Gestion des listes de diffusion, autour de SYMPA

Benoît MARCHAL
Direction du Numérique
Université de Lorraine
28, rue Lionnois
54000 NANCY

Résumé

Sympa est un gestionnaire de listes de diffusion complet et paramétrable.

Mais dans une université importante (10 000 personnes, plus de 62 000 étudiants), l'ordre de grandeur pose souci, par exemple, le nommage des listes.

Il faut en faciliter l'usage (création, type de liste prédéfini …) à la fois pour les néophytes, mais aussi pour les administrateurs confirmés.

La cohérence, le peuplement automatique sont nécessairement en liaison avec le système d'information. Il faut mettre en place des rappels automatiques pour « éviter » les oublis de validation ou d'authentification, des listes privilégiées pour écrire en « shuntant » les scénarios spécifiques … et enfin un certain nombre de routines et de configurations adaptées plus ou moins à la structure.

Alors comment peut-on réaliser cela ? À l'université de Lorraine, nous avons écrit des interfaces dédiées, des scripts périodiques, fait des adaptations en utilisant les possibilités du logiciel (custom conditions, scénarios, filtres …).

Ce sont plus de 5000 listes pour le personnel, 2000 listes pour les étudiants et d'autres pour des partenaires qui sont gérées sur nos différents robots.

Cette présentation veut donner un aperçu de ce que nous avons réalisé. Pourquoi ne pas le partager avec d'autres ?

Mots-clefs

SYMPA, Écosystème, SI

1 Introduction

En 2012, les quatre universités de Lorraine fusionnent pour former l'université de Lorraine. Dès le début, la problématique des adresses électroniques est posée.

Les adresses personnelles sont traitées avec le logiciel ZIMBRA. Une même personne peut avoir alors plusieurs adresses valides.

Fort d'une longue expérience avec SYMPA dans chacune des quatre anciennes structures, il est décidé de le conserver et de l'appliquer pour toutes les adresses génériques.

On définit alors un nommage des adresses, puis les moyens à mettre en œuvre pour faciliter et suivre les créations et les usages.

Avec SYMPA et ses nombreuses personnalisations possibles, il est possible d'écrire relativement facilement des interfaces de création de listes, ou bien des scripts de suivi d'exploitation.

Je ne parlerai pas des configurations « habituelles » des listes (inclusions …), juste des développements spécifiques réalisés chez nous, et je ne donnerai que quelques exemples de réalisations.

2 Contexte

L'université de Lorraine utilise trois robots (au sens SYMPA) pour ses personnels (univ-lorraine.fr), ses étudiants (etu.univ-lorraine.fr) et ses associations (asso.univ-lorraine.fr). Et nous gérons des robots pour des partenaires : l'université de la grande région (transfrontalière : Belgique, Luxembourg, Sarre, Lorraine) ainsi que pour l'école d'architecture de Nancy.

Ce sont plus de 9 000 listes gérées avec, pour la plus peuplée, environ 62 000 adresses d'étudiants.

Chaque jour, notre SYMPA envoie entre 10 000 et 350 000 messages avec une moyenne supérieure à 100 000 (chiffres 2018). Il traite en moyenne 5 100 messages entrants par jour.

L'application fonctionne sur une machine virtuelle biprocesseur avec 12 Go de RAM et un système Centos 7. L'installation est faite en utilisant les paquets (RPM) fournis.

Environ 4 000 listes utilisent des peuplements automatiques : requêtes LDAP, MySQL ou Oracle ainsi que des inclusions de listes, etc.).

3 SYMPA, petit rappel

SYMPA, Système de Multi-Postage Automatique (https://sympa.org/) est un gestionnaire de listes de diffusion écrit en Perl.

À l'origine développé par le CRU et RENATER, il concurrençait des logiciels comme Majordomo ou Listserv. Aujourd'hui, une communauté plus élargie travaille sur son développement.

En dehors de sa fonction d'envoi de messages aux abonnés d'une liste en fonction des droits, il permet de déléguer la configuration aux propriétaires : l'émission de messages, les abonnements, la visualisation de la liste et de ses abonnés, l'archivage, le stockage de documents, etc.

Il peut gérer des domaines de messagerie différents (ou « robots »), sur une même instance.

On se reportera à la documentation pour connaître l'ensemble des fonctionnalités. Dans cet article, on reviendra seulement sur certaines d'entre elles.

À l'origine, la majorité des tâches se faisait au travers de la messagerie électronique, et depuis de nombreuses années, une interface web facilite le travail. L'identification était et reste basée sur l'adresse électronique de l'utilisateur.

Une gestion fine des droits est possible en liaison avec le système d'information (utilisation d'un annuaire LDAP, d'une identification SSO, etc.).

