RAPPORT DE STAGE

Développement d'un outil d'exploration des séries temporelles provenant de différentes sources d'objets connectés

GANZA Mykhailo

Septembre 2016

Tuteur en entreprise : Monsieur François NAÇABAL
Tuteur académique : Madame Christelle LECOMTE

Entreprise d'accueil : Maya Technologies, Grenoble
Établissement/Formation : Université de Rouen / Master 2 GEII

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué à la réussite de mon stage et qui m'ont supporté dans le travail.

Tout d'abord, je veux remercier mon maître de stage, François Naçabal, chef de projet au sein de l'entreprise Maya Technologies, pour son accueil et le temps passé ensemble. Grâce à sa confiance en mes compétences et la liberté offerte pour résoudre une problématique posée, j'ai pu acquérir des nouvelles aptitudes. Son intelligence nous a permis de nous poser de bonnes questions, pour trouver des solutions correctes, afin d’accomplir totalement la mission.

Je tiens à remercier mon professeur Mme Christelle Lecomte qui était de grand support pour toutes les étudiants en recherche de stage. Ces conseils m'ont permit de trouver ce stage qui est en totale adéquation avec mes attentes.

Je remercie également toute l'équipe des ingénieurs de l'open space de Maya Technologies pour leur accueil, leur esprit d'équipe et l'ambiance amicale.

Enfin, je tiens à remercier toutes les personnes qui m'ont aidé avec la correction de ce rapport de stage : mon amie Sara et ma famille.

Introduction

Une entreprise américaine de conseil et de recherche dans le domaine technique, Gartner, préviens qu'en 2020 le nombre de objets connectés dans le monde va atteindre 20.8 milliards[1]. Sachant que le chiffre estimée par la même entreprise pour l'année de 2016 et de 6.4, cela montre la grande vitesse à laquelle ce domaine se développe et quelles sont ses perspectives pour le futur.

Dans le cadre de mon Master 2 GEII à l'Université de Rouen, j'ai souhaité réaliser mon stage dans une entreprise qui se spécialise dans le domaine des objets connectés. Ce secteur à été choisi car il est en forte demande actuellement et le sera longtemps car de plus en plus d'objets « smart » vont s'introduire dans la vie des gens. Une autre raison est technique : les compétences d'un ingénieur de ce domaine doivent être polyvalentes, un esprit ouvert pour les nouveaux technologies est obligatoire. C'est cette aptitude, d'être à jour de plusieurs technologies, que je vise à développer.

Une mission proposée par Maya Technologies, m'a intéressée parce que je voulais acquérir de l'expérience dans le domaine de l'IoT (*Internet-of-Things*, objets connectés) et accroître mes compétences de développeur. Le sujet de travail proposé consiste à concevoir une solution logicielle capable de lire et analyser les données provenant de différentes types des objets connectés. La problématique principale rencontrée est l'hétérogénéité des objets connectés actuels, et de leurs données.

En première partie nous décrirons l'entreprise et son produit MayaNet, qui sert de cadre à ce stage. Ensuite, nous étudierons le contexte académique et les recherches réalisées. A la fin de rapport nous présenterons le travail réalisé, avant de dresser un bilan de celui-ci.

[1] http://www.gartner.com/newsroom/id/3165317

Table des matières

Partie 1 : Présentation

1 Présentation de l'entreprise

Maya Technologies est une société de services en ingénierie qui se concentre sur la « conception de systèmes microélectroniques embarquées »[2]. Ses ingénieurs sont compétents en matière de systèmes embarquées et sont disponibles aussi bien pour le développement logiciel que pour la conception matérielle en électronique et microélectronique. L'entreprise intervient notamment dans les secteurs de l'aéronautique, du multimédia, de la téléphonie, de l'énergie et de la santé.

Fondée en 2007 par Philippe Mattia à Grenoble (France), Maya Technologies compte 130 personnes (2013) et a des centres à Paris, Toulouse, Grenoble et Valence. Depuis le 2015, la recherche et le développement se concentrent sur le marché de l'IoT, qui n'est autre qu'une évolution logique des systèmes embarquées, désormais connectés. Une des solutions pour l'IoT proposées par Maya Technologies est le *middleware* MayaNet.

MayaNet est « un middleware conçu pour les besoins de collecte et de traitement de données provenant de tout type d'objets connectées »[3]. C'est une solution logicielle de gestion de flux de données sur un réseau filaire (ou sans fil) des systèmes embarquées pour la collecte et traitement des données. Les protocoles de connexion supportées sont par exemple GSM, Wi-Fi, Bluetooth, KNX, USB, etc.

Une évolution éventuelle de MayaNet et le passage en version 3. Un des axes d'amélioration visée est le support natif de la gestion de données provenant de différentes sources (différents environnements de prise des mesures).

2 Présentation de contexte de stage

2.1 IoT

IoT, *Internet of Things* (internet des objets, les objets connectés), est une extension d'internet classique (« un réseau des réseaux ») vers le monde physique grâce aux systèmes embarqués. Généralement, ces systèmes réalisent l'acquisition des données de leur environnement qui sont ensuite partagées via le ou les réseaux. Un exemple d'objet connecté peut être un moniteur de fréquence cardiaque, ou un capteur de luminosité, smartphone, etc. Tout équipement électronique ayant la possibilité d'échanger des données via un réseau, peut être considéré comme un objet connecté.

Le concept de connexion à internet des équipements électroniques pour fournir certains données est apparu pour la première fois à 1982 à l'Univeristé Carnegie-Mellon (Etats-Unis). Il s'agit d'un bricolage de distributeur de Coca Cola pour sonder à distance (via un réseau APRANET) la présence (ou absence) et la température des bouteilles dans le distributeur[4]. Ensuite, depuis les

[2] http://www.maya-technologies.com/historique-chiffres-cles
[3] http://www.maya-technologies.com/actus/maya-annonce-la-version-2-0-de-son-middleware-iot-mayanet
[4] https://www.cs.cmu.edu/~coke/history_long.txt

années 1999 le domaine des objets connectées à commencé à prendre de l'ampleur avec la présentation par Bill Joy d'un protocole de communication D2D (*Device to Device*) : un réseau de capteurs qui vont « fusionner les systèmes embarqués avec la vie quotidienne »[5]. Durant la même année, le terme « Internet of Things » a été inventé par Kevin Ashton[6]. Un autre éventement important dans l'histoire d'IoT est la création en 2005 d'une carte Arduino, qui a eu un grand d'impact sur les objets connectés[7], surtout pour les amateurs de domotique. Depuis les années 2013 le marche d'IoT est en pleine expansion avec la croissance des smartphones, montres connectées, capteurs cardiaques et différentes sortes des objets « smart » ayant une connexion internet. Et de plus en plus d'acteurs d'industrie numérique (Google, Samsung, etc) participent pour apporter des innovations et gagner un part de marché de ce secteur.

Les objets connectées ont pénétré différents domaines remplissant diverses tâches. Les plus notables sont la médecine (« santé connectée ») où les objets connectés sont utilisés pour le monitoring en temps réel des indicateurs de santé d'un patient. Un autre domaine important d'IoT est la domotique, avec toute une panoplie de capteurs de luminosité, vidéosurveillance, réfrigérateurs connectées, etc. Un autre exemple d'application des objets connectés sont les capteurs environnementaux, qui observent l'état de certains grandeurs physiques (qualité de l'eau, indices de séisme) pour un but prévention et de protection de l’environnement

Vue le nombre des applications possibles et la quantité des objets connectées actuel, une des évolutions logiques est la standardisation d'IoT. Sachant que le principe même de fonctionnement d'internet est basé sur l'utilisation des mêmes protocoles standards par toutes les équipements, les objets connectes pour être considérés comme des vrais objets « things » d'internet, doivent être standardisés, et être capables d’interagir entre eux. Actuellement la communication avec les objets connectées est basée sur les protocoles définis et standards comme TCP/IP, HTTP, etc. Mais au dessous des protocoles connus, dans une couche des données métier, le protocole de communication entre les objets connectés n'est pas du tout homogène d'une solution à une autre. Par exemple, la nature des données échangées par un réseau de capteurs de luminosité et une infrastructure des réfrigérateurs connectés est complètement différente. Il y a un besoin de langage standard pour que tous les objets connectes, quel que soit leur fournisseur, puissent communiquer entre eux.

