Conception d'un cube OLAP alimenté par un flux de données

Matthieu Ruiz

2008D018

2008

Département Informatique et Réseaux
Groupe IC2 : Interaction, Cognition et Complexité

46 rue Barrault 75013 Paris

Georges Hébrail & Annie Danzart

Conception d'un cube OLAP alimenté par un flux de données

Matthieu Ruiz

17 mars 2008 - 16 mai 2008

Résumé

Les cube de données (encore appelés OLAP) constituent aujourd'hui une part importante de la composante décisionnelle du système d'information des entreprises. Ils permettent à l'utilisateur décideur d'explorer interactivement des informations synthétiques sur le fonctionnement de l'entreprise.

Or, de plus en plus d'informations sont disponibles sous la forme de flux continus ("data stream") et il devient nécessaire pour les entreprises de les analyser en temps réel afin d'accroître leur capacité concurrentielle. Ainsi, il existe des systèmes de gestion de flux de données ("Data Stream Management Systems") permettant d'opérer des requête ad-hoc sur un ou plusieurs de ces flux, flux qui ne sont plus alors nécessairement stockés en base de données.

L'objectif de ce projet est donc de tester l'un de ces systèmes, tel que celui, commercial, proposé par la société Aleri : Aleri Streaming Platform. Pour ce faire, un cube de données (implémenté grâce au composant Aleri Live OLAP) sera alimenté en temps réel par le flux des logs d'accès au site web www.infres.enst.fr.

Le présent document propose, dans un premier temps, une étude des données de ce flux et une proposition de schéma pour le cube. Une étude du produit de la société d'Aleri sera ensuite développée. Enfin, ces deux études se recouperont afin de donner lieu, dans une troisième partie, à l'élaboration d'un prototype et son implémentation.

Table des matières

Première partie

Étude des données

Chapitre 1

Contexte et source des données

Le site web www.infres.enst.fr est hébergé par un serveur HTTP Apache sur l'une des machines du réseau interne de l'ENST, machine que l'on nommera infres5.enst.fr dans la suite de ce document. Chaque requête HTTP, à destination de ce serveur, est mémorisée sous la forme d'un "access log" dans un fichier dédié de cette machine. Une solution pour accéder aux logs du site web serait donc de lire périodiquement ce fichier et de relever les différences constatées par rapport à la précédente lecture. Cette proposition pose cependant deux problèmes dans le cadre de cette étude. Tout d'abord, il conviendrait alors d'accéder physiquement à ce fichier, via le réseau, ce qui n'est pas idéal en terme de sécurité. De plus, et surtout, une telle méthode serait contraire avec l'objectif de l'analyse d'un flux de données temps réel. La solution retenue consiste donc en la création d'un script bash dont l'exécution sur la machine infres5.enst.fr permet la création d'une "socket", en mode connecté TCP afin de garantir l'ordonnancement et la fiabilité du transfert, non "multithreadée", écoutant sur le port 8999. Dès qu'un client se connecte sur cette "socket", le script lui redirige les "access logs" en temps réel générant ainsi un flux de données à proprement parler. La figure, ci dessous, résume cette génération :

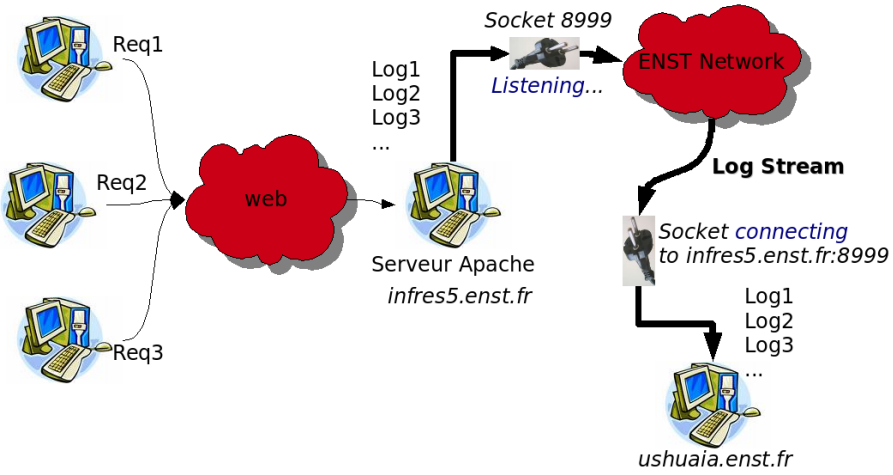

FIG. 1.1 – Diagramme simplifié de la génération du flux de données

Chapitre 2

Analyse des données

2.1 Combined Log Format

Les éléments du flux de données sont des "access logs" crées par le serveur Apache. Le format de ces logs est déterminé dans un fichier de configuration du serveur grâce à la ligne suivante :

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""
```

Il sera considéré, dans la suite de cette étude, que cette configuration est immuable. En effet, il n'est pas envisagé de modifier ce fichier de configuration, changement dont la prise en compte impliquerait notamment le redémarrage du serveur. Dès lors, examinons sans plus attendre à quoi ressemble un log sous ce format :