On utilise pour cela des scénarios, personnalisables en fonction de la tâche à effectuer. De très nombreux sont livrés directement avec le logiciel. Un mécanisme de surcharge permet de gérer des priorités (dans l'ordre d'importance : liste, famille, domaine, instance, défaut). i.e. si un scénario existe au niveau d'une liste il est exécuté, même si un scénario de même nom existe au niveau du robot.

En voici un exemple : lors de l'envoi d'un message à une liste privée, le scénario send.private ci-dessous est évalué (c'est le scénario par défaut) .

```
title.gettext restricted to subscribers
is_subscriber([listname],[sender]) smtp,dkim,smime,md5 -> do_it
is_editor([listname],[sender])     smtp,dkim,smime,md5 -> do_it
is_owner([listname],[sender])      smtp,dkim,smime,md5 -> do_it
true()                             smtp,dkim,md5,smime ->
reject(reason='send_subscriber')
```

Si l'émetteur appartient aux abonnés, est propriétaire ou modérateur de la liste, le message est envoyé. Dans le cas contraire, le message est refusé et l'émetteur reçoit un

message lui signalant le problème. On peut remarquer qu'il est possible d'utiliser des variables (entre crochets) permettant une écriture plus facile du scénario.

On se reportera à la documentation officielle pour connaître la syntaxe d'un scénario.

Dans le même esprit, les messages générés par Sympa utilisent des templates écrits en TT2, avec le même principe de priorité (liste, famille…). Ils permettent donc une adaptation fine au besoin.

Ci-dessous, un exemple de template utilisé pour renvoyer, par messagerie, le refus d'un message par un modérateur de la liste.

```
[%# reject.tt2 ~%]
From: [% fromlist %]
Subject: [%"Your message has been rejected"|loc|qencode%]

[%|loc(list.name,domain,rejected_by)%]Your message for list %1@%2
has been rejected by the moderator (%3).[%END%]

[%|loc(subject)%](Subject of your mail: %1)[%END%]

[% IF conf.wwsympa_url -%]
[%|loc(list.name)%]Check %1 list usage:[%END%]
[% 'info' | url_abs([list.name]) %]
[% END -%]
```

On peut remarquer que le message est localisé « [%|loc(subject)%] ». SYMPA utilise pour cela des fichiers « .mo ».

4 Les customs Conditions et les scénarios

La fusion des quatre universités, en 2012, a nécessité de mettre en œuvre des moyens pour rendre équivalentes les différentes adresses d'une même personne. En effet, il n'était pas envisageable de faire table rase du passé et chacun voulait pouvoir garder son ou ses anciennes adresses tout en utilisant sa nouvelle. Cela n'a été possible qu'en modifiant les scénarios par défaut, et donc par l'introduction de conditions personnalisées : « custom conditions ».

Si on revient sur le scénario send.private de l'exemple ci-dessus, il va se réécrire en :

```
title.gettext restricted to subscribers

CustomCondition::is_subscriber([listname],[sender],[domain])
smtp,dkim,smime,md5    -> do_it
```

```
CustomCondition::is_editor([listname],[sender],[domain])
smtp,dkim,smime,md5    -> do_it

CustomCondition::is_owner([listname],[sender],[domain])
smtp,dkim,smime,md5    -> do_it

true()
smtp,dkim,md5,smime    -> reject(reason='send_subscriber')
```

On voit qu'il suffit, tout d'abord, de remplacer le test 'is_subscriber' par 'CustomCondition::is_subscriber' puis d'écrire le module Perl correspondant ([1]) et de mettre celui-ci dans le répertoire adéquat <répertoire d'installation de sympa>/custom_conditions.

Ce module (is_subscriber.pm dans notre exemple) doit contenir une méthode 'verify' qui retourne la valeur 1 si la condition est vérifiée, -1 sinon. Dans cette méthode, nous récupérons l'ensemble des adresses d'une personne, mémorisées dans l'annuaire de l'établissement (attribut 'mail' ou 'udlMail') et pour chacune, nous vérifions la condition (« est-ce que l'adresse est abonnée à la liste? » toujours pour notre exemple). Un grand merci à Dominique Lalot de l'université d'Aix-Marseille pour son aide.

Tous les scénarios utiles ont été traités : envoi de messages, visibilité des abonnés d'une liste, etc.

Un autre usage des scénarios est la possibilité de faire évaluer un scénario systématiquement avant tout autre. Voici notre scénario « include.send.header » :

```
CustomCondition::is_privilegie( [custom_vars->privilege], 'on',
'<adresse d'une liste>', [sender], [domain])    smtp,smine ->
request_auth

CustomCondition::is_privilegie( [custom_vars->privilege], 'on',
'<adresse d'une liste>', [sender], [domain])    md5,dkim   ->
do_it
```

Ce scénario est regardé avant tout envoi de message. Les listes ayant un topic([2]) particulier possèdent une variable « privilege » positionnée à « on ». Si l'émetteur appartient à la liste '<adresse d'une liste>', et après vérification de son identité, il shunte le scénario de la liste. On y retrouve, par exemple, le président, le directeur général des services, etc. Un script vérifie périodiquement les topics des listes pour positionner cette variable dans la configuration (fichier config).

