale. Le serveur Icecast ne se suffit pas à lui-même : il faut lui fournir un flux audio en permanence pour que la radio ait un fonctionnement minimal. Même pour cette tâche sommaire, les rares outils d'automatisation qui existent sont difficiles à mettre en œuvre et demandent des compétences techniques que beaucoup de gens n'ont pas.

Voici les logiciels qui existent sur le marché actuellement :
- IceS (`http://www.icecast.org/`) est un programme qui prend des fichiers sur un disque ou qui recupere une entrée depuis une carte son, et qui l'envoie à un serveur icecast. La version 0.3 supporte le mp3, alors que IceS2 est un encodeur Vorbis ;
- OddCastDSP (`http://www.oddsock.org/tools/dsp_oddcast/`) est un client pour winamp qui permet d'envoyer un flux à un serveur icecast. Étant limité à winamp, il n'a pas non plus l'ambition de permettre de créer une radio web ;
- DarkIce (`http://darkice.sourceforge.net/`) permet d'envoyer directement le flux récupéré depuis une carte son au serveur icecast, après l'avoir compressé à la volée ;
- SHOUTcast (`http://www.shoutcast.com/`) est un concurrent de IceCast, gratuit mais propriétaire, distribué par nullsoft (éditeur de Winamp). Le flux streamé **Ne serait-ce pas un pléonasme ?** est au format mp3.

Il y a donc là un réel besoin, d'autant que les technologies multimédias attirent de plus en plus de gens. Même de simples particuliers ont maintenant les logiciels nécessaires pour monter un film numérique ou mixer la musique qu'ils ont composée. L'absence de logiciel d'assistance à la mise en place d'une radio représente donc un vrai manque.

[1] `http://www.winamp.com/`
[2] `http://windowsmedia.com/download/download.asp`
[3] `\fixme{http://}http://www.apple.com/fr/itunes/download/`
[4] `http://www.xmms.org/`

2.2 Choix du $\boxed{\text{langagage}}$ de programmation

Notre projet est essentiellement réalisé dans le langage OCaml. Ce choix s'explique par les points suivants :

- une première mouture de Savonet avait été réalisée dans ce langage par Samuel Mimram et David Baelde. Cette version avait été développée très vite, n'était pas évolutive ce qui justifie le fait de reprendre le projet depuis le début (reprendre le travail déjà effectué de manière à le rendre évolutif et modulaire demanderait plus de travail que de repartir à zéro). Cette expérience a cependant permis de montrer que le langage OCaml pour sa propreté et son haut-niveau sémantique était parfaitement adapté au projet ;
- tous les programmeurs du projet sont familiers de ce langage alors que leurs connaissances dans d'autres langages sont plus disparates. Par conséquent, travailler en OCaml permettra de supprimer le surcoût induit par une formation des programmeurs à un autre langage $\boxed{\textbf{Même à C/C++ ? ?}}$;
- comme mentionné plus haut, travailler en OCaml obligera à interfacer des bibliothèques qui existent en C mais pas en OCaml. Le surcoût induit ne devrait pas être énorme car ce travail s'intègrera parfaitement au reste du projet (notamment il faudra de toutes façons développer des bibliothèques spécifiques). En revanche, la communauté des programmeurs OCaml bénéficiera de l'apport de ces bibliothèques pour de futurs logiciels ;

Le projet touchant à des sujets très hétérogènes (web, algorithmique — pour le strider —, interfaçage de bibliothèques) d'autres langages sont manipulés (C, PHP, éventuellement Perl) mais il demeure que le langage qui semblait bien adapté, pour l'ensemble du projet, est le langage OCaml.

Vous êtes convaincus mais pas convaincants. état des lieux ? Discussion ?

2.3 Objectifs

Le projet que nous avons proposé vise à créer un logiciel permettant — à l'instar de ceux évoqués plus haut — de mettre en place une radio streamée, mais de façon beaucoup plus aisée que ce qui se fait actuellement. Nous avons proposé trois grands axes de développement :

ce qui précède ne paraît pas permettre de mettre en place une radio ... (dynamicité de la playlist, etc.)

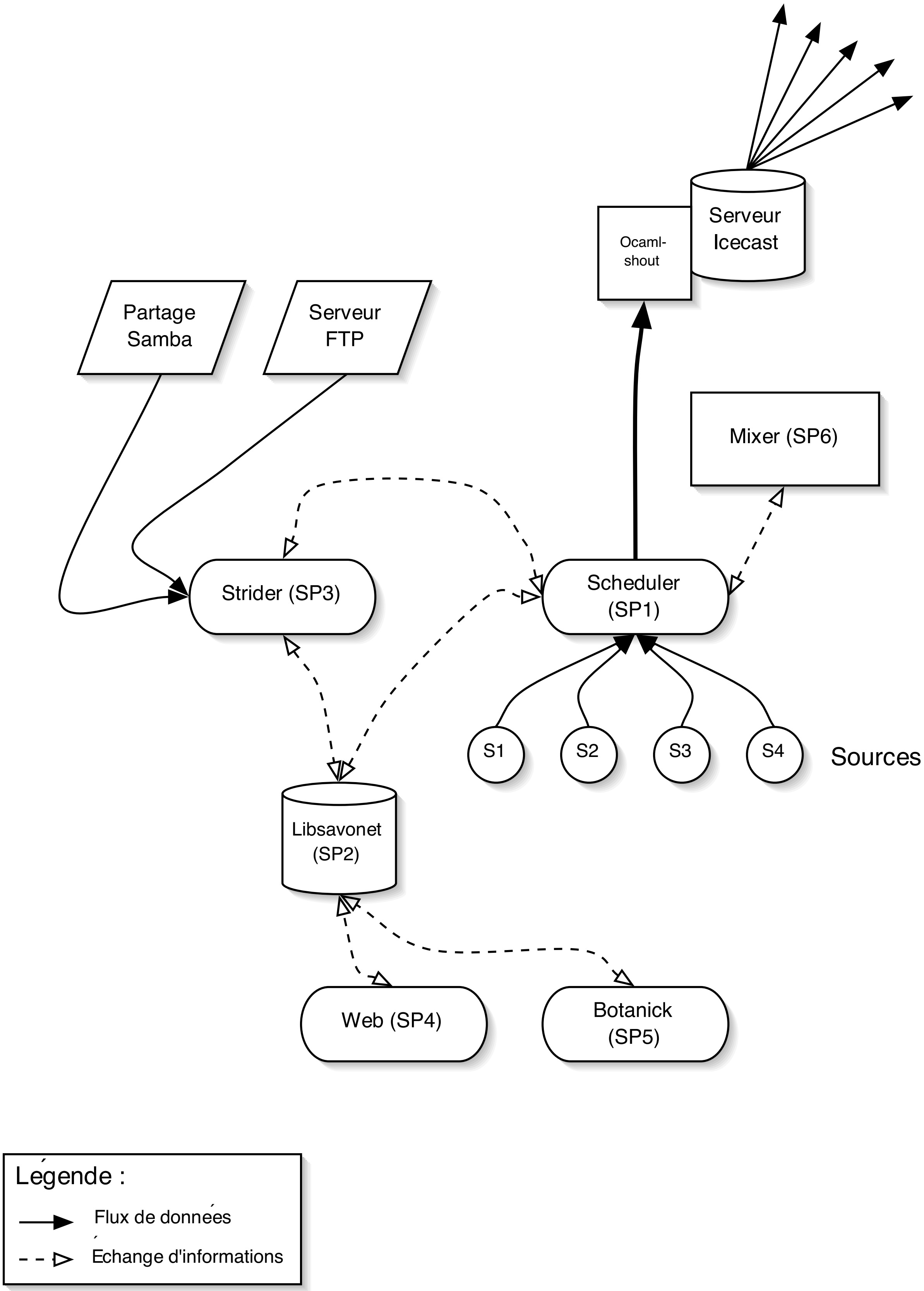
- développement d'un certain nombre de bibliothèques pour le langage OCaml qui n'existent pas à l'heure actuelle. On peut citer une bibliothèque d'accès en lecture aux fichiers d'un partage suivant le protocole Samba, une autre offrant les même possibilités pour des fichiers situés sur un serveur FTP ; des bibliothèques de conversions de formats audio telles que ogg, mp3 ou wav ; une bibliothèque de mixage de flux audio au format wav ;
- développement d'outils permettant la mise en place et la maintenance facile d'une radio numérique diffusée en streaming sur internet. Ces outils permettront d'indiquer une liste de fichiers audio à utiliser comme contenu pour la radio, de spécifier précisément une programmation horaire (par exemple pour alterner des phases de musique et de prise d'antenne par un

animateur, ou bien pour diffuser une émission contenue dans un certain fichier à une heure donnée). Par ailleurs, des pages web seront fournies avec le logiciel qui seront facilement personnalisables et permettront au créateur de la radio de lui consacrer une page web sur laquelle seront disponibles des informations sur l'émission en cours, les émissions précédemment diffusées, etc., ainsi qu'une possibilité d'administration de la radio à distance. Il sera possible de créer plusieurs canaux par radio (par exemple pour faire des radios thématiques : un canal de jazz, un canal d'émissions à thèmes, etc.). Ces canaux pourront être définis et contrôlés via l'interface web ;
– développement d'outils susceptibles d'améliorer le fonctionnement de la radio au fil du temps, en tenant comptes des vœux des utilisateurs, de leurs goûts, des liens de parenté entre différents styles de musique, ... Par exemple le site web comportera aussi une rubrique *auditeurs* qui permettra aux auditeurs enregistrés d'interagir avec la radio (voter pour un morceau ou demander à écouter un morceau, par exemple). Ces outils intégreront des programmes intelligents qui tenteront d'établir des informations pertinentes sur les goûts des auditeurs sur les différents morceaux diffusés ainsi que les liens entre eux.

manque une description haut-niveau

2.4 Axes de réflexions

Certaines parties du projets demandent une étude précise qu'il convient de ne pas négliger. Citons particulièrement deux points importants :
– le noyau de Savonet, qui s'occupe de fournir à l'utilisateur les moyens de programmer facilement sa radio : afin d'obtenir le maximum de souplesse dans la programmation, un petit langage de programmation est développé. Pour concevoir le langage, il faut déterminer quelle expressivité est nécessaire. La programmation de la radio peut sommairement apparaître sous la forme d'une suite d'instructions de la forme « entre telle heure et telle heure, mettre la radio dans le mode M » ; certains modes prédéfinis seront fournis en standard comme par exemple le mode « jouer les morceaux d'une liste », « laisser la parole à l'animateur », « laisser l'antenne libre aux choix des auditeurs ». Pour les animateurs plus exigeants et qui maîtrisent le langage OCaml, il sera toujours possible d'écrire soi-même un module en OCaml ayant la bonne interface et qui implémente un mode M extrêmement tordu (comme par exemple un mode dans lequel ne passe que des morceaux dont l'auteur a un nom qui commence par la lettre B et dont le titre est un ~~palyndrome~~ **palindrome**) ;
– le Strider, qui est la partie du projet qui s'occupera du côté « radio intelligente **?** capable d'évoluer » demande du travail de recherche afin de déterminer les algorithmes à mettre en place pour améliorer de manière significative l'intelligence de la radio vis à vis des goûts des auditeurs.