```
berlioz.enst.fr - - [23/Apr/2008:19:43:12 +0200]
"GET /~hebrail/publications.html HTTP/1.0"
200 24281 "http://www.infres.enst.fr/~hebrail/"
"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.13)
Gecko/20060601 Firefox/2.0.0.13 (Ubuntu-edgy)"
```

Détaillons chaque donnée de ce log :

berlioz.enst.fr (*%h*) :
: Il s'agit de l'adresse IP du client qui a fait la demande de requête au serveur. Ici, cette adresse IP se présente sous la forme d'un nom de domaine. En effet, le serveur Apache essaye de résoudre lui même l'IP en nom de domaine. S'il y arrive, il place ce nom de domaine en lieu et place de l'IP brute. Il convient de préciser ici que l'IP (résolue ou non) ne correspond pas nécessairement à la machine derrière laquelle est assis l'utilisateur (réel ou virtuel dans le cas de robot). En effet, il suffit que l'utilisateur soit "derrière" un proxy pour que l'IP présente dans le log ne coïncide pas avec celle de la machine de celui-ci. C'est d'ailleurs typiquement le cas ici, la machine berlioz.enst.fr étant un proxy.

- (*%l*) :
: Cette information indique le nom de l'utilisateur de la machine distante comme supposé dans le protocole RFC 1413. Ici, cette information n'est d'aucune utilité car le serveur Apache, dans sa configuration, n'essaye même pas de déterminer cette information. La valeur de ce champ sera au final systématiquement la même : "-".

- (*%u*) :
: Il s'agit de l'identité de la personne faisant la requête comme déterminé dans le protocole d'authentification de HTTP. Si la requête ne requiert pas d'authentification, la valeur de ce champ est, comme ici, "-" .

[23/Apr/2008 :19 :43 :12 +0200 (%t)] :
Date exacte GMT à laquelle le serveur a terminé de traiter la requête. Il aurait été possible de formater cette date d'une autre manière mais on considérera que ce format est immuable également.

"GET /~hebrail/publications.html HTTP/1.0" (\"%r\") :
Voici la requête exacte du client. On peut en extraire trois types d'information bien distincts. Tout d'abord la méthode utilisée par le client : Il s'agit de la méthode "GET". Ensuite, on trouve la ressource exacte demandée : /~hebrail/publications.html. Enfin, il est spécifié le protocole utilisé par le client : "HTTP/1.0".

200 (% > s) :
Il s'agit ici du code de statut que le serveur renvoie au client. Cette information est précieuse car elle révèle si la requête a résulté en un succès (code commençant par 2), une redirection (code commençant par 3), une erreur causée par le client (code commençant par 4) ou bien encore une erreur imputée au serveur (code commençant par 5). Le lecteur qui souhaite avoir l'ensemble des codes possibles, pourra trouver cette information dans la spécification HTTP (RFC2616 section 10).

24281 (%b) :
Cette entrée indique la taille de l'objet (en octet) retourné par le serveur, sans inclure les "headers". Tel qu'est configuré le serveur Apache ici, en cas de réponse sans contenu, ce champ vaudra "-" et non 0.

"http ://www.infres.enst.fr/~hebrail/" (\"%{Referer}i\") :
Voici la première des deux entrées qui caractérise le "Combined Log Format" par rapport au "Common Log Format". Il s'agit exactement du champ "Referer" contenu dans le "header" de la requête HTTP. Celui-ci symbolise le site dont le client prétend se référer. Dans notre cas, on peut interpréter qu'il y a un lien sur la page http ://www.infres.enst.fr/~hebrail/ redirigeant vers la page indiquée dans la requête : /~hebrail/publications.html.
Précisons ici que le champ "Referer" est un champ *optionnel* du "header" de la requête HTTP. Certains navigateurs offrent notamment, à leurs utilisateurs, la possibilité de désactiver le remplissage de ce champ. Ainsi, s'il n'est pas renseigné, il sera généralement noté "−" dans le log, ou bien encore, de manière très épisodique et hors standard, "".

"Mozilla/5.0 (...) Firefox/2.0.0.13 (Ubuntu-edgy)" (\"%{User − agent}i\")) :
Ce champ est le deuxième ajouté par le "Combined Log Format" par rapport au "Common Log Format" : le "User Agent". Lorsqu'un utilisateur visite une page web, cette chaîne de caractères est généralement envoyée pour identifier l'agent de l'utilisateur. Concrètement, ces agents sont les applications clientes qui envoient les requêtes aux serveurs web, selon, ici, le protocole HTTP. La gamme des agents est très vaste puisqu'elle englobe aussi bien les navigateurs webs classiques que les robots d'indexation en passant par les navigateurs brailles pour les personnes possédant un handicap. Malheureusement, l'histoire du web ayant été marquée par la domination de certains types de navigateurs, cette chaîne de caractères a été analysée par les serveurs pour retourner un contenu web parfois plus pauvres à des navigateurs moins populaires bien que ceux ci pussent les traiter correctement. C'est à cause de cette gestion de contenu discriminatoire que les navigateurs ont commencé à "falsifier" cette chaîne afin de prendre l'identité de navigateurs plus renommés. L'exemple historique est Internet Explorer qui utilise une chaîne commençant par « Mozilla/version (compatible ; MSIE version... », afin de recevoir le contenu destiné à Netscape Navigator, son rival principal au début de son développement. On doit noter qu'il ne s'agit pas d'une référence au navigateur Mozilla Open Source, qui a été développé beaucoup plus tard, mais au nom de code originel du navigateur, qui était également le nom de la mascotte de la société Netscape. Ce format de chaîne a été depuis copié par d'autres, en partie parce qu'Internet Explorer est devenu, à son tour, le navigateur dominant. Ainsi, cette pratique ne semble pas prêt de disparaître, malgré des efforts de standardisation.

Sur notre exemple, bien que la chaîne commence par «Mozilla/5.0 ... », le vrai navigateur se trouve plus loin et il s'agit de Firefox dans sa version 2.0.0.13. Par ailleurs, on peut remarquer que la chaîne recèle de renseignements tels que l'architecture de la machine client, son OS, ou bien encore sa distribution. Dans cet exemple, il s'agit d'une distribution Linux, Ubuntu-Edgy, installée sur un Intel Pentium Pro CPU.
Ces informations complémentaires, couplées à l'émergence des terminaux mobiles connectés au Web, tendent à faire exploser les combinaisons possibles de "UserAgentStrings".

Nous avons identifié, dans cette section, quels étaient les champs d'information au sein de l'unité de base du flux de données : Le "Combined Log Format". Avant d'envisager une structure de cube de données pour synthétiser un tel flux, il serait intéressant de posséder des éléments de volumétrie concernant celui-ci : Quel en est le "débit" moyen et maximum ? Combien de valeurs différentes relevons nous pour chacun des champs de ce flux ? Pour ce faire, nous proposons d'analyser, dans la prochaine section, un échantillon de ce flux.

2.2 Etude de la volumétrie du flux

L'échantillon de données considéré pour cette analyse est celui obtenu par le flux des logs du premier mars 2008 (00 :20 :03) au dix-sept mars 2008 (16 :17 :36) soit deux grosses semaines. Cette échantillon est considérable puisqu'il consiste en 1 451 213 logs. Un script a permis le "parsing" et l'importation de 1 450 574 de ces logs au sein d'une base de données MySQL. Nous reviendrons notamment sur les expressions régulières qui ont permis le "parsing" dans la partie III. L'exécution et le recoupement d'un ensemble de requêtes SQL sur cet échantillon a permis non seulement d'analyser le débit du flux comme explicité en section 2.2.1, mais aussi la volumétrie des différents champs qui le composent comme décrit en section 2.2.2. Le lecteur trouvera l'ensemble des résultats numériques de cette étude en annexe A.

2.2.1 Prévision du débit

En première approche le débit moyen du flux peut être calculé à partir des données précédentes : 1 451 213 logs en environ 17 jours soit 400 heures (compte tenu des heures tronquées du premier et dernier jour) ou bien encore 1 380 000 secondes. **On obtient alors un débit moyen de l'ordre de 1 log/s**. Cependant, ce calcul est très grossier puisque il fait une moyenne qui considère, par essence, que la fréquentation du site www.infres.enst.fr est la même 24 heures sur 24 et 7 jours sur 7. Or, on peut se douter qu'une telle fréquentation est loin d'être aussi régulière. La figure 2.1, obtenue à l'aide d'un ensemble de requêtes SQL sur l'échantillon nous conforte dans cette idée.

On peut tout d'abord constater que la fréquentation du site est plus faible les week-ends (1,2,8,15,16 mars) que les jours de semaine. De plus, il convient de noter que le nombre de requêtes peut varier assez fréquemment du simple au double. Attachons nous désormais à trouver le pire cas, c'est à dire où le débit serait maximal. Sans être exhaustif, focalisons nous simplement sur la journée du lundi 10 mars, c'est à dire là où culmine le nombre de requêtes (124 694 pour être exact). La figure 2.2 met en évidence le nombre de requêtes par tranche horaire pour ce jour là.

Sans grande surprise pour ce site français, on remarque un fort contraste entre la fréquentation nocturne et celle diurne, notamment pendant les heures de bureau (La courbe présente une forme quasi gaussienne de 9h00 à 19h00). Le pic de fréquentation est obtenu dans la tranche horaire "12h-13h" avec 9330 requêtes soit **approximativement un débit maximum de 2,5 log/s**.

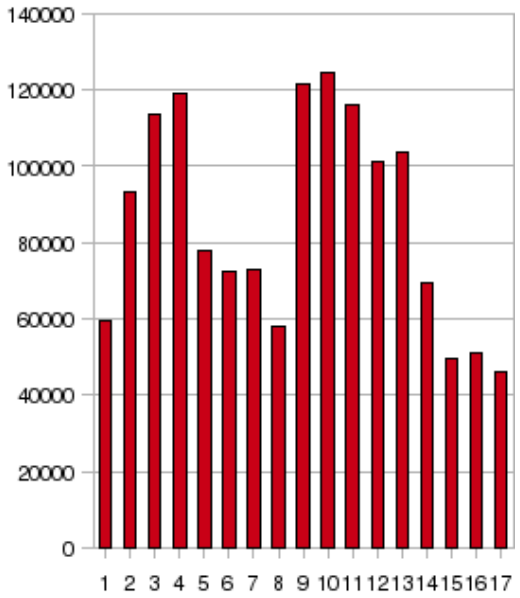

FIG. 2.1 – Nombre de requêtes traitées par le serveur et par jour, du 01-03-08 au 17-03-08

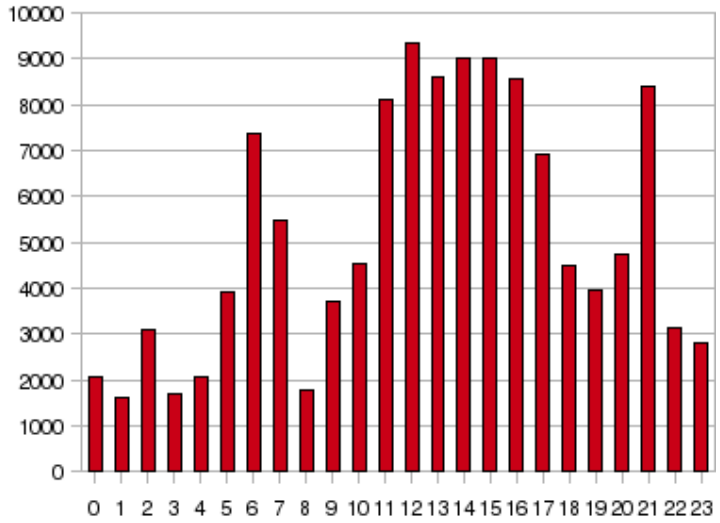

FIG. 2.2 – Nombre de requêtes traitées par le serveur et par heure le 10-03-08

Bien que cette analyse de débit maximum se contente d'une moyenne sur la tranche horaire la plus fréquentée du jour échantillonné où culmine le nombre de requêtes, on peut considérer, avec une marge de sécurité, que **le débit du flux de données ne devrait pas dépasser 3 à 4 log/s**. C'est cette fourchette de débit qui sera considérée comme borne supérieure dans la suite de cette étude. Elle sera notamment mise en regard avec la capacité d'absorption du DSMS.

Maintenant que le débit du flux a été précisé, il est nécessaire de porter son attention au contenu des données elles mêmes : En quelle mesure sont elles hétérogènes ? En fonction de quel champ de données considéré ?

2.2.2 Mesure de l'hétérogénéité des données

Dans cette section, seuls les champs de données du flux, comme décrits dans la section 2.1 sur le "Combined Log Format" et parmi lesquelles l'hétérogénéité est a priori inconnue, feront l'objet d'une analyse.

Nous pouvons, dès lors, isoler les champs concernant la méthode, le protocole et le code de statut associé à chaque requête car ils sont fixés par les spécifications du protocole HTTP. Ainsi, on peut considérer comme déterministes les contenus de ces champs en ce sens où ils ne peuvent prendre guère plus qu'une dizaine de valeurs différentes.

Par ailleurs, une telle étude sur la quantité d'octets transférés par requête, n'est d'aucune utilité car on pressent qu'un tel champ sera une mesure agrégée au sein du cube et non pas une dimension. Nous reviendrons sur ce point dans le chapitre 3.

L'IP, ou le "Who ?"

Considérons tout d'abord le champ contenant l'information sur l'IP du client (ou de son proxy). Comme expliqué dans la section 2.1, le serveur Apache essaye de résoudre chaque IP en nom d'hôte et, en cas de succès, inscrit ce dernier dans le log en lieu et place de l'IP. La figure 2.3 , ci dessous, met en évidence le ratio entre les IP résolues par le serveur et celles qui ne le sont pas :

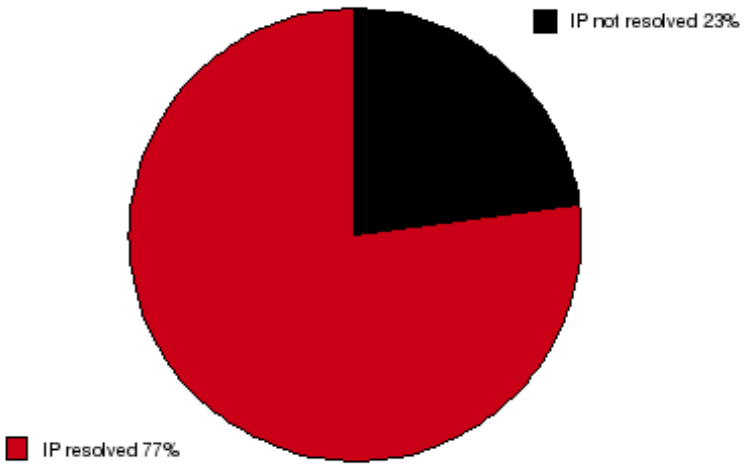

FIG. 2.3 – Ratio entre les IP résolues ou non par le serveur

On constate qu'environ un quart des IP correspond à des IP non résolues, ce qui est loin d'être négligeable. Autrement dit, un pré-traitement devra être mis en place pour déterminer si l'IP a été résolue ou non, puis, en fonction, un traitement spécifique sera appliqué au champ. Ces traitements seront mis en exergue lors de la conception des dimensions du cube au chapitre 3. Par ailleurs, l'étude de l'hétérogénéité des IP non résolues ne semble pas pertinente puisque conserver une telle IP "brute", en soi, n'apporte guère d'information lisible par un humain. Dès lors, focalisons notre étude sur l'hétérogénéité des IP résolues. La figure 2.4, page suivante, met en évidence l'évolution du nombre d'IP résolues distinctes sur les jours

échantillonnés.

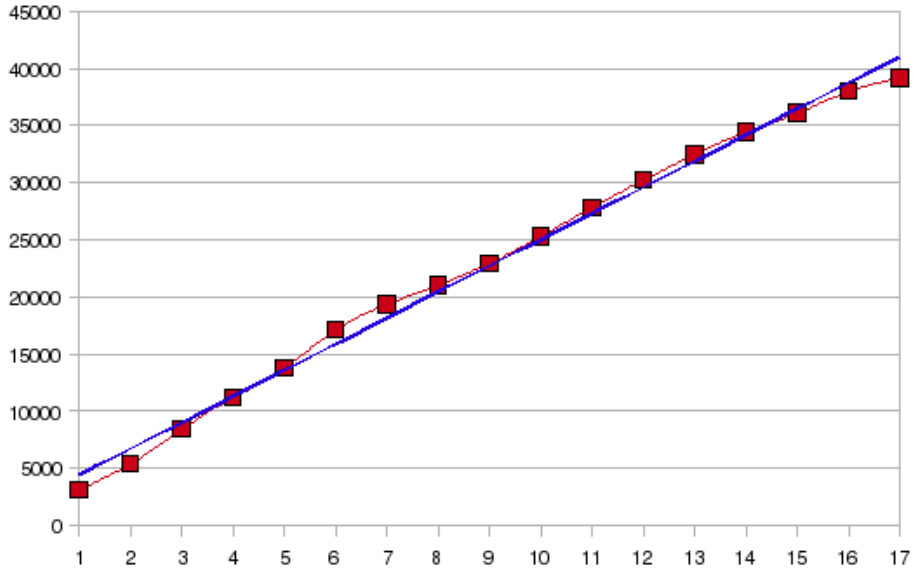

FIG. 2.4 – Evolution du nombre d'IP résolues distinctes du 01-03-08 au 17-03-08

On remarque que cette évolution est linéaire au fil des jours analysés à tel point qu'une régression linéaire coïncide avec un coefficient de 0,9941 (droite en bleue sur le graphique). A partir de ce seul échantillon on ne peut donc anticiper un ralentissement du nombre de valeurs distinctes mais seulement une évolution selon l'équation suivante, avec x en jour :

$$2290 * x + 2101 \tag{2.1}$$

Cette équation est une modélisation peu optimiste puisqu'elle propose une évolution constante alors que si nous avions disposé d'un échantillon plus macroscopique nous nous apercevrions probablement d'un ralentissement de cette évolution. Aussi, c'est avec ce pire cas que seront "dimensionnées" les dimensions du cube au chapitre 3. De plus, on peut calculer que la **longueur moyenne des chaînes de caractères parmi les IP résolues distinctes est de 33**.

Les "hits", ou le "What ?"

Portons désormais notre attention sur les ressources demandées au sein des requêtes. Ces ressources sont communément appelées "hits" dans la sphère de la web analyse. Ainsi, avec cette définition, toute requête au serveur peut être considérée comme tel. Par exemple, une image téléchargée pour être affichée au sein d'une page est un "hit". De même, une simple page web contenant quatre images résultera en cinq "hits" pour le serveur. On comprend alors pourquoi l'utilisation de cette mesure pour vanter la popularité d'un site web est totalement erronée en ce sens où elle gonfle artificiellement le nombre de pages réellement visualisées ou "page view". C'est cette dernière notion qui est plus intéressante en terme d'analyse puisqu'elle essaye de mesurer les documents visualisés par un internaute. Ces documents peuvent être de plusieurs types :

- Pages web telles que : .html, .htm, .php, .asp, etc...
- Documents à proprement parler comme : .pdf, .xls, .doc, etc...

– Fichiers plein texte : .txt, .java (et autres fichiers sources de programmation), etc...

Les fichiers qui sont généralement considérés comme faisant partie d'une page web, sans en être le contenu principal, ne sont pas des "pages view". C'est typiquement le cas des images et autres fichiers de style (.css) ou scripts (.js, etc...).

Une première requête SQL parmi tous les "hits" de l'échantillon révèle que 224 076 d'entre eux sont distincts. Or, parmi ces derniers, un certain nombre correspond à des requêtes POST associées à des arguments (situés après le " ?" de la requête). En quelle mesure ces arguments rajoutent-ils alors de l'hétérogénéité parmi ces "hits" ?
Pour ce faire, il suffit d'associer à un même "hit" tous les "hits" qui lui sont dérivés via l'ajout d'arguments. Par exemple, deux "hits" distincts tels que /GEOserviceWeb/localisationDemo.xml ?bidon=6819634754 et /GEOserviceWeb/localisationDemo.xml ?bidon=6373345605 (tous deux extraits de l'échantillon) seront considérés comme le même "hit" /GEOserviceWeb/localisationDemo.xml atteint deux fois. Une deuxième requête SQL ,"tronquant" de cette façon les "hits" qui possèdent des arguments, diminue de plus de 50% leur hétérogénéité : En effet au bout des dix-sept jours de l'échantillon ce ne sont plus que 91 321 "hits" distincts qui sont relevés. L'évolution de leur cardinalité est présentée sur la figure 2.5 ci dessous :

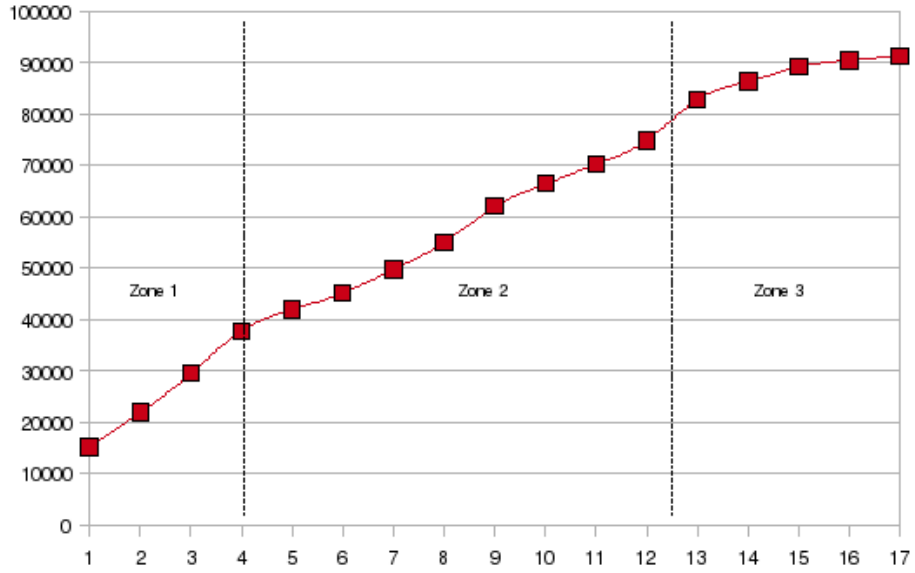

FIG. 2.5 – Evolution du nombre de "hits" distincts (arguments non compris) du 01-03-08 au 17-03-08

Trois zones se dissocient graphiquement : Les deux premières qui pourraient être qualifiées de zones d'apprentissage (moins marqué dans la seconde), et une troisième zone de "stabilisation progressive". C'est cette dernière zone qui est pertinente à long terme et qui peut être modélisée par une régression logarithmique de coefficient 0,9933 et d'équation :

$$3576 * \log(x) + 8625 \tag{2.2}$$

Une fois encore, cette modélisation est pessimiste car, à long terme, elle risque de dépasser le nombre de "hits" plafonné par la structure intrinsèque du site lui même, structure que l'on considérera comme relativement stable. Or, nous ne disposons pas de l'arborescence du site pour cette étude. On se contentera donc dans la suite de cette modélisation. Ajoutons que la **longueur moyenne des chaînes de caractères parmi les "hits" distincts (arguments non compris) est de 42**.

La répartition des 91 321 "hits" distincts, selon leur type comme décrit précédemment, est présentée sur la figure 2.6, page suivante.

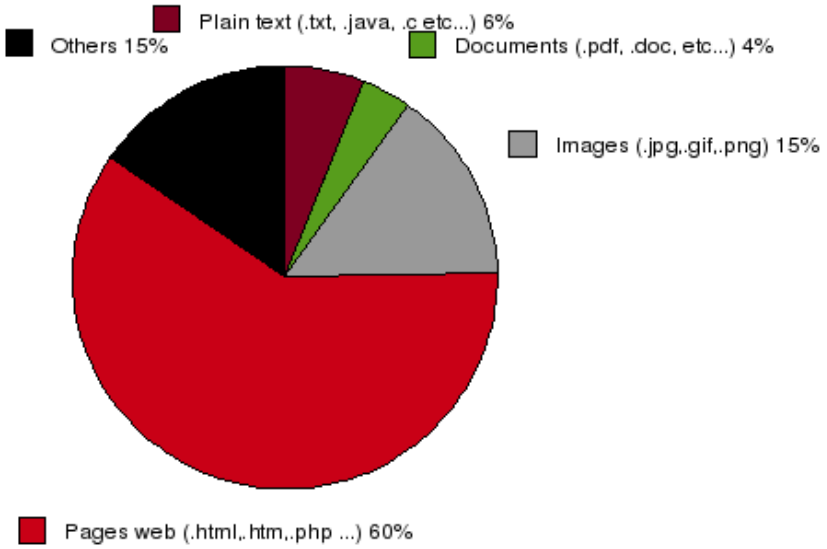

FIG. 2.6 – Répartition des "hits" distincts (arguments non compris)

Les "User Agents", ou le "How ?"

Les deux champs restants à analyser sont le "User Agent" et le "Referer". La figure 2.7 met en évidence l'évolution du nombre des chaînes "User Agents" distinctes, relevées dans l'échantillon.

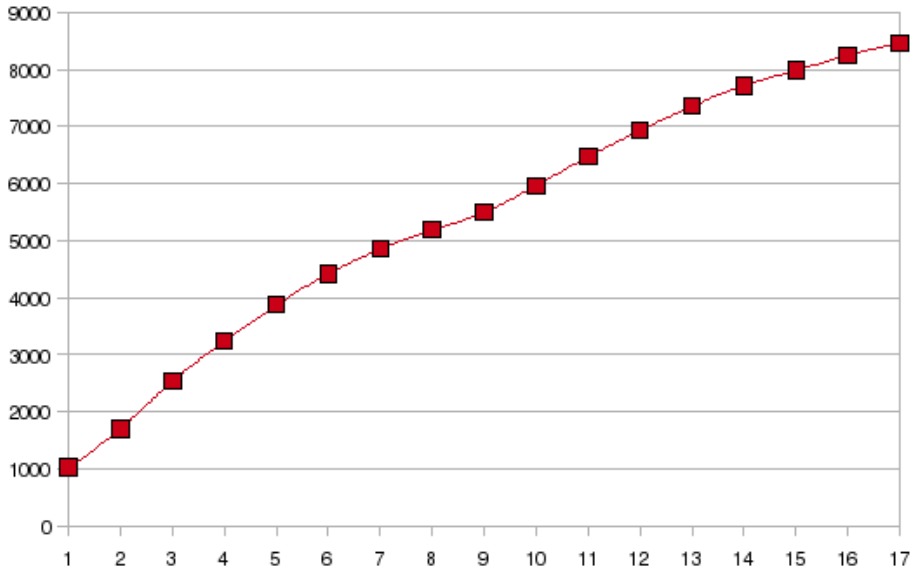

FIG. 2.7 – Evolution de "User Agents" distincts du 01-03-08 au 17-03-08

Aussi surprenant que cela puisse paraître, cette évolution ne présente pas de signe de stabilisation. Sans doute est ce ici la preuve de la multiplication des configurations possibles pour

l'application cliente comme explicité dans la section 2.1. De plus, de nouveaux "User Agents" paraissent régulièrement que ce soit à cause de l'évolution des navigateurs, des robots d'indexation, ou bien encore des platformes et des distributions. Le site www.useragentstring.com ([5]) tente de référencer toutes les "UserAgentsStrings" existants. Pourtant, une étude attentive montre que sa base de données est encore loin d'être complète. Ainsi, il semble réellement peu intéressant de stocker ces chaînes de caractères qui regorgent d'informations aussi utiles qu'inutiles, et dont l'hétérogénéité est vouée à croître.

Les "Referers", ou le "From ?"

Les "referers", contrairement aux "hits", ne sont pas nécessairement des URLs internes au site www.infres.enst.fr. En effet, et c'est justement là l'un des enjeux de la web analyse que de les identifier, les "referers" peuvent aussi être des **URLs externes**. La figure 2.8, ci-dessous, explicite la répartition de ces "referers" parmi tous les "hits" de l'échantillon :

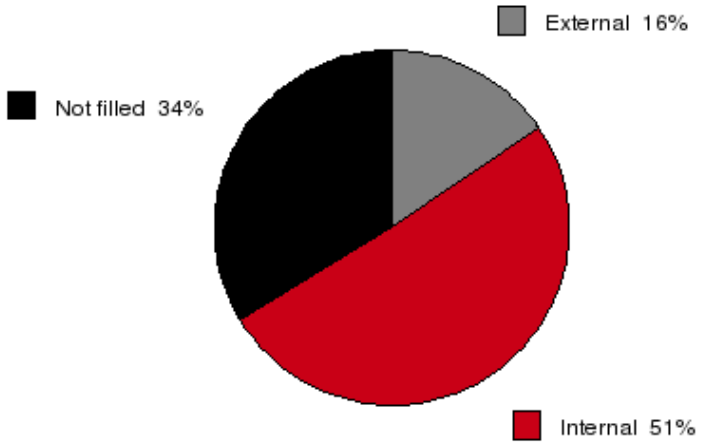

FIG. 2.8 – Répartition des "referers" parmi tous les "hits" du 01-03-08 au 17-03-08

Rappelons que le champ "referer" est optionnel dans le "header" de la requête HTTP (cf section 2.1). Effectivement, 34% des "hits" n'ont pas de "referers" renseignés : cela arrive notamment dans le cas de la visite du site par des robots d'indexation ou "spiders". Par ailleurs, seul 16% des "hits" proviennent d'une URL externe au site www.infres.enst.fr. Cependant, cette répartition pourrait être trompeuse dans la mesure ou elle souligne l'impact du "referer" sur les "hits" et non sur les pages réellement visualisées ou "pages views", comme expliqué précédemment dans le paragraphe 2.2.2. La figure 2.9 rectifie donc la répartition en ne considérant que les "pages views".

Il est alors très intéressant de constater que le poids des "referers" internes s'effondre au profit notamment de ceux externes, dont le poids, quant à lui, double en culminant à près de 30%. Ce renversement de tendance s'explique principalement par deux phénomènes dont la corrélation est négative : D'un côté, nous avons vu auparavant qu'un "page view" entraîne de multiples "hits" (2.2.2) pour lesquels le "referer" est alors la page web en question, soit un "referer" interne. De l'autre, la majorité des liens externes pointent sur une page web, à proprement parler, donnant lieu à un "page view" et non uniquement un simple "hit".

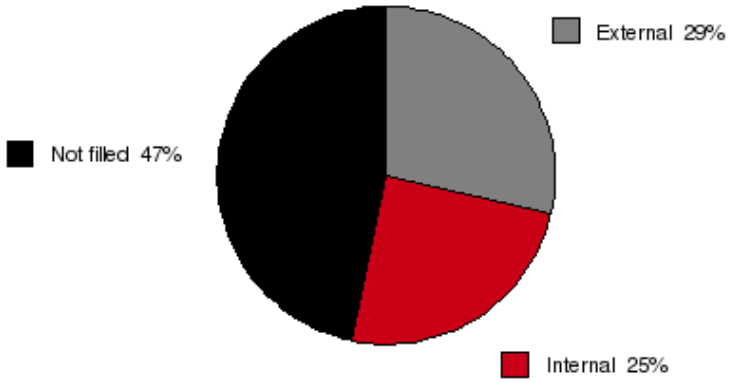

FIG. 2.9 – Répartition des "referers" parmi tous les "pages views" du 01-03-08 au 17-03-08

Poussons dès lors l'analyse en examinant de plus près les "referers" externes. Ces derniers peuvent revêtir plusieurs formes :

- **Les requêtes directes dans les moteurs de recherches** : Explicitons ce cas par un extrait d'un des logs de l'échantillon : Le "page view" /~hebrail/publications/sfc04.pdf est associé au "referer" http ://www.google.com/search ?q=serie+temporelle&hl=fr&lr =&start=30&sa=N. Ici, le "referer" regorge d'informations interessantes. Effectivement, son analyse livre que ce "page view" est le fruit d'une interrogation dans le moteur de recherche Google avec les mots-clés suivants : "serie" et "temporelle". Dès lors, pas question ici de "tronquer" les arguments situés après le " ?" comme celà avait été suggéré pour l'analyse des "hits" puisqu'ils sont, en quelque sorte, la "substantifique moelle" de la web analyse.