1. On le retrouve en annexe de cet article.

2. Le topic est un attribut d'une liste permettant son classement. Une liste peut avoir plusieurs topics.

Il est possible d'ajouter des variables sur une liste autant que nécessaire, on en retrouvera un autre usage ci-dessous.

Un dernier exemple est l'usage des custom conditions pour mettre un répondeur automatique sur les listes (https://www.sympa.org/contribs/vacation). C'est possible depuis la version 6.2.9 de Sympa.

Nous positionnons trois variables dans la configuration de la liste ('vacation_start' la date de début, 'vacation_end' la date de fin et éventuellement 'vacation_exclude_list' une liste d'exclusion) ainsi qu'un message d'absence dans le répertoire mail_tt2 de la liste. Nous modifions alors le scénario pour intégrer un appel pour envoyer (ou non) le message de vacation.

Par exemple :

```
CustomCondition::vacation([list->address],[custom_vars-
>vacation_start],[custom_vars->vacation_end],[custom_vars-
>vacation_exclude_list],[sender],[msg_header->Subject])
smtp,smime,md5,dkim -> do_it

include public
```

La méthode « verify » de la custom condition « vacation » va regarder et récupérer, pour une liste donnée, les valeurs définies ci-dessus et si la date actuelle est dans la fourchette, on envoie un message à l'émetteur (variable [sender]), sauf si celui-ci est dans la liste des exclus.

Dans cet exemple, la dernière ligne indique de continuer avec le scénario « send.public » habituel. On voit donc qu'il est possible d'intégrer relativement facilement cette fonctionnalité supplémentaire.

Alors pourquoi n'avons-nous pas rajouté ces lignes dans le scénario que nous avons vu plus haut « include.send.header » ? Pour le moment, les messages de vacation sont très peu nombreux et nous ne voulons pas charger le serveur par une évaluation systématique rarement utile.

Attention, nous ne sommes pas dans une véritable gestion des messages d'absences à laquelle nous sommes habitués. En effet, un message est systématiquement envoyé à l'émetteur (et non un tous les X jours).

5 Les interfaces

Pour le moment, nous sommes restés dans les possibilités habituelles du logiciel.

La problématique de la création des listes a été traitée de plusieurs manières différentes en fonction de leur usage. Nous interdisons la possibilité à un usager de demander la création d'une liste directement au travers de l'interface de Sympa.

Pour le nommage des listes, nous avons décidé de commencer autant que possible le nom de la liste soit par la composante (ex : « dfoip- »), soit par l'application (ex : « apogee- »). Le séparateur est systématiquement un tiret, le point est réservé pour les personnes.

Lorsque le renseignement existe dans l'annuaire, nous inscrivons le ou les services informatiques locaux comme propriétaires privilégiés.

5.1 Listes étudiantes : LIDIE

Pour les listes permettant de contacter une population d'étudiants, la sous-direction Système d'Information, Études et Développement a développé un logiciel spécifique LIDIE ([3]) à partir des données d'Apogée.

Voici quelques captures d'écrans reprises de la documentation en ligne :

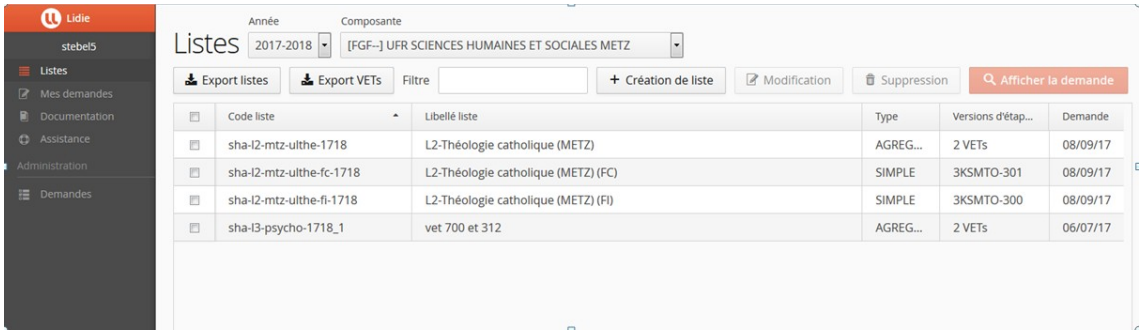

Figure 1 - Liste d'une composante

Cet écran permet pour une année et une composante sélectionnée :