Un des avancement dans cette direction est le système EPC[8], qui en perspective pourra être utilisé comme une base commune pour développer un tel protocole universel. Le langage de programmation JavaScript est vue[9] aussi comme un langage possible des toutes les IoT car c'est un langage qui évolue très rapidement et se trouve partout sur internet actuellement : cote client, serveur, et s'applique également aux systèmes embarqués[10].

5 https://www.technologyreview.com/s/404694/etc-bill-joys-six-webs/
6 http://www.rfidjournal.com/articles/view?4986
7 http://www.forbes.com/sites/gilpress/2014/06/18/a-very-short-history-of-the-internet-of-things/3/#540eb98143c5
8 https://fr.wikipedia.org/wiki/Code_produit_%C3%A9lectronique
9 https://blog.jscrambler.com/javascript-the-perfect-language-for-the-internet-of-things-iot/
10 https://cylonjs.com/

Il est important de préciser que l'IoT ne correspond pas à une seule technologie, il s'agit plutôt d'un systèmes des systèmes, une infrastructure à plusieurs tiers. Et comme les objets connectés sont en plein développement, la panoplie des secteurs pour R&D dans l'IoT est très vaste : développement des microprocesseurs et cartes spécifiques (ex : Intel Edison), différentes sortes des capteurs miniaturisées (ex : accéléromètre, thermomètre etc), solutions *middleware*, sécurité des données, etc. Un autre aspect important de l'IoT est l'analyse des données acquises.

2.2 Analyse des données

L'aspect analyse données dans l'IoT prend de plus en plus d'ampleur. Les informations acquises ont beaucoup de valeur, car elles peuvent être analysées afin d'en tirer des indices importants pour la prise de décisions stratégiques. Par exemple, clôturer l'utilisation de certains cellules dans un réseau de capteurs de luminosité à cause de la puissance négligente mesurée. Dans ce cas, il est possible d'analyser les données manuellement : il suffit juste d'analyser toutes les relevés. Aussi, un module d'analyse des données peut faire une partie d'un produit final. Par exemple la détection de chute de rythme cardiaque mesurée par la montre connectée d'un patient.

Si nous voulons concevoir un outil générique (pour plusieurs types des objets connectés) d'analyse des données pour l'IoT on sera en face de trois grands problèmes: volume, approche, et l'hétérogénéité des types de stockage et formats. Vue la quantité d'objets connectés et haute fréquence des mesures effectuées, les données acquises peuvent être assez volumineuses. De ce fait, l'IoT se pose fortement sur les pratiques de *Big Data*[11] pour la gestion et le traitement des énormes masses des données. Un autre point problématique est le choix de l'approche. C'est une question de recherche d'un algorithme le plus adapté pour tirer les conclusions nécessaires à partir d'un ensemble des données acquises. Il s'agit d'un domaine des statistiques, l'utilisation des classificateurs, recherche des tendances, étude de similitude, etc. Ces techniques sont très souvent utilisé dans l'IoT pour l'exploration de données.

La dernière problématique, dans le contexte d'un outil générique d'analyse des données, est la diversité des formats et les façons de stocker les données. Comme l'on a vu dans le chapitre précédent, le marché des objets connectées actuel se repose sur les protocoles commun de communication, mais pas jusqu'au point d'utiliser un format des données standard, c'est-à-dire identique pour tous les types d'objets connectés.

Cet hétérogénéité se complexifie encore plus par l'utilisation de différents outils de stockage : souvent les données sont stockées dans les bases des données relationnelles (comme MySQL, Oracle), ou les bases des données orientées document (MongoDB, Couchbase), qui sont de plus en plus utilisées vu leur structure de données flexible. Rien n'empêche d'utiliser les fichiers texte (JSON, CSV, ARFF), ou d'autres formats rares ou exotiques (photos, fichiers binaires, etc).

La question de multitude des formats et types de stockage est importante à résoudre, si nous voulons concevoir un outil générique d'analyse

[11] http://www.sas.com/en_ph/insights/articles/big-data/big-data-and-iot-two-sides-of-the-same-coin.html

des données, car cet outil doit pouvoir accéder à plusieurs types de stockage et lire les données de structure variable.

Partie 2 : Étude de problématique

Dans cet partie on va citer l'objectif initial et générique de stage, et expliquer les choix prises pour définir un cahier des charges spécifique.

1 Objectif initial

Dès le départ, l'objectif visé du stage est le suivant :

« Concevoir un outil de l'exploration des données et les relations entre les données provenant de sources différentes d'objets connectés»

Il s'agit d'une application qui sera capable de faciliter l'exploration des données et les relations entre les données qui ont été générées par des sources différents. Par exemple, comparer l'évolution d'une mesurée horaire de la température dans une pièce avec les relevés de la température venant d'un site météorologique.

Le terme « sources différentes » corresponds à le fait que les données peuvent être générés par différentes types des objets connectés, qui se trouvent dans les environnements différentes. Mais comme il s'agit de développer un logiciel, on emploi un terme « source » pour dire source des données. C'est qui correspond à le type de stockage de données : bases des données, fichiers, etc. Donc, comme illustré dans une figure 1, notre application se situe entre les données déjà acquises et l'utilisateur.

La problématique de « sources différentes » peut se conclure à trois points principaux à gérer si nous voulons concevoir un outil d'exploration générique. Il s'agit tout d'abord de désynchronisation des instants de prise de mesures entre plusieurs sources, les types des stockages variées et les formats des données différents. L'application doit pouvoir gérer cette nature hétérogène des données de l'IoT.

Les concepts « exploration des données» et « relation entre les données» sont à définir par le stagiaire. Une seule contrainte à suivre est que l'outil doit pouvoir travailler avec au moins deux types des stockages : les bases de données MongoDB et MySQL. Bien entendu, le format et la structure des données ne sont pas imposées. Aucune autre contrainte n'est présentée.

Cela revient au stagiaire de choisir les technologies les plus adaptées pour résoudre la problématique et définir le cahier de charges de l’application. Vue la nature générique de l'objectif, le stagiaire est amenée à explorer la problématique posée, afin de définir les frontières et spécificités nécessaires pour apporter une solution concrète et satisfaisante.

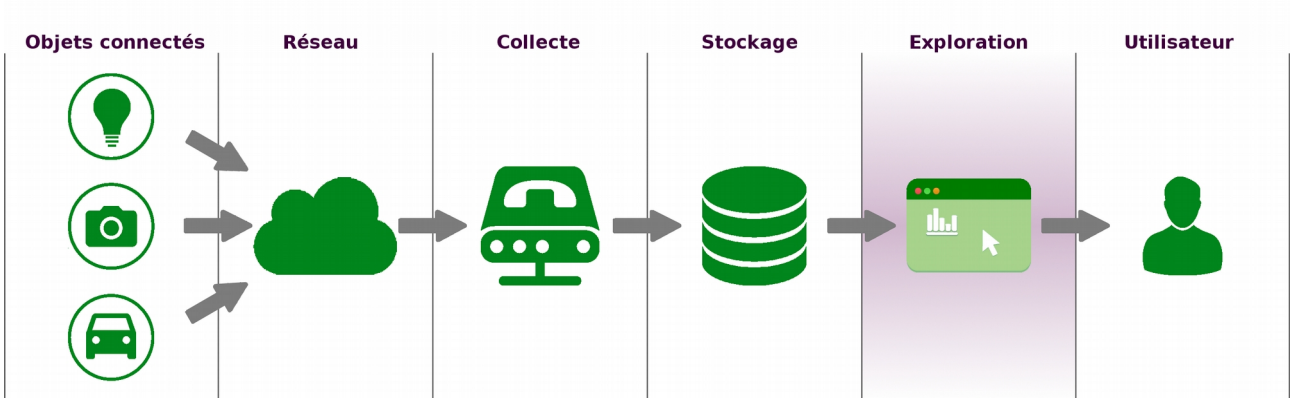


Figure 1: CHEPAAAA

2 Étude initiale

L'objectif initial est assez étroit et impose que l'utilisation des systèmes des stockage (MongoDB et MySQL). Mais il y a un autre contrainte sous-jacente: le stage se porte sur le domaine de l'IoT, et nous pouvons déjà en tirer une conclusion : l'outil d'exploration doit au moins être capable de travailler avec les données spécifiques à l'IoT.