- **Les pages cachées ou requêtes indirectes** : Les moteurs de recherches mettent en cache les pages web et celles du site www.infres.enst.fr ne font pas exception. Le "referer" peut indiquer que l'utilisateur a provoqué un "hit" à partir de la visualisation d'une page mise en cache par un moteur de recherche. Or, le "referer" indiquera également à partir de quel(s) mot(s)-clé(s) l'utilisateur aura été amené à consulter cette page mise en cache ! Ce type de "referer" est sans doute le plus riche en terme d'information et de web analyse puisqu'il permet de détecter un certain nombre des "pages view" qui n'auraient jamais été détectées. En effet, quand un utilisateur visualise une page mise en cache, il fait un "page view" sur un serveur du moteur de recherche, puis des "hits" sur le serveur web proposant la page considérée mais uniquement afin de récupérer les documents secondaires (images, etc...). Du point de vue de ce dernier n'apparaît alors aucun "page view". Pourtant, l'utilisateur visualise bien, en théorie, le contenu de la page ! Dès lors, on comprend, en terme de web analyse, la problématique des pages cachées. Mais prenons, sans plus attendre, un exemple, extrait d'un des logs de l'échantillon : Le "hit" considéré est /~danzart/mysql/illustration.html, associé au "referer" http ://209.85.173.104/search ?q=cache :kqfJXQnAMsQJ :www.infres.enst.fr/~ danzart/mysql/php.phtml+mysql+php+WHERE&hl=fr&ct =clnk&cd=1&client=firefox-a. Malgré l'apparence "repoussante" de ce "referer", essayons, tout de même, d'en

extraire l'information fondamentale. Avant toute chose, il convient de préciser que l'adresse "209.85.173.104" est une des IP des serveurs de cache de Google. Le terme "search ?q=cache" (codage pour le moteur Google) corrobore le fait qu'il s'agit d'une page mise en cache, dont l'URL initiale est située un peu plus loin dans le "referer" : www.infres.enst.fr/~danzart/mysql/php.phtml. De plus, les mots-clés associés à cette recherche sont "mysql", "php" et "WHERE". Ignorons dans cette analyse les autres informations, moins pertinentes. Pour résumer, ce "referer" nous indique, à lui seul, que l'internaute a interrogé le moteur de recherche Google avec les mots-clés cités précédemment ; Puis, parmi les réponses qui lui ont été proposées, il a choisi de visualiser la page /~danzart/mysql/php.phtml, mise en cache par Google, page à partir de laquelle il alors fait un "page view" vers /~danzart/mysql/illustration.html. Le lecteur averti aura déjà remarqué que le lien menant vers cette dernière page était présent en bas de celle mise en cache.

– **Les boites mails** : Ce scénario, bien qu'assez exotique dans le cas du site étudié ici, mérite d'être cité. Le "hit" provient d'un clic sur un lien, dans le corps d'un mail, au sein d'une boîte électronique : webmail, squirrelmail pour les boites internes à l'ENST, ou les classiques Gmail, Yahoo, etc... pour les autres. Le "referer" est alors inexploitable puisque l'URL qu'il contient est sécurisée, car associée à une session du proporiétaire de la boîte. Pour le site www.infres.enst.fr en question, cette situation pourrait être fréquente lorsque l'administration et/ou un professeur envoie par mail un lien pour les élèves. Néanmoins, on conçoit plus facilement l'interêt d'isoler ce type de "referer", dans le cadre d'un site commerciale, suite à une campagne publicitaire par "mailing" !

– **Les autres sites externes** : Il s'agit de toutes les URLs externes qui ne rentrent pas dans les catégories précédentes et qui sont, à priori, exploitables directement c'est à dire sans traitement spécifique.

Dans la suite de cette étude, la catégorie "pages cachées" sera fusionnée avec celle des "recherches dans les moteurs de recherches". En effet, ces deux catégories résultent d'interrogations de moteurs de recherches, à l'aide de mots-clés. Par ailleurs, seules les pages cachées par le moteur Google seront détectées.
Maintenant qu'une classification des "referers" a été précisée, il serait intéressant de quantifier et prévoir l'évolution de leur hétérogénéité. Les figures 2.10 et 2.11 nous donnent quelques pistes de réflexions selon deux éclairages complémentaires.

On constate, tout d'abord, que ce sont les "referers" internes, et ceux externes de type "moteurs de recherches" qui contribuent le plus intensément à faire croître l'hétérogénéité. Cependant, bien que leurs contributions respectives soient relativement équilibrées, notamment lors des premiers jours échantillonnés (de l'ordre de 40% pour les moteurs de recherches contre 50% pour ceux internes), les "referers" liés aux moteurs de recherches tendent, inexorablement, à apporter d'avantage de diversité (plus de 60% pour les moteurs de recherches contre moins de 30% pour ceux internes au bout des deux semaines échantillonnées). Nous pouvons même affirmer, avec confiance, que cette tendance ne s'inversera pas. En effet, la diversité des "referers" internes est, en fait, intimement liée à l'hétérogénéité des "hits"'. Or, il a été constaté dans le paragraphe 2.2.2 que cette dernière évoluait de manière logarithmique. En effet, en faisant l'hypothèse, légitime, d'une évolution négligeable de la structure du site www.infres.enst.fr, cette hétérogénéité est même bornée sur une fenêtre temporelle grande. A l'inverse, chaque "referer" de type "moteurs de recherches" contient une information assez imprévisible : les mots-clés rentrés par l'internaute lui même. L'évolution de l'hétérogénéité des "referers" de ce type peut ainsi être modélisée, avec un coefficient de 0,9952, par l'équation suivante (x en jours) :

$$2378 * x + 15 \tag{2.3}$$

Bien sûr, cette modélisation est, une fois encore, extrêmement pessimiste puisque les

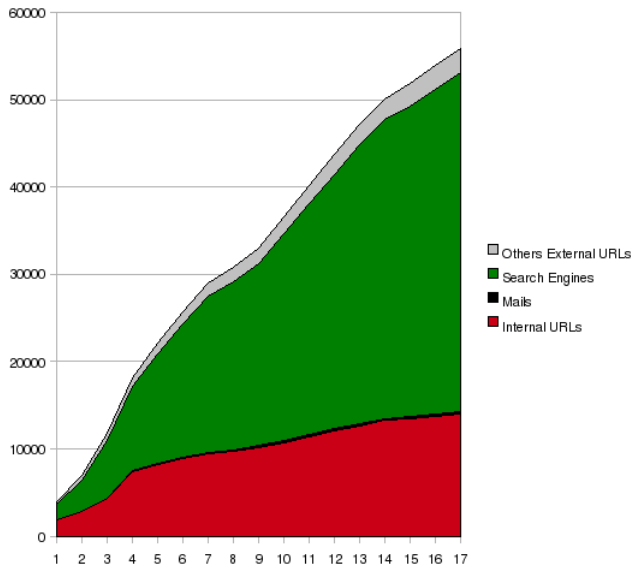

FIG. 2.10 – Evolution des "referers" distincts, selon leur type, du 01-03-08 au 17-03-08

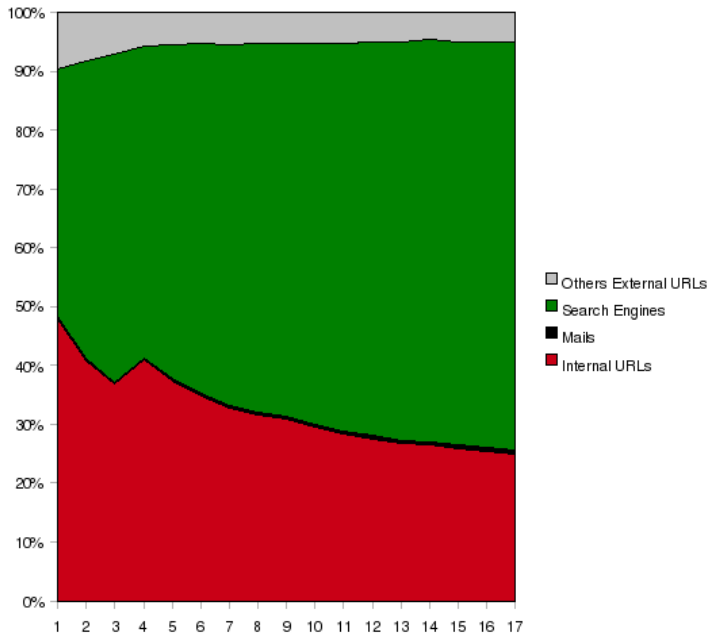

FIG. 2.11 – Evolution des "referers" distincts, selon leur type, en pourcentage

"referers" associés aux moteurs de recherches contiennent, pour la plupart, d'autres informations que nous avons ignorées (la langue, l'encodage etc...). Ainsi, deux recherches avec des mots-clefs identiques au sein d'un même moteur de recherche, mais ce dernier dans deux langues différentes, produira deux "referers" distincts. Pire, dans des conditions strictement identiques et avec les mêmes mots-clés, mais ces derniers rentrés dans un ordre différent, deux "referers" distincts seront pourtant considérés. En outre, cela ne serait peut être pas si inintéressant si l'on considérait cette ordre non pas comme aléatoire pour l'internaute

mais comme une hiérarchisation de ses idées, au regard de sa recherche. Néanmoins, dans le cadre de cette étude préliminaire, nous nous contenterons de cette modélisation pessimiste. Soulignons l'écrasante domination de Google, pour lequel 94% des "referers" distincts, de type "moteurs de recherches", lui sont associés, suivi de Yahoo avec 2,6% puis de MSN avec 1,4%.

Notons également la proportion négligeable de "referers" de type "mail", "referers" qu'il serait tout de même pertinent de repérer et d'isoler lors de la construction du cube afin de "purifier" la dernière catégorie de "referers" : les autres URLs externes. La proportion de ce dernier type (dont la longueur moyenne est de 40) est relativement constante, oscillant entre 5% et 7%. L'évolution de l'hétérogénéité de cette catégorie peut être modélisée, avec un coefficient de 0,9861, par l'équation suivante (x toujours en jours) :

$$147 * x + 411 \tag{2.4}$$

Synthèse

L'analyse de cet échantillon a permis de dégager différentes caractéristiques de notre flux de données :

- Le débit moyen en est de 1 log/s mais ce dernier est en fait extrêmement variable, aussi bien entre deux jours, qu'entre deux plages horaires. Le débit horaire, en pointe, est estimé entre 3 à 4 log/s.
- Quatre types d'informations qualitatives semblent pertinents pour décrire un "access log" c'est à dire **un fait** : le "Who", le "What", le "From" et enfin le "How".
- Une des caractéristiques des faits est mesurable : il s'agit de la quantité d'octets transférés suite au traitement d'une requête.
- Le "Who" peut revêtir deux formes : une adresse IP ou un nom d'host. Leurs proportions (respectivement 23% et 77%) impliquent un traitement spécifique. L'évolution de l'hétérogénéité des noms d'host peut se modéliser, linéairement, à l'aide de l'équation 2.1. Leur longueur moyenne est de 33 caractères.
- Le "What" est représenté par un "hit" c'est à dire une ressource demandée. Un sous ensemble des "hits" est les "pages views" c'est à dire une ressource correspondant à un contenu principal (une page web, un document pdf etc...). Par opposition, le complémentaire des "pages views" dans l'espace de tous les "hits", correspond à des ressources dont le contenu est secondaire (images, javascript etc...). Un "page view" implique généralement de multiples "hits". Des arguments peuvent être associés à un "hit". Cependant, leur prise en compte, tandis qu'elle augmente de plus de 100% l'hétérogénéité des "hits" sans argument, n'apporte guère d'information supplémentaire. L'évolution de la diversité des "hits" (sans prise en compte d'éventuels arguments) peut se modéliser, logarithmiquement, avec l'équation 2.2. Leur longueur moyenne est de 42 caractères.
- Le "How" est caractérisé par la "UserAgentsString" indiquant l'application cliente qui a procédé à la requête. Les politiques historiques des navigateurs, leur évolution incessante, ainsi que la multiplication des plateformes webs (terminaux mobiles etc...) rendent difficile l'établissement d'une base de données exhaustive et stable de ces chaînes. Par conséquent, il semble indispensable de procéder à un "parsing" permettant d'extraire les informations les plus pertinentes : Le type de navigateur (éventuellement sa version), ainsi que le type d'OS.
- Le "From" est représenté par le "referer", une chaîne optionnelle contenant une URL. Celle ci peut être interne au site www.infres.enst.fr ou externe. Ce sont les URLs de ce dernier type (longueur moyenne de 40) dont l'analyse paraît la plus pertinente. Les équations 2.3 et 2.4 modélisent l'évolution de leur hétérogénéité.

Chapitre 3

Structure du cube de données

Fort des informations acquises dans le chapitre précédent (résumées dans la synthèse 2.2.2), la présente section propose une structure de cube données pour modéliser le flux. Les dimensions retenues, ainsi que les hiérarchies qui leur sont associées, sont, tout d'abord, présentées de manière textuelle. Puis, la figure 3.2 propose un schéma en "flocon" pour un stockage de type ROLAP. Enfin, le tableau 3 résume la structure du cube en proposant notamment une estimation volumétrique à l'année.

La dimension "When"

Il s'agit de la dimension "Temps" classique que nous renommons ainsi par cohérence avec le nom des autres dimensions. L'alimentation de notre cube étant réalisée en "temps réel", une granularité au niveau journalier, comme il fait classiquement, serait inadaptée au projet. Ainsi, la granularité retenue sera, ici, l'unité horaire. Par ailleurs, une granularité en terme de plages horaires est mise en place : Ce ne sont pas moins de six zones journalières qui sont distinguées :

- 9h00-12h00 : Horaires de bureau — Matinée
- 12h00-14h00 : Pause déjeuner
- 14h00-19h00 : Horaires de bureau — Après-midi
- 19h00-23h00 : Soirée
- 23h00-09h00 : Nuit

Les hiérarchies de cette dimension sont les suivantes :

$$HourDayMonthYear > DayMonthYear > MonthYear > Year\ Hour > TimeSlot$$

La dimension "Who"

Cette dimension précise la provenance géographique de la requête HTTP. Avant traitement, cette information se présente dans le log sous la forme d'une adresse IP brute ou d'un nom d'host. Or, c'est cette dernière qui offre l'information la plus précise. Par conséquent, la granularité la plus fine de cette dimension sera le nom d'host. Puisqu'une IP brute, non résolue, ne présente pas d'intérêt à être stockée en elle même (L'attribution dynamique à la connection n'assure même pas que ce soit la même machine utilisé pour deux logs relevés avec la même IP), elle sera associée dans la dimension à un nom d'host "unknown". Cependant, cela serait une erreur que de "jeter" ces IP sans en extraire la moindre information. En effet, toute IP, résolues ou non, possède intrinsèquement une information géographique de "haut niveau" : la localisation en terme de pays. Il suffit de multiplier les nombres qui composent l'adresse (aux coefficients près) et de se référer à une base de données telle que celle proposée gratuitement par le site ip-to-country ([12]) pour obtenir le pays correspondant. En fait, ce

type de base de données peut être perçue comme un répertoire téléphonique mais pour internet. De même, un nom d'host se termine par le nom de code du pays correspondant (.fr, .es, etc...). Or, la base de données de ip-to-country, associe également les noms de codes des pays avec les pays eux mêmes. A l'exception des noms d'host spéciaux (.net, .org, etc...), il est donc possible d'associer un nom d'host à un pays.

L'unique hiérarchie de cette dimension est donc la suivante :

$$Host > Country$$

A raison de 33 octets en moyenne par noms d'host, de 7 octets en moyenne par pays, d'un "header" d'une dizaine d'octets et d'une clé primaire de 4 octets, un tuple de cette dimension ferait en moyenne 54 octets. Soit, en injectant dans l'équation 2.1, une volumétrie pour une année de :

$$[10 + 4 + 33 + 7] * [2290 * 365 + 2101] \sim 45Mo \tag{3.1}$$

Les dimensions "Method", "Protocol" et "Status"

On ne s'attardera pas sur ces dimensions statiques et extrêmement simples puisqu'elles stockent respectivement la dizaine de méthodes HTTP qu'il est possible de rencontrer dans une requête, les deux protocoles existants (HTTP ou HTTPS), ainsi que les différents codes de statut. Elles ne possèdent aucune hiérarchie particulière. Leur taille est complétement négligeable.

La dimension "What"

Cette dimension caractérise, pour un fait donné, la ressource demandée, c'est à dire un "hit". Cependant, ceux qui sont associés à des arguments dans les requêtes seront tronqués comme expliqué dans la section 2.2.2. Chaque "hit" est précisé par quatre autres attributs, qui sont illustrés avec l'exemple ~hebrail/enseignement.html :

- "HighestLevel" : Noeud le plus haut dans l'arborescence du site www.infres.enst.fr auquel il appartient. Dans l'exemple, il s'agit de ~hebrail.
- "DocumentType" : Identifie le type de la ressource (cf section 2.2.2). Ici, vaut "html".
- "IsPageView" : Boolean caractérisant le fait que le "hit" en question appartient au sous ensemble des "pages views" ou non. Ici, vaut "1", puisque qu'une page html est l'exemple typique d'un "page view".
- "IsPersonalType" : Boolean qui caractérise la fonction de la ressource. Typiquement deux possibilités : Personnel ou institutionnel. Toujours dans cet exemple, cet attribut vaut "1".

Cette dimension est composée des deux hiérarchies suivantes :

$$Hit > DocumentType > IsPageView$$
$$Hit > HighestLevel > IsPersonalType$$

A raison de 42 octets en moyenne par nom d'host, de 7 octets en moyenne par "HighestLevel", de 3 octets en moyenne par "DocumentType", de 1 octet par boolean, d'un "header" d'une dizaine d'octets et enfin d'une clé primaire de 4 octets, un tuple de cette dimension ferait en moyenne 68 octets. Soit, en injectant dans l'équation 2.2, une volumétrie pour une année de :

$$[10 + 4 + 42 + 7 + 3 + 1 + 1] * [3576 * \log(349) + 8625] \sim 1,2Mo \tag{3.2}$$

La dimension "How"

Il s'agit des informations pertinentes sur l'application cliente qui a procédé à la requête, informations enfouies dans le "UserAgentsString" (cf section 2.2.2). Les attributs retenus sont les suivants :

- "Navigator" : Navigateur avec sa version exacte. Exemple : "MSIE 7".
- "NavigatorType" : Type du navigateur. Ici, "Microsoft".
- "OS" : "Linux", "Windows" ou "Macintosh". Toujours dans le même exemple : "Windows".

L'unique hiérarchie de cette dimension est donc la suivante :

$$Navigator > NavigatorType > OS$$

Le volume de cette dimension est totalement négligeable.

La dimension "From"

Cette dimension relève de l'analyse des "referers". En plus de la clé primaire, cette dimension possède trois autres attributs :

- "Url" : L'URL exacte. En fait, l'URL n'est précisée que pour celles qui sont externes et non de type "moteurs de recherches" (on précise alors le nom du moteur) ou bien encore de type "mails" (on indique alors uniquement "mail" dans le champ). Les URLs internes ne sont, quant à elles, pas répertoriées étant donné le peu d'interêt qu'elles présentent. En lieu et place, on précise juste "internal".
- "UrlSubType" : Sous catégorie de l'URL : "SearchEngines", "Mails", "Personal", etc... comme décrit dans la section 2.2.2.
- "UrlType" : Catégorie de plus haut niveau pouvant prendre trois valeurs : "NotFilled", "External" or "Internal".

L'unique hiérarchie de cette dimension est donc la suivante :

$$Url > UrlSubType > UrlType$$

Deux cas sont à distinguer pour la taille moyenne de l'attribut "Url" : Ou bien, il s'agit d'une adresse externe, à proprement parler (pas de moteurs, mails etc...), et on l'a estimé dans la section 2.2.2 à 40 octets ; Ou bien, il s'agit de tout autre type d'URLs, et alors, ce n'est pas l'URL elle même qui occupe l'attribut mais le nom du moteur de recherche ou bien encore "internal" etc...soit à peu près 6 octets de moyenne. Estimons l'attribut "UrlSubType" et "UrlType" à une moyenne de 9 octets, et considérons, comme pour les précédentes dimensions, un "header" d'une dizaine d'octets ainsi qu'une clé primaire de quatre octets.

De plus, envisageons la possibilité d'un shéma en flocon avec une table "KeyWords" contenant les mots-clés utilisés par les internautes dans les moteurs de recherches. Une association (n,m) existe entre cette table et la table "From" : Une URL peut ainsi être reliée à zéro ou à plusieurs mots-clés et, réciproquement, un mot-clé est utilisé dans au moins une URL, voire plus. Dès lors, deux URLs, quant bien même elle correspondraient à un même moteur de recherche, devront avoir deux clés primaires différentes afin de se distinguer dans la table de jointure. Ainsi, en distinguant la taille des tuples en fonction du contenu de l'attribut "Url", et en en injectant dans les équations 2.3 et 2.4, on obtiendrait au bout d'un an :

$$[10+4+9+9+6]*[2378*365+15]+[10+4+9+9+40]*[147*365+411] \sim 37Mo \quad (3.3)$$

La table des faits

Cette table contient les logs, agrégés selon la granularité la plus fine des dimensions précédentes. Deux mesures permettent cette agrégation : La première additionne le nombre de logs agrégés tandis que la seconde cumule la quantité d'octets transférés. La volumétrie de cette table est très difficile à estimer. Contentons nous du pire cas, c'est à dire sans agrégation, en considérant un stockage ROLAP (un espace mémoire réservé pour chaque cellule pleine) : L'échantillon contient environ 1,4 millon de logs pour 2 semaines soit environ 33,6 millons de logs à l'année. La volumétrie, au bout d'un an, serait donc de :

$$[10 + 9 * 4] * 33600000 \sim 1,5Go \tag{3.4}$$

Tables	Evolution	Hiérarchies	Volumétrie/tuple	Volumétrie/ar
"When"	Statique	$HourDayMonthYear > DayMonthYear > MonthYear > Year$ $Hour > TimeSlot$	$\sim$ 50 octets	$\sim$ 430ko
"Who"	Dynamique	$Host > Country$	$\sim$ 54 octets	$\sim$ 45 Mo
"Method"	Statique	$Method$	-	Négligeable
"Protocol"	Statique	$Protocol$	-	Négligeable
"Status"	Statique	$Status > Class$	-	Négligeable
"What"	Dynamique	$Hit > DocumentType > IsPageView$ $Hit > HighestLevel > IsPersonalType$	$\sim$ 68 octets	$\sim$ 1,2 Mo
"How"	Statique	$Navigator > NavigatorType > OS$	-	Négligeable
"From"	Dynamique	$Url > UrlSubType > UrlType$	$\sim$ 38 ou 72 octets	$\sim$ 37 Mo
AggregatedLogs"	Dynamique	None	$\sim$ 46 octets	$\sim$ 1,5 Go

FIG. 3.1 – Récapitulatif des tables du cube avec une estimation de leur volumétrie

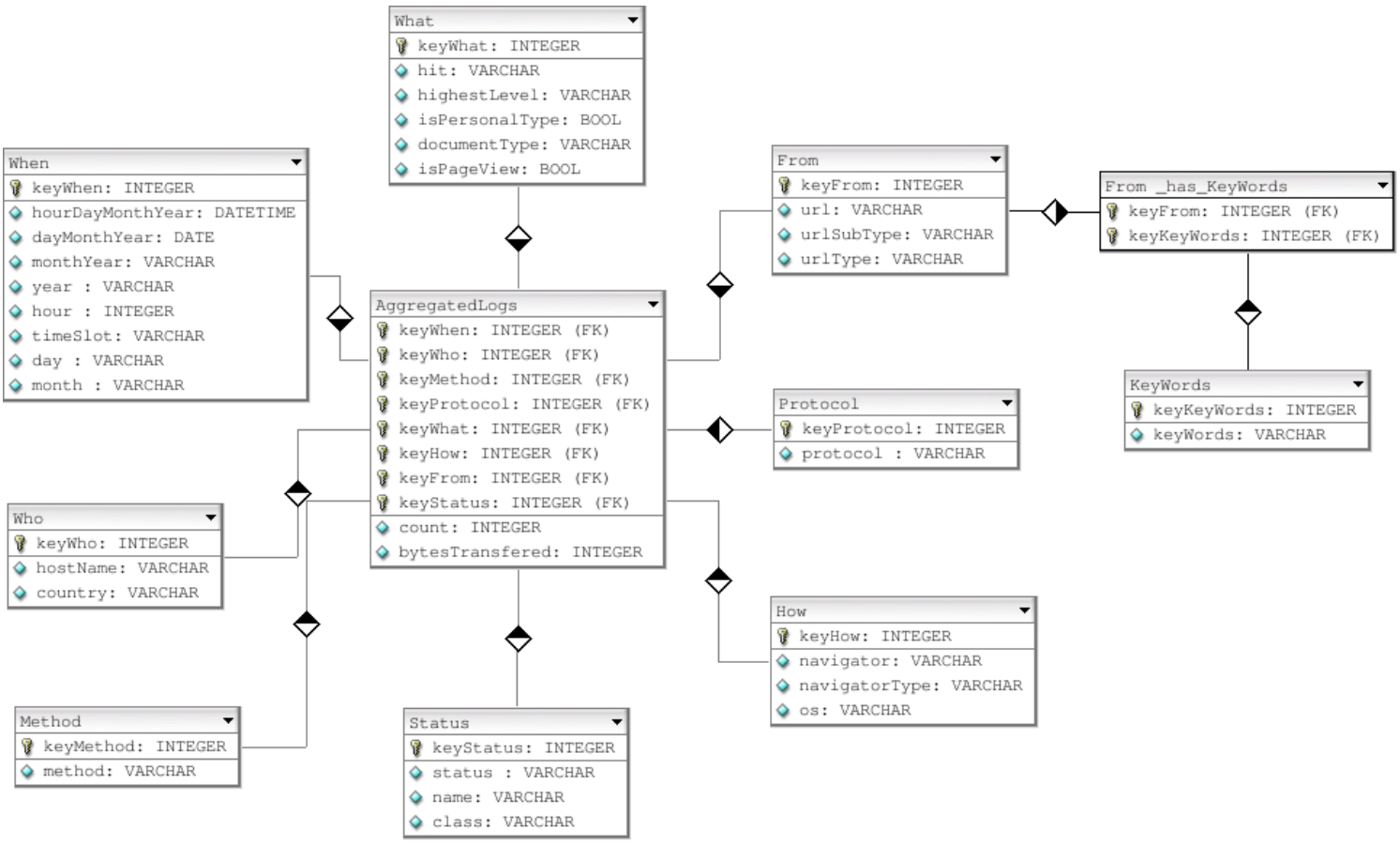

FIG. 3.2 – Shéma en flocon de la structure du cube

Deuxième partie

Étude du DSMS : Aleri Streaming Platform

Chapitre 4

Le modèle de données

Cette partie, charnière, n'a pas pour vocation de présenter exhaustivement la "Aleri Streaming Platform" dans sa version 2.4. Il s'agit, ici, de fournir au lecteur les éléments de base de ce DSMS afin qu'il puisse comprendre sereinement la proposition de prototype décrite en partie III.