- l'affichage des listes ouvertes ;
- l'affichage des demandes relatives aux listes ;
- la demande de création d'une nouvelle liste ;
- la demande de modification d'une liste ;
- la demande de suppression d'une liste ;
- la demande d'extraction d'un fichier excel des listes ouvertes ;
- la demande d'extraction d'un fichier excel des VETs([4]) ouvertes avec leurs listes associées,

3. LIDIE : LIste de Diffusion Etudiante

4. VET : Version d'étape permettant de connaître le parcours d'un étudiant

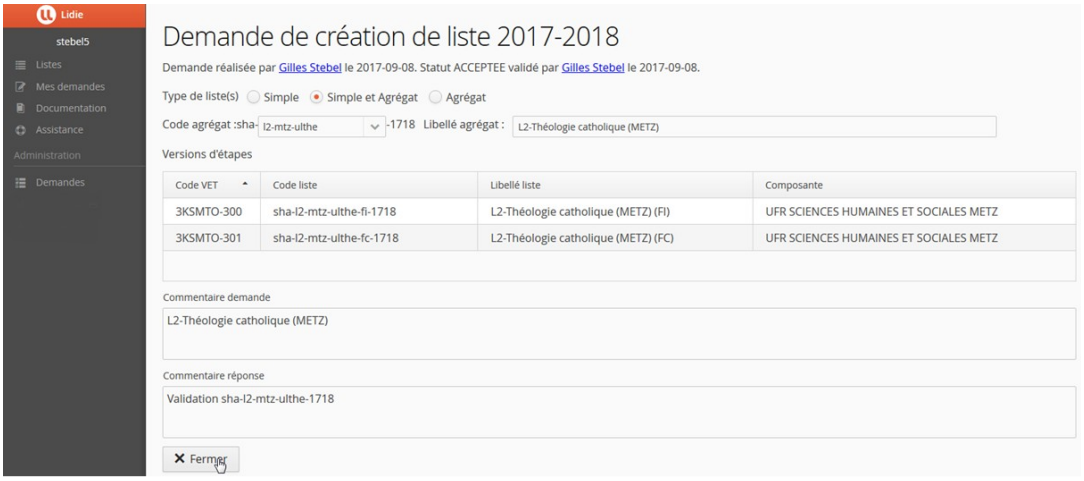

Figure 2 - Affichage de la demande relative à la liste sha-l2-mtz-ulthe-1718

Dans la procédure :

- Les personnels autorisés peuvent, à partir d'une VET, demander la création d'une liste pour l'année en cours ou bien demander une liste comprenant plusieurs VET ;
- Le nommage est normalisé et comprend : la composante, la formation, un champ libre et l'année (sous une forme yyYY avec yy l'année de début et YY l'année de fin, i.e. 1920 pour l'année universitaire commençant en septembre 2019) ;
- La demande est ensuite validée par les responsables de la scolarité de la composante.

Une liste peut donc être une union de VET (agrégat), une même VET peut appartenir à plusieurs listes.

Le nom validé est alors introduit dans l'annuaire de l'établissement comme un attribut de chacune des VET et un script quotidien est chargé de créer, modifier ou supprimer les listes demandées.

Pour cela, il parcourt la liste des VETs ouvertes sur l'année demandée et regarde les modifications apparues. Le peuplement se fait ensuite par une requête LDAP depuis l'annuaire de l'établissement en fonction des VET et de l'année de la demande. Nous utilisons pour cela un attribut multivalué, composite : supannEtuInscription.

```
filter (supannEtuInscription=*[anneeinsc=2019]*[etape={LOC}
{VET}5KIND0-500]*)
```

Nous gardons les listes sur deux années, ce qui nous offre la possibilité de rappeler les étudiants inscrits l'année précédente dans une formation.

La suppression de l'année précédente se fait directement dans LIDIE, sans intervention de notre part (sous-direction Infrastructure). Le nom de liste disparaissant de l'annuaire, notre script quotidien supprime alors les listes.

Pour information, nous gardons les étudiants dans l'annuaire six mois après leur dernière inscription.

5.2 Les autres listes

Pour faciliter les demandes, nous avons tout d'abord défini quatre profils de listes selon certains critères :

- scénarios utilisés pour émettre, visualiser les abonnés … ;
- positionnement de l'archivage ;
- etc.

Ensuite une interface a été développée permettant à tout un chacun, identifié à l'université, de faire une demande. Lors du remplissage :

- le nom demandé est vérifié dès la saisie (pour éviter un doublon), on propose l'un des trois robots gérant notre domaine ;
- Les champs obligatoires sont clairement identifiés et une aide en ligne est proposée, adaptée au contexte ;
- Le demandeur est automatiquement pré-saisi comme propriétaire, mais il peut invalider ce choix et rajouter d'autres adresses ;
- Il peut rajouter directement des modérateurs ou des abonnés ;
- Lorsque des adresses de liste sont détectées, on propose alors de faire des inclusions des membres plutôt que de l'adresse de la liste.