Serveur
Icecast
Ocaml-
shout
Partage
Samba
Serveur
FTP
Mixer (SP6)
Strider (SP3)
Scheduler
(SP1)
S1
S2
S3
S4
Sources
Libsavonet
(SP2)
Web (SP4)
Botanick
(SP5)
Légende :
Flux de données
Échange d'informations

3 Organisation du travail

Voici à présent un descriptif plus détaillé des différents sous-projets de Savonet et de leur avancement. Suivent des descriptions pour chaque sous-projet des objectifs précis atteints et à atteindre.

3.1 Vue d'ensemble

Savonet se décompose de la manière suivante :

LiquidSoap : aussi appelé noyau de Savonet, ce sous-projet s'occupe de tout ce qui concerne la programmation horaire de la radio. Plus généralement, ce sous-projet développe le « scheduler » **ordonnanceur** qui a pour mission de fournir en permanence un flux audio au serveur Icecast. Ce sous-projet est donc au cœur de Savonet ;

LibSavonet : ce sous-projet a pour objectif de définir le protocole de communication et de fournir les fonctions de dialogue entre les différents modules de Savonet. Plus précisément, les modules au cœur de Savonet (LiquidSoap, Strider) auront souvent besoin de communiquer avec les interfaces utilisateurs (Soapdish, Botanick). LibSavonet doit assurer le dialogue entre ces modules ;

Strider : ce sous-projet a déjà été décrit un peu plus haut. Il s'occupe de donner à la radio son évolutivité, en déterminant et en utilisant le plus d'informations possibles sur les utilisateurs, leurs goûts, et les fichiers disponibles à la diffusion. Ce sous-projet interface pour le langage OCaml des bibliothèques qui existent déjà pour le langage C **Tout existait en C (cf la non discussion)**. Ces bibliothèques pourront resservir à d'autres développeurs OCaml. Ce sont OCaml-SmbClient (pour la lecture de fichiers d'un partage Samba), OCaml-Ftp (même chose, mais avec le protocole FTP) et OCaml-mp3Id3 (pour la lecture des informations contenues dans les tags Id3 des fichiers au format mp3).

le descriptif haut-niveau ne me semble pas du tout correspondre à la description technique. N'auriez-vous pas regroupé deux modules totalement indépendant ? La collecte des sources et "big brother pour le meilleur de la radio" ?

Soapdish : ce sous-projet, d'une importance capitale,

alors que la collecte des fichiers n'est qu'accessoire !

assure le contact entre la radio et ses utilisateurs (tant auditeurs, qu'animateurs et administrateurs) en fournissant l'interface web de la radio. À peu près tout ce qui nécessite un dialogue entre les utilisateurs et le logiciel se fera via une interface web **cohérence avec Botanik !** : celle-ci doit donc être à la fois facile d'utilisation, fonctionnelle et facilement personnalisable pour le créateur d'une radio ;

Botanick : ce sous-projet très indépendant a pour objectif de fournir un bot IRC afin de permettre aux utilisateurs de faire des demandes ou des votes directement à partir d'un terminal IRC au lieu de passer par l'interface plus lourde du web.

Mixer : il s'agit là de fournir un ensemble d'outils permettant la manipulation de contenu audio. Une grosse tâche est l'écriture de la bibliothèque OCaml-Mixer qui fournit des fonctions permettant le mixage de flux audio. Par ailleurs, ce sous-projet interface des bibliothèques du langage C vers le langage OCaml. Ce sont les bibliothèques OCaml-Shout (pour le dialogue avec un serveur Icecast), OCaml-Vorbis (pour l'encodage et le décodage de fichiers audio au format ogg/vorbis), OCaml-Mad (pour la conversion du format mp3 vers le format wav).

Mentionnons aussi l'existence d'un sous-projet qui a pour objectif d'assurer la cohérence générale du projet. Ce sous-projet comprend une équipe chargée des questions juridiques, qui tente de différencier les utilisations de la radio qui sont légales de celles qui ne le sont pas. Sylvain Chevillard assure une autre partie de ce sous-projet de cohérence générale en coordonnant les efforts, en supervisant la documentation et l'harmonie des outils communs (makefiles, journaux de débuggage, etc.).

3.2 LiquidSoap (SP1)

3.2.1 Équipe de développement

– David Baelde (Responsable 100 %)
– Florent Bouchez (100 %)
– Sylvain Chevillard (50 %)
– Etienne Duchesne (100 %)

3.2.2 Description

Le noyau est un composant fondamental du générateur de flux LiquidSoap. Son rôle est de créer un programme intéressant à diffuser, à partir de briques de bases, qui diffusent des choses élémentaires.

Nous appelons ces briques de base des *sources*. On peut concevoir par exemple une source « diffusion d'un fichier de la liste par défaut », « diffusion d'un fichier obtenu après une recherche dans les bases de données », ou encore une source « dire l'heure », « dire les infos », « diffuser un direct », etc.

La diversité des sources n'est rien sans un grand nombre de possibilités d'organisation de celles-ci. On veut par exemple pouvoir mêler plusieurs sources en garantissant des quotas de diffusion, ou bien spécifier le type de diffusion en fonction de la date ou l'heure, programmer des enchaînements, des fonds sonores, etc.

3.2.3 État actuel

L'architecture du noyau a été fixée. Nous avons imaginé sinon toutes, du moins un grand nombre de possibilités, et l'architecture que nous avons retenue devrait pouvoir se plier à la plupart des exigences concevables. Nous nous sommes arrêtés sur un noyau structuré en graphe orienté acyclique. Aux feuilles du graphe sont placées les sources. Nous appelons *opérateur* un nœud du noyau. Ce nom se comprend de lui-même lorsqu'on sait quel est le rôle des nœuds du noyau : ils prennent le contenu que leur transmettent leurs fils via les arêtes entrantes et le transforment de façon à se comporter en sortie comme une sorte de source plus complexe.

Typiquement, on peut avoir un opérateur sur lequel sont branchées deux sources $S1$ et $S2$. L'opérateur a pour règle de passer $S1$ le matin et $S2$ l'après-midi. Il fournit donc en sortie un contenu enrichi, et se comporte vis à vis du reste du graphe (et donc des opérateurs qui sont au-dessus de lui) comme une source spéciale qui diffuserait un contenu différent le matin et le soir.

Nous avons fixé le protocole de communication entre source et opérateur, en définissant ces deux classes **lesquelles ?** dans `types.ml`.

Nous nous sommes aperçus au passage que la gestion des sous-flux à l'intérieur du noyau pouvait être faite de façon très économique en mémoire, et donc éviter des allocations, et surtout de coûteux glanages de cellules. Initialement, les recopies et allocations étaient nombreuses. Nous avons mis au point deux outils dans le module `tools/mixer.ml` qui permettent de résoudre deux problèmes de façon efficace. Ces problèmes peuvent paraître triviaux, mais on n'y pense pas immédiatemment quand on programme dans un langage de haut niveau comme OCaml. Heureusement, OCaml possède aussi des fonctionnalités impératives, qui nous ont permis de contrôler finement l'utilisation des ressources.

– Le premier problème est la gestion de l'*overflow.* Quand on décode du mp3 ou du ogg, on ne sait pas combien de wav on va obtenir. On accumule donc en fait le résultat de la décompression jusqu'à obtenir la quantité souhaitée. Et on garde le surplus pour la suite. La structure d'accumulation a été réécrite afin d'éviter les recopies et allocations.
– Le second problème est l'*underflow.* Dans le noyau, on se passe des blocs de wav de taille fixée. Ceci est nécessaire pour pouvoir faire des mixages facilement. Le problème survient lorqu'une source arrive en fin de fichier, et n'a pas de quoi remplir un bloc. Elle doit donc pouvoir rendre des blocs partiels. En résolvant ce problème, on a mis en place un protocole qui évite les recopies dans toute la chaîne de transmission d'un sous-flux. Un opérateur donne son bloc à remplir à un fils, puis à un autre, jusqu'à ce que le bloc soit complet. Finalement, chaque opérateur possède $n-1$ blocs alloués, où n est le nombre de fils. Et le pseudo-opérateur au sommet, le gestionnaire du canal, en possède 1. Pendant l'exécution, il n'y a aucune allocation.

Je n'ai pas suivi pourquoi ... J'ai l'impression qu'il n'y a pas de "pipeline" le long de l'arbre. Pas de problème de temps réel ? Tout est toujours produit bien avant que l'on en ait besoin ? Ou est-ce la cause des "parasites" de la démo.

LiquidSoap est désormais une application utilisable et conviviale, du moins autant que la plupart des outils UNIX. La programmation du noyau peut se faire au moyen d'un script `#!` : ainsi, on peut lancer facilement un canal radio en spécifiant :

– sur la ligne `#!`, les paramètres de la connexion ;
– et dans le corps du script, la programmation du noyau.