La "Aleri Streaming Platform" peut être vue comme une "machine" capable de réaliser du "Complex Event Processing" ou "CEP". Son objectif est d'offrir, à son utilisateur, l'opportunité d'analyser des flux de données, en temps réel : Traitements spécifiques pour des données réunissant certaines conditions, visualisation temps réel de données agrégées etc...

Deux aspects de la platforme sont à dissocier :

- Le "Aleri Platform server" : C'est lui qui traite toutes les données entrantes, sous la forme d'événements temps réel.
- La partie "Authoring tools" : APIs pour les développeurs, utilitaires en ligne de commande (Requêtes ad-hoc, "upload" de fichiers de données, simulation d'un client "publisher" etc...), et enfin un IDE complet, basé sur Eclipse, afin de créer, visualiser et tester les applications.

Une telle application se construit autour d'un **modèle de données** qui contient les opérateurs et les règles qui seront appliqués aux données entrantes afin d'en produire des résultats. La "Aleri Streaming Platform" propose un langage de haut niveau permettant de se concentrer sur la logique métier, sans se préoccuper des autres aspects. En de nombreux points, la création d'une application pour la "Aleri Streaming Platform" est similaire à celle d'une application fondée sur une base relationnelle. Néanmoins, la différence fondamentale réside sur le fait que la "Aleri Streaming Platform" traite des données volatiles alors qu'une base relationnelle opère sur des données statiques. Dans ce dernier cas, les données sont insérées puis interrogées. La "Aleri Streaming Platform", quant à elle, inverse en quelque sorte ce concept : On définit, dans un premier temps, le modèle de données c'est à dire un ensemble de requêtes successives, puis, au fur et à mesure que les données arrivent, ces dernières "traversent" le modèle. Cette architecture de type "dataflow" est 100% "event-driven", traitant les messages quand ils arrivent et incrémentant les résultats en temps réel.

Le présent chapitre propose de définir les éléments de base d'un modèle de données au sens de la "Aleri Streaming Platform".

4.1 Les "Data Streams"

Le bloc de base d'un modèle de données est le "data stream". Il y a deux types de "streams" :

- Les "source streams" : Il s'agit de points d'entrées pour des données provenant du monde extérieur.
- Les "derived streams" : Ce sont le fruit d'opérations sur une ou plusieurs autres "streams" du modèle.

La "Aleri Streaming Platform" utilise un paradigme relationnel : Les "source streams" peuvent être pensées comme des tables tandis que les "derived streams" peuvent être pensées comme des vues matérialisées. Ainsi, les champs d'une "stream" sont appelées des colonnes. Un modèle de données consiste en une ou plusieurs "source streams", ayant des colonnes définies, et un ensemble d'opérations produisant une ou plusieurs "derived streams". Ce sont ces dernières qui contiennent et produisent les résultats. Un évènement entrant peut être inséré ("insert") en tant que tuple dans une table, mis à jour ("update") dans la table ou encore détruit ("delete") dans cette même table.

Les "source streams"

On en distingue deux types :

- Les "base streams" : Permet d'importer des données du monde extérieur, données possédant dejà une clé primaire. Les données peuvent se voir appliquer un "insert", un "update" ou bien encore un "delete".
- Les "auto streams" : Il s'agit d'une classe bien particulière de "source streams", optimisée pour les insertions. De plus, elles permettent la génération automatique de clés primaires si les données entrantes n'en sont pas dotées intrinsèquement.

Les "derived streams"

Elles peuvent revêtir différentes formes :

- Les "filter streams" : Prennent en entrée une unique "stream" et propose en sortie un sous ensemble des tuples à partir d'un critère filtrant.
- Les "compute streams" : Prennent en entrée une unique "stream" et calcule, à partir des champs entrants, les nouveaux champs en sortie.
- Les "aggregate streams" : Prennent en entrée une unique "stream", regroupe les tuples selon certains critères et produit une "output stream" contenant l'information agrégée ("sum", "count", "average" etc...) pour chacun de ces groupes. De fait, il y a en sortie un tuple agrégé par groupe de tuples.
- Les "union streams" : Combinent deux ou plus "input streams" qui doivent alors impérativement avoir le même format.
- Les "join streams" : Combinent les tuples de deux ou plus "input streams" et produisent une unique "output stream" où les tuples sont combinés à partir des expressions individuels de chaque nouvelle colonne.
- Les "copy streams" : Produisent simplement une copie de la "input stream" avec la possibilité d'appliquer des règles de rétention différentes.
- Les "flex streams" : "Streams" programmables à partir d'un jeu de méthodes définies dans un script SPLASH (explicité en section 4.3) et s'appliquant aux données entrantes.

Toutes les "streams" sont associées avec un magasin ou "store", définissant de quelle manière la sortie du flux est stockée.

4.2 Les "Stores"

On en distingue trois types :

- Les "memory stores" : Ils permettent de stocker les données dans la mémoire vive. Il n'y a pas de restriction sur leur utilisation si ce n'est qu'en cas de crash du système, toutes les données seront perdues.
- Les "log stores" : Permettent de faire de la "disk-based" persistance à partir de fichiers et donc de récupérer les données même en cas de crash système.
- Les "stateless stores" : Ne retiennent aucune donnée : Ces magasins opèrent sur un enregistrement à la fois et le "jette" une fois la sortie produite. De sorte, les "stateless stores" sont extrêmement efficaces notamment pour faire des calculs intermédiaires. Cependant, ils restent assez restrictifs et ne peuvent pas être utilisés avec tous les types de "streams".

4.3 Les expressions et les scripts "SPLASH"

On nomme "expression" un ensemble d'opérations sur les tuples d'une (ou plus) "input stream" afin de déterminer les tuples de la "output stream". Ces expressions sont notamment utilisées dans les "filter streams" pour exprimer le critère filtrant, les "compute streams" afin de calculer les tuples en sorties etc...Un certain nombre d'opérations sont mises à la disposition du concepteur de modèles. Nous ne les citerons pas de manière exhaustive mais soulignons, malgré tout, les grandes catégories auxquelles elles appartiennent :

- Les opérateurs arithmétiques : Il s'agit classiquement des additions, soustractions etc... Ils ne peuvent s'appliquer que sur des champs numériques.
- Les opérateurs de comparaison : "=", ">=", etc... A l'instar des précédents, ils ne peuvent s'appliquer que sur des champs numériques.
- Les opérateurs boolean : "AND", "OR", etc...Comparent uniquement des int32 et retournent des int32.
- Les fonctions arithmétiques : "log", "sin", etc ... Elles ne peuvent s'appliquer que sur des champs numériques.
- Les fonctions d'agrégation : Typiquement utilisées dans les "aggregate streams", elles s'appliquent sur tout type de champs. Elles offrent un large panel de possibilités à partir d'un groupe de tuples : "avg", "first", "min" etc ...
- Les fonctions opérant sur les chaînes de caractères : leurs possibilités sont assez limitées et se cantonnent à la concaténation de chaînes, à l'inversion de la casse, etc...
- Les fonctions de dates et d'heures
- Les fonctions de conversion de type : "String" à int32 etc...
- Les fonctions définies par l'utilisateur : Il est important de souligner que la platforme offre aussi la possibilité de créer ses propres fonctions en Java ou en C++. Il y a, bien sûr, certaines restrictions en terme de type d'arguments et de valeurs de retour mais nous ne les détaillerons pas ici, afin de ne pas alourdir cet écrit. Ces fonctions peuvent être appelées au sein d'une expression. Créer de telles fonctions, sur mesure, peut notamment s'avérer utile lorsque les fonctions/opérateurs précédents ne parviennent pas à répondre aux besoins d'un traitement spécifique.

Si une simple combinaison de ces fonctions ne suffit pas, il est aussi possible de déclarer des variables et de faire des traitements itératifs (boucles etc...) dans un script SPLASH ayant une syntaxe proche de celle du C. Ceci est utilisé dans les "flex streams".

Nous conseillons au lecteur, qui souhaiterais approfondir la création de modèles de données, de consulter la référence [8] de la bibliographie.

Chapitre 5

Les interfaces

Il a été évoqué, dans la section précédente, la présence de données venues du "monde extérieur". En effet, une fois le "data model" crée, et le "Aleri Platform server" prêt à traiter des "incomming events", il convient de posséder un moyen de lui envoyer des données : Il s'agit du mécanisme de publication, explicité dans la section 5.1. De même, une fois que les données ont traversé le "dataflow", il serait intéressant de pouvoir en explorer les résultats, notamment au sein d'un cube de données : Il s'agit du mécanisme de souscription détaillé en section 5.2. La figure 5.1, extraite de la page web http ://www.aleri.com/products/aleri-streaming-platform/, schématise ces deux interfaces avec la "Aleri Streaming Platform" :

FIG. 5.1 – Diagramme illustrant la publication et la souscription

5.1 La publication

La "Aleri Streaming Platform" propose aux développeurs une API afin de créer leur propre "publisher" ou "adaptater". Les objet fondamentaux auquels ces "adaptaters" peuvent publier des données sont les "source streams" définies précédemment. L'interface de publication permet notamment aux applications clientes de modifier le contenu du "store" de ces dernières en leur communiquant des "enregistrements" couplés avec un ordre d'exécution : "insert", "update", "delete" etc...Trois langages sont proposés pour cette API : Java, C++ ou bien encore .NET. Seule l'API Java sera étudiée dans ce document.

L'interface expose un certain nombre de fonctionnalités en en cachant l'implémentation. En effet, les classes de l'API possèdent des constructeurs privés et c'est grâce à l'utilisation d'un pattern "Factory" (class **SpFactory**) que l'application peut les instancier. Malheureusement, le "design" de cette API, tout du moins dans sa version 2.4, n'autorise pas l'utilisation

d'exceptions : De fait, toute fonction de l'API renvoie un code d'erreurs (zéro ou autre) afin d'indiquer la réussite ou non de son traitement interne. On peut déplorer ce "design" "anti programmation objet", qui, après discussion avec les ingénieurs d'Aleri, semble évoluer dans la version 3.0 de la platforme.

Le mécanisme de publication, très hiérarchisé, est non seulement présenté ci dessous mais aussi résumé dans les figures 5.2 et 5.3 :

1. Instanciation, via la "Factory", d'un objet **SpPlatformParms**. C'est lui qui contient toutes les informations nécessaires à la connection avec la platforme (nom d'host, numéros de ports, nom d'utilisateur, mot de passe, nom de la "source stream" , etc...).
2. Instanciation, via la "Factory", d'un objet **SpPlatformStatus** auquel on pourra faire appel afin d'obtenir les informations du statut de la dernière méthode appelée de l'API.
3. Instanciation, via la "Factory", d'un objet **SpPlatform** auquel on donne en paramètre les objets **SpPlatformParms** et **SpPlatformStatus** précédents.
4. Instanciation, via le **SpPlatform** précédent, d'un objet **SpPublication** : C'est lui qui sera véritablement responsable de la publication des enregistrements ou **SpStreamDataRecord** expliqués ci dessous.
5. Instanciation, de **SpStreamDataRecords** : Ils encapsulent l'information à transmettre à la "stream" ainsi que la commande associée ("insert", "update" etc...).
6. On fournit alors une collection de ces **SpStreamDataRecords** à l'objet **SpPublication** qui peut les transmettre à la "stream", via une "socket", et selon un mode de publication choisi : l'un après l'autre de manière synchrone , par bloc transactionnel (une erreur sur l'un des enregistrements annule tous les autres), par bloc "enveloppe" (un unique accusé pour une collection d'enregistrements) etc...

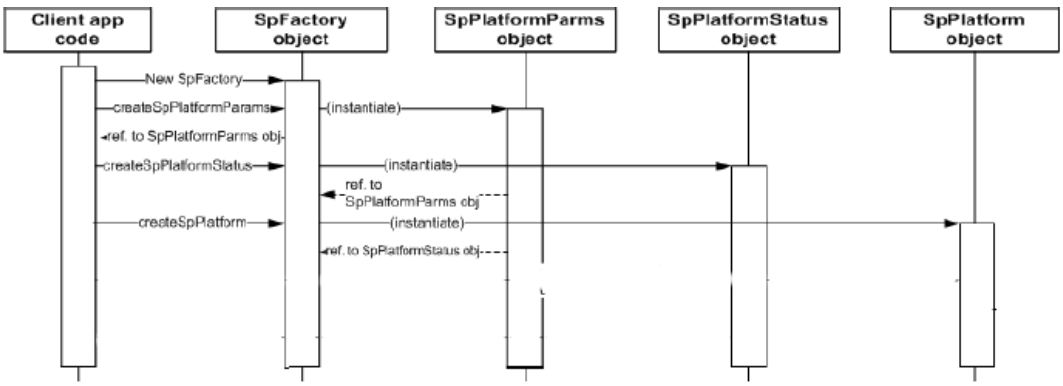

FIG. 5.2 – Initialisation des objets associés au mécanisme de publication

Une fois publiés dans une "source stream", les enregistrements sont entièrement pris en charge par la platforme, suivant le "dataflow" mise en place dans le modèle de données.
Nous conseillons au lecteur, désireux d'approfondir ce mécanisme de publication, de consulter la référence [9] de la présente bibliographie.

5.2 La souscription avec le composant "Aleri live OLAP"

Lors de l'écriture de ce rapport, aucune documentation pour ce composant n'a malheureusement pue être arrachée des mains de la société Aleri. Il semblerait que ce composant ne soit encore qu'au stade d'une version beta...

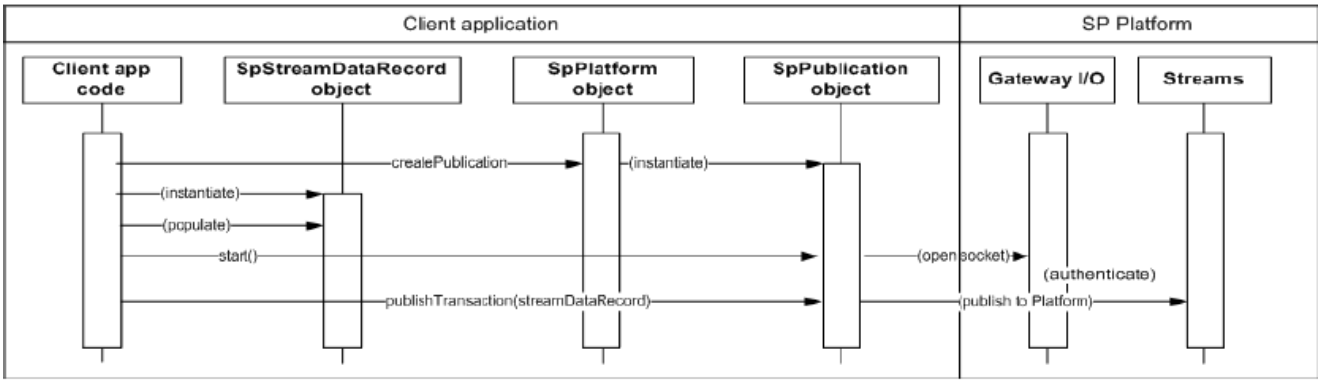

FIG. 5.3 – Illustration du mécanisme de publication

Bilan et émergence de problématiques

Le lecteur dispose, désormais, des éléments de base pour construire un "data model" et publier des données au sein de la "Aleri Streaming Platform". Faut-il encore maintenant appliquer ces connaissances, récemment acquises, au cas de notre flux de logs. Or, ceci n'est pas sans soulever plusieurs problématiques :

- Comment concevoir notre "adaptater" pour qu'il "relie" la source du flux de logs, définie dans le chapitre 1 avec la "Aleri Streaming Platform" ?
- Le flux se compose d'une succession de logs au "Combined Log Format" comme décrit en section 2.1. Mais sous quel format allons nous les publier dans la "Aleri Streaming Platform" ? Autrement dit, quelles seront les colonnes de la "source stream" qui accueillera ces données ?
- Comment assurer alors le passage du format "Combined Log" au format de la "source stream" ?
- Une fois les données publiées, comment allons nous construire le modèle de données pour qu'il puisse permettre la construction du cube OLAP décrit dans le chapitre 3 ? Quel(s) type(s) de "streams" utilisés ? Avec quel(s) type(s) de "stores" ? Les expressions et le langage "SPLASH" suffiront ils à traiter les données où faudra il créer des fonctions, sur mesure et extérieures à la platforme, comme explicité dans la section 4.3 ?
- De quelle(s) source(s) de données, extérieure(s), devrons nous disposer pour construire les dimensions du cube ? Comment les intégrer, alors, au modèle de données ?

La partie III suivante, au regard de ces problématiques, propose une maquette pour implémenter ce cube de données.

Troisième partie

Proposition d'un prototype

Chapitre 6

Publication du flux de données

6.1 Du serveur Apache à la "Aleri Streaming Platform"

Le chapitre 1 a mis en évidence la source du flux de données : une "socket" écoutant sur le port 8999 de la machine infres5.enst.fr, prête à transmettre, en temps réel, les "access logs" au "Combined Log Format". Cependant, avant d'envisager l'acheminement de ce flux vers la "Aleri Streaming Platform", il convient, aussi, de savoir quel va être le point d'entrée dans le "data model". Autrement dit, quelle est la "source stream" qui va accueillir les données du flux ? Deux options de conception se présentent alors, comme décrit dans la section 4.1 : Une "base stream" ou bien encore une "auto stream".
C'est la nature même des données du flux qui oriente naturellement ce choix vers une "auto stream". **En effet, les logs ne sont pas dotés intrinsèquement d'une clé primaire**. Il serait donc intéressant de pouvoir la générer automatiquement par le DSMS : Seule l'"auto stream" offre une telle possibilité. De plus, les logs n'ont pas vocation à être stockés sous leur forme brute, mais à être traités afin de les agréger dans la table des faits comme expliqué dans le chapitre 3. Par conséquent, la publication des logs résulte uniquement en des "insert" et non pas en des "update" ou "delete". Les restrictions imposées par l'utilisation d'une "auto stream" ne sont donc pas handicapantes, bien au contraire.
La figure 6.1 montre cette "auto stream", telle qu'elle apparaît au sein de l'IDE Studio :

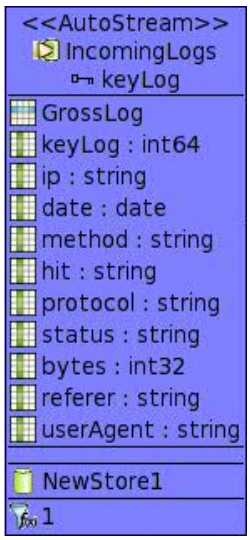

FIG. 6.1 – Le point d'entrée des logs dans le "data model" : une "auto stream"

Nous reviendrons, incessamment sous peu, sur le choix des colonnes de cette "stream" dans la section 6.2. Ici, le rôle de l'adaptateur va donc être de permettre l'acheminement des données de infres5.enst.fr à la "Aleri Streaming Platform", cette dernière étant exécutée sur une machine dédiée : ushuaia.enst.fr. Ainsi, il va devoir réaliser de manière chronologique les opérations suivantes :

1. Établissement d'une "socket" connectée au port 8999 de la machine infres5.enst.fr
2. Initialisation des objets de publication décrits en section 5.1 (**SpPlatformParms**, **SpPlatformStatus**, etc...). Cela va notamment établir une connection locale, par "sockets", entre le "thread" java et la "I/O Gateway" de la platforme.
3. Puis, en boucle infinie, pour chaque log lu :
   (a) "Parsing" du log afin d'identifier et d'isoler les différents champs qui le composent.
   (b) Conversion des champs en adéquation avec le typage de données requis par l'API de publication d'Aleri.
   (c) Encapulation de ces données dans un objet java **Collection**, dans l'ordre requis par le format de l'"auto stream".
   (d) Encapsulation de la **Collection** et d'un ordre d'insertion dans un nouvel objet **SpStreamDataRecord**.
   (e) Publication synchrone de ce **SpStreamDataRecord** dans la platforme via l'objet **SpPublication**.
   (f) Attente du "acknowledgment code".

La figure 6.2, ci dessous, résume de manière macroscopique, en quelle mesure l'adaptateur joue un rôle de "connecteur" entre deux flux aux formats différents :

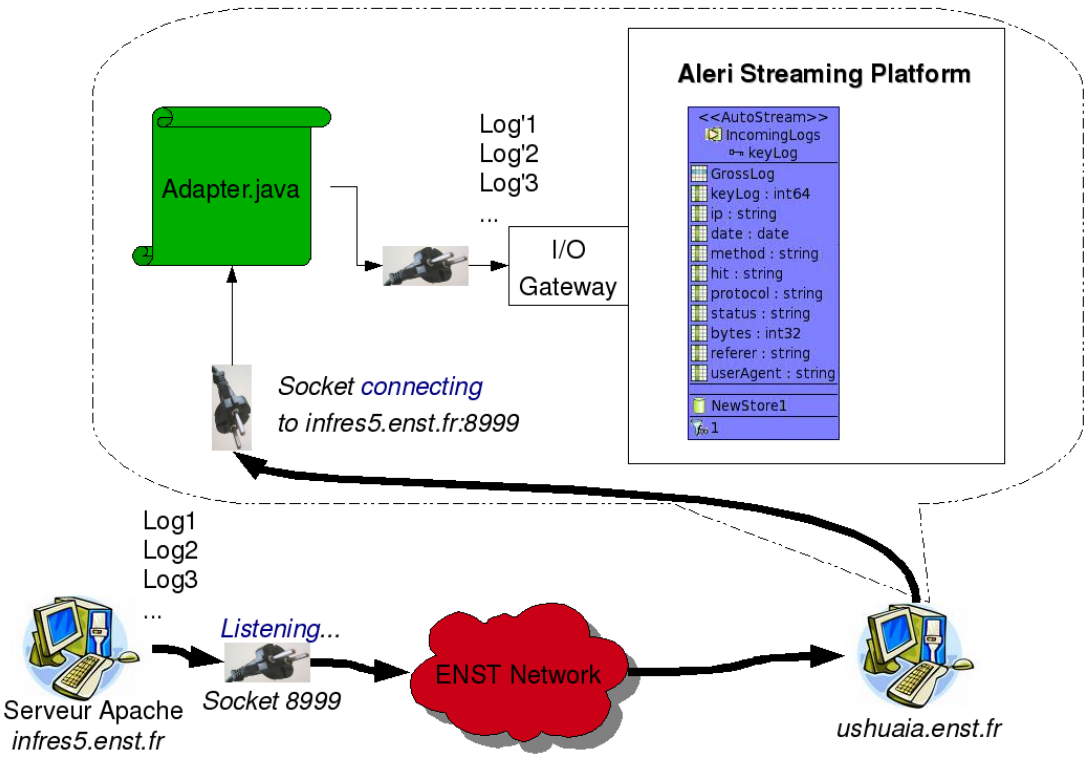

FIG. 6.2 – Acheminement du flux du serveur Apache à la platforme via l'adaptateur

6.2 Parsing du "Combined Log Format"

L'algorithme de l'adaptateur, présenté dans la section précédente, fait mention d'une étape de "parsing" (étape (a)). En effet, il est bien nécessaire, à un moment ou à un autre du "dataflow", d'isoler les différents champs du log, champs qui vont alors recevoir un traitement spécifique. Une possibilité aurait pu consister en la réalisation de cette étape au sein même de la "Aleri Streaming Platform", dans le "data model". Dans ce cas, l'adaptateur aurait simplement publier une seule "string" contenant la ligne entière du log. Pourtant, il a été préféré d'effectuer ce "parsing" au niveau de l'adaptateur et ce pour trois raisons principales :

- La complexité : Les opérations sur les chaînes de caractères pouvant être utilisées dans les "expressions" (cf section 4.3) sont relativement basiques et ne proposent pas, en outre, le recours aux expressions régulières. Il aurait donc fallu créer des "user defined" fonctions, en Java, qui auraient été appelées au sein des "expressions". Ces fonctions auraient été notamment alourdies par l'utilisation d'exceptions. Dès lors, il paraît bien plus raisonnable de faire ce traitement dans l'adaptateur, lui même en Java natif.
- La gestion des erreurs : Le "parsing" des logs, en Java ("matching" d'expressions régulières) peut lever des exceptions en cas d'échec. Retenons simplement ici que le processus de "parsing" est source d'erreurs et que ces dernières devront être mémorisées ce qui implique le recours à l'ouverture/écriture de fichiers (nous reviendrons sur ce point dans la section 6.3). Or, il n'est vraiment pas adapté de faire appel à ce type de mécanismes dans des "user defined" fonctions de la platforme (complexité, noms des fichiers écrits en dur, performances douteuses, etc ...).
- La logique "métier" : Le "parsing" des logs, dans une simple perspective d'identification des champs, semble être d'avantage un pré-traitement du flux, qu'un véritable traitement "métier" pour construire et alimenter le cube.

Les colonnes de la "source stream" correspondent, grosso modo, à l'identification des champs du "Combined Log Format" du chapitre 2.1, à quelques exceptions près :

- Les deuxièmes et troisième champs (%*l* et %*u*) ne sont pas utilisés car ils n'interviennent pas dans la structure du cube.
- Le cinquième champ, précisant la requête, est éclaté en trois : La méthode, le protocole, et le "hit". En effet, ces trois paramètres interviennent dans trois dimensions différentes.

Les expressions régulières mises au point pour le "parsing" sont présentées ci dessous. Elles mélangent les notions de "capturing groups" et de "non-capturing groups" couplées avec la syntaxe Java pour les expressions régulières. Afin de ne pas alourdir le récit, ces expressions ne seront pas commentées. Cela dit, pour les interpréter au mieux, nous invitons le lecteur à consulter la documentation de Sun sur la création et l'utilisation de "Patterns" ([12]).