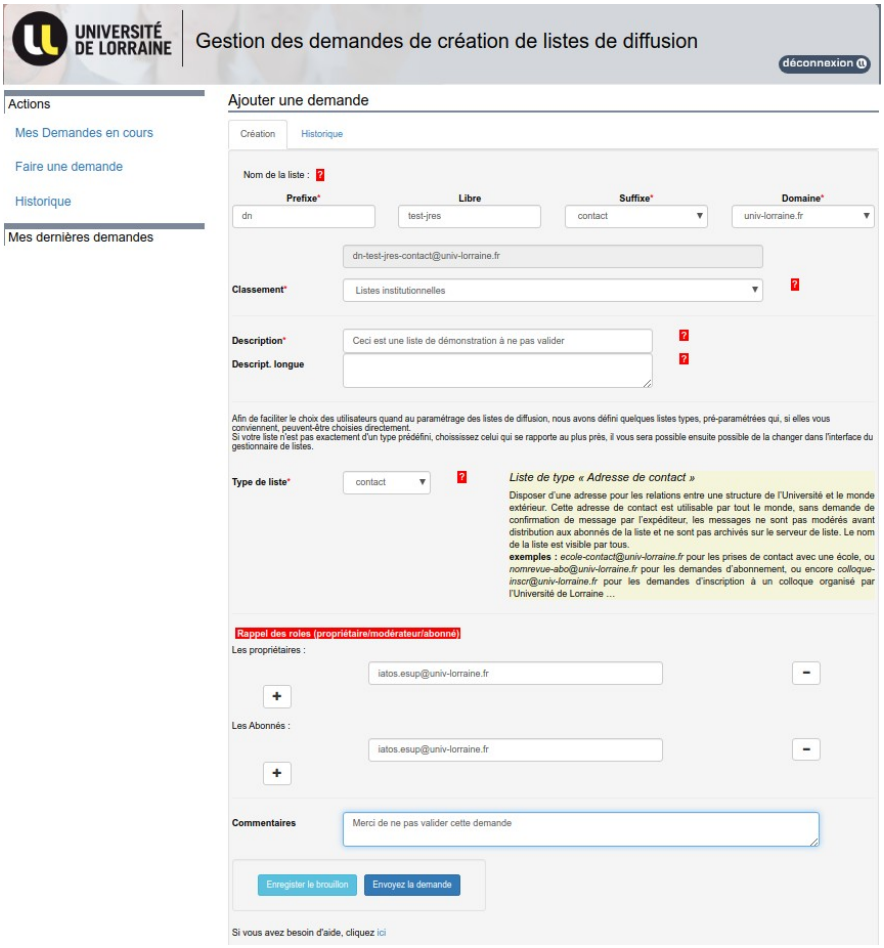

Figure 3 - : Interface de demande

Le demandeur peut suivre l'état de sa demande. En effet, un groupe de personnes est chargé de valider celle-ci pour assurer la cohérence du nommage. Il peut donc y avoir des allers-retours, toujours depuis cette interface, entre le demandeur et les validateurs. Comme dans le paragraphe précédent, le service informatique local à la structure est rajouté dans les propriétaires si le renseignement existe dans l'annuaire.

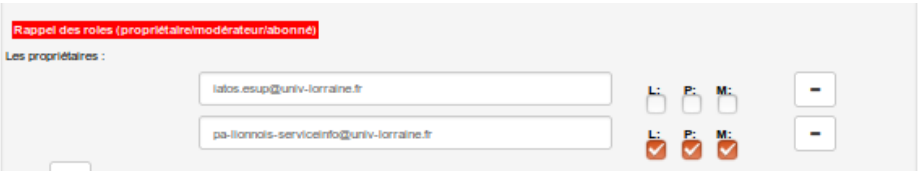

Figure 4 - Ajout automatique du service informatique lors de la validation

Lorsque la demande est validée, la liste est créée et peuplée au besoin. Le demandeur et les validateurs reçoivent à chaque étape un message électronique.

Auparavant, nous devions reprendre la demande et, au travers de l'interface fournie par SYMPA, refaire les différentes manipulations, les validateurs n'étant pas listmaster.

L’usager a accès à son propre historique.

Cette interface ne permet pas de rajouter des peuplements automatiques par des requêtes LDAP ou SQL.

On utilise ensuite l’interface habituelle de SYMPA pour la gestion de la liste.

C’est une application web, client-serveur, qui s’exécute sur l’infrastructure hébergeant les applications web standards de l’université. Côté serveur sympa, un processus serveur est en écoute des demandes lors de la validation.

Les validateurs utilisent la même interface. Ils ont accès au bouton de validation ou de rejet. Les administrateurs peuvent aussi suivre l’historique global.