Sur la figure 1 donnée en exemple, on programme un canal où il y a un jingle tout **tous** les 10 morceaux,

ou les 5 ? peu lisible ! N'aurait-il pas mieux valu un fichier de conf + lisible + une moulinette convertissant Ocaml ?

du jazz le matin, du hard-rock la nuit. En *prime-time*, on passe des morceaux demandés par les utilisateurs, ou, s'il n'y a pas de requêtes, des morceaux habituellement appréciés des auditeurs. Et on précise que s'il y a un problème, on peut toujours retomber sur la playlist par défaut, qui n'est jamais en panne, et rend donc le noyau infaillible. Nous n'avons pas encore toutes les sources et tous les opérateurs utilisés dans cet exemple (voir un peu plus bas pour un état des lieux plus précis).

```
#!/usr/local/bin/liquidsoap -host sparc7.residence.ens-lyon.fr

let defaut = playlist ["/usr/share/radio/default.pls"] in
let jingle = playlist ["/usr/share/radio/jingle.pls"] in
let jazz = request ["je veux du jazz"] in
let hard = request ["je veux du hard rock"] in
# Le langage de requêtes est un peu fantaisiste, mais c'est l'idée ...
let democratie = fallback [] [ user_requests [] ; user_preferences [] ] in
  fallback [] [
   quotas ["1";"5"] [
     jingle ;
     date_switch ["0-6h";"7-10h";"11-17h";"18-23h"]
                 [hard ; jazz ; democratie ; hard ] ;
   defaut ]
```

Fig. 1 – Comment créer sa radio ?

Remarquons au passage qu'on utilise désormais une instance de LiquidSoap par canal radio. Cela permet de répartir les calculs sur plusieurs machines si le nombre de canaux à gérer est trop grand. Cette préoccupation est nouvelle : on a en effet constaté que le décodage vers le wav et l'encodage depuis le ogg, qui ont tous deux lieu en permanence, constituaient des calculs importants. On pourra, si le développement va suffisamment loin, développer un serveur de canaux qui lance à la demande une instance de LiquidSoap pour gérer un nouveau canal, sur une machine pas trop chargée. Cependant, il s'agit là d'une option qui n'est pas dans nos priorités car on peut très bien se satisfaire d'une gestion manuelle de la charge des machines et du démarrage des canaux, dans un premier temps.

Nous avons actuellement une seule famille de sources : celles qui jouent des fichiers locaux, au format mp3, wav ou ogg. Et une seule famille d'opérateurs : les sélecteurs par prédicat. Leur tâche consiste à diffuser leur source fille *src* si le prédicat associé à *src* est vrai au moment du changement de source. Cette famille contient notamment la sélection par tranche horaire.

Il faut préciser que notre travail de réflexion sur l'architecture nous a amené à concevoir quelque chose que nous jugeons particulièrement intéressant, et vers lequel nous avons axé notre développement. Ce point n'a pas été évoqué dans le précédent livrable parce que nous n'avions pas encore une idée assez claire du noyau pour dégager cette problématique, mais il aurait dû s'y trouver si nous y avions pensé.

Tout part de la constation que LiquidSoap a pour but d'automatiser certaines parties de la diffusion, de façon à ce qu'il ne soit pas nécessaire qu'un opérateur humain s'occupe en permanence de la radio. Mais, comme pour tout problème d'automatisation, la tolérance aux pannes est un critère déterminant. Des pannes, il peut en survenir à tout instant : problème de transmission d'un fichier sur le réseau, défaillance d'une source, partage Samba déconnecté… Nous avons donc jugé qu'il était important que le noyau soit capable, dans une certaine mesure à tout le moins, d'assurer que la diffusion se fera toujours sans problème. Nous avons donc mis au point plusieurs analyses statiques du noyau. Pour l'instant, on peut détecter certaines panne à l'avance (vérification de vivacité). On entend par panne la situation où il n'y a aucune source à diffuser : pas de source prête parmi celles qui sont sollicitées par l'arborescence d'opérateurs. Notre algorithme peut garantir, au moment de la programmation de la radio, que celle-ci résistera aux pannes.

Nous avons aussi mis au point un algorithme vérifiant qu'une source n'est jamais utilisée par plusieurs opérateurs à la fois (vérification de sûreté). On vérifie au passage que chaque source est accessible depuis la racine. Cela empêche d'éventuels blocages dus au fait qu'une source essaie de diffuser pour deux opérateurs simultanément (cas qui n'est pas censé se produire dans une bonne programmation de radio, mais duquel il faut tout de même se méfier : une programmation de radio un peu complexe peut sembler exempte de ce genre de problèmes sans l'être effectivement). En outre on fait ainsi de la détection de « code mort » puisqu'on détecte les sources qui apparaissent dans la programmation mais qui correspondent en fait à des cas impossibles. Ces sources sont donc inutiles et il n'est pas nécessaire de les créer.

Cet algorithme n'est pas encore implémenté en lui-même. Cependant nous avons déjà écrit les outils de base qui serviront à sa réalisation : intersection, union et vacuité de tranches horaires.

3.2.4 Rétractions

L'architecture choisie permet de diffuser un enregistrement en direct, par exemple un animateur, ou une retransmission du concert privé des VRP. La méthode `countdown` des opérateurs est particulièrement utile dans ce cas : la source diffusant le flux reçu depuis un processus d'enregistrement distant peut indiquer à ce processus dans combien de temps il va être diffusé. Cette méthode a en fait été mise en place uniquement pour ce type de sources. « 3 . . . 2 . . . 1 . . . Antenne ! ».

Je suppose qu'il serait trop dur de parler d'un mécanisme de compte à rebours !

Cependant, la réalisation du client permettant l'enregistrement parait assez difficile dans le temps imparti. Éventuellement, elle pourrait être assurée par quelqun d'extérieur au sous-projet. Le principal problème vient de ce qu'on ne trouve pas de bibliothèque indépendante de l'architecture permettant la capture audio. Il est absurde, trop long, et peu portable de réécrire les codes d'acquisition pour les différents systèmes (OSS, ALSA, OSX, Windows,. . .), comme le font malheureusement beaucoup de logiciels actuellement. Ce n'est pas insurmontable en soi, mais il reste peu de temps, et beaucoup de choses plus prioritaires à faire. Peut être que le plus simple est que la source en question se fasse passer pour un serveur Icecast auprès d'un client DarkIce ?

La reconfiguration dynamique du noyau ne sera pas non plus implémentée. Tout d'abord, cette tâche nécessite un avancement important des sous-projets LibSavonet et Soapdish, ce qui repousse d'autant le moment où nous pourrons effectivement la réaliser. Mais surtout, il s'agit là d'une tâche délicate, dont la réalisation est rendue encore plus difficile par le soucis que nous avons à présent de faire un noyau sûr. En effet, pour reconfigurer dynamiquement le noyau sans mettre à bas tout le travail sur la vivacité et la sûreté entrepris par ailleurs, il faut prendre garde à ne modifier dynamiquement que des parties inactives du noyau, et encore, en s'assurant qu'elles ne vont pas devenir actives avant d'avoir été réinitialisées. L'analyse du problème est assez délicate. Nous n'avions pas envisagé cet aspect des choses auparavant de sorte que le travail ne parassait pas très difficile. Mais à présent, notre priorité va à la sûreté plutôt qu'à la reconfiguration dynamique.

Toutefois, il est clair que cette fonctionnalité pourra etre implémentée un jour. Rien ne s'oppose à une modification d'une partie du graphe, *quand cette partie n'est pas active*. Et le parseur utilisé pour analyser le programme du noyau peut servir pour analyser un programme décrivant un sous-graphe.

Cependant, nous n'aurons vraisemblablement pas le temps de le faire avant la remise du projet.

3.2.5 Priorités

Nous devons rapidement être capables d'utiliser le module OCaml-Fetch, afin de rapatrier des fichiers distants pour les jouer.

L'autre extension très importante des sources est évidemment le support des requêtes utilisateurs. Nous devons pour cela intégrer LibSavonet ; tout dépend donc de ce sous-projet.

Si les choses progressent vite on devrait pouvoir créer des sources utilisant la synthèse vocale pour prononcer divers messages.

3.2.6 Estimation de la durée de développement

En deux semaine nous devrions avoir une bibliothèque riche de sources et d'opérateurs, au comportement correct. La grande question, pour le support des requêtes, est le temps que mettra LibSavonet à être fonctionnelle.

Cela fait, il nous restera un peu de temps pour débuguer, nettoyer et documenter. Finalement, la radio ne sera pas tout à fait aussi riche que prévue, mais le noyau permet des programmations complexes, dans un langage extensible, et surtout agréable d'utilisation **discutable**, le tout en garantissant une certaine sûreté de la programmation.

3.3 LibSavonet (SP2)

3.3.1 Équipe de développement

– Stéphane Gimenez (Responsable : 100 %)
– Jean-François Pineau (100 %)

3.3.2 Description

Ce sous-projet consiste en l'élaboration d'une bibliothèque de communication pour OCaml capable de gérer des appels à distance afin de permettre la réalisation dans le cadre du projet global :

– d'un ensemble logiciel qu'utiliseront les différents clients de la radio (dont les deux sous-projets Soapdish et Botanick qui sont des relais « user-friendly », qui peuvent être considérés comme des clients multi-utilisateurs) afin que les utilisateurs puissent interagir avec leur radio préférée. Strider

rentrera finalement aussi dans la catégorie des utilisateurs de cette bibliothèque, afin de pouvoir le séparer de la radio elle-même ;
– d'un ensemble logiciel côté serveur, grâce auquel les différents composants internes de la radio qui ont besoin de l'intervention des utilisateurs pourront s'enregistrer afin que celle-ci leur transmette les requêtes qui leur sont adressées.

Nous avons pu constater (probablement à cause de la portée générale proposée) que la distinction client-serveur n'était pas très justifiée, et les concepts semblaient relativement symétriques de ce point de vue là. Les besoins spécifiques aux clients et au serveur qu'il est nécessaire de réaliser sont très peu nombreux. Concrètement, il ne parait donc pas nécessaire de distribuer deux fichiers de bibliothèque distincts.