```
// Ip RegEx
re_ip="(^[^\\s]+)";
// Date RegEx : [dd/mmm/20XX:time]
re_date="((?:.*\\[)((\\d\\d)(?:/)(\\w{3})(?:/)(20\\d{2})(?::)(\\d
    {2}:\\d{2}:\\d{2})(?:.*))(?:\\]))";
// Request RegEx
re_request="((?:.*\")(GET|POST|HEAD|PUT|DELETE|OPTIONS|PROPFIND)
    (?:\\s)(/[^?]*)(?:.*\\s)(HTTP|HTTPS))";
// Status and Bytes sent RegEx
re_status_and_bytes_sent="((?:.*\"\\s)(\\d{3})(?:\\s)((\\d+)|(-))
    (?:\\s))";
// Referer RegEx
```

```
re_referer="((?:.*\")((http.*)|(-))(?:\"\\s\"))";
// User Agent RegEx
re_user_agent="([^\"]*)";

Pattern pattern_log = Pattern.compile(re_ip+re_date+re_request+
    re_status_and_bytes_sent+re_referer+re_user_agent);
```

Enfin, précisons que ces expressions ont été obtenues avec une démarche itérative, à partir de l'échantillon, afin d'améliorer sans cesse le nombre de logs dont le "parsing" était fructueux. Avec ces dernières expressions, le taux de réussite est de 99,96%, les échecs étant liés à des cas non nominaux et non expliqués.

6.3 "Design" de l'adaptateur

L'objectif de cette section est d'expliciter certains choix de conception technique à propos du "design" de l'adaptateur, notamment en terme de modularité. Tout d'abord, il est donné la possibilité à son utilisateur de choisir deux types de sources de données, et ce de manière paramétrable :

- Une "socket" sur une machine distante : Cas du régime permanent de l'alimentation du cube.
- Un fichier : Cette option permet de publier un fichier de logs dans la platforme. Cela offre des possibilités non seulement pour vérifier l'adaptateur et/ou le "data model" mais aussi pour réaliser des tests de performance en imposant à la platforme un débit de données très élevé.

De plus, l'adaptateur ayant pour vocation de fonctionner en régime permanent et de manière automatique, il a été conçu de façon à traiter toutes les exceptions pouvant se produire au sein de sa boucle. De sorte, même si une exception est levée pendant le "parsing" d'un log, la conversion de ces champs ou bien encore sa publication, celle ci sera enregistrée dans un fichier (choisi en paramètre), accompagnée du log ayant posé problème. L'avantage est alors double : D'un coté, cela rend robuste l'adaptateur qui ne s'arrête pas de publier même si un log pose problème, et d'un autre côté, l'erreur est mémorisée de manière durable au lieu d'être simplement imprimée à la console. Ainsi, l'administrateur de l'adaptateur peut consulter ce fichier d'erreur afin de connaître, à posteriori, les logs qui n'ont pas été publiés dans la platforme c'est à dire ceux qui n'ont pas participés à l'alimentation du cube. En effet, un des fondamentaux des cubes de données n'est-il pas de connaître exactement les données qui ont participées ou non à leur alimentation ?

Le choix, dans le "design" de l'adapateur, d'une publication non seulement synchrone mais aussi individuelle va dans ce sens puisqu'elle seule permet une granularité d'erreurs au log près. En effet, que ce soit avec une publication asynchrone, ou bien encore "par bloc", il serait impossible de savoir les logs qui ont été publiés avec succès ou non.

Étant donné que l'API de publication proposée par la société Aleri n'utilise pas les exceptions mais seulement des codes de retour d'erreurs (cf section 5.1), il a été nécessaire de "wrapper" et de hiérarchiser les erreurs dans les propres classes d'exception de l'adaptateur.

Le lecteur, souhaitant regarder plus en détail l'implémentation du processus de publication, est invité à regarder le code source de l'adaptateur, ci-joint au présent document.

Maintenant que les logs, éclatés en plusieurs champs, sont publiés au sein de la platforme, il est nécessaire de construire les dimensions et la table des faits du cube, comme décrit au chapitre 3. Autrement dit, il est temps de passer à la construction du "data model".

Chapitre 7

Construction du "data model"

La construction d'un modèle de données afin de mettre en place la structure d'un cube comme celle décrit au chapitre 3 est un vaste chantier. A partir d'une "auto stream" alimentant la platforme avec les logs sous leur forme brute, quelle combinaison de "streams" et d'"expressions" (cf chapitre 4) allons nous utiliser afin de construire chacune des dimensions du cube, ainsi que la table des faits ? Comment articuler entre elles ces "streams" ?

7.1 Élaboration des dimensions

Les élaborations de chacune des dimensions, ainsi que les points-clés associés, sont présentées successivement par ordre de difficulté d'implémentation. Commençons, dès lors, par les dimensions "statiques" :

La dimension "Protocol"

Une dimensions statique telle que celle-ci possède la propriété d'avoir ses tuples connus par avance. Ainsi, il est possible de "remplir" cette dimension avant même le commencement de l'alimentation du "data model" en logs. On retrouve ici le concept d'importer des données du "monde extérieur" au sein de la platforme : le concept de "source stream". Mais, ici, c'est une "base stream" et non pas une "auto stream" qui va être choisie, étant donné, notamment, que les données possèdent ici une clé primaire. La "Aleri Streaming Platform", permet d"uploader" un fichier .xml de données au sein de cette "base stream". En voici, le contenu :

```
<ProtocolDimension ALERI_OPS="i" keyProtocol="1" protocol="HTTP"/>
<ProtocolDimension ALERI_OPS="i" keyProtocol="2" protocol="HTTPS"
    />
```

ProtocolDimension est le nom de la "base stream", l'opération associée aux données est une insertion ("i") et enfin **keyProtocol** et **protocol** sont les colonnes. Maintenant que cette dimension est construite et remplie, comment permettre, alors, l'identification du protocole des logs ? Une "compute stream" **UnknownProtocol** va tout d'abord "prélever" de l'"auto stream" **IncomingLogs** uniquement la clé primaire du log ainsi que le protocole qui lui est associé. Il est impératif ici de garder la clé primaire des logs car c'est elle seule qui permettra de les identifier, bien plus tard dans le "dataflow", notamment lors d'une jointure "post traitement" qui sera explicité en section 7.2. Ce concept sera utilisé pour chaque dimension. Une fois le protocole du log isolé, la "join stream" **IdentifiedProtocol** va permettre l'identification du protocole en terme de clé primaire de la dimensions "Protocol". Concrètement, lorsqu'un nouveau log arrive, cette **IdentifiedProtocol** fait la jointure entre **ProtocolDimension** et **UnknownProtocol** avec le critère suivant :

```
ProtocolDimension.protocol=UnknownProtocol.protocol
```

La figure 7.1 présente, ainsi, le "dataflow" associé à l'identification du protocole :

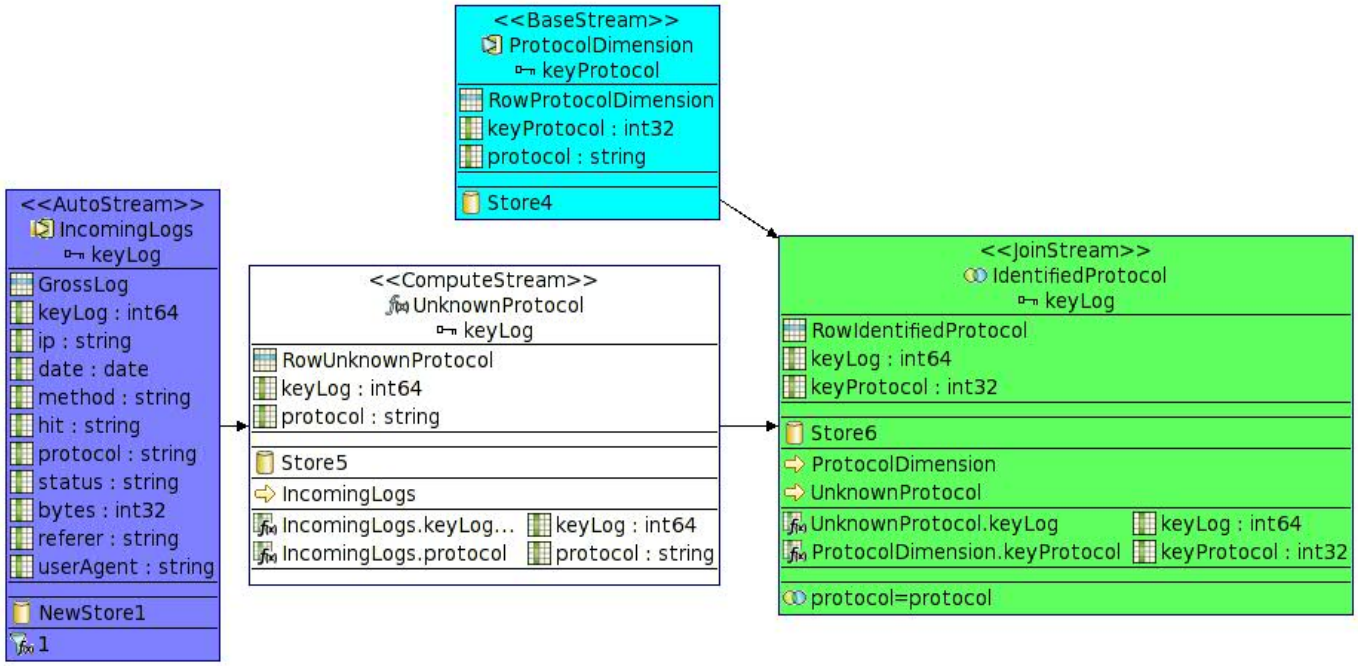

FIG. 7.1 – "Dataflow" permettant l'identification du protocole des logs

Les dimensions "Method" et "Status"

La construction de ces dimension ainsi que l'identifaction de la méthode/statut des **IncomingLogs** est strictement identique, dans la démarche, à celle précédente mais avec les données suivantes pour la dimension "Method" :

```
<MethodDimension ALERI_OPS="i" keyMethod="1" method="GET"/>
<MethodDimension ALERI_OPS="i" keyMethod="2" method="POST"/>
<MethodDimension ALERI_OPS="i" keyMethod="3" method="HEAD"/>
<MethodDimension ALERI_OPS="i" keyMethod="4" method="PUT"/>
<MethodDimension ALERI_OPS="i" keyMethod="5" method="DELETE"/>
<MethodDimension ALERI_OPS="i" keyMethod="6" method="OPTIONS"/>
<MethodDimension ALERI_OPS="i" keyMethod="7" method="PROPFIND"/>
```

Ou bien encore pour la dimension "Status" :

```
<StatusDimension ALERI_OPS="i" keyStatus="1" status="100" class="
    Information" name="Continue"/>
<StatusDimension ALERI_OPS="i" keyStatus="2" status="101" class="
    Information" name="Switching Protocols"/>
<StatusDimension ALERI_OPS="i" keyStatus="3" status="102" class="
    Information" name="Processing"/>
<StatusDimension ALERI_OPS="i" keyStatus="4" status="200" class="
    Success" name="OK"/>
<StatusDimension ALERI_OPS="i" keyStatus="5" status="201" class="
    Success" name="Created"/>
<StatusDimension ALERI_OPS="i" keyStatus="6" status="202" class="
    Success" name="Accepted"/>
...
```

Aussi, il ne sera pas redétaillé tout le mécanisme de construction. Les "streams" impliquées dans le processus sont ici respectivement **MethodDimension**, **UnknownMethod** et **IdentifiedMethod** pour la dimension "Method" et **StatusDimension**, **UnknownStatus** et **IdentifiedStatus** pour la dimension "Status". Le "dataflow" qui en résulte est présenté sur la figure 7.2, les "streams" étant décrites en mode "Image" plutôt que "Compartment" afin de gagner en concision.

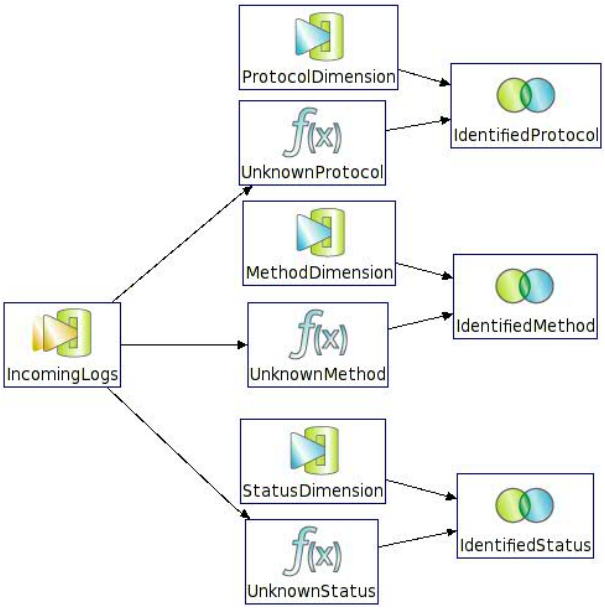

FIG. 7.2 – "Dataflow" rajoutant l'identification des méthodes/statuts des logs

La dimension "When"

Voici encore une dimension statique mais dont la conception est légèrement différente. Tout d'abord, elle se démarque des précédentes par le nombre de tuples qui la composent. En effet, autant auparavant les dimensions ne possédaient que quelques tuples, autant ici, à raison d'un tuple par heure, cela en fait plus de 8760 à créer pour une seul année ! Ajoutons à cela que le nombre des colonnes est beaucoup plus important, comme le montre cet exemple de tuple :