Cette interface a permis la création de plus de 2 300 listes depuis son ouverture en avril 2016.

Des fonctionnalités supplémentaires sont en cours de réflexion, mais nous ne voulons surtout pas redévelopper l’interface web native de SYMPA.

5.3 Visualisation des abonnements

L’interface web de SYMPA permet de rechercher les abonnements et fonctions d’un utilisateur en appui sur les scénarios des listes (visibilité). Une question assez récurrente est le besoin pour nos collègues du service aux usagers (SU) de passer outre et d’obtenir l’ensemble des abonnements d’une personne ou bien les abonnés d’une liste précise. Avec Sympa, le seul moyen de tout voir est d’être listmaster, ce qui n’est pas évidemment pas possible dans une université comme la nôtre.

La sous-direction SU a développé un portail spécifique pour donner accès à un certain nombre d’applications (par exemple une gestion de l’Active Directory). Nous avons intégré des procédures permettant de :

– connaître les listes d’un utilisateur (abonnement manuel ou automatique, propriétaire privilégié ou non, modérateur) ;

– connaître les abonnés d’une liste.

La sécurité d’accès se fait directement par le portail.

5.4 Listes pour groupes de travail

Dernièrement, nous avons mis en exploitation un dépôt de fichiers pour groupe de travail, utilisant le logiciel Nextcloud. Depuis l’interface de gestion développée spécifiquement, il est possible de demander une liste privée, associée au groupe, créée automatiquement.

Notre interface de gestion des groupes alimente l’annuaire. La liste utilise alors une inclusion par requête LDAP. Chaque personne de l’université possède un attribut « udlGroup », multivalué, permettant de connaître son appartenance à un groupe.

Nous sommes toujours dans une application client/serveur et le démon spécifique sur notre serveur Sympa effectue les opérations de création et de destruction de liste.

5.5 Abonnement à des listes spécifiques :

À la demande de la présidence, nous avons mis en place des listes ‘expression-libre’ et ‘expression-syndicale’. L’abonnement est automatique (en fonction de la catégorie de la personne). Pour faciliter la gestion, l’utilisateur retrouve ces listes dans l’interface de gestion de son compte : https://sesame.univ-lorraine.fr/. Le programme modifie un attribut dans la fiche LDAP, pour chaque liste, qui est pris en compte par la requête de peuplement de la liste.

5.6 Pourquoi avons-nous développé nos propres protocoles client /serveur ?

Avec Sympa, il est possible d’utiliser SOAP pour réaliser des actions directement avec le serveur web (https://sympa-community.github.io/manual/customize/soap-api.html).

Un grand nombre de fonctions est possible, mais il nous manquait principalement la gestion des inclusions, que ce soit au travers de listes ou de requêtes LDAP ou SQL.

Le développement de modules ou de programmes en Perl n’est pas un problème pour nous.

La première difficulté peut venir du manque de documentation des fonctions (la documentation de programmation est directement intégrée dans le code et elle est accessible grâce à la commande perldoc, encore faut-il trouver le bon module).

La deuxième est l’évolution des fonctions dans les différentes versions, et surtout l’obsolescence de certaines, lorsqu’elles sont remplacées par de nouvelles.

Il est parfois utile de parcourir directement la base de données ou les fichiers de configuration pour des questions de rapidité. Là aussi, il faut bien faire attention au changement de structure. Les dernières versions ont passé un certain nombre d’attributs de la liste du fichier de configuration à la base de données, les propriétaires ou les modérateurs en sont un exemple.

Actuellement, nous avons plusieurs serveurs([5]) en écoute, spécifiques à chaque développement. Une des tâches va être de les fusionner pour n’avoir qu’un seul programme pouvant exécuter les différentes opérations. Cela va poser rapidement la problématique de l’identification du client. A l’heure actuelle, c’est un CGI qui envoie les demandes sur un serveur en écoute filtré par iptables. Avec un seul serveur en écoute, on ouvrira forcément plus d’accès.

6 Les scripts d’exploitation

6.1 Gestion des abonnements invalides

Les abonnements manuels, nombreux dans nos listes, provoquent de nombreuses incohérences si l’adresse de l’abonné n’existe plus. Un des premiers programmes a consisté à supprimer, de manière automatique, ces abonnements pour les adresses de

5. Démon au sens processus sur le serveur SYMPA

notre domaine. Depuis que les propriétaires/modérateurs sont en base de données, nous envisageons de faire la même chose, mais se pose alors le problème de l'absence potentielle de propriétaires pour une liste.

Pour les domaines que nous ne gérons pas (adresse d'abonné hors des domaines *.univ-lorraine.fr) ce n'est pas possible facilement, et nous n'avons rien fait pour le moment.