L'objectif est de centraliser tous les dialogues de ce type afin de ne pas se retrouver avec $x \times y$ protocoles de communications (x étant le nombre de composants de la radio susceptibles de réclamer un contact avec les auditeurs et y le nombre d'interfaces utilisateurs, les deux pouvant être très grands, étant donné le niveau de modularité du projet). Ce couple de bibliothèques permettra d'apporter la cohésion nécessaire dans ce genre de projet.

Un point important est celui de la fonction d'administration :
– de la radio elle-même ;
– des différents canaux qu'elle propose, qui pourra être déléguée à d'autres administrateurs ;
– des différents types de requêtes possibles sur les canaux de la radio.

La bibliothèque s'occupe de fournir une gestion d'authentification et de droits d'accès complexes, adaptée, en particulier, aux besoins de ce type de service.

3.3.3 Choix de programmation

Dans un premier temps, il semblait qu'il puisse être utile de spécifier le protocole de communication réseau, normalement interne au sous-projet, dans un langage clair afin que de nouveaux types de clients n'ayant pas la possibilité d'utiliser les interfaces proposées puissent être tout de même envisagés. Mais nous nous sommes aperçus que la transmission de requêtes de type indéfini ('a en OCaml) seraient très pratique et qu'il serait dommage de se priver des méthodes de sérialisation (« marshaling ») proposées dans la bibliothèque standard d'OCaml. Cette bibliothèque n'est pourtant pas entièrement adaptée à nos besoins car elle repose en fait en interne sur des spécificités d'implémentation du langage en cassant la vérification de typage et sa spécification ne précise pas de sécurité d'exécution lorsqu'on reçoit des données d'un type différent de celui attendu. Il est tout de même possible de l'utiliser dans le cadre de clients jugés sûrs par authentification et qui respectent le protocole. Une extension dans ce sous-projet est donc apparue, et consisterait en l'implémentation d'un « marshaling » sûr, afin d'accepter des clients quelconques.

pas tout compris du paragraphe précédent...

Nous comptions sur le fait que ces bibliothèques puissent être implémentées sans faire appel à des spécificités du type de service (radio) que l'on cherche à réaliser dans le projet global. Et nous nous étions réservé **s** le droit, dans le cas où une réalisation trop générale ne serait pas réellement satisfaisante de perdre **de** la généralité selon les besoins qui apparaîtraient en cours de développement. Néanmoins dans l'état d'implémentation actuel nous n'avons pas ressenti le besoin de perdre la moindre généralité, et somme$\boxed{\texttt{s}}$ confiants quant à la conservation de cette généralité jusqu'au terme du projet.

3.3.4 Structure du sous-projet

Le sous-projet se découpait dans la proposition préalablement faite en cinq composants qui permettent de construire les bibliothèques présentées. Les fichiers contenant le code sont détaillées ci-dessous, il peut être intéressant de regarder les interfaces `.mli` associées. Un schéma récapitulatif de la structure du sous projet pourra être trouvée un peu plus bas.

- *Module de streaming/parsing.* Il s'occupe de la gestion du réseau et permet de transformer les objets du langage de programmation en information qui peut être communiquée au travers du réseau, ainsi que de réaliser l'opération inverse.
  Trois blocs logiciels :
  - `src/net.ml` implémente les primitives réseau (création de serveur, envoi et réception de données) ;
  - `src/client.ml` implémente le protocole de connexion du client ;
  - `src/server.ml` implémente le protocole de connexion du serveur.
- *Module de collection de types de requêtes/distribution de requêtes.* Il permet aux composants de la radio d'enregistrer le type de requêtes qu'ils sont aptes à recevoir. Il fait alors en sorte que les requêtes des utilisateurs soient envoyées aux bons composants de la radio. Et, éventuellement, si une réponse est attendue, il va transmettre la réponse de la radio à l'utilisateur.
  Un bloc logiciel :
  - `src/agent.ml` implémente l'objet agent qui sera utilisé pour les communications. Un agent est une entité qui sait traiter un message de type particulier. Il peut aussi nommer d'autre agents comme sous-agents en proposant aux autres agents leur accès au travers d'un nom. On obtient ainsi une arborescence qui permet de trouver un contact particulier parmi les agents. On peut donc envoyer sous le nom d'un certain agent un message requête à un autre agent, celui-ci après traitement peut répondre au premier à l'aide d'un message réponse. Un « thread » est bien évidement créé pour chaque traitement de message étant donné que ceux-ci peuvent prendre un temps quelconque.
- *Module d'authentification.* L'utilisateur envoie son login et son mot de passe lorsqu'il envoie une requête. Le module d'authentification s'occupe de vérifier la validité du mot de passe. Il est envisageable d'utiliser une base de donnée$\boxed{\texttt{s}}$.

  cryptage ? sécurité ?

  Deux blocs logiciels :
  - `src/user.ml` spécifie le profil d'un utilisateur contenant en particulier les éléments nécessaires à son authentification ;
  - `src/auth.ml` implémente la gestion de la base de données concernant l'authentification, avec pour contrainte une sécurité en cas d'arrêt brutal (non désiré) du programme.
- *Module de gestion des droits d'accès.* La gestion des droits d'accès consiste

à vérifier qu'une personne qui effectue une requête possède les droits d'accès appropriés. La requête en question peut être de changer les droits d'accès. Ces derniers types de requêtes seront récupérés par la bibliothèque elle-même.
Un bloc logiciel :
  - `src/access.ml` implémente la gestion des droits d'accès pour l'instant sur la base générale d'objets de type 'a pouvant accéder selon autorisation à des objets de type 'b. Une gestion de groupes d'utilisateurs est envisageable.
- *Interfaçage de la bibliothèque client.* Cela sera réalisé en collaboration avec le sous-projet Soapdish.
  Code situé dans la hiérarchie à l'extérieur du sous-projet LibSavonet.

Un nouveau module a dû être ajouté :
- *Module de gestion d'historiques (« logs »)* . L'utilité d'un tel module dans ce genre de réalisation est assez évidente, il avait simplement était **été** oublié. Un bloc logiciel :
  - `src/log.ml` implémente une gestion de fichiers d'historiques contenant des messages correctement datés **sens ?** de niveaux d'importance et d'origine spécifiables.

3.3.5 Détail des réalisations et échéances.

- *Définition concrète des interfaces.*
  Priorité : 0 Travail : 2ps **2 personnes-secondes, 2 pico secondes ?**
  Ce sous-projet devant permettre aux utilisateurs de communiquer avec la radio, il a fallut tout d'abord commencer par se mettre d'accord avec les chefs des autres sous-projets (Soapdish, Botanick, LiquidSoap et Strider) afin de décider des requêtes qui vont se transmettre, de leur structure, etc. Cela a été fait en premier et a nécessité une bonne semaine afin de bien choisir la solution la mieux adaptée pour tous. Cette définition a évolué plusieurs fois et n'est toujours pas considérée comme stable à l'heure actuelle.
- *Module de streaming/parsing.*
  Priorité : 1 Travail : 1ps
  Ce module a été réalisé grâce au module de « marshaling ». Et a donc nécessité moins de travail que prévu (1ps à la place de 2ps).Il est disponible sous une forme qui ne sera probablement que très légèrement modifiée par la suite.
- *Module de collection de types de requêtes/distribution de requêtes.*
  Priorité : 1 Travail : 4ps
  Erreur dans l'estimation du temps, 2ps était illusoire, un minimum de 4ps sera nécessaire, c'est en fait le point difficile. L'interface est réellement difficile à spécifier, non seulement pour des raisons de programmation (implémentation en bordure de ce que permet le langage de programmation **? ?**) mais aussi d'aisance d'utilisation. De premiers résultats ont étés obtenus, des exemples de serveurs et de clients proposés dans `sample` permettent déjà les premiers éléments de communication (ping-pong au sein du serveur, et requête simple au travers d'un lien réseau mettant en jeu serveur et client). L'implémentation devra être terminée et éventuellement

une fois de plus complétée.
- *Interfaçage de la bibliothèque client.*
  Priorité : 1   Travail : 2ps
  Cette tache nécessite que la spécification des agents soit totalement fixée. Il serait intéressant d'avoir une implémentation totalement fonctionnelle pour cela, néanmoins il devient urgent de débloquer la dépendance pour le sous-projet qui en dépend (Soapdish) et un développement parallèle doit commencer dès à présent.
- *Module de gestion des droits d'accès.*
  Priorité : 2   Travail : 2ps
  Il peut être intéressant d'arriver le plus vite possible à avoir une radio fonctionnelle, à laquelle on rajoutera des modules pour l'améliorer. Ainsi, on peut estimer dans un premier temps que tous les utilisateurs ont tous les droits, et n'implémenter ce module que dans un second temps. Nous espérons à présent pouvoir faire cela en 2ps.
  Pour l'instant seule une réflection quant aux spécifications nécessaires a été réalisée. Ceci afin de prévoir son intégration dans les modules déjà implémentés et en cours d'implémentation.
- *Module d'authentification.*
  Priorité : 2   Travail : 2ps
  Pour les mêmes raisons, ce module n'est pas prioritaire, il a pourtant été réalisé car facilement parallélisable au sein du projet : réalisable en attendant une dépendance interne. Il a nécessité 1 personne semaine. Une personne semaine de plus pourrait être nécessaire si l'on souhaite une gestion plus optimisée de la base de données.
  Un module gérant de façon fictive (`Auth.Dummy`) l'authentification a été réalisé. Un module gérant de façon non cryptée (`Auth.Uncrypted`) l'authentification a été réalisé. On peut bien sûr envisager (hors du cadre du projet) l'implémentation d'un module crypté. Le code est structuré de façon a **à** rendre son insertion très facile.
- *Module de gestion d'historiques*
  Priorité : 1   Travail : 1ps
  Implémenté.