Vue cet première précision importante, le milieu de recherche initial avait pour le but d'explorer et d'apprendre la façon dont les observations provenant des objets connectés sont sauvegardées. Il s'agit des séries temporelles : une manière intuitive d'organiser les mesures au fil de temps.

2.1 Séries temporelles

Une série temporelle, ou une série chronologique, est une suite de valeurs numériques qui évolue dans le temps. La plupart de temps les séries temporelles sont représentées par un ensemble des associations clef-valeur avec une clef-valeur spécifique d'horodatage que l'on appelle *timestamp*.

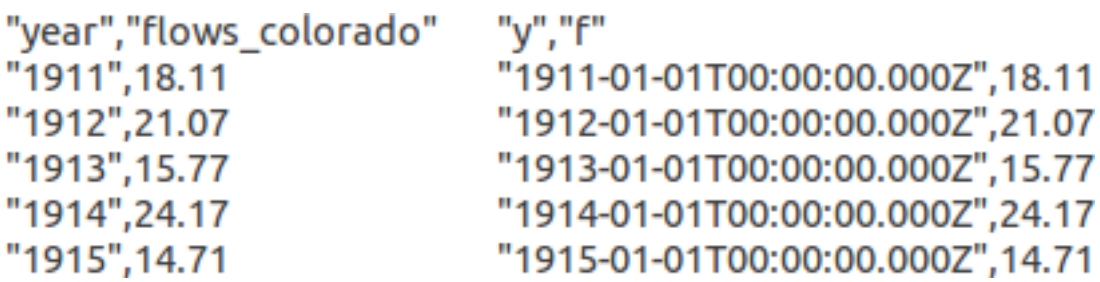
Le but d'une majorité des objets connectés est d'observer l'évolution de certains grandeurs physiques au file de temps. Ainsi l'utilisation des séries temporelles dans l'IoT est adapté : chaque observation est accompagnée d'un *timestamp* avec la date de prise de mesure. Il faut préciser bien sûr, qu'il s'agit des domaines d'application ou l'utilisation des séries temporelles est optimale. Par exemple, pour une caméra de vidéosurveillance connectée, l'utilisation de ce type d'organisation des données ne sera pas vraiment adaptée et naturelle.

Les séries temporelles peuvent être sauvegardée dans les sources différentes (stockages) avec un format variable. Par exemple voici une même série temporelle, mais présentée par différentes types de stockage comme les fichiers JSON (à gauche) et CSV (à droite) :

```
"year","flows_colorado"
"1911",18.11
"1912",21.07
"1913",15.77
"1914",24.17
"1915",14.71
```

```
[
    {"year": 1911, "flows_colorado": 18.11},
    {"year": 1912, "flows_colorado": 21.07},
    {"year": 1913, "flows_colorado": 15.77},
    {"year": 1914, "flows_colorado": 24.17},
    {"year": 1915, "flows_colorado": 14.71}
]
```

Ou on peut également imaginer un cas où les données sont sauvegardées sous le même type de stockage, mais la structure et format ne sont pas les mêmes :



"year","flows_colorado"	"y","f"
"1911",18.11	"1911-01-01T00:00:00.000Z",18.11
"1912",21.07	"1912-01-01T00:00:00.000Z",21.07
"1913",15.77	"1913-01-01T00:00:00.000Z",15.77
"1914",24.17	"1914-01-01T00:00:00.000Z",24.17
"1915",14.71	"1915-01-01T00:00:00.000Z",14.71

Dans l'exemple à droite on peut voir une autre problématique mineure qui peut être rencontrée lors de gestion des séries temporelles : le format de *timestamp* est aussi variable. La plupart du temps le *timestamp* suit un format standard ISO8601, ou souvent l'on utilise un *timestamp* unix : le nombre des secondes écoulées depuis le 1er janvier 1970. Mais parfois le format de temps peut être spécifique, par exemple dans le cas d'une série temporelle avec les mesures mensuelles annuelles du niveau des précipitations. Donc, cet aspect variable des séries temporelles doit aussi être géré si nous voulons concevoir un outil qui pourra travailler avec plusieurs formats et sources des données.

2.2 Socle de base

A l'issue de cette étude préliminaire, l'on se fixe un invariant de base, sur lequel l'outil d'exploration des données peut se fonder : on travaille uniquement avec les séries temporelles car c'est une façon naturelle de stocker les mesures. C'est qui implique, par la définition des séries temporelles, que l'application va fonctionner seulement avec les jeux de données ayant les caractéristiques suivantes :

- chaque échantillon d'une série temporelle doit être accompagné par un seul *timestamp*
- chaque échantillon doit avoir au moins un attribut numérique (une mésure)
- les *timestamps* doivent suivre un ordre chronologique

Bien sûr, même si la série temporelle est une façon très répandue de stocker des observations, il peut y avoir d'autres solutions pour organiser les données. Évidement ces formats spécifiques ne seront pas acceptés par l'application. Mais le choix de se concentrer sur les séries temporelles est argumentée par :

- l'utilisation des séries temporelles est compatible avec MayaNet
- c'est un façon d'organiser les données largement répandu
- pour arriver à un outil fonctionnel, il faut bien imposer ces variables

Ainsi l'on a une une pierre angulaire commune pour toutes les types de stockage et formats possibles: les données sont représentées par les séries temporelles.

2.3 Terminologie

Étant donné le fait que l'on ne travaille qu'avec les séries temporelles, afin d'éviter la confusion terminologique, nous allons définir explicitement certains expressions clés. Sur la Figure 2, on peut voir un exemple d'une série temporelle (en format CSV) et une description de ces éléments principaux.

Une « série temporelle » ou « *dataset* » (ou même « les données ») est composée d'une ensemble des « intstances ». Une instance, (ou « échantillon ») dispose d'une ensemble d'attributs. Il y a deux types des attributs : attribut d'horodatage (« *timestamp* ») et un attribut numérique aussi appelle « mesure », ou « valeur mesurée », « valeur ».

Lorsque on parle de format des données, il s'agit de la connaissance des attributs d'une série temporelle.

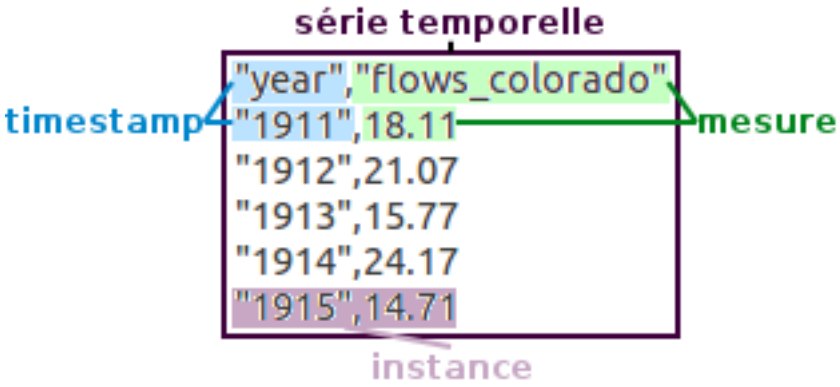


Figure 2: série temporelle

3 Première implémentation

En se basant sur le premier mini-étude sur les séries temporelles, la première application naïve est apparue. On va exposer ici les choix technologiques réalisées lors de sa développement, ainsi que ses problématiques. Tout cela pour illustrer les étapes de la déduction d'un objectif concret.