```
<WhenDimension ALERI_OPS="i" keyWhen="43" hourDayMonthYear="
    2008-05-2T18:00:00" dayMonthYear="2008-05-2T00:00:00" monthYear
    ="May 2008" hour="18" timeSlot="Office Afternoon" day="Friday"
    month="May" year="2008" />
```

Dès lors, il paraît difficilement concevable de générer "à la main" un fichier XML si conséquent : il y a donc une réelle nécessité d'automatiser ce processus. La solution retenue fût de créer un script Ruby, sur mesure, afin de remplir cette tâche rébarbative. Le langage Ruby a été choisi de part sa capacité à produire rapidement un code concis et souple. Ainsi, une cinquantaine de lignes de code suffisent à s'accomplir de ce travail d'alimentation. Le lecteur pourra notamment trouver l'intégralité de ce script dans l'annexe B de ce document.

La deuxième différence entre cette dimension et les trois précédentes réside dans le "dataflow" lui même et en particulier au niveau de la complexité des expressions utilisées dans la "computeStream" **UnknownDate**. L'une des difficultés, ici, réside dans l'identification de la date inscrite dans le log avec un unique tuple de la "stream" **WhenDimension**.

En effet la granularité de cette date n'est pas la même dans les deux "streams" (à la seconde près pour l'une, à l'heure près pour l'autre). Dès lors, il n'est pas question de recopier tel quel la date du log dans la "computeStream" : il va falloir la transformer. La première idée serait sans doute d'utiliser la colonne **hourDayMonthYear** qui identifie de manière unique un tuple de la dimension, et de tronquer la date du log au niveau de l'heure. Malheureusement, les "date and time functions" proposées par la "Aleri Streaming Platform" ne permettent que de tronquer une date au niveau journalier (fonction *trunc*). Cependant, il existe une fonction qui permet d'extraire l'heure d'une date (fonction *datepart*). Ainsi, c'est le couple **dayMonthYear** et **hour** qui va permettre d'identifier de manière unique le tuple de la "stream" **WhenDimension**.

De plus, la date des logs est au format 'UTC', impliquant un décalage d'une ou deux heures (en fonction de l'heure d'été/d'hiver) avec l'heure française. Néanmoins, la platforme propose également une fonction *totimezone* permettant de convertir une date d'une "timezone" à une autre.

Ces deux problématiques conduisent donc aux expressions suivantes dans la "stream" **UnknownDate** :

```
UnknownDate.dayMonthYear=trunc(totimezone(IncomingLogs.preciseDate,'UTC','Europe/Paris'))
UnknownDate.hour=datepart('hh',totimezone(IncomingLogs.preciseDate,'UTC','Europe/Paris'))
```

La jointure entre **UnknownDate** et **WhenDimension** s'exprime quant à elle par le système :

```
UnknownDate.dayMonthYear=WhenDimension.dayMonthYear and
UnknownDate.hour=WhenDimension.hour
```

La figure 7.3, ci dessous, fait un zoom sur le "dataflow" permettant l'identification de la date des logs :

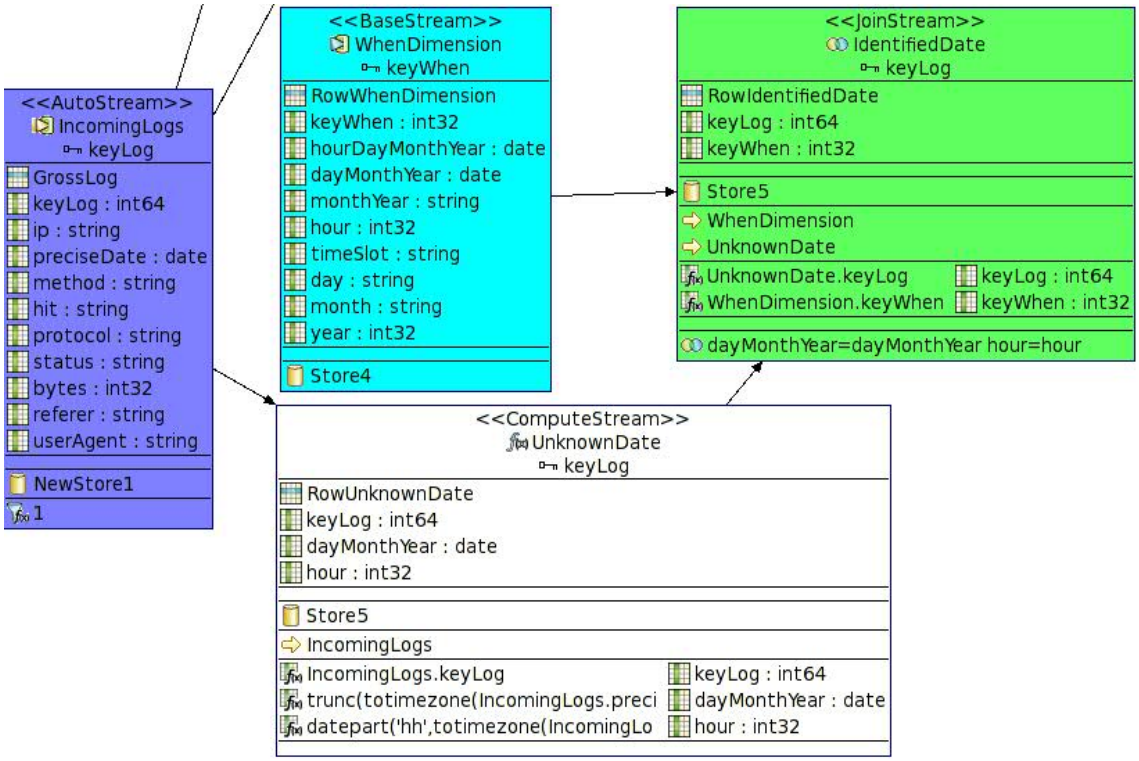

FIG. 7.3 – Zoom sur l'identification de la date des logs via la dimension "When"

La dimension "What"

Cette dimension est la première des dimensions dynamiques c'est à dire que ses tuples ne sont pas connus à l'avance. La manière de la concevoir va donc différer des précédentes : il ne s'agit plus d'alimenter la dimension par un fichier xml chargé dans une "source stream", mais de la créer, en temps réel, en fonction des logs eux mêmes. Le "dataflow" est présenté, ci dessous, sur la figure 7.4 :

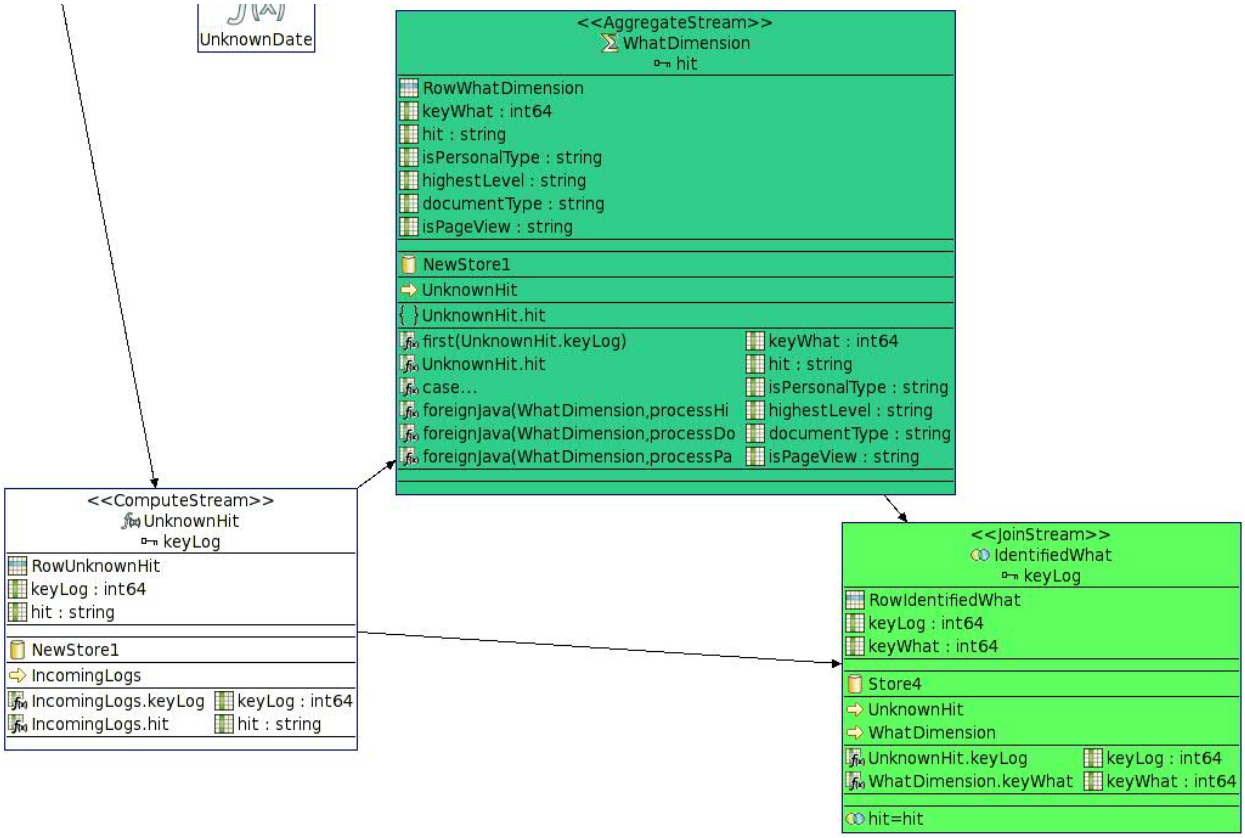

FIG. 7.4 – "Dataflow" de l'alimentation de la dimension "What"

Bien qu'une "compute stream", avec un "memory store" soit toujours utilisée afin d'isoler la partie du log qui va être traitée (rappelons que cette isolation est nécessaire du fait que l'"auto stream" est associée avec un "stateless store"), un nouveau type de "stream" fait son apparition : une "aggregate stream". Ce dernier type sera notamment employé, par la suite, à chaque fois qu'une dimension dynamique entrera en jeu. Ses colonnes correspondent exactement au schéma de la dimension "What" (cf chapitre 3), et sa clause d'agrégation repose sur les "hits" de **UnknownHit**. Chacune des colonnes est associée à une expression spécifique qui sera expliquée sous peu. Mais avant, précisons le mécanisme d'agrégation :

Lorsqu'un "hit" arrive dans la "stream" **UnknownHit** il est tout d'abord stocké dans un "memory store" avec une politique de rétention (fenêtre temporelle ou nombre constant d'enregistrements) qui sera fixée lors du déploiement. Puis, un "event" est transmis aux deux "derived stream" : **WhatDimension** et **IdentifiedWhat**. La première va détecter s'il n'existe pas déjà un tuple dans son "store" ayant la même valeur. Si oui, alors ce "hit" a déjà été traité au moins une fois et est déjà présent dans la dimension. Si non, alors il s'agit d'un "hit" qui n'a jamais été rencontré auparavant. Dès lors, un nouveau tuple est crée dans la dimension et chacun de ses attributs est calculé. En parallèle, la "join stream" **IdentifiedWhat** essaye d'identifier dans **WhatDimension**, un enregistrement pouvant correspondre

au "hit" de **UnknownHit**. Si aucun "hit" ne coïncide, la valeur "NULL" est imposée à **IdentifiedWhat.keyWhat**. Or, cette situation peut tout à fait se produire même si elle semble signifier que la dimension "What" est défaillante. Cependant, gardons à l'esprit que la dimension "What" est en alimentation permanente. Si le "hit" en question n'a jamais été rencontré, le temps nécessaire à son traitement et à son apparition définitive dans la dimension sera probablement plus grand que le temps mis pour faire la jointure. Mais, fort heureusement, dès que le nouveau tuple sera crée dans la dimension, la "join stream", va alors mettre à jour la jointure, remplaçant la valeur "NULL" par la clé primaire correspondante de la dimension.

Portons désormais notre attention sur les "expressions", au sens Aleri, associées aux colonnes de **WhatDimension**.Tout d'abord **WhatDimension.hit** est une simple recopie de **UnknownHit.hit**, qui devient, du fait de l'agrégation, une clé primaire. Cependant, nous ne souhaitons pas que ce soit le "hit" lui même qui jour ce rôle (le "hit" serait alors recopié dans chaque fait !) mais un "integer" à part entière. Une manière simple de créer une clé primaire est alors de recopier la clé primaire du log correspondant au premier "hit" rencontré d'un groupe. C'est cette solution qui aura été retenue pour calculer **WhatDimension.keyWhat**. Le traitement requis pour déterminer si le "hit" est de type institutionnel ou personnel est suffisamment léger pour être traité directement par l'API de la platforme :

```
case
when like(UnknownHit.hit,'/~%')=1 then 'Personal'
when like(UnknownHit.hit,'/people/%')=1 then 'Personal'
else 'Institutional'
end
```

Cependant, il n'en est pas de même pour les autres colonnes qui nécessitent un usage conséquent des expressions régulières. L'unique solution est alors d'utiliser des "user defined functions" (cf section 4.3) et de les appeler dans les expressions. Ainsi les expressions des colonnes **WhatDimension.highestLevel**, **WhatDimension.documentType** et **WhatDimension.isPageView** sont respectivement :

```
foreignJava(WhatDimension,processHighestLevel,'(String;)String;',UnknownHit.hit)
foreignJava(WhatDimension,processDocumentType,'(String;)String;',UnknownHit.hit)
foreignJava(WhatDimension,processPageViewDetection,'(String;)String;',UnknownHit.hit)
```

Elles font alors appel, dans une autre JVM, aux fonctions de la classe suivante :

```
import java.util.regex.*;

public class WhatDimension{
        public static String processHighestLevel(String hit){
                Matcher m = Pattern.compile("(^/[^/]*)").matcher(
                    hit);
                if (m.find()){return m.group(1);}
                else {return "Unknown";}
                }

        public static String processDocumentType(String hit){
                Matcher m = Pattern.compile("([.][a-z]{1,4}$)").
                    matcher(hit);
                if (m.find()){return m.group(1);}
                else {return ".html";}
                }

        public static String processPageViewDetection(String hit){
```

```
                    if (Pattern.matches("(^(([^.]*)|(.*[.](html|htm|
                        php|ps|asp|doc|ppt|pdf|xls|txt|c|java|ada)))$)"
                        ,hit)){return "PageView";}
                    else {return "Not PageView";}
                    }
}
```

Encore une fois, nous ne détaillerons pas les expressions régulières employées ici et invitons le lecteur à consulter la documentation de Sun sur la création et l'utilisation de "Patterns" ([12]).

La dimension "How"

La dimension "How" est dynamique dans le sens où les "UserAgentStrings" le sont. En effet, les versions des clients web évoluent sans cesse et il serait intéressant que la dimension "apprenne" d'elle même en s'auto-alimentant. De plus, il est laborieux et difficile, fâce à la multitude de configurations possibles d'écrire et de tenir à jour un fichier XML contenant les tuples de cette dimension. Sa conception réside donc sur l'utilisation d'une "Aggregate stream", comme précédemment, mais introduit un nouveau concept : l'utilisation d'une "flex stream" au lieu d'une "compute stream". Quelle est précisément leur différence ? D'un côté, la "compute stream" définie pour chacune des colonnes du flux de sortie, une expression particulière permettant de la calculer. C'est, par ailleurs, ce que nous avons utilisé jusqu'à présent au sein du modèle. De l'autre côté, une "flex stream" définie une méthode pour chacune de ces "input streams". C'est méthode est implémentée au sein d'un script "SPLASH" dans lequel toutes les colonnes du flux de sortie sont calculées. Le grand avantage de ce type de script est de pouvoir ainsi corréler les calculs entre plusieures colonnes du flux de sortie. Mais prenons, sans plus attendre, l'exemple de cette dimension afin d'illustrer ce concept :

Déterminer le client et sa version exacte nécessite le "parsing" de la "UserAgentString". Ce traitement est notamment confié à une "user defined" fonction JAVA de la classe **HowDimension**. Cependant, une fois le client et sa version détectée, le calcul de la colonne "client" est presque déjà fait ! En effet si "clientVersion" vaut "Firefox/2.0.13" alors "client" est en fait "Firefox". Il n'est alors pas nécessaire de refaire le "parsing" de la "UserAgentString" en entier mais seulement de l'effectuer sur "clientVersion" c'est à dire le **résultat du calcul précédent**. C'est ce concept de "réutilisation" des calculs précédents qui caractérise sans doute le mieux l'intérêt d'une "flex stream". Dans le cas très précis de cette dimension, nous aurions pu recourir à une "compute stream" et faire, à plusieurs reprises, le "parsing" de la "UserAgentString" bien que cette méthode gaspille d'avantage de ressources. Cependant, dans des cas où les calculs des colonnes sont très corrélés entre eux, l'utilisation d'une "flex stream" est alors indispensable.

La figure 7.5, page suivante, met en relief le "dataflow" relatif à l'alimentation de la dimension "How" et à l'identification de l'application cliente.

Nous laissons le soin au lecteur de découvrir la syntaxe de ce premier script "SPLASH", très simple pour cette dimension :

```
{
string clientVersion := foreignJava(HowDimension,processClientVersionIdentification,'(String;)St
string client;
string isBrowser;
string os;

if (like(clientVersion,'Unknown')=1) {client := clientVersion; isBrowser := 'Unknown';}
else {
```

FIG. 7.5 – "Dataflow" de l'alimentation de la dimension "How"

```
client := foreignJava(HowDimension,processClientIdentification,'(String;)String;',clientVersion)
isBrowser := foreignJava(HowDimension,processBrowserDetection,'(String;)String;',clientVersion)
}

os := foreignJava(HowDimension,processOsDetection,'(String;)String;',IncomingLogs.userAgent);

output [ keyLog = IncomingLogs.keyLog; |
         clientVersion = clientVersion;
         client = client;
         isBrowser = isBrowser;
         os = os; ];
}
```

Ci dessous, le lecteur trouvera également l'implémentation de la classe **HowDimension** où sont regroupées les fonctions appelées par le script "SPLASH" précédent :

```
import java.util.regex.*;

public class HowDimension{
        public static String processClientVersionIdentification(
            String hit){
                Matcher m = Pattern.compile("((MSIE\\s[^;]*)|((
                    Firefox|Netscape|Opera|Safari|Konqueror|(Yahoo!
                     Slurp)|msnbot|Googlebot|VoilaBot|(FAST
                    Enterprise Crawler))[^\\s]*))").matcher(hit);
                if (m.find()){return m.group(1);}
```