3.4 Strider (SP3)

3.4.1 Équipe de développement

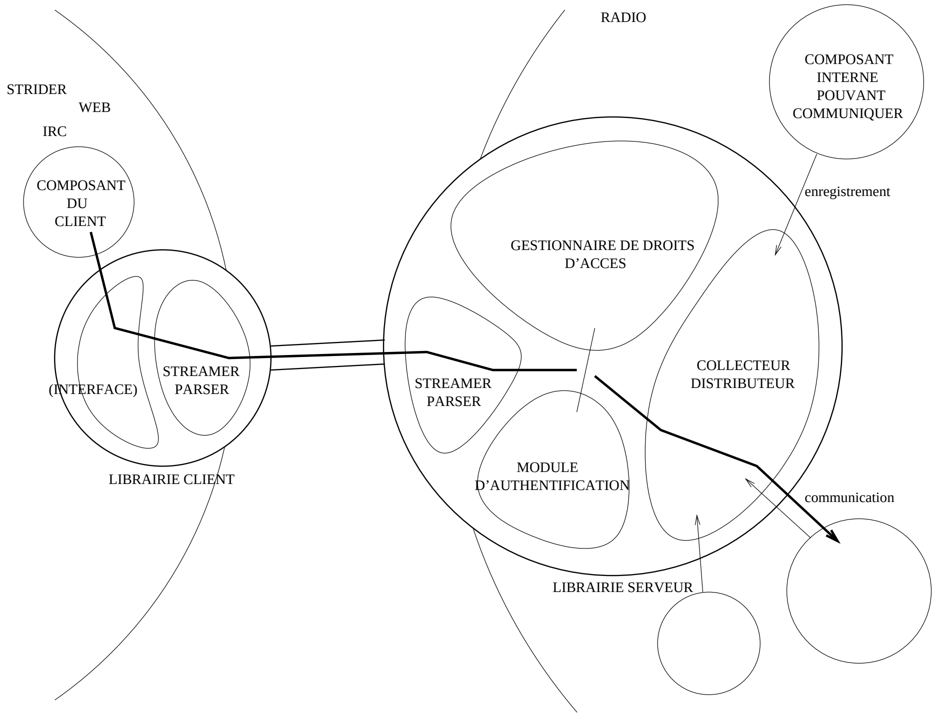
- Guillaume Burel (Responsable 100 %)
- Florent Becker (90 %)
- Samuel Mimram (50 %)
- Jean-Baptiste Rouquier (100 %)
- Martin Devautour (90 %)
- Gaétan Richard (30 %)

3.4.2 Description

L'objet de ce sous-projet est de permettre à la radio d'étoffer ses possibilités de programmation en donnant la possibilité d'indexer de manière à la fois automatique et intelligente de grands ensembles de morceaux. Ces morceaux, destinés à être diffusés par la radio, doivent être classés, de façon à ce qu'on puisse

FIG. 2 – Schéma récapitulatif du fonctionnement de LibSavonet

ACCÈS dans la figure



les retrouver en fonction de leur auteur, mais aussi (par exemple) de critères comme leur popularité ou leurs similitudes. Pour cela, strider maintient deux bases de données, l'une permettant de savoir où sont les *fichiers* (par exemple s'ils sont accessibles par un réseau, et comment) et l'autre permettant de garder les informations plus abstraites sur les *morceaux*. Il contient de plus un langage de requête permettant d'exprimer les critères sur lesquels seront choisi **s** les morceaux (les requêtes seront par exemple : «où puis-je trouver un morceau des Green Foxes populaire et durant moins de 3 minutes ?»)

Le premier objectif de ce sous-projet est de s'occuper des bases de données contenant les informations sur les fichiers à récupérer. Deux approches complémentaires sont utilisées :

1. faire périodiquement le tour des machines accessibles et ajouter à la base de données les nouveaux fichiers ; **et retirer ceux qui ont disparus ?**
2. mettre à jour la base de données quand la récupération d'un fichier n'a pas pu s'effectuer correctement.

Il faut aussi s'occuper de récupérer les informations dans les fichiers. Ceci peut se faire en cherchant dans les fichiers les tags Id3 pour les fichiers mp3 (on utilise la norme Id3 version **s** 1.1 et 2.4 **pourquoi cette restriction ?**), ou l'équivalent de ces tags que l'on trouve dans les fichiers au format vorbis. Malheureusement ces tags sont souvent absents, incomplets voire incorrects. Ces informations seront donc améliorées en regardant le chemin/nom du fichier et en les comparant aux données préexistantes dans la base de données (par exemple pour le nom de l'artiste).

Le deuxième objectif est de rendre la playlist par défaut « intelligente ». Pour cela, il faut faire du *data-mining*, comme chez Casino© : si on achète du citron et du sel, on a droit à une réduction sur la Tequila[5] ; ici si les auditeurs demandent fréquemment Alizée et Lorie sur un canal, il faudra mettre également Tatu sur la playlist. Pour cela il va falloir faire des statistiques sur les requêtes de façon rapide, que l'on puisse utiliser pour créer la liste par défaut.

À terme l'objectif serait que, en l'absence d'autre contenu prioritaire à diffuser (requêtes explicites, jingle, flash info, ...), on tienne compte des goûts des auditeurs habituels.

Une telle démarche populiste n'est-elle pas le degré zéro de la programmation radiophonique ?

3.4.3 Dépendances

Ce sous-projet a de nombreux liens avec LiquidSoap, puisque celui-ci se sert des informations de la base de données, et de la playlist par défaut. Ce sous-projet aura besoin des informations recueillies par LiquidSoap pour les statistiques d'écoute, ainsi que des informations recueillies par l'interface Web (projet Soapdish) pour avoir un retour des auditeurs (vote pour le morceau en cours par exemple). **botanik ? ?**

Les requêtes à la base de données se font à l'aide de LibSavonet, il y a donc une forte dépendance avec ce sous-projet.

[5] L'abus d'alcool est dangereux pour la santé, à consommer avec modération

3.4.4 État actuel

première base de données La partie chargée de remplir la base de données à partir des fichiers présents sur un réseau Samba ou FTP est en cours de test. Son architecture permet de déterminer les conditions dans lesquelles les serveurs seront scrutés. Actuellement, elle permet d'indexer des fichiers mp3 et ogg. L'exécutable chargé de l'indexation est `strider`, mais son nom pourra changer pour plus de clarté (`strider` est censé être le programme chargé de retrouver les morceaux à partir des requêtes.)

`strider` parcourt le réseau, dont la configuration est enregistrée dans sa base de données,

je suppose qu'il ne parcourt que des répertoires autorisés par leurs propriétaires... ce n'est pas parcequ'un tiroir est ouvert qu'on a le droit de le fouiller ... Mécanisme d'autorisation décrit où ? Stockage de login/mots de passes ? Sécurité de la BD contenant les infos personelles ? (CNIL ?)

de manière équitable, et recense les fichiers mp3 et ogg, dont il extrait les tags, qu'il indexe dans sa base de données. Il utilise pour ce faire OCaml-Fetch (cf paragraphe correspondant). Si celle-ci est étendue à de nouveaux protocoles, `strider` sera automatiquement étendu de la même manière, moyennant une redéfinition du format de la base de données.

On obtient ainsi une première table ; cette table est destinée à être raffinée, afin de construire la base d'où seront faites les requêtes.

`strider` est conçu pour être lancé comme un démon qui, à chaque `SIGALRM` reçu, lance le parcours d'un nouveau serveur, de manière multi-threadée.

informations recueillies dans les fichiers Pour recueillir les informations dans les fichiers mp3, une bibliothèque a été créée (OCaml-mp3Id3). Celle-ci tente de respecter au maximum les normes définies sur `http://www.id3.org/`.

pourquoi ne respectez-vous pas la norme ?

Elle permet aussi de récupérer des informations sur le fichier mp3 lui-même (taille, durée, encodage, ...).

Pour les fichiers ogg, ces informations sont recueillies dans la bibliothèque OCaml-Vorbis.

Ces bibliothèques pouvant être réutilisées à d'autres fins, elles permettent de récupérer des informations dont nous n'avons ensuite pas besoin.[6]

... inutiles dans le cadre de ce projet.

P majuscule pour la footnote pour l'anecdote...

Nous ne garderons que les informations suivantes pour les stocker dans la table `songs` :

- `proto` : le protocole utilisé pour rapatrier le fichier
- `server` : le nom du serveur
- `dir` : le chemin d'accès sur le serveur
- `filename` : le nom du fichier

[6] pour l'anecdote la norme Id3 v2.4 prévoit explicitement l'inclusion de l'image d'un poisson coloré lumineux

- `title` : le titre (fourni à partir du tag Id3/vorbis)
- `artist` : le nom de l'artiste
- `album` : le titre de l'album
- `year` : la date (pas forcément uniquement l'année)
- `comment` : un commentaire éventuel
- `tracknum` : la piste du morceau dans l'album
- `genre` : le genre (une chaîne de caractères quelconque, a priori pas forcément un genre défini dans la norme Id3 v1)
- `duration` : la durée du morceau
- `samplerate` : la fréquence de l'échantillonage
- `stereo` : le mode d'enregistrement
- `bitrate` : la taille de chaque échantillon
- `encoding` : l'encodage (échantillonage à taux fixe ou variable)
- `filesize` : taille du fichier
- `normalize` : facteur de normalisation (rempli par la suite, uniquement après une première diffusion)
- `status` : le fichier est-il rapatriable ?

requêtes Nous avons défini les requêtes qui pourront être faites à la base de données. Celles-ci seront transmises à la base de données grâce à LibSavonet, nous auront donc ensuite à définir plus précisément comment.

Une requête est du type suivant

```
type request =
  | And of request list
  | Or of request list
  | Range of (criterion * bool) * comparison * request
  | Leaf of field
```

`Range((critère, true), comparaison, requête)` exécute la requête `requête`, puis la trie suivant `critère` (en ordre inverse si on a mis `false` à la place **mal dit**) et enfin ne conserve que ceux dont le numéro satisfait `comparaison`. Un critère porte sur l'année, la durée ou la popularité.

pourquoi cette restriction ? pourquoi ne puis-je pas sélectionner que des morceaux d'introduction d'album ?