3.1 Module unique

Au début du stage l'outil à développer était vu comme une seule unité de traitement, qui peut recevoir en entrée des séries temporelles provenant de différents types de stockages et ayant des formats différents, et en sortie pourra fournir, sous un certain format, les indicateurs caractérisant les données. Et s'il le faut, les indices caractérisant les relations entre les données.

Gardant cet idée dans la tête, la première tentative pour répondre à un problème a été proposée. L'application était écrite en Java, utilise Morphia, et travaille avec MongoDB.

MongoDB

Le choix de MongoDB, comme type de stockage de base, est argumenté

par le fait que c'est une base de données orientée document, et par rapport à les bases de données classiques (relationelles), MongoDB permet d'avoir un format flexible des données. En quelque sorte, MongoDB (avec Morphia) était considérée comme une solution pour la problématique de gestion de plusieurs sources des données, parce que cette base peut facilement travailler avec différents formats des données.

Aussi, MongoDB est une base de données sur laquelle MayaNet compte migrer dans le future (encore une fois, à cause de dynamisme du format). Cette base de données sera le type de stockage privilégié dans MayaNet.

Vue ces deux arguments, la première application devra stocker les séries temporelles dans une base de données MongoDB.

Java

L'utilisation de Java comme langage de programmation a été aussi influencée par MayaNet car celle-ci utilise Kura[12]. Kura est un *framework* Java « basé sur OSGi pour les applications M2M s'exécutant dans les passerelles de services »[13]. Il s'agit d'une solution logicielle faisant un intermédiaire (passerelle) entre un réseau filaire ou sans fil d'objets connectées (capteurs, etc) et un réseau local (ou internet).

Donc, en se basant sur le principe que l'outil qu'on vise à développer, sera utilisée pour l'IoT et notamment MayaNet, l'utilisation de Java semble appropriée.

Morphia

Encore une fois, le choix de Morphia est argumenté par la compatibilité avec MayaNet. MayaNet utilise les objets POJO pour définir les données métiers. POJO (*Plain Old Java Object*) est un simple classe Java, qui par rapport aux JavaBeans, ne suit pas des conventions strictes.

Morphia est un *wrapper* de driver Java pour MongoDB qui se sert des POJO (en ajoutant quelques annotations) pour communiquer avec une base de donnés. Morphia est un couche d'abstraction de MongoDB qui facilite beaucoup l'interface entre les objets métier et la base des données.

L'idée derrière l'utilisation de Morphia est que, pour tout type d'objets connectes gérés par MayaNet, il faudra créer des objets métiers (des POJO). Donc, comme les POJO sont crées dans toutes les cas, pour toutes les « things », un outil générique d'analyse des données pourra utiliser Morphia comme un interface entre n'importe quel format d'objets métiers et la base des données.

Du coup, l'utilisation de Morphia est vue comme une réponse à une problématique de multitude des sources et formats des données. Car par la suite, chaque format sera défini dans les POJO, et Morphia offre une façon universelle (même pour toutes les POJO) d'écriture et lecture dans la base de

[12] http://www.eclipse.org/kura/
[13] http://www.electronique-mag.com/article8518.html

données MongoDB.

Récapitulatif

La première tentative naïve de répondre à une problématique posée est largement inspirée par les choix technologiques de MayaNet, dans le but d'être compatible avec cette dernière. C'est un application qui est capable de travailler avec les formats de données différentes grâce à l'utilisation de Morphia et MongoDB. Le but d'un étape de développement qui devrait suivre est d'englober cette application par une couche de traitement de données.

3.2 Inconvénients

Le première approche de développer un outil d'analyse des données a permit de se familiariser avec la problématique de multitude des sources et avoir une première initiation à les données réelles. Mais la solution proposée pouvait être améliorée sur plusieurs points.

Tout d'abord, l'utilisation de Java. L'objectif initial n'impose pas la compatibilité de langage avec MayaNet, c'était une décision de stagiaire de partir vers une solution qui sera à priori facilement intégrable. De plus, MayaNet utilise Kura (et donc Java) pour les couches très bases de communication entre les objets connectés et la passerelle. Par contre, l'application que l'on vise à développer doit servir à l'utilisateur pour explorer les données, et donc elle se place logiquement dans un couche plus haut (Interface Homme-Machine), loin des objets métiers de MayaNet.

Un second axe d'amélioration par rapport à la première approche est le support prévu d'au moins deux types de stockage : MongoDB et MySQL. Morphia est utilisable seulement avec MongoDB. Par contre il existe une DAO (un outil d'abstraction des données comme Morphia) Hibernate OGM, qui permet de travailler avec MongoDB et MySQL, mais les POJO ne sont pas interchangeables entre les deux[14]. Donc cette approche pour résoudre la problématique de multitude des formats de stockage n'est pas satisfaisante.

Mais la leçon apprise lors du développement de cette première approche est une réflexion sur la bonne façon d'aider l'utilisateur à explorer les données, et surtout les relations entre les données. Cet fonctionnalité n'a pas était rajoutée à cette première application parce que, cela ne rentre pas dans les objectifs visés, mais encore une fois, elle a permis d'initier la recherche des solutions pour offrir la possibilité d'explorer les données.

4 Exploration des donnés

En général le terme « exploration des données » correspond à un acte d'observation et analyse des données acquises, afin de tirer des conclusions sur les données. Dans le cadre du stage, l'étendue de ce terme n'est pas vraiment précisée dans l'objectif initial. Il peut s'agir d'utilisation des techniques statistiques, ou d'une simple visualisation par une table. Il va falloir fixer un sens concret pour ce terme afin d'apporter une solution réelle.

Il y plusieurs façons d'analyser et explorer les données, mais nous allons

[14] http://hibernate.org/ogm/faq/

étudier seulement les deux les plus communes et convenables. Tout d'abord, l'utilisation des techniques statistiques avancées de machâge des données pour en tirer des connaissances. Un autre façon, plus simple mais aussi efficace, est une visualisation des données via différentes sortes de graphes pour mettre en évidence leurs caractéristiques importantes facilement analysables par l'humain. Les deux approches ont des défauts et avantages.

Les techniques statistiques qu'on peut utiliser pour explorer les données, et surtout la relation entre les données, peuvent être simples comme la recherche de maximum ou minimum, du filtrage, une moyenne glissante. Mais ils peuvent aussi être avancées comme, par exemple, l'étude des tendances, l'utilisation des estimateurs, techniques d'extrapolation, classification, etc. L'avantage d'utiliser des méthodes statistiques est qu'elles sont très performantes pour trouver une valeur cachée dans un grand volume de données. Le problème est que, si on vise à développer un outil purement « statistique », une base de cet outil doit présenter des techniques avancées et complexes de traitement des données, et donc, la mise en place des ces méthodes risque d'être assez complexe, dans le temps imparti. Par contre, rien n’empêche d'appliquer au moins les caractéristiques statistiques simples.

La façon la plus simple et évidente, pour interpréter et explorer les données est l'utilisation de graphiques de différentes sortes comme les histogrammes, les nuage des points, etc. Les avantages et les défauts de cette approche sont opposés à celles d'une approche statistique : la visualisation par les graphiques est simple à mettre en place, mais l’étendu d’exploration des données, et des relation entre elles, n'est pas autant avancée.