```
                else {return "Unknown";}
                }

        public static String processClientIdentification(String
            client){
                Matcher m = Pattern.compile("((MSIE|Firefox|
                    Netscape|Opera|Safari|Konqueror|(Yahoo! Slurp)|
                    msnbot|Googlebot|VoilaBot|(FAST Enterprise
                    Crawler)))").matcher(client);
                if (m.find()){return m.group(1);}
                else {return "Unknown";}
                }

        public static String processBrowserDetection(String client
            ){
                if (Pattern.matches("(^(MSIE|Firefox|Netscape|
                    Opera|Safari|Konqueror).*$)",client)){return "
                    Browser";}
                else {
                        if (Pattern.matches("(^((Yahoo! Slurp)|
                            msnbot|Googlebot|VoilaBot|(FAST
                            Enterprise Crawler)).*$)",client)){
                            return "Crawler";}
                        else {return "Unknown";}
                    }
                }
        public static String processOsDetection(String hit){
                if (Pattern.matches("(^.*Windows.*$)",hit)){return
                     "Windows";}
                else {
                        if (Pattern.matches("(^.*Macintosh.*$)",
                            hit)){return "Macintosh";}
                        else {
                                if (Pattern.matches("(^.*X11.*$)",
                                    hit)){return "Linux";}
                                else {return "Unknown";}
                        }
                }
        }
}
```

On peut souligner que seuls les navigateurs Internet Explorer, Firefox, Netscape, Opera, Safari et Konqueror sont pris en compte dans le "parsing". De la même manière, seuls les robots d'indexation de Google, Yahoo, MSN, Voila et "Fast Enterprise" sont identifiés précisément.

La dimension "From"

La conception de cette dimension repose exactement sur les concepts utilisés pour la dimension "How". La figure 7.6 en résume le "dataflow". Le script "SPLASH" associé à la "flex stream" est cependant ici nettement plus élaboré. On peut notamment remarquer l'utilisation d'une boucle *while* afin d'identifier les mots-clés qu'a utilisé l'internaute (pour les moteurs Google, Yahoo, MSN et AOL uniquement). En effet, étant donné que les "user defined" fonctions ne peuvent avoir de types de retour complexes tels que des tableaux ou des collections, le script interroge tout d'abord la classe **FromDimension** via sa méthode *processNumberKey-*

Words pour connaître le nombre de mots-clés, puis boucle pour les récupérer un par un via la méthode *processOneKeyWord*. Nous invitons le lecteur à consulter de nouveau le chapitre 3 afin de comprendre la signification de chacunes des colonnes de la dimension.

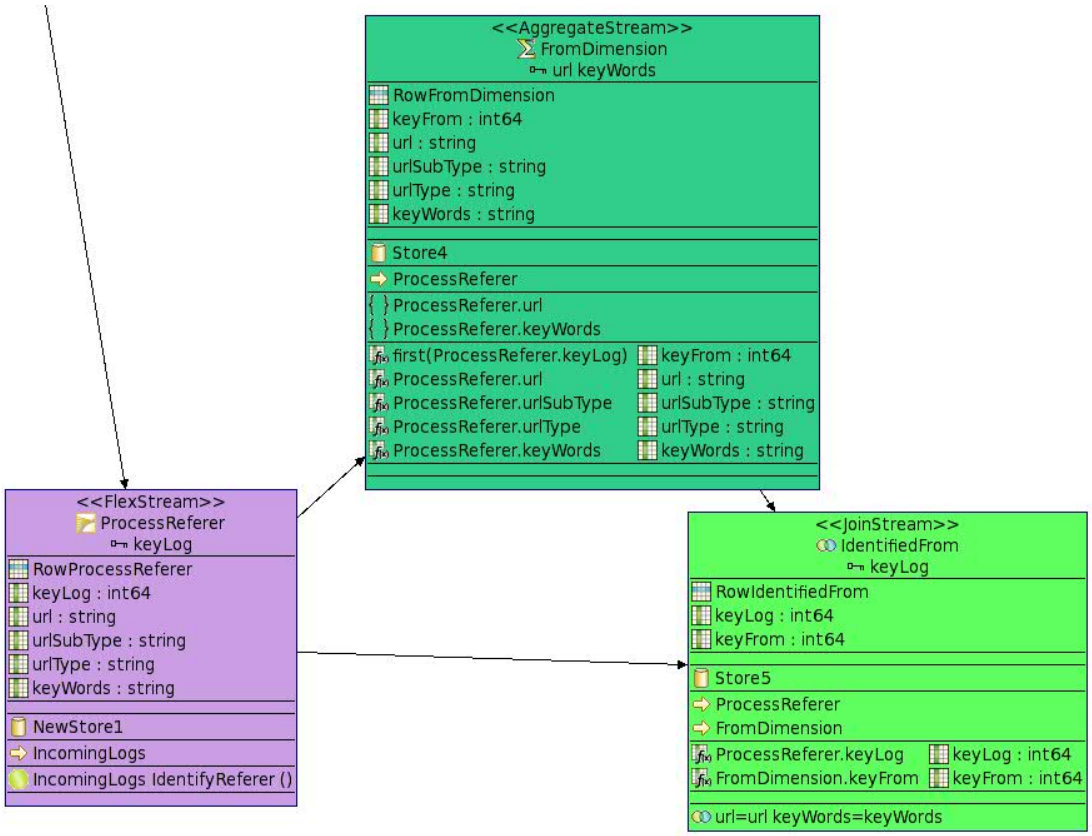

FIG. 7.6 – "Dataflow" de l'alimentation de la dimension "From"

```
{
string url;
string urlSubType;
string urlType;
string keyWords;
string currentKeyWord;
int i;
int numberOfKeyWords;

keyWords:='';

if(like(IncomingLogs.referer,'-')=1)
{
    // The referer is not filled, no use to process it further.
    url := 'Not Filled'; urlSubType := 'Not Filled'; urlType := 'Not Filled';
}
else
{
    // Process the url.
    url := foreignJava(FromDimension,processUrlIdentification,'(String;)String;',IncomingLogs.re
    if (like(url,'Internal')=1)
```

```
    {
        // It's an internal one.
        urlSubType := 'Internal'; urlType := 'Internal';
    }
    else
    {
        // The urls which remain should be external.
        urlType := 'External';
        // Let's determine the urlSubType.
        if (like(url,'%search%')=1)
        {
            // The referer comes from a search engine.
            urlSubType := 'Search Engine';
            // Identify KeyWords from most famous search engines : Google|Yahoo|MSN|AOL
            if (like(url,'Google search')=1 or like(url,'Yahoo search')=1 or like(url,'MSN searc
            {
        // Identify KeyWords.
            numberOfKeyWords:=foreignJava(FromDimension,processNumberKeyWords,'(String;)I',Incon
        i := 1;
        while (i<(numberOfKeyWords+1))
        {
            currentKeyWord:=foreignJava(FromDimension,processOneKeyWord,'(String;I)String;',Inco
            // All the characters of the keyword are in lower case.
            currentKeyWord:=lower(currentKeyWord);
            keyWords:=concat(keyWords,currentKeyWord);
            keyWords:=concat(keyWords,' ');
            i:=i+1;
        }
       // Remove last space.
       keyWords:=rtrim(keyWords);
   }
        }
        else
        {
            if (like(url,'%Mail%')=1)
            {
            // The referer comes from a mail box.
            urlSubType := 'Mail';
            }
            else
            {
                // The referer comes from an other web sites.
            urlSubType := 'Others web sites';
            }
        }
    }
}

output [ keyLog = IncomingLogs.keyLog; |
         url = url;
         urlSubType = urlSubType;
         urlType = urlType;
         keyWords = keyWords;
       ];
```

```
}
```

Ci dessous, le lecteur trouvera également l'implémentation de la classe **FromDimension** où sont regroupées les fonctions appelées par le script "SPLASH" précédent :

```
import java.util.regex.*;

public class FromDimension{

        public static String processUrlIdentification(String
            referer){
                if (Pattern.matches("(^http://(www){0,1}([.]|-)
                    {0,1}(inf|infres|perso|vreng|bd|cartodyn)+[.](
                    enst|telecom-paristech).*)",referer)) return "
                    Internal";
                else if (Pattern.matches("(^http://(www|images)
                    {1}[.](l[.]){0,1}(google){1}.*)",referer))
                    return "Google search";
                else if (Pattern.matches("(.*search.yahoo.*)",
                    referer)) return "Yahoo search" ;
                else if (Pattern.matches("(.*search[.](msn|live)
                    .*)",referer)) return "MSN search" ;
                else if (Pattern.matches("(.*search[.]ke[.]voila
                    .*)",referer)) return "Voila search" ;
                else if (Pattern.matches("(.*aol/(search|
                    imageDetails){1}.*)",referer)) return "AOL
                    search" ;
                else if (Pattern.matches("(.*altavista.*results
                    ?.*)",referer)) return "Altavista search" ;
                else if (Pattern.matches("(^http://([0-9]{1,3}[.])
                    {3}[0-9]{1,3}/search\\?q=cache:.*)",referer))
                    return "Google cached page search";
                else if (Pattern.matches("(.*(webmail|(mail.(live|
                    google|yahoo).com)|squirrelmail).*)",referer))
                    return "Mail" ;
                else if (Pattern.matches("(.*search.*)",referer))
                    return "Others search" ;
                // Other cases : it's an external Url.
                else return referer;
        }

        public static int processNumberKeyWords(String
            search_engine_referer){
                String [] tab= search_engine_referer.split("\\+");
                return tab.length;
        }

        public static String processOneKeyWord(String
            search_engine_referer,int index){
                // "q=" for Google|MSN or "p=" for Yahoo or "query
                    =" for AOL.
                String regexp="(?:.*(?:q|p|query)=([^\\+&]*)";
                int i;
                for (i=0;i<(index-1);i++) { regexp+="(?:\\+)
                    ([^\\+&]*)";}
```

```
                regexp+=")";
                Matcher m = Pattern.compile(regexp).matcher(
                    search_engine_referer);
                if (m.find()){return m.group(index);}
                else {return "";}
        }
}
```

La dimension "Who"

Cette dernière dimension présente une originalité remarquable par rapport aux précédentes : Elle nécessite, pour son alimentation, l'appuie de données tiers. En effet, afin de déterminer le pays associé à une adresse IP ou à un nom d'host, il est bien nécessaire de recourir à une base de données externe. Chacun de ses tuples se présente de la manière suivante :

- Une plage d'adresses IP sous la forme d'une *Ip_from* et d'une *Ip_to* qui sont tous deux des "int64".
- Une codification à deux ou trois lettres par pays : Par exemple, "FR" pour la France.
- Un nom complet de pays associé à cette plage.

Dès lors, deux processus distincts sont possibles pour connaître le pays d'une IP :

Soit elle est résolue, et il suffit de regarder sa codification (*.fr, *.es etc...) et de la chercher dans la précédente table. Soit elle est non résolue (de type A.B.C.D) et il faut la convertir sous la forme d'un "int64" selon la formule suivante :

```
IP Number = A x (256*256*256) + B x (256*256) + C x 256 + D
```

Une fois cette conversion réalisée, il suffit de chercher le tuple correspondant dans la table pour en déterminer le pays. Notons, cependant, que lorsque le nom d'host fini par une codification réservée (*.net, *.com, *.edu etc...) il est alors impossible de retrouver le pays avec cette méthode.

En terme de "streams", c'est une "source stream" **IpToCountry** qui accueille les données tiers qui sont "uploader" à l'initialisation du modèle. Cette dernière, ainsi que **IncomingLogs**, sont alors les entrées de la "flex stream" **ProcessIp** qui assure la détermination du pays selon le processus explicité ci dessus. Puis, à l'instar des précédentes dimensions, une "aggregate stream" et une "join stream" sont utilisées. La figure 7.7 met en exergue ce "dataflow".

Si l'on décortique un petit peu le script "SPLASH" de la "flex stream" **ProcessIp**, on s'aperçoit qu'il commence à déterminer si l'IP a été résolue ou non par le serveur Apache, et ce grâce à l'appel de la fonction Java : *WhoDimension.processIpIdentification*. Notons, quelque soit le cas, l'usage novateur, par rapport aux précédents scripts, d'itérations afin de parcourir les enregistrements de **IpToCountry**. On peut remarquer qu'une fois encore l'usage d'un script "SPLASH" est particulièrement adapté puisque les calculs des champs des "output records" sont fortement corrélés entre eux.

```
{
string host; string country;string isIp; int64 ipNumber; string domaineCode;

string isIp := foreignJava(WhoDimension,processIpIdentification,'(String;)String;',IncomingLogs
if (like(isIp,'Ip')=1)
{
// It's a non resolved IP.
host := 'Not resolved';
    // Compute the IP number.
```

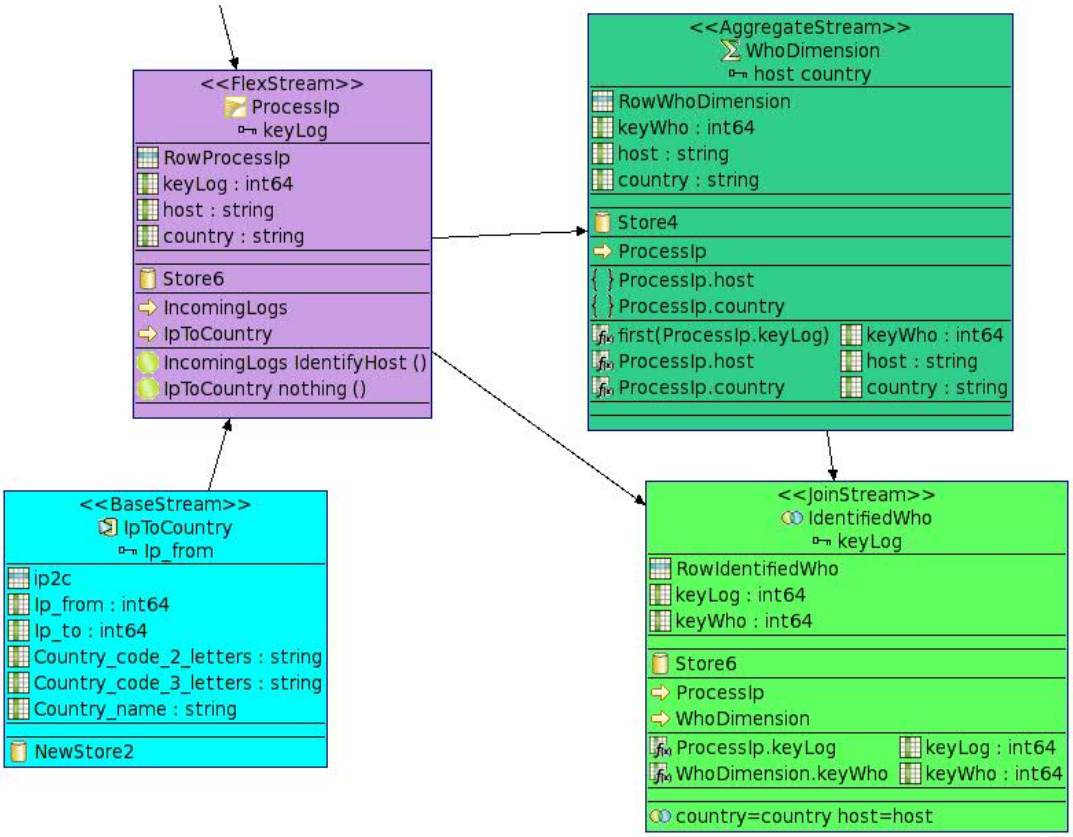

FIG. 7.7 – "Dataflow" de l'alimentation de la dimension "Who"

```
    ipNumber := foreignJava(WhoDimension,computeIpNumber,'(String;)J',IncomingLogs.ip);

    // Find in the IpToCountryStream the country.
    for (record in IpToCountry_stream){
        if ((ipNumber >= record.Ip_from) and (ipNumber <= record.Ip_to))
        {
            country:= record.Country_name;
            break;
        }
    }
    if (isnull(country)) {country:='UNKNOWN';}
}
else
{
    // The hostName has been resolved by the Apache server.
host := IncomingLogs.ip;
// Get the domaine code of this host name.
    domaineCode:= foreignJava(WhoDimension,processDomaineCode,'(String;)String;',IncomingLogs.ip
    domaineCode:= upper(domaineCode);
    // Find in the IpToCountryStream the country.
    for (record in IpToCountry_stream where Country_code_2_letters=domaineCode ){
            country:= record.Country_name;
            break;
        }
    if (isnull(country)) {country:='UNKNOWN';}
```

```
}

output [ keyLog = IncomingLogs.keyLog; |
         host = host;
         country = country;
       ];

}
```

Ci dessous, le lecteur trouvera l'implémentation de la classe **WhoDimension** où sont regroupées les fonctions appelées par le script "SPLASH" précédent :

```
import java.util.regex.*;

public class WhoDimension{

        public static String processIpIdentification(String ip){
                if (Pattern.matches("(^([0-9]{1,3}[.])
                    {3,3}[0-9]{1,3}$)",ip)){return "Ip";}
                else {return "Host";}
        }

        public static long computeIpNumber(String ip){
                long ipNumber;
                Matcher m = Pattern.compile("(^([0-9]{1,3})
                    [.]([0-9]{1,3})[.]([0-9]{1,3})[.]([0-9]{1,3})$)
                    ").matcher(ip);
                if (m.find()){
                ipNumber=Integer.parseInt(m.group(2))*(long)
                    16777216+Integer.parseInt(m.group(3))*65536+
                    Integer.parseInt(m.group(4))*256+Integer.
                    parseInt(m.group(5));
                return ipNumber;
                }
                else return 0;
        }