Les paramètres `field` sont définis par

```
type field =
  | Title of textfield_parameters
  | Artist of textfield_parameters
  | Comment of textfield_parameters
  | Path of textfield_parameters
  | Server of textfield_parameters
  | Name of textfield_parameters

  | Year of comparison
  | Duration of comparison
  | Popularity of comparison

  | Genre of string
```

interprétation des requêtes En ce qui concerne la gestion des requêtes, la structure dans laquelle elles seront représentées en interne a été définie. Nous avons essayé de faire en sorte que le langage soit suffisamment puissant pour répondre aux besoins des administrateurs, tout en restant relativement facile à manipuler. Il reste encore à traduire ces requêtes internes en requêtes SQL, que l'on pourra envoyer à la base de données au travers de la bibliothèque OCaml-mysql.

Les principaux problèmes que nous rencontrons actuellement proviennent du fait que mysql n'implémente qu'une sous-partie du langage SQL, et ne nous permet pas de faire la traduction aussi simplement que nous l'avions prévu lors de la spécification. **pourquoi, alors, avoir choisi mysql ? ? ?**

statistiques Une réflexion est en cours sur le moyen d'intégrer des statistiques sur le nombre de diffusions et les votes des utilisateurs sur le morceau en cours. Elles serviront à remplir le champ « popularité ».

OCaml-Fetch Pour le parcours du réseau, Strider utilise une interface globale OCaml-Fetch regroupant Samba, FTP et la bibliothèque Unix **? ? ?** . La partie concernant le FTP est réalisée, celle sur le protocole Samba, OCaml-SmbClient est écrite mais il reste des problèmes de conflits de variables entre la bibliothèque utilisée (écrite en C) et OCaml en cours de résolution.

L'interface est écrite mais elle demande encore quelques temps afin d'être améliorée : le principal problème est de trouver une vision communes aux protocoles utilisés et de permettre une extension à d'autres protocoles.

3.4.5 tâches **Tâches** futures

deuxième base de données Nous devons maintenant construire la deuxième base de données qui contiendra les informations plus pertinentes qui seront utilisées pour les requêtes. Pour l'instant une approche simple est envisagée, c'est à dire rapprocher uniquement les fichiers possédant les mêmes caractéristiques (titre, artiste, ...).

Cette base de données contiendra non plus des fichiers, mais des morceaux.

élimination des typos des titres, etc ? Comment est-ce fait ?

Elle contiendra donc les informations sur le morceau (titre, artiste, album, année, piste, genre, durée), ainsi qu'une liste de pointeurs vers les entrées de la première base de données qui correspondent à ce morceau. Ces pointeurs seront classés suivant leur qualité, selon un ordre qu'il nous reste à définir et qui fera intervenir les informations sur la fréquence d'échantillonage, le mode, ...

data-mining L'architecture de cette deuxième base de donnée devrait permettre de faire du data-mining. Une API sera mise en place, et si on dispose de suffisamment de temps, quelques pistes seront explorées, comme l'utilisation de Google ou Amazon pour explorer les ressemblances entre les morceaux, ou l'enregistrement des préférences par utilisateur ou par canal.

tests La gestion de la playlist par défaut devra être jugée « satisfaisante ». Il est difficile de définir ici des critères objectifs pour ceci. Il faudra donc envisager des tests avec des personnes qui n'ont a priori **italique** pas de liens avec l'équipe Strider.

Une série de tests devra alors être effectuée, pour juger de l'« intelligence » de la playlist par défaut.

3.5 Soapdish (SP4)

3.5.1 Équipe de développement

- Laurent Beaudou (90 %)
- Louis Esperet (90 %)
- Mathias Péron (Responsable 100 %)
- Damien Regnault (100 %)

3.5.2 Objectifs

L'objectif est de développer l'interface web de la radio, c'est-à-dire un ensemble d'outils permettant tant l'écoute interactive de la radio que son administration.

De cet objectif découle la co-existence de différents acteurs :
- auditeur radio ;
- administrateur de l'interface web ;
- administrateur radio.

Le portail web que nous livrerons devra être configurable par nos clients, de telle sorte que chaque radio puisse avoir sa propre identité.

3.5.3 Etat **État** actuel

Le site internet **Internet**, en l'état sur le cvs, est visible à l'adresse `http://sparc7.residence.ens-lyon.fr`.

Le développement a suivi trois axes :
- le plus important d'entre eux est la mise en place du squelette des pages. À présent terminé, il va permettre de développer les outils d'interaction avec les utilisateurs (BibMusic par exemple).
  Ce squelette se compose d'un bandeau, où apparaissent en bonne place les informations actuelles du canal écouté par l'utilisateur, ainsi qu'un espace pour effectuer une opération de login. Enfin le reste de la page a été divisé en trois colonnes : un espace principal flanqué de deux colonnes de menu **s** qui comportent d'un côté un accès à la radio et à ses canaux, de l'autre un module de requête ;
- le second axe a été la validation en XHTML1.1 de notre code qui nous a amené, et ce de façon surprenante, à re-spécifier totalement les paramètres d'affichage à l'aide des feuilles de style CSS.
  pourquoi ? expliquer ?
  Ce fut donc une phase de développement à la fois enrichissante (apprentissage fin de l'utilisation des feuilles de style) et bénéfique pour la compatibilité du site internet avec l'ensemble des butineurs ;

- enfin, des outils ont déja été développés, ou sont en cours de développement : le système interne de nouvelles est en phase terminale et la page de transmission de requête en élaboration, en collobaration **collaboration** avec Strider. Ce qui ce **se** cache derrière ce travail, c'est la mise à niveau des développeurs dans le langage PHP et dans l'utilisation de MySql, ce qui inclut l'utilisation de PhpMyAdmin.

Ci-dessous, chaque développeur relate plus précisement l'état d'avancement :
- *le squelette* clairement indenté le code fait appel à plusieurs javascripts pour améliorer la navigation (navigation *paresseuse* !). Javascript est surtout intervenu dans la barre d'avancement de la chanson en cours ainsi que le système horaire. Le site se résume à l'unique fichier index.php qui est fondé sur une approche de taille minimale, jouant sur des inclusions de fichiers. Un ensemble non négligeable de variables permet déja de personnaliser le site, variables regroupées là encore dans un fichier à part. Enfin le site étant visible par l'ensemble des protagonistes du projet, le squelette est bel et bien le fruit d'âpres discutions **discussion** qui garantissent la simplicité quant à l'emplacement des éléments.
- *XHTML1.1* La page web a été passée au « validator » du w3c, disponible sur le site `http://validator.w3.org/`. Mettre la page web aux normes du w3c nous assure que celle-ci sera compatible avec **la plupart des -> les** navigateurs **digne de ce nom**.
  Dans un premier temps, nous avons répertorié des fautes de syntaxes récurrentes que nous avons listées pour éviter de les commettre de nouveau par la suite. En XHTML1.1, il faut :
  - placer toutes les valeurs des attributs entre " " ,
  - mettre les attributs length, height en feuilles de styles css ,
  - fermer toutes les balises.

  n'est ce pas ce que l'on trouve dans tout "howto" parlant de la migration de html à xhtml ?

  Corriger ces erreurs était simple mais long car elles venaient fréquemment. C'est pourquoi nous avons choisi de valider la page dès maintenant car ces fautes peuvent être très facilement évitées lorsque le codeur est averti.
  Dans un deuxième temps, nous avons corrigé les erreurs plus spécifiques au code. Certaines ont pu être corrigées facilement. D'autres ont été signalées à leur codeur pour éviter de mauvaises interprétations qui auraient pu aboutir sur des bogues ultérieurement. Par exemple un problème de gestion de lignes et de colonnes dans un tableau, ou un problème avec le symbole & dont la syntaxe change selon l'utilisation que l'on en fait. Dans l'état actuel des choses, il ne reste plus que 13 fautes à corriger pour rendre la page conforme ;
- *modules de nouvelles* mon **? ? ?** travail a consisté à gérer le système de news. La base de donnée *savonet* sur sparc7 se compose (pour l'instant) de 2 tables : *news* et *channel*.
  La table *channel* contient la liste des canaux (des strings de 10 caractères maximums) et de leur identifiant, ainsi qu'un bref commentaire pour chaque canal.
  La table *news* contient pour chaque news l'auteur, le titre, la date et

l'heure où elle a été postée, un bit qui dit si cette news est globale (elle apparaîtra ainsi sur les news de tous les canaux), de manière optionnelle le canal auquel elle est destinée, un bit qui dit si elle a été archivée, ainsi qu'un identifiant unique.

booleen ? c'est amusant de voir choisir un langage hat niveau puis d'entendre parler de bits

Selon que l'on désire regarder les news ou les archives d'un canal, les news de la table avec un bit *archived* à 0 ou à 1, avec un bit *all_channel* à 1 ou un identifiant de canal correspondant au canal visité sont affichées.
En mode administrateur, il est possible d'effacer ou d'archiver n'importe quelle news. Cela supprime la news de la table ou cela modifie son bit *archived* selon l'action.
Sinon on peut toujours poster une news sur un canal spécifique. Concrètement cela insère la news dans la table, avec la date et l'heure courante. Il y a une petite manipulation pour afficher la date au format français ;

– *modules de requête*
  – en ce qui concerne les requêtes passées au moyen du site web, l'état actuel des choses est une boîte pour les requêtes simples (avec simplement les champs "artiste" et "titre") ainsi qu'une page de requête avancée où les champs sont beaucoup plus nombreux et permettent de cibler plus précisément la chanson ou le fichier désiré. De plus, il y a aussi sur cette page un champ réservé pour passer directement une requête dans le langage que nous donnera le strider,
  – pour la being-done list **? ? ?**, la page des requêtes avancées est en train de subir un remodelage dû à la discussion de la dernière réunion de projet pour laisser encore plus de possibilités à l'utilisateur moyen,
  – l'attente de la bibliothèque des agents pour passer nos requêtes au Strider a finalement pris fin et la to-do list est donc d'interfacer le PHP vers OCaml à l'aide de OCaml-CGI afin d'obtenir le plus vite possible un site web par lequel les requêtes pourront transiter.