Les critères sur lesquelles le stagiaire se base pour définir le terme « l'exploration des données » dans le contexte d'un outil à développer sont:

- le temps de développement
- la montée en compétence de stagiaire
- l’utilité pour un cas d'usage réel
- découverte de DC.js (chapitre suivante)

Vu tous ces points, l'exploration des données sera réalisée grâce à un outil de visualisation des graphiques dynamiques DC.js (qui permet aussi de mettre en évidence la relation entre les données) et quelques statistiques simples à mettre en place pour fournir plus d'information.

Cette solution est un mélange des deux approches exprimées ci-dessous : on utilisera une visualisation de graphiques, relativement avancée, associée à certaines techniques statistiques, relativement simples. Et surtout, c'est la solution la plus réaliste pour parvenir à fournir un outil fonctionnel.

5 DC.js

DC.js est une librairie JavaScript basée sur Crossfilter.js et D3.js.

5.1 Crossfilter.js

Crossfileter.js est une librairie open source d'exploration de grandes masses de données. Elle permet d'explorer les relations entre plusieurs attributs au sein d'une série temporelle.

Pour expliquer l'intérêt de l'utilisation de Crossfilter.js on invite à consulter une page officielle de Crossfilter[15] avec un exemple d'utilisation concret que l'on va exposer ici. On précise que le chargement de cette page pourra prendre quelques secondes. Un autre remarque à noter et que les graphes ont été générées via D3.js et sont cliquables (comme expliqué par la suite).

L'exemple est construit à partir d'un extrait de *dataset*[16] des vols réalisés entre 1 janvier et 31 avril 2001. Chaque instance de cet base a 29 attributs (mesures), dans cet l'exemple seulement 4 ont été étudies : l'heure de départ, le retard à l'arrivée (en minutes), la distance parcourue (en milles) et la date de départ.

Chaque attribut étudié est représenté par un histogramme. Sur chaque histogramme il est possible de choisir une plage des valeurs, c'est-à-dire appliquer un filtre (la raison pourquoi ils ont cliquables). Grâce à Crossfilter.js, le filtre sera affectée à toutes les autres histogrammes. Il est possible d'utiliser plusieurs filtres à la fois.

Tout cela nous permet, par exemple, d'explorer les relations suivantes entre plusieurs mesures (cf Figure 3):

- quels sont les jours de la semaine avec les vols les plus lointains ?
- quels sont les distances de vols à choisir, pour avoir le moins de retard (imaginons que c'est ce qu'on cherche)?
- quels heures de départ faut-il choisir pour avoir le plus de retard (si on aime bien passer du temps en avion ou l'aéroport, par exemple)?

On peut dire que certains relations sont absurdes, mais ils existent, et donc sont explorables par Crossfilter.js. En fait, les correspondances entre n'importe quels attributs au sein d'une même instance sont explorables par cette librairie.

La puissance de Crossfilter.js est que les filtres sont appliqués en temps réel (interactivement), ce qui implique que lorsque les données en entrée sont modifiées, les caractéristiques calculées (la sortie) sont modifiées instantanément. Cela permet d'explorer l'influence de chaque mesure sur le résultat final. Par illustrer cela, prenons un exemple d'une série temporelle sur laquelle nous appliquons Crossfilter.j pour compter le nombre d'occurrences de chaque valeur mesurée (Figure 4). En entrée, nous avons un *dataset* avec les données à analyser, en sortie nous avons un *dataset* avec le résultat de