6.2 Gestion des rappels

Lors d'une modération ou d'une authentification, Sympa envoie un message que l'utilisateur oublie trop souvent, que ce soit un message d'authentification ou un message signalant une modération à effectuer.

En analysant les diverses files d'attente, nous pouvons envoyer des messages de rappels, en plus de celui envoyé par Sympa. Nous avons choisi de le faire deux fois : après une heure et après 24 heures. Mais, malgré ceci, nous avons toujours des messages qui restent dans les files d'attente (environ entre 70 et 100 messages).

Notre script tourne pour cela en crontab toutes les heures, il est paramétrable sur la période de rappel (par défaut 1 h et 24 h).

6.3 Gestion des suspensions

Dans la version 6.1, nous configurions, pour certaines listes, une interdiction des suspensions. Cela concerne les listes officielles, comme, par exemple l'ensemble du personnel ou bien des étudiants. Cela se passait en modifiant le template wwsympa suspend_request.tt2 et en positionnant une variable dans la configuration de la liste.

Bien que celui-ci soit toujours présent dans la distribution, ce template semble ne plus être utilisé en version 6.2.

Aujourd'hui, nous avons modifié le template suboptions.tt2 pour autoriser ou non les suspensions, toujours en fonction du positionnement de la même variable sur la liste. Il est alors aussi possible d'interdire à un utilisateur de se mettre en « nomail » pour ces listes, sans revenir dans les options de la liste.

Un script quotidien assure une vérification, et, au besoin, force les valeurs en parcourant la base de données.

6.4 Gestion des listes sans abonnés

Au cours du temps, les abonnements à une liste évoluent, que ce soit de manière manuelle ou par inclusion (de liste ou par requête). Il existe donc potentiellement des listes sans abonné. Cela pose deux soucis :

- on garde des adresses valides avec des messages qui se perdent en monopilisant des adresses potentiellement utiles,
- l'émetteur n'est pas prévenu que son message n'est pas distribué.

Le deuxième peut être traité dans le scénario « include.send.header » en rajoutant une ligne de type :

```
equal([list→total],'0') smtp,md5,smime -> reject(tt2=list_empty)
```

Il faut ensuite modifier le template de rejet reject.tt2 pour rajouter la condition « list_empty ». Cette ligne teste le nombre d'abonnés de la liste (attribut list→total) et s'il est nul, rejette le message et retourne un courrier à l'émetteur.

Le premier problème concerne plus directement les propriétaires qu'il faut prévenir. Nous avons développé un script, mais il est rarement utilisé, car les propriétaires, avec le recul, se sentent très peu concernés par ce souci. Dommage !

6.5 Et d'autres

Dans nos cartons, d'autres procédures sont soit à l'étude, soit partiellement réalisées.

6.5.1 Modification des abonnements et des droits d'une personne

En effet, dans notre université, les personnes changent relativement souvent de fonction et sont remplacées (ou non) par d'autres. C'est très compliqué à automatiser, car il faut comprendre dans quel cadre une personne (abonné, propriétaire ou modérateur) a été inscrite dans la liste. Pour le moment notre traitement est encore manuel.

6.5.2 Changement de nom d'une structure

Un deuxième point est le changement de nom d'une structure interne.

Le dernier exemple qui me vient à l'esprit est le changement de dénominations des ESPE en INSPE.

Il faut alors renommer chaque liste, en espérant que le nouveau nom n'existe pas déjà et peut-être positionner un alias de l'ancien nom vers le nouveau pendant une certaine durée. Nous utilisons en parallèle une gestion des alias dans notre annuaire.

Nous avons donc développé un script qui va utiliser l'action « rename_list » de SYMPA. Mais ce n'est pas si simple. En effet, il ne faut pas oublier de regarder l'ensemble des inclusions pour garder la cohérence, et donc, modifier directement la base de données (table inclusion_table dans les dernières versions de SYMPA), ou bien les fichiers de configuration de chaque liste dans les versions les plus anciennes.

Cela devient plus compliqué depuis qu'une partie des configurations a été reportée dans la base de données. Par exemple, la sous-commande de SYMPA citée ci-dessus ne fonctionne pas si la liste que l'on veut renommer est présente dans d'autres listes.

Enfin, il reste toujours des reliquats dans les fichiers de configurations, qui changent de nom d'attribut au cours des versions.

6.5.3 Cohérence de la base de données de SYMPA

La première version installée en 2012 a vécu des mises à jour nombreuses. À une certaine période, des accès directs à la base de données ont peut-être perturbé celle-ci.

Même si le fonctionnement quotidien semble correct, nous nous sommes rendu compte de certaines incohérences dans la base mais aussi entre celle-ci et le système de fichiers..

Nous sommes en train de regarder un script qui vérifiera que :

- Pour tout répertoire de l'arborescence des listes ([6]), il existe bien une entrée dans la table list_table (et inversement) ;
- Pour toutes les références à une liste dans une table quelconque, il existe bien une entrée dans la table list_table.