3.5.4 Travail à venir

Les périodes de développement, et le découpage des tâches ont subi de profondes modifications. Et ce à deux titres : un problème de dépendances qui a été mal évalué est en cause.
Le développement de l'interface web n'aurait dû intervenir qu'une fois que les outils auxquels elle devait accéder aient été terminée **orthographe !**.
En effet d'un côté l'interface doit avoir le moyen d'accéder aux informations de Strider– pour les requêtes principalement – mais également elle a **également [ici]** besoin de certaines informations radio. La spécification récente de LibSavonet, avec la notion d'agent, ne résoud **orthographe** en rien cette dépendance, puisqu'il nous faut à présent savoir **connaître** le format des messages vers Strider.
Cependant, le temps a été mis à profit pour effectuer l'ensemble de la base du site : l'intégralité de ce qui était implémentable sans cette contrainte de dépendance avec Strider a été implémentée.

Au dessus : mise en page hideuse

Les modules restant à implémenter vont être traités dans l'ordre suivant :

– mettre soigneusement au point l'outil de requête en collaboration avec les développeurs de Strider. Des scripts OCaml-CGI seront utilisés pour accéder à Strider. L'utilisation des agents deLibSavonet **espace** a été clarifiée.
– la foire aux questions : il peut paraître bizarre de la placer avec un tel niveau de priorité. On souhaite cependant que dès à présent tous les sous-projets la remplisse au fur et à mesure de leurs implémentations. Cette directive générale devra prendre effet très rapidement.
– mettre en place l'authentification des utilisateurs. Les sessions PHP ont été explorées sur des petits exemples.
– les pages des canaux avec conjointement le développement de BibMusic où l'ensemble des données que l'on souhaite collecter a été mis à jour.

Mentionons **orthographe** que l'outil GetText n'est plus une priorité au sous-projet. il s'avère en effet que cela apporte peu à l'ensemble et qu'il est préférable de se concentrer sur les outils radio.
Il est à noter que la partie administration web n'a pas été mentionnée car elle sera développée conjointement aux outils précédents (à l'instar du système des nouvelles).
Cependant le contenu de la partie administration radio devra s'affranchir d'un plus petit nombre d'élément implémenté que nous ne l'avions prévu, et ce pour la cause évoquée plus haut : le sous-projet web ne pourra pas fournir des outils développés dans les autres sous-projets quelques heures avant la remise du projet. Ceci est un point à ne pas oublier dans le cas d'un nouveau projet impliquant le développement d'une interface web : sa programmation doit s'effectuer en aval et non réellement de front avec le reste du projet.

Enfin de nouvelles discussions vont être provoquées avec les penseurs du SP 7.1 (sous-projet juridique) : il s'avère en effet que les résultats de ce sous-projet sont directement applicables à l'interface web : nos pages de collecte de données par exemple devront respecter les recommandations de la CNIL.

Ainsi, le temps qui est passé du livrable1 au présent document, a été mis a profit pour l'apprentissage de l'ensemble des langages à aborder pour la mise en place du code.

pourquoi ? pourquoi ne pas avoir tout fait en ocaml que tout le monde connaissait ? ? ?

L'implémentation devrait donc à présent suivre son cours sur un rythme beaucoup plus soutenu. Cette période nous aura surtout appris l'enjeux du sous-projet web ainsi que son interaction avec la totalité des autres sous-projets à l'écoute desquels nous devons rester.

3.6 Botanick (SP5)

3.6.1 Équipe de développement

Sattisvar Tandabany (Responsable 100 %) Florent Becker (10 %)

3.6.2 Description

Ce sous-projet est une autre interface user-friendly **italique**, qui permet d'envoyer des requêtes à la radio sans passer par l'interface web qui peut être lourde à gérer. En particulier, le bot **c'est quoi ?** peut être utilisé en mode texte. Les objectifs du bot reprennent pour beaucoup ceux de l'interface web. Le bot doit permettre une interaction inter-utilisateurs, et l'administration de la radio moyennant une authentification des utilisateurs.

Le bot prend part à un canal IRC et interprète dans un langage quasi-naturel des commandes pour accéder aux requêtes des utilisateurs. Les utilisateurs sont reconnus par leur pseudonyme, en s'identifiant auprès du bot, grâce à une commande.

Le bot sera donc capable de fournir les outils suivants :

– *un bot par canal :* **autoreplication ?** chaque canal pourra avoir son propre bot et son lot d'auditeurs authentifiés. Chaque canal possède son bot : cela implique que le programme est lancé autant de fois que nécessaire ;
– *système d'authentification :* le bot permet à un ensemble d'auditeurs authentifiés d'user de leurs différents droits sur la radio (droits d'administration, d'envoi de requêtes, de programmation ...) gérée par la LibSavonet ainsi que leurs préférences (voir plus bas) ;
– *envoi de requêtes à la radio pour un canal :* toutes les possibilités offertes par Strider en matière de requêtes pourront être demandées au bot. Les méthodes de sélection d'une chanson pourront être précises ou floues. On peut demander un type de musique, un chanteur particulier ou encore la chanson précise d'un album précis d'un chanteur précis. Ou encore indiquer directement en dur le fichier que l'on veut passer. Tout ceci au travers d'une syntaxe facile et proche d'un langage naturel ;
– *préférences d'auditeurs authentifiés :* chaque auditeur authentifié peut créer et modifier une liste de commandes qui lui est propre. Ces commandes se définissent à l'aide de primitives. Par exemple, l'auditeur authentifié sous le nom *lambda*

  vu le contexte, lamb(a)da m'eut paru plus approprié...

  s'est créé la commande « Jackson ». Lorsque l'auditeur en question (et lui seul) s'adresse au bot et lui envoie cette commande, le bot comprend la requête « passe Billie Jean de Michael Jackson sur le canal Geek » par exemple ; pour la gestion des préférences, il faut encore discuter pour savoir où cela sera stocké. Si LibSavonet se charge aussi de conserver ces préférences, elles seront utilisables depuis le web aussi. Mais il est plus vraisemblable que les commandes personnalisées seront plus internes à Botanick, afin de pouvoir utiliser le générateur de lexeur **analyseur lexical ?** à la volée (voir plus bas) ;

– *administration radio :* certains utilisateurs authentifiés comme tels, auront le pouvoir d'administrer la radio, via les fonctions fournies par LibSavonet.

Un critère de réussite du projet est la faculté du bot à correctement interpréter les commandes et à les mettre effectivement en œuvre. Un autre point a été rajouté ici. Pouvoir configurer le bot de façon à contrôler le langage qu'il utilise pour répondre et le langage qu'il comprend. Cela implique de créer un générateur de lexeur à la volée. Je réutilise pour cela le module Genlex d'OCaml complété pour permettre une plus grande souplesse quant au langage compris. De plus, comme la configuration ne nécessite qu'une édition d'un fichier de configuration, on peut demander au bot de modifier lui même ce fichier de configuration afin de prendre en compte de nouveau vocable : soit de façon automatique, soit de façon suggérée (par une commande).

3.6.3 Dépendances

Botanick doit donc être conduit, au vu de sa description, en collaboration avec le sous-projet Soapdish. De plus, il dépend de la bibliothèque LibSavonet pour ce qui est de la communication des requêtes au noyau et de l'administration de la radio.

3.6.4 Estimation de la durée de développement

Le protocole de communication sur IRC est déjà écrit. La conception du langage, en vue des commandes, peut être mise en place sans attendre que ces commandes soient effectives dans les différents modules avec qui le bot communique. Toute la partie compréhension modulaire est fini **orthographe**. Un simple fichier de configuration est à éditer pour changer la syntaxe reconnue. En ce qui concerne la reconnaissance de nouvelles fonctionnalités, cela n'est pas configurable et ne figure pas dans les points à réaliser. Le projet a fait un grand bond grâce aux générateur de lexers à la volée.

marant à l'occurence précédente de ce barbarisme, l'orthographe était différente...

La gestion des utilisateurs authentifiés et de leurs droits, ne dépend d'aucun autre module. Une harmonisation avec le serveur web est envisagée, ainsi qu'une participation à toute réunion de l'équipe web sur ce sujet. Ceci devrait être fait en une semaine. S'il reste un peu de temps (et il semblerait, au vu de l'avancée du projet, qu'il en reste), l'extension à d'autres supports comme le mail ou ICQ, pourra être envisagée.

3.7 Mixer (SP6)

3.7.1 Équipe de développement

– Samuel Mimram (80 %)
– Julien Cristau (100 %)

3.7.2 Description

Le but de ce sous-projet est de faire l'interface entre les fichiers sonores récupérés auprès des différentes sources, et le serveur icecast.

On doit fournir au serveur icecast, par l'intermédiaire de la bibliothèque OCaml-Shout, un flux au format ogg/vorbis. On voudrait donc pouvoir convertir à la volée les fichiers sources dans ce format, après un pré-traitement pouvant consister (par exemple) en :

- fade-in/fade-out au changement de morceau ;
- normalisation du niveau sonore ;
- encodage en différents bitrates pour la diffusion en fonction de la bande passante des clients.

Pour cela on utilisera différentes bibliothèques interfacées : OCaml-Shout pour l'envoi des données à un serveur Icecast, OCaml-Mad pour le décodage de fichiers au format mp3, OCaml-Vorbis pour le décodage des fichiers au format ogg/vorbis et le réencodage final.

3.7.3 Réalisation effective

Par rapport aux ambitions que nous nous étions fixées, cette tâche est un véritable succès. En effet les bibliothèques ont été interfacées afin d'y appeler les fonctions dont nous avions besoin :

- la bibliothèque OCaml-Shout permet d'envoyer des données au serveur icecast. La bibliothèque shout a été intégralement interfaçée ce qui permet de communiquer avec les serveurs de diffusion audio. Les formats ogg et mp3 sont supportés ;
- la bibliothèque OCaml-Mad permet de décoder des fichiers audio au format mp3 vers des données au format wav, bloc par bloc ;
- la bibliothèque OCaml-Vorbis permet de manipuler les fichiers audio au format ogg. Elle permet en particulier de décoder, d'encoder, de lire les en-têtes ou d'obtenir des informations sur les paramètres d'encodage de fichiers ogg.

Nous arrivons maintenant par exemple à convertir à la volée des fichiers du format mp3 vers le format ogg pour les diffuser.

Il est à souligner que de nombreuses difficultés ont été posées par le manque (voire l'absence totale) de documentation pour ces bibliothèques. Il nous a fallu comprendre et intégrer le code source de programmes les utilisant, ces programmes étant eux-même complexes et quasiment pas commentés.

Ces bibliothèques ont de plus été développées dans l'optique d'être réutilisables et sont entièrement indépendantes. Ainsi un projet OCaml nécessitant la manipulation de fichiers audio mp3 ou ogg ou la communication avec un serveur icecast devrait pouvoir rapidement et facilement réutiliser nos bibliothèques. Des packages Debian et Gentoo en ont de plus été faits pour faciliter l'installation pour les utilisateurs de programmes utilisant ces bibliothèques.

Enfin nous avons, en collaboration avec les membres du noyau, réalisé une bibliothèque de manipulation de buffers audio qui permet de les mixer et changer leur volume ainsi que de convertir leur format en un format canonique et ce, sans perte audible de qualité.

La tâche étant maintenant quasiment terminée, Samuel va rejoindre pleinement l'équipe Strider et Julien va aider Sylvain à homogénéiser et finaliser les fichiers autoconf et les Makefiles ainsi qu'à interfacer LibSavonet pour qu'elle soit utilisable depuis l'interface web en PHP.

? ? ? Je ne comprends pas cette conclusion vu les parasites à chaque début de morceaux pendant plusieurs seconde lors de la démo...

Cette remarque ne sera prise en compte car le L2 est évalué avant la demo. EC.

3.8 Administration du projet (SP7)

3.8.1 Équipe de développement

- Louis Esperet (10 %)
- Laurent Beaudou (10 %)
- Martin Devautour (10 %)
- Sylvain Chevillard (50 %)

On distingue deux sous sous-projets qui ont trait à l'administration du projet :

3.8.2 SP 7.1 : Affaires juridiques

S'occupent de ce sous-projet Louis Esperet, Laurent Beaudou et Martin Devautour. Ce sous-projet ne se fixe pas vraiment d'objectifs précis : le but est de collecter un maximum de renseignements pour éclaircir la situation juridique d'une radio streamée en réseau local. Il y a plusieurs problèmes à resoudre :

- en sachant qu'il est illégal de diffuser des morceaux des droits desquels on ne s'est pas acquitté, est-il illégal de fournir un logiciel qui permettra cette diffusion ? Kaaza est-il l'objet de procès du même type que Napster ? Pourquoi ?
- le fait de diffuser sur un réseau local change-t-il quelque chose vis-à-vis des droits d'auteurs ? Est-ce considéré comme de la diffusion grand public ?
- Une radio streamée a-t-elle les mêmes obligations (quotas, droits, playlist) qu'une radio hertzienne ?
- Est-il légal (vis à vis de la CNIL) de stocker les préférences et les goûts des utilisateurs. Faut-il les mettre dans une base de données ou les stocker dans des cookies (donc sur les machines des utilisateurs) ?

3.8.3 Questions juridiques : objectifs et état d'avancement

L'objectif principal est simplement d'avoir les idées un peu plus claires sur les questions juridiques liées à notre projet (par exemple : savoir si le simple fait de distribuer un logiciel permettant de diffuser illégalement de la musique est, en soi, illégal).

Pour l'instant, nous nous sommes simplement documentés en consultant les sites internet de la CNIL et de la SACEM, et en épluchant la jurisprudence (française) concernant les droits de diffusion.

Voici ce qui ressort du site de la SACEM :

- Pour les webradios streamées générant des recettes via leur site web, la SACEM perçoit 6 % des recettes du site avec un minimum garanti de 145 euros HT par mois et avec une majoration de 5,35 euros HT par tranche de 100 000 pages vues par mois.
- Pour les webradios exploitées par des particuliers ou des associations à but non lucratif et ne générant pas de recette via leur site Internet, la SACEM

perçoit un forfait de 72,5 euros HT par mois avec une majoration de 2,7 euros HT par tranche de 100 000 pages vues par mois.

La SACEM écrit également que *concernant l'étendue de l'autorisation applicable aux WebRadios telle que définies précédemment, elle ne couvre pas les services à la demande permettant aux internautes soit d'agir sur la composition du programme en vue de sa modification ou de sa personnalisation, soit de sélectionner des parties du programme pour les écouter au moment voulu par eux. Cependant, le diffuseur est autorisé à proposer des archives de son programme de radio représentant une durée maximale d'un mois de programmation destinées exclusivement à l'écoute en streaming par les internautes à l'exclusion de toute faculté d'écoute oeuvre* **utiliser œ** *par oeuvre* **utiliser œ** *à la demande et de téléchargement.*

Si on considère ce qu'on propose plutôt comme un service d'écoute à la demande il faut payer à la SACEM :

- Une redevance proportionnelle aux recettes réalisées, avec un taux de 12% ramené, à titre provisoire, à 8 % du prix hors taxes payé par le consommateur.
- Cette redevance est assortie d'un minimum garanti de 0,10 euros **pas d'espace**, pour la première année d'exploitation, par oeuvre musicale mise à disposition dans la base de données.

Nous avons contacté le responsable d'une radio streamée, afin de profiter de son expérience sur la question (plus précisément pour savoir si le fait que la radio soit en intranet uniquement peut avoir une incidence sur les tarifs de la SACEM). Cet échange devrait nous apporter des informations complémentaires dans les semaines à venir. Nous allons rédiger à l'intention des futurs administrateurs de radio, un document pour les informer des différentes règles à respecter pour rester dans un cadre légal : droits de diffusion, règles imposées par la CNIL... Ce document d'information est très important puisqu'il permettra au particulier de connapitre **orthographe** exactement les conditions dans lesquelles il peut créer une radio tout en restant légal **tout en respectant la législation**. Ce document pourra être considéré comme la synthèse de nos recherches sur le sujet. Si cela se justifie, c'est-à-dire si certaines questions que nous avons résolues ne rentrent pas précisément dans le cadre de ce document d'information, nous rédigerons un second document traitant des questions annexes. Ce document sera lui aussi distribué avec Savonet. Il comprendra éventuellement un petit historique des précédentes affaires qui peuvent être reliées à notre projet (procès de Napster, création de la première radio streamée, etc.)

3.8.4 SP 7.2 : Documentation et supervision du projet

Cette tâche est assurée par Sylvain Chevillard. Il s'agit de superviser les différents sous-projets afin d'harmoniser les standards, les routines de compilations (Makefile et autres gnu-eries). C'est aussi Sylvain Chevillard qui s'occupe de la documentation du projet à l'usage des utilisateurs et qui coordinera la documentation interne.

3.8.5 Documentation générale : objectifs et état d'avancement

Les deux livrables, que j'ai **?** entièrement supervisés et relus, et dont j'ai **?** écrit une bonne partie, introduisent à mon sens assez bien le lecteur à l'architecture générale de Savonet, à sa problématique et ses difficultés. On peut donc considérer ces deux document comme partie intégrante de la documentation interne.

En outre, j'ai rédigé un document appelé `archi_generale.tex` qui présente plus en détail que les livrables les articulations des différents projets les uns par rapport aux autres. J'ai assuré aussi la traduction en anglais d'un document initialement rédigé par Samuel Mimram qui présente le projet de façon vague et générale et intitulé `architecture.tex`. Ce document est plus une présentation dédiée à un usage externe. C'est d'ailleurs ce qui a motivé sa traduction rapide.

Un gros travail de documentation reste encore à effectuer, tant interne qu'externe. Mes principales difficultés viennent du fait que je ne peux pas écrire tout moi-même, mais que les membres des divers sous-projet n'ont pas forcément suffisamment de temps pour s'occuper d'écrire de la documentation. Le fonctionnement que nous adoptons est donc généralement le suivant : sur la base d'une discussion avec les membres d'un sous-projet, je rédige ou bien un membre du sous-projet rédige une ébauche de document, à partir de laquelle nous travaillons en commun pour arriver à un document final accessible à quelqu'un d'étranger au projet.

J'ai insisté et j'insisterai à nouveau pour que le code soit commenté à l'aide de OCaml-Doc afin de le rendre facilement compréhensible par tous. En ce qui concerne le SP4 (Soapdish), j'ai insisté pour que le code qui représente de la mise en page soit clairement différencié du code qui sert à implémenter telle ou telle fonctionnalité (javascript, accès aux bases de données, script cgi, etc.) Mon objectif est que l'utilisateur de Savonet ne soit pas obligé de passer par les pages web proposées par Soapdish. Il doit pouvoir écrire son propre site comme ça l'enchante (pour l'intégrer par exemple parfaitement à un site préexistant) mais en même temps, il ne doit pas lui être difficile d'aller récupérer dans le code source des pages proposées par Soapdish le code de telle ou telle fonctionnalité pour l'intégrer à sa page.

En ce qui concerne la documentation interne, j'ai pour objectif de fournir deux documents internes au SP1 : « How to write a source » et « How to write an operator » pour permettre à chacun de développer ses propres sources et opérateurs de façon standard. Par ailleurs, je prévois un document externe : « How to program your radio » **utiliser le tilde pour les espaces insécable** pour expliquer la syntaxe du langage de programmation du noyau. Il y aura de nombreux exemples types de programmation de radio, dans lesquels pourront puiser les animateurs de radio Savonet.

Je prévois une documentation interne au SP2 visant à expliquer comment créer et faire communiquer un agent avec le reste de la radio. Cette partie est encore très floue et demande un travail de discussion important avec les membres de ce sous-projet.

Enfin, il y aura bien sûr une documentation externe de Savonet, expliquant commet **orthographe** installer et utiliser Savonet.

Julien Cristau, qui a fini son travail dans le SP6 m'assistera grandement dans le travail d'unification des Makefile et de création des scripts de configuration et d'installation, ces tâches me demandant vraisemblablement trop de travail dans les semaines à venir.

Dommage que vous ne prevoyez nulle part d'apprendre à utiliser un correcteur orthographique. Où est la bibliographie ? La synthèse en 3.8.3 sans une riche biblio sera absolument inutile !