[15] http://square.github.io/crossfilter/

[16] http://stat-computing.org/dataexpo/2009/the-data.html

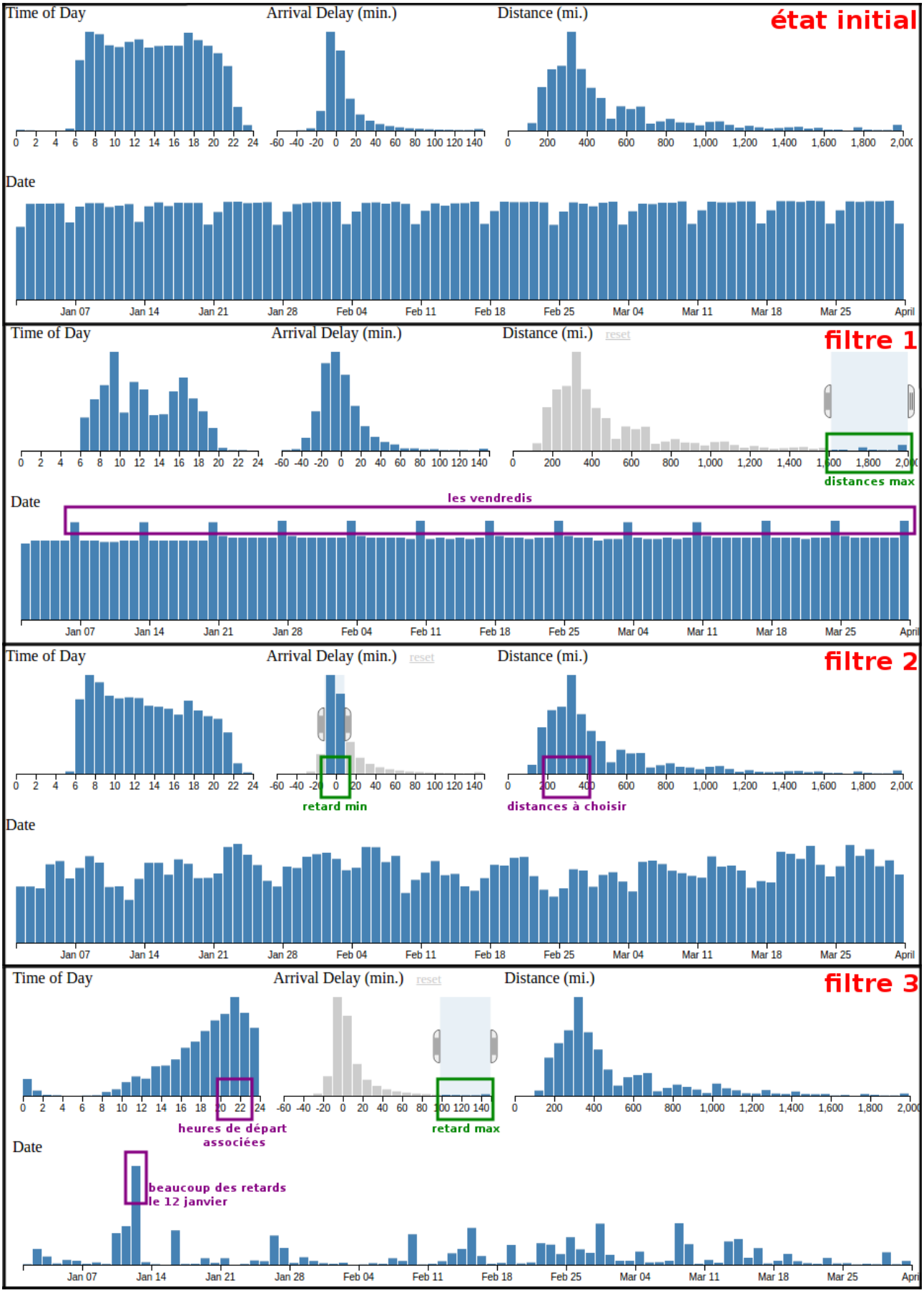


Figure 3: l'exemple de Crossfilter.js rendu avec D3.js

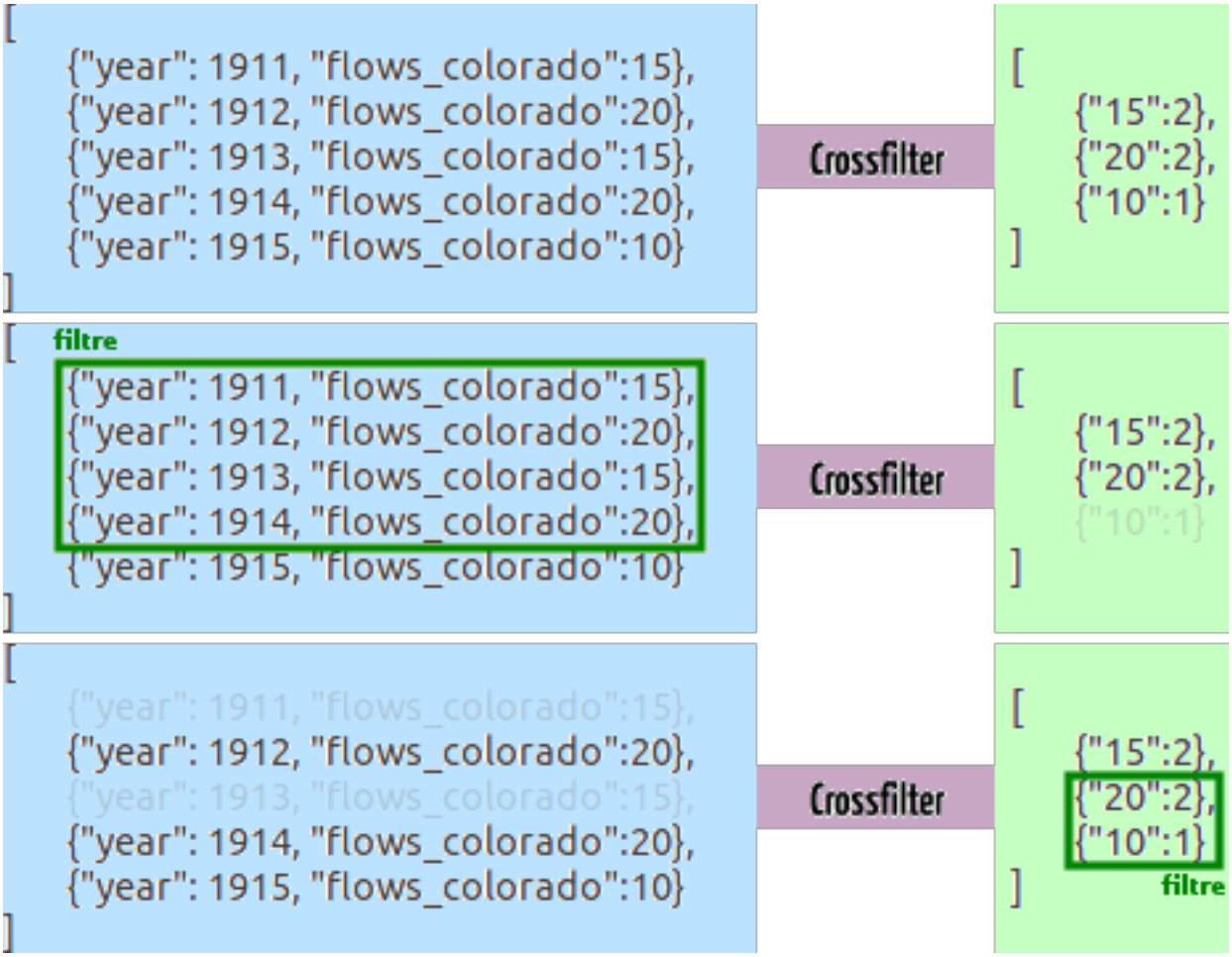


Figure 4: filtres de Crossfilter.js

l'analyse. Si à l'instant *t* les données en entrée sont modifiées (on applique un filtre) le *dataset* en sortie est instantanément modifiée.

Comme il a été dit précédemment, Crossfilter.js supporte les filtres multidirectionnelles, c'est-à-dire dans le cas de cet exemple, on peut appliquer un filtre sur les caractéristiques (les sorties), pour voir comment les données à l'entrée sont changées.

En conclusion, Crossfilter.js est un outil puissant d'analyse de grands volumes de données, mais il ne s'agit pas d'une bibliothèque graphique, c'est plutôt une machine d'exploration de données. Pour visualiser les relations dynamiques mis en évidence par Crossfilter.js, il faut utiliser une autre librairie JavaScript de graphisme. Il est possible d'utiliser n'importe quelle bibliothèque de son choix, mais pour tirer le maximum de Crossfilter.js, il faut considérer deux points importants: l’interfaçage et le dynamisme.

Comme Crossfilter.js travaille directement avec les données (*dataset* en entrée et en sortie), si nous voulons l'utiliser avec une librairie de rendu visuel, cette dernière doit être capable de générer les graphes directement (ou avec un minimum d'effort) à partir des données. Donc, l'interfaçage avec Crossfilter.js se traduit par l'interfaçage avec les données. Un autre point est le dynamisme. Pour profiter d'utilisation des filtres interactives de Crossfilter.js, la bibliothèque graphique doit supporter les transitions, *morphing*, le passage d'objets visuelles d'un état à un autre, pour pouvoir visualiser le changement

de *dataset* (en entré ou sortie). En plus, comme l'application des filtres se traduit par la modification des données, la librairie graphique doit être capable de détecter un tel changement.

Un bon candidat ayant une bonne compatibilité avec Crossfilter.js est D3.js. Ces deux bibliothèques JavaScript sont souvent utilisées ensemble (comme dans le première exemple de ce chapitre), mais il peut être utilisée avec d'autres librairies graphiques comme Chart.js[17] ou NVD3[18].

5.2 D3.js

D3.js (*Data-Driven Documents*) est une bibliothèque JavaScript graphique qui permet la visualisation des données numériques sous la forme des graphiques dynamiques. Elle est conforme W3C et utilise les technologies SVG (*Scalable Vector Graphics*) et CSS (*Cascading Style Sheets*). Les point forts de cette librairie graphique est dynamisme (les objets graphiques peuvent réaliser des transitions) et le fait qu'elle est orientée-données (« *data-driven* »).

Pour expliquer les avantages de D3.js il faut tout d'abord se rappeler de quelques bases du web. Toutes les éléments d'une page html (les *tags*) sont organisés dans une hiérarchie qu'on appelle DOM (*Document Object Model*). Les nœuds de la DOM peuvent être modifiées par plusieurs langages de programmation, c'est ce qui va impacter la page HTML.

D3.js utilise des SVG pour la génération des objets visuelles. Les SVG sont les images vectorielles, et chaque élément d'image fait partie de la DOM. C'est ce qui permet de les modifier facilement et c'est la raison pour laquelle les graphes de D3.js sont dynamiques et peuvent « bouger ».

Ensuite, D3.js est « *data-driven* » parce que elle utilise le data-binding. Il s'agit du lien entre les données bruts (sous un format JSON) et les éléments de la DOM (les SVG) pour le rendu visuel. Ce lien garantit que lorsque les données sont changées, les graphes vont « bouger ».

Pour conclure sur D3.js, il faut dire que ce n'est pas une librairie de génération des graphiques scientifiques, mais plutôt de rendu des objets visuels. Nous invitons à consulter la page principale de D3.js[19] pour se rendre compte de toutes les domaines d'application de cette bibliothèque graphique. Cette flexibilité de génération des objets visuels implique que langage graphique de D3.js est assez bas niveau: on travaille avec les rectangles, cercles, sélecteurs des éléments HTML. C'est qui implique que le temps d'apprentissage nécessaire pour visualiser des graphes simples est important.

5.3 DC.js

DC.js (*Dimensional Charting*) et basé sur deux bibliothèques JavaScript : Crossfilter.js pour filtrage multidirectionnel entre les données, et D3.js pour le rendu des objets visuelles dynamiques. Ces deux librairies vont naturellement bien ensemble, et peuvent même être utilisées en tant que tel sans aide de DC.js. Ce qui permettra de générer des visualisations plus flexibles et configurables, mais il y a quelques raisons importantes pour l'utilisation de

[17] https://github.com/nsubordin81/Chart.dc.js/tree/master
[18] http://stackoverflow.com/questions/19094015/crossfilter-d3-brush-and-nvd3-integration
[19] https://d3js.org/

DC.js.

Tout d'abord, la courbe d'apprentissage de D3.js est exponentielle, il faut passer beaucoup de temps pour apprendre comment l'utiliser pour rendre des graphiques simples. Bien entendu, le temps fourni à cet apprentissage va permettre de développer des visualisations splendides, complexes et spécifiques, comme par exemple celle-ci[20] , réalisée en pure D3.js. Mais comme en chapitre 2.3 de la Partie 2, nous avons décidé de travailler qu'avec les séries temporelles, pour les analyser il nous suffit d'un ensemble des graphiques classiques et simples (histogramme, nuage des points, etc). Et DC.js nous offre un nombre important des graphiques déjà configurées, il faut juste passer les données pour qu'ils soient visualisés.

Donc, DC.js est une autre pierre angulaire de stage, sur laquelle nous nous basons pour « explorer les données », et voir les « relations entre les données ». En langage DC.js cela se traduit par l'exploration des relations entre les attributs d'une série temporelle.

5.4 SVG

DC.js est basée sur D3.js, et comme nous l'avons dit avant, elle utilise des SVG pour générer les objets visuels. La puissance de SVG vient de fait que c'est une image vectorielle, et chaque son composant est référençable via DOM. C'est ce qui permet de mettre à jour facilement certains éléments de l'image, mais cela a un inconvénient. Pour être repérable via DOM, chaque composant d'image doit porter avec lui des métadonnées. Et si l'image à rendre et assez complexe, comportant de milliers des éléments vectoriels, la génération sera ralentie. Donc cela peut engendrer de la latence lorsque nombre des formes à dessiner est important.

Pour illustrer ceci, l'on peut comparer[21] SVG avec un autre technologie souvent utilisé pour visualisation des images en web: les canvas. Pour le DOM, le canvas représente un trou avec des pixels. On ne peut pas référencer les formes dans le canvas, pour mettre à jour l'image il faut le recharger. Par contre comme il s'agit juste d'un ensemble des pixels, sachant aussi qu'ils sont générées par le processeur graphique, le temps de réponse est presque invariable en fonction de nombre des éléments à produire.

Donc, dans les cas d'une grande taille des objets visuels à dessiner, DC.js sera long. Cet aspect doit être géré pour satisfaire au mieux l'expérience d'utilisateur.

5.5 Format des données

Une autre point à préciser à propos de DC.js est le format des données. La structure des données avec laquelle travaille Crossfilter.js, et donc DC.js, est une table au format JSON. JSON, comme XML, est un format des données textuel. Il comporte des paires clef-valeur, et des listes de valeurs. Si l'on stocke une série temporelle en JSON, chacun document JSON doit comporter un timestamp et une ensemble des attributs numériques.

---

[20] http://www.nytimes.com/newsgraphics/2013/09/07/director-star-chart/

[21] http://smus.com/canvas-vs-svg-performance/

Une précision importante à noter à propos de Crossfilter.js est que, en fait, il est possible d'explorer les relations entre plusieurs valeurs mesurés seulement au sein de la même instance. En langage JavaScript cela vaut dire que seulement la comparaison entre les valeurs au sein d'un seul JSON (cf Figure 5) est possible.

Concrètement cela vaut dire que Crossfilter.js peut établir des filtres interactives seulement au sein d'une seule série temporelle. Encore une fois, en langage JavaScript cela signifie que Crossfilter.js n'accepte qu'une seule table JSON. Et donc, comme dans un exemple de la Figure 5, il va falloir fusionner plusieurs séries temporelles en une, si on veut pouvoir comparer ses attributs. Plus de détails sur les aspects subtils de cette fusion sont précisés dans le chapitre 6.

6 Fusion des séries temporelles

Concernant l'utilisation de DC.js, si l'on veut profiter de ses capacités pour comparer plusieurs séries temporelles, il va falloir les combiner de sorte d'en avoir qu'une seule. Une façon de le faire est de réaliser une fusion horizontale, un peu comme dans la Figure 5.

Cette fusion n'est une opération compliquée à mettre en place, mais il y a quelques problèmes à résoudre avant de le faire. En fonction des conditions dans lesquels les objets connectes réalisent les mesures, la période de la prise de mesure peut varier. De plus, il se peut que les *timestamps* d'une série temporelle soient complètement désordonnées (c'est-à-dire que les mesures sont prises irrégulièrement dans le temps). Et donc, si nous voulons comparer les deux suites de mesures provenant de différentes sources, nous risquons de réaliser des fusions fausses si nous ne respectons pas la correspondance des *timestamps* entre les deux séries temporelles.

Donc il faut traiter toutes les séries temporelles avant de les fusionner, afin qu'elles soient synchronisées. Les deux séries temporelles sont synchronisées (ou homogènes) s'ils ont le même nombre d'instances, et que les *timestamps* des instances adjacentes sont égaux. C'est-à-dire qu'avant de fusionner toutes les séries temporelles, il faut s'assurer qu'ils ont toutes les mêmes instantes de prise de mesures, dans le même ordre. Or cela est rarement le cas si les séries proviennent de différents sources.

Donc, il va falloir pré-traiter les données, pour qu'ils soient homogènes. Il y a plusieurs façons de le faire. Par exemple, entre un ensemble de séries temporelles, nous pouvons laisser seulement les mesures qui ont été faites aux instants inclus dans toutes les séries temporelles. Ou nous pouvons faire une interpolation des points (mesures) manquant d'une série mais présentes dans une autre. Aussi on peut remplacer chaque trou (instance manquante) par des attributs zéro. Ou même choisir de ne pas accepter à traiter les suites des mesures qui ne sont pas compatibles. Cette question de homogénéité fais partie de problématiques auxquelles le stagiaire doit répondre et trouver la solution la plus approchée de l'objectif initial.

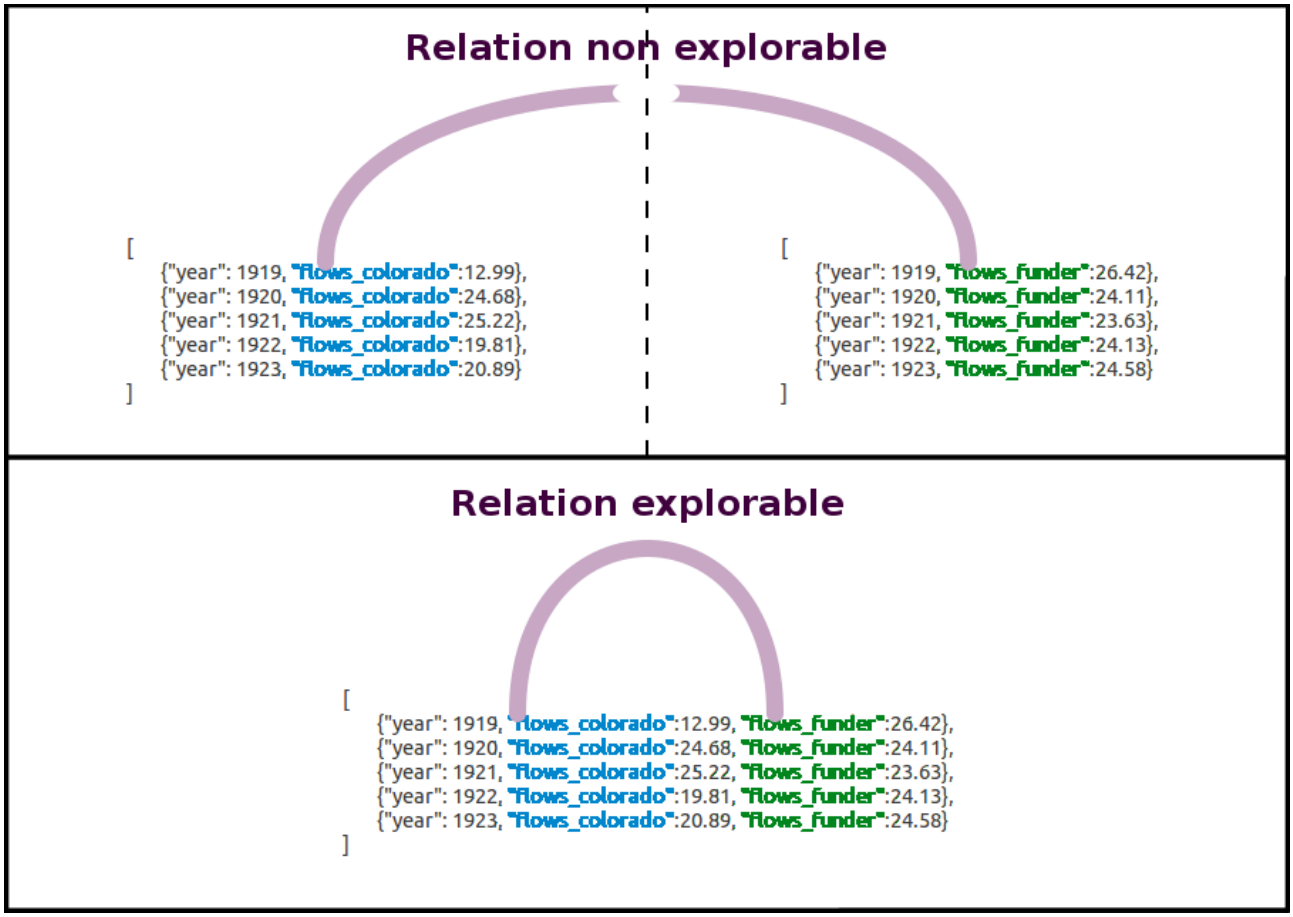


Figure 5: fusion des séries temporelles

7 Nouveau objectif concret

Un étape d'étude nous a permis de définir le cahier de charges de l'application et implanter un objectif précis. On se fixées déjà que notre outil ne travaille qu'avec les séries temporelles. De plus, comme nous avons décidé dans un chapitre 4 de la Partie 2, le terme « l'exploration des données » vaut dire l'exploration visuelle. Nous allons utiliser des graphiques pour mettre en évidence certains caractéristiques des séries temporelles. De plus, pour approfondir l'exploration, on va calculer quelques statistiques simples sur les données.

En ce qui concerne les « relations entre les données », il s'agit de la découverte de l'influence des valeurs d'un attribut d'une série temporelle sur l'autre. Cet influence sera mise en évidence grâce à les filtres des Crossfilter.js qui fait partie de DC.js qu'on va utiliser pour la visualisation.

Voici donc le nouveau objectif détaillée:

« Concevoir un outil graphique d'exploration des séries temporelles provenant de différentes sources d'objets connectés»

Partie 3 : Travail réalisé

Dans cet partie nous allons exposer toutes les fonctionnalités développées lors de stage.

1 Deux modules

Pour répondre au mieux aux exigences du cahier de charges et maximiser la ré-utilisabilité de l'application, nous avons décidé de le diviser en deux modules indépendantes. La première module est une boite à outil de traitement des séries temporelles. Le second est une application graphique qui englobe le premier module et offre la possibilité de travailler avec plusieurs types de stockages des séries temporelles, ainsi qu'il permet de les explorer grâce aux graphiques de DC.js.

Les raisons pour choisir une telle architecture sont multiples. Tout d'abord, il y a un besoin de traiter plusieurs formats et sources de données. Pour répondre à cette contrainte, une solution logique est de développer une brique principale, qui réalise le traitement d'un certain format des séries temporelles, et ensuite, concevoir des convertisseurs pour chaque format spécifique, autre que supportée par la brique. Cette approche est beaucoup plus optimisée que d'avoir un traitement spécifique pour chaque format de série temporelle. De plus, de point de vue d'évolutivité du code, cette architecture est beaucoup plus accueillante pour les modifications et améliorations, car les algorithmes de traitement des séries temporelles sont détachées de format des données.

Une autre raison est, encore une fois, la compatibilité avec MayaNet. Si les solutions développées pendant ce stage seront demandées par les clients de MayaTechnologies, il sera beaucoup plus simple de rajouter un module de traitement dans un projet existant que de « casser » une autre application pour y tirer du code et reconstruire les fonctionnalités nécessaires. De plus, il se peut que pour certains raisons une autre librairie graphique que DC.js soit choisi pour le rendu des graphiques, dans ce cas, si l'application qu l''on a conçu et à 100 % attachée à DC.Js, ça ne sera pas possible.

Finalement, même si le module de traitement des séries temporelles seul ne remplit pas toutes les exigences du cahier des charges, c'est une partie de travail réalisée qui risque d'être utilisée le plus souvent. Et donc, l'application graphique en quelques sorte, joue le rôle d'un exemple de configuration de module de traitement. Et de même, l'application graphique fournit les exemples de mise en place des graphiques de DC.js pour explorer les données.

2 Module de traitement

Un module de traitement des séries temporelles, qu'on appelle *tsproc* (*TimeSeries Processor*), est un module indépendant Node.js qui prend en entrée un ensemble des séries temporelles, un JSON avec la configuration de toutes les séries, et offre plusieurs façons de les traiter.

De ça conception le module était prévu pour pré-traiter les données avant de les servir à DC.js. C'est la raison pour laquelle il contient certains

fonctionnalités spécifiques comme l'interpolation ou fusion horizontale des *datasets*, nécessaires pour DC.js et l'exploration interactive des données. Mais *tsproc* a fini par être une boite à outils pour les séries temporelles en general. Actuellement l'utilisation de se module ne se limite pas exclusivement pour servir les graphiques.

2.1 Choix technologiques

Le module est écrit en JavaScript. Le choix de ce langage est argumenté par la compatibilité avec MayaNet, car la parti IHM de MayaNet tourne sous un serveur Node.js. De plus, JavaScript est un langage qui prend de plus en plus d'ampleur pour l'utilisation de côte de serveur, et aussi en IoT.

Les séries temporelles sont stockées par une table de documents JSON, et c'est le seule format acceptée par ce module de traitement. Les autres formats doivent être transformées pour être accepté par le module. JSON est utilisée car c'est un format acceptée par Crossfilter.js. De plus, il est native dans JavaScript et indépendant de langage de programmation et s'utilise partout actuellement.

Même si l'on fige la forme sur laquelle les séries temporelles sont stockées (une suite des JSON), le format d'organisation des données reste toujours variable comme dans un exemple de chapitre 2.1 de Partie 2. Plus précisément, les noms des attributs peuvent être différentes. Il s'agit de la problématique posée dès le début de la multitude des formats. Pour la résoudre, le format de chaque série temporelle (description des ces attributs) doit être passé à l'entrée du module dans un JSON de configuration. Donc, la solution à une problématique de multitude des formats de séries temporelles consiste juste à spécifier le format de chaque série.

2.2 Première approche

On va exposer ici la première approche de développement d'un module de traitement des séries temporelles.

Delta

Comment nous l'avons vu dans le chapitre 7 de la Partie 2, pour être capable d'explorer plusieurs séries temporelles par DC.js, ils doivent être homogénéisées et fusionnées. La première idée pour synchroniser plusieurs suites de mesures est d'utiliser un delta de temps. Il s'agit de découper le temps entre la mesure la plus ancienne et la plus récente de toutes les séries temporelles en *n* morceaux de taille delta. Ensuite, toutes les mesures qui se trouvent dans une plage horaire d'un delta donnée sont composées par une certain opération mathématique (calcul de somme ou moyenne), à une seule mesure. A l'issue de cet algorithme toutes les séries temporelles ont la même période (égale à delta) des mesures et pourront être fusionnées.

Cette approche peut être utile pour certains types de mesures, par exemple les relèves de la pluie annuelle (et donc si la delta est égale 5 ans, on somme les relèves par année), mais pour les séries temporelles où les mesures ne sont