        public static String processDomaineCode(String host){
                Matcher m = Pattern.compile("(^.*[.]([a-z]*)$)").
                    matcher(host);
                if (m.find()){
                return m.group(2);
                }
                else return "Unknown";
        }
}
```

7.2 Élaboration de la table des faits

Un schéma valant mieux qu'un long discours, cela doit rester valable avec trois! Ainsi, Les figures 7.8, 7.9 et 7.10 résument comment les traitements spécifiques à chacune des dimensions se regroupent dans la "join stream" **IdentifiedLog**. On comprend ici pourquoi il était nécessaire de conserver la clé primaire du log dans chaque sous "dataflow". Enfin, les

logs sont agrégés dans la table des faits **AggregatedLogs**, mettant alors à jours les deux mesures du groupe dans lesquels ils sont insérés : Le nombre de logs agrégés dans le groupe ainsi que la quantité totale d'octets transférés.

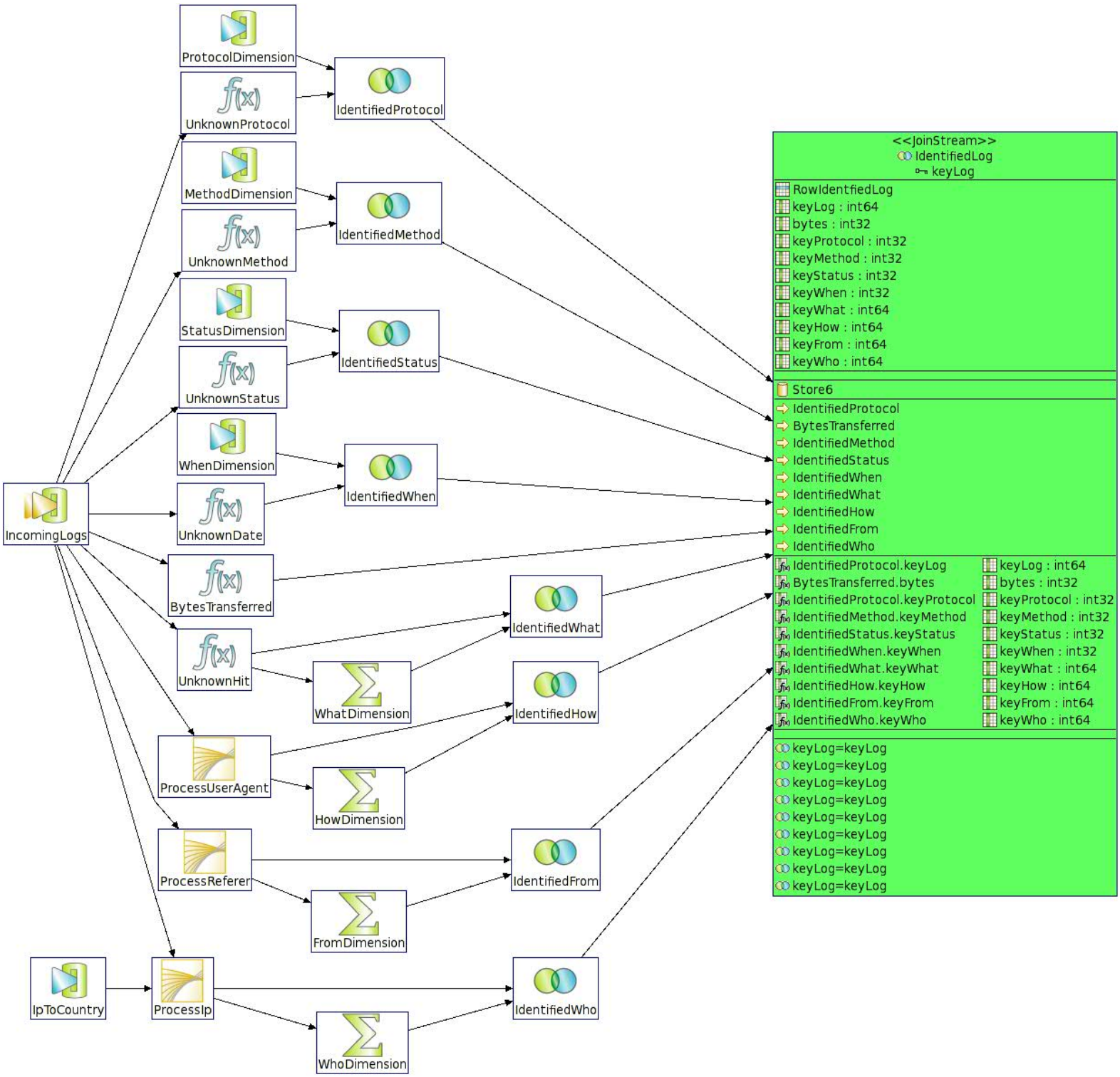

FIG. 7.8 – Zoom sur la jointure des différents traitements effectués en amont

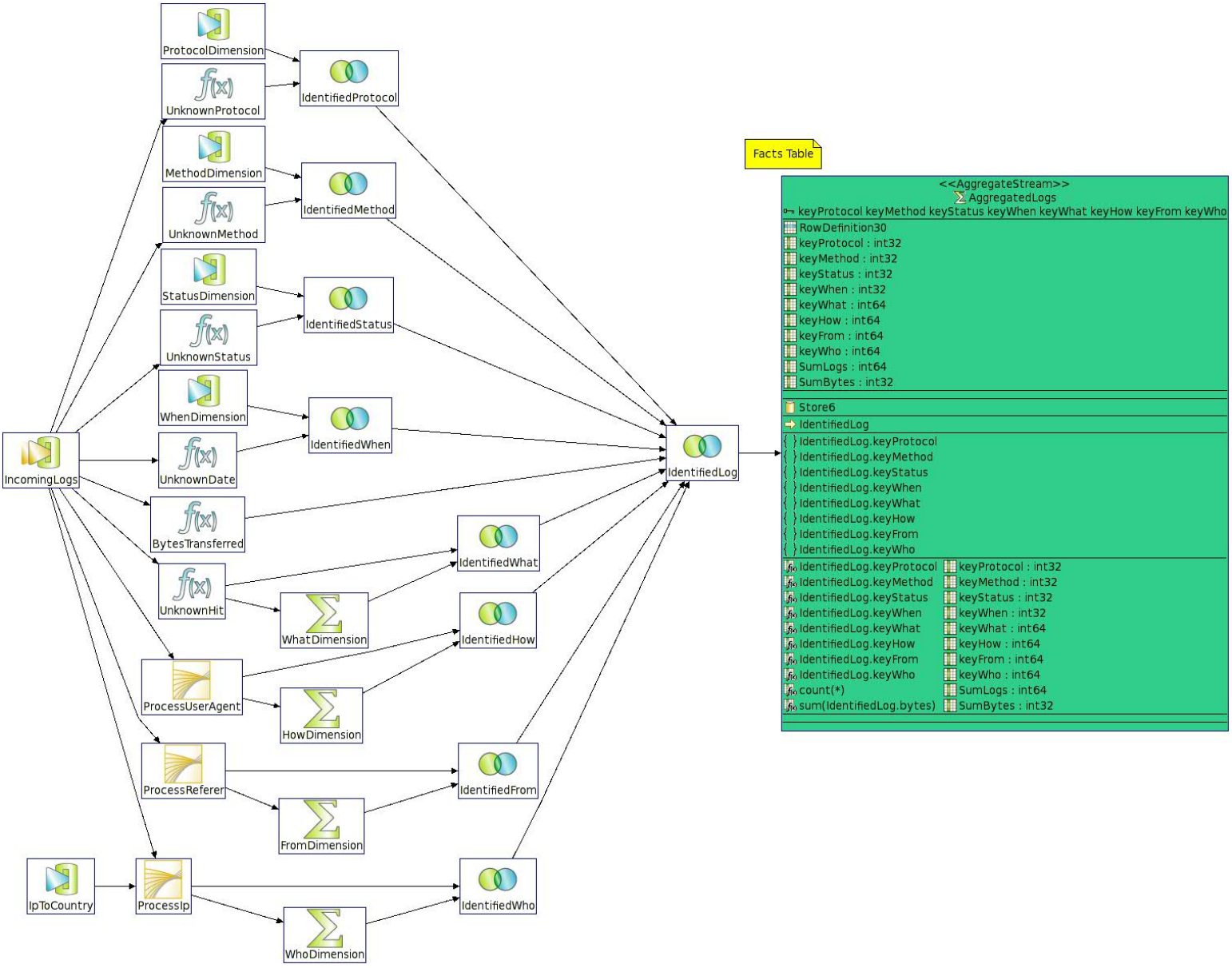

FIG. 7.9 – Zoom sur la table des faits : **AggregatedLogs**

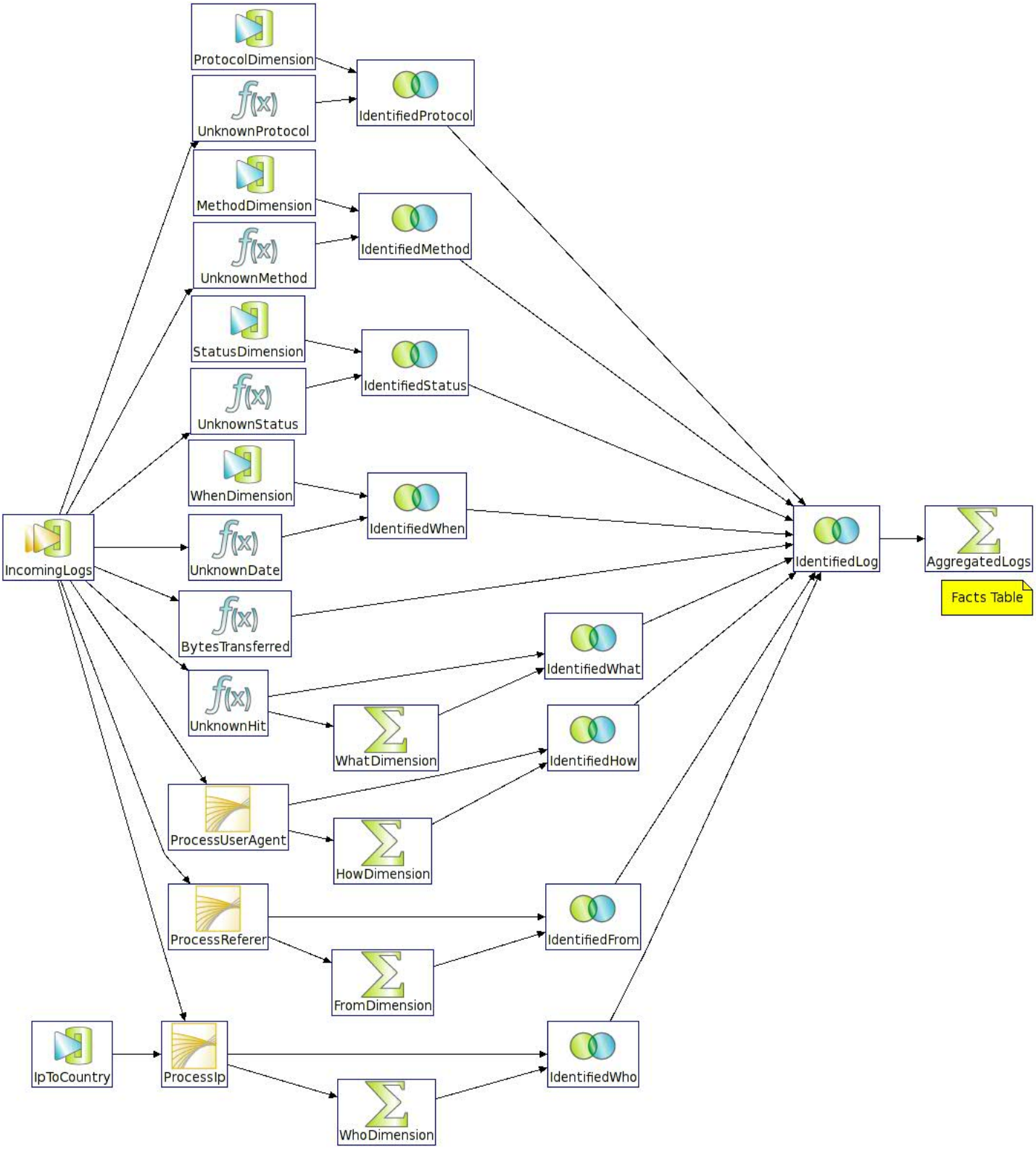

FIG. 7.10 – "Dataflow" complet jusqu'à l'alimentation de la table des faits

Chapitre 8

La souscription au composant "Aleri Live OLAP"

Lors de l'écriture de ce rapport, aucune documentation pour ce composant n'a malheureusement pue être arrachée des mains de la société Aleri. Il semblerait que ce composant ne soit encore qu'au stade d'une version beta...

Quatrième partie

Bilan et perspectives

La construction et l'alimentation temps réel d'un cube de données, alimenté par le flux des logs d'accès au site web www.infres.enst.fr, fût un projet ambitieux et ce envers de nombreux aspects :

Une première problématique rencontrée fût tout d'abord la conception d'un schéma pour le cube OLAP, schéma se devant d'être à la fois pertinent en terme de web analyse, mais aussi raisonnable en terme de volumétrie. Une fois cet objectif dual à l'esprit, la complexité des données rencontrées, couplée à leur forte hétérogénéité s'est rapidement avérée être une réelle difficulté. En effet, les "access logs" sont intrinsèquement "confus" du fait de la conjonction de multiples facteurs : la politique historique des navigateurs, le manque de standardisation, la multiplicité des acteurs (navigateurs, terminaux mobiles, moteurs de recherches, robots d'indexation), la naissance encore récente de la web analyse, etc...

De fait, à partir d'un échantillon s'étalant sur une fenêtre temporelle de deux semaines, une analyse approfondie des données a été effectuée afin d'identifier les informations pertinentes et modéliser l'évolution de leur hétérogénéité. L'ensemble de cette étude a notamment été présentée dans la partie I et a permis d'aboutir au schéma en flocon d'un cube OLAP à huit dimensions.

Dès lors, une nouvelle problématique fût d'implémenter ce dernier en s'appuyant sur les fonctionnalités offertes par la récente "Aleri Streaming Platform" et son composant "Aleri Live OLAP". Un tableau présentant l'ensemble des concepts introduits pas ce DSMS a été brossé dans la partie II : "Complex event processing", modèles de données, "stores", expressions et scripts "SPLASH", interfaces et mécanisme de publication, etc...

Un prototype fût alors présenté dans la partie III du présent document. La méthodologie employée a permis d'isoler chronologiquement le travail en trois phases : Dans un premier temps, la publication des données au sein de la platforme, puis, dans un second, la construction du modèle de données en lui même, et enfin la souscription à ce dernier via le composant "Aleri Live OLAP".

Une fois la publication prise en charge par un adaptateur Java, robuste et modulaire, les dimension du cube furent construites, au sein du modèle, par ordre de difficulté : des dimensions statiques à volumétrie faible ("Protocol", "Method", "Status") aux dimensions fortement dynamiques, nécessitant , quant à elles, une alimentation temps réel ("What","From"), voire même l'appuie de données tiers (telle que la dimension "Who"). Ainsi, ce projet a permis de balayer l'ensemble des fonctionnalités de la platforme et constitue, en tant que tel, une réelle illustration de ce qu'il est possible de faire avec ce DSMS. Soulignons, par ailleurs, que le déploiement de ce projet nécessiterait une analyse de performance ainsi qu'une étude précise des politiques de rétention associées aux nombreux "stores" du modèle.

Finalement, on ne peut que regretter amèrement le fait que la société Aleri n'a pas laissé évaluer son composant "Aleri Live OLAP". Pire, pas une seule documentation n'a été fournie sur ce composant, et ce malgré les requêtes répétées auprès de cette entreprise. Le manque d'outil de souscription laisse donc ce cube OLAP sans réelles possibilités d'exploration et de "reporting". Il s'agit d'une déception d'autant plus grande que ce cube semblait présenter, que ce soit pour le site particulier www.infres.enst.fr ou pour un quelconque autre site au prix de quelques modifications, des possibilités de web analyse relativement étendues au regard notamment de ce que peuvent offrir certains outils professionnels.

Cinquième partie

Annexes

Annexe A

Synthèse de l'échantillon

Days	Hits/day	Distinct Hits **with** arguments	Distinct Hits **without** arguments	Distinct User-Agent	Distinct IP resolved	Distinct IP not resolved
1	59470	20814	15217	1034	3120	839
2	93286	57991	21973	1698	5424	1539
3	113737	80494	29533	2534	8441	2742
4	119139	93199	37862	3237	11187	3859
5	77722	98532	42002	3882	13760	4968
6	72633	102778	45292	4416	17121	5946
7	72973	108755	49822	4858	19358	6766
8	57796	118326	55075	5186	21062	7350
9	121554	146442	62063	5501	22946	7918
10	124694	181176	66466	5964	25331	8814
11	116031	189709	70259	6476	27838	9657
12	101143	195455	74885	6937	30249	10504
13	103941	210400	82857	7362	32494	11366
14	69559	215229	86398	7718	34458	12083
15	49745	218686	89224	7984	36107	12643
16	51137	220485	90511	8253	37995	13233
17	46014	224076	91321	8466	39162	13736
Total :	1450574					

FIG. A.1 – Synthèse de l'analyse des données du 01-03-08 au 17-03-08

Referer Repartition	Count		Percentage (%)	
	hits	pages views	hits	pages views
Not filled	488381	260076	**33,67**	**46,67**
Internal	736829	137660	**50,8**	**24,7**
Mail Box	1589	1147	0,11	0,21
Search Engine + Cached pages	79889	63357	5,51	11,37
Others URLs	143886	95082	9,92	17,06
External	225364	159586	**15,54**	**28,63**
Total :	1450574	557322	100	100

FIG. A.2 – Répartition des "referers" ("hit" et "page view") du 01-03-08 au 17-03-08

Day (March 08)	Internal URLs	Mails	Search Engines	Others External URLs	All
1	1904	17	1669	379	3969
2	2850	24	3525	573	6972
3	4333	32	6595	832	11792
4	7440	49	9629	1054	18172
5	8261	62	12555	1214	22092
6	8923	124	15293	1375	25715
7	9471	166	17792	1568	28997
8	9736	172	19263	1659	30830
9	10174	181	20857	1765	32977
10	10754	207	23637	1919	36517
11	11396	219	26457	2094	40166
12	12029	241	29189	2236	43695
13	12612	246	31922	2355	47135
14	13244	258	34176	2356	50124
15	13461	261	35553	2617	51892
16	13746	264	37212	2707	53929
17	13979	282	38815	2805	55881

FIG. A.3 – Evolution de l'hétérogénéité des "referers" par type du 01-03-08 au 17-03-08

Annexe B

Dimension When : Alimentation

Voici le script Ruby permettant l'alimentation de la dimension "When". Deux paramètres sont à configurer : D'une part le fichier ".xml" où seront enregistrés les tuples et, d'autre part, les mois qui sont précisément à générer. Chaque mois est décrit par son nom, son nombre de jour, son premier jour ainsi que son année. L'ensemble des mois à générer est stocké dans un tableau associatif. Ci dessous, le script génère ainsi la dimension "When" pour le mois de "Mai 2008" qui a 31 jours et qui commence par un jeudi" :

```
# Insert here the months you want to generate
$monthsToBeGenerated=[{:name=>"May" ,:days=>31,:firstDay=>"Thursday
    " ,:year=>"2008" }]
# Insert here the file where the records will be generated.
$output=File.open("when.xml","w")

# Global variables : Shall not be changed
$monthNumberToMonthName={"January"=>"01",
                         "February"=>"02",
                         "March"=>"03",
                         "April"=>"04",
                         "May"=>"05",
                         "June"=>"06",
                         "July"=>"07",
                         "August"=>"08",
                         "September"=>"09",
                         "October"=>"10",
                         "November"=>"11",
                         "December"=>"12"}

$days=["Monday","Tuesday","Wednesday","Thursday","Friday","
    Saturday","Sunday"]
$primary_key=0

# Attribute a timeSlot to a given hour.
def hourToTimeSlot(hour)
        if hour < 9 then return "Night"
        elsif hour < 12 then return "Office Morning"
        elsif hour < 14 then return "Lunch"
        elsif hour < 19 then return "Office Afternoon"
        elsif hour < 23 then return "Evening"
        elsif  hour < 24 then return "Night"
        end
```

```
end

# For each month to be generated.
$monthsToBeGenerated.each do |month|
        # For each day in this month.
        1.upto(month[:days]) do |dayNumber|
                # Update currentDayName.
                currentDayName= $days[($days.index(month[:firstDay
                    ])+dayNumber-1)%7]
                # For each hour.
                24.times do |hour|
                        $primary_key+=1
                        # Add the dimension name.
                        currentRow='<WhenDimension ALERI_OPS="i" '
                        # Add the primary key.
                        currentRow << "keyWhen=\"#{$primary_key}\"
                            "
                        # Add the hourDayMonthYear field; ex:
                            "2008-05-10T15:00:00"
                        currentRow<<"hourDayMonthYear=\"#{month[:
                            year]}-#{$monthNumberToMonthName[month
                            [:name]]}-#{dayNumber}T#{hour}:00:00\"
                            "
                        # Add the dayMonthYear field; ex :
                            "2008-05-10T00:00:00"
                        currentRow<<"dayMonthYear=\"#{month[:year
                            ]}-#{$monthNumberToMonthName[month[:
                            name]]}-#{dayNumber}T00:00:00\" "
                        # Add the monthYear field; ex : "may 2008"
                        currentRow<<"monthYear=\"#{month[:name]}
                            #{month[:year]}\" "
                        # Add the hour field.
                        currentRow<<"hour=\"#{hour}\" "
                        # Add the timeSlot field.
                        currentRow<<"timeSlot=\"#{hourToTimeSlot(
                            hour)}\" "
                        # Add the day.
                        currentRow<<"day=\"#{currentDayName}\" "
                        # Add the month.
                        currentRow<<"month=\"#{month[:name]}\" "
                        # Add the year.
                        currentRow<<"year=\"#{month[:year]}\" "
                        currentRow << "/>"
                        $output.puts currentRow
                end
        end
end
```

Table des figures

Bibliographie

[1] Le data warehouse - Kimball, Reeves, Ross, Thornthwaite - ISBN 2-212-11600-4 - Groupe Eyrolles : Partie 2 sur la modélisation dimensionnelle.

[2] http ://httpd.apache.org/docs/1.3/logs.html : Documentation sur les "Combined Log Format" au sein du serveur Apache.

[3] http ://en.wikipedia.org/wiki/Web_analytics : Généralités sur la web analyse. Les liens sur les "referers" et sur les "User Agents" offrent également des éléments solides de compréhension.

[4] http ://www.antezeta.com/search-engine-cache.html : Article interessant sur les pages cachées de Google et leur expression dans les "referers" en tant qu'URL.

[5] http ://www.useragentstring.com/pages/useragentstring.php : Site qui essaye de regrouper tous les "UserAgentsString" mais qui ne propose pas une base données téléchargeable.

[6] http ://ip-to-country.webhosting.info/node/view/6 : Une base de données "csv" gratuite et régulièrement mise à jour proposant une correspondance "Plage d'addresses IP - Pays" (login requis).

[7] AleriTutorial.pdf - Aleri : Prise en main de la "Aleri Streaming Platform v2.4" .

[8] AuthoringGuide.pdf - Aleri : Documentation sur la création de "Data models" : "Data streams", "Stores", le langage "SPLASH", etc...

[9] InterfacesGuide.pdf - Aleri : Comment publier des données au sein de la "Aleri Streaming Platform v2.4", partie sur l'API Java.

[10] UtilitiesGuide.pdf - Aleri : Documentation sur tous les utilaires Aleri : "sp_server", "sp_query", "sp_upload", "sp_convert" etc...

[11] http ://java.sun.com/j2se/1.5.0/docs/api/ : Documentation de l'API Java 1.5.

[12] http ://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html : Les expressions régulières en Java.

Dépôt légal : 2008 – 4ème trimestre
Imprimé à l'Ecole Nationale Supérieure des Télécommunications – Paris
ISSN 0751-1345 ENST D (Paris) (France 1983-9999)

TELECOM ParisTech

Institut TELECOM - membre de ParisTech

46, rue Barrault - 75634 Paris Cedex 13 - Tél. + 33 (0)1 45 81 77 77 - www.telecom-paristech.frfr

Département INFRES