6.5.4 Et encore

Enfin un dernier exemple est le positionnement dans la « Global Access List » (GAL) de Zimbra des listes que l'on veut rendre facilement accessibles (complétion de l'adresse lors de la frappe).

Nous utilisons l'attribut « visibility » de la liste. Si cette liste est publique ou réservée à l'intranet, on l'introduit dans la GAL. Dans le cas contraire, on la supprime de la GAL si elle y figurait.

7 Conclusion

Comme j'ai essayé de vous le montrer, la seule installation d'un logiciel comme Sympa ne suffit pas dans notre cas.

Le développement d'un environnement supplémentaire, plus ou moins personnalisé, nécessite des ressources et du suivi. Le passage d'une version à une autre est donc long, car il demande une vérification et une adaptation éventuelle de l'ensemble de nos scripts. À l'heure actuelle, nous sommes en version 6.2.42 et nous avons donc deux versions de retard.

L'expérience d'autrui est très utile, encore faut-il en avoir connaissance. Cette présentation a pour but de permettre la compréhension et le partage de nos réalisations.

Certaines réalisations sont très paramétrées et donc facile à transposer, d'autres seront plus difficiles. Mais il ne faut pas hésiter à me demander. On cherchera un moyen plus moderne si la demande est forte.

6. La structure de données de SYMPA est à la fois dans une base de données et dans une arborescence de fichiers.

Annexe

Récupération des adresses d'une personne dans l'annuaire de l'UL

```
package ULSympa;

use Exporter;
use vars qw(@ISA @EXPORT @EXPORT_OK %EXPORT_TAGS $VERSION);
@ISA = ('Exporter');
@EXPORT = qw( &alternate);

use Net::LDAP;

use strict;

my $refLDAP =
{
  hosts    =>  [ "ldapw1.univ-lorraine.fr" ],
  basedn   => "ou=people, dc=univ-lorraine, dc=fr",
  filter   => "(|(mail=%s)(udlMailAlias=%s))",
  attrs    => [ "mail", "udlMailAlias" ],
};

######################################################################
# recoit une adresse mail en parametre
# retourne un tableau de toutes les adresses mail valides
######################################################################

sub alternate {
    my $mail2check = shift;
    return $mail2check unless ($mail2check =~ /\@/);
    my @TMailsValides = (lc($mail2check));
    my $refAttrs = $$refLDAP{attrs};
    my $ldap = Net::LDAP->new($$refLDAP{hosts}) or return
@TmailsValides;
    my $mesg = $ldap->bind('', version => 3 ) or return
@TmailsValides;
    my $filter = "$$refLDAP{filter}";
    $filter =~ s/\%s/$mail2check/sg;

    $mesg = $ldap->search (  # perform a search
                           base   => $$refLDAP{basedn},
                           filter => $filter,
                           attrs => $$refLDAP{attrs},
                           scope => "one"
                           );
    $ldap→unbind();
    return  @TMailsValides if ($mesg->count == 0);

    my $entry = $mesg→shift_entry;
```

```
    foreach my $attr (@$refAttrs){
       my $refVal = $entry->get_value($attr, asref =>1);
       next unless (defined($refVal));
       foreach my $val (@$refVal){
                my $mail = lc($val);
                next unless($mail =~ /\@/);
                push (@TMailsValides, $mail) unless (grep
/^$mail$/, @TmailsValides);
       }
    }
    return @TmailsValides;
}
## Packages must return true.
1;
```

Exemple de package pour la custom condition is_subscriber

J’ai laissé volontairement l’envoi des traces dans le journal de SYMPA. On affiche les paramètres reçus.

```
#!/usr/bin/perl
use lib '/usr/share/sympa/lib/';
package CustomCondition::is_subscriber;

use strict;
use Sympa::List;
use Sympa::Log;
use ULSympa;

my $log = Sympa::Log→instance;

sub verify {
    my $listName = shift;
    my $sender = shift;
    my $domain = shift;

    $log→syslog(
              'notice',
              'custom: is_subscriber: sender "%s", liste "%s",
domain = "%s"',
              $sender,
              $listName,
              $domain,
      );

    # Création de l'objet List
    my $list = ( $listName =~ /\@/ )? new
Sympa::List($listName) : new Sympa::List($listName, $domain);
    return -1 if( !defined($list));
```

```
    # Récupération des adresses alternatives de l'envoyeur
    my @alternatemail=ULSympa::alternate($sender);

    # Test de l'appartenance de l'adresse aux abonnés de la liste
    foreach my  $mail (@alternatemail){
        return 1 if $list→is_list_member($mail);
     }

    # L'envoyeur n'appartient pas à la liste
    return -1;
}

## Packages must return true.
1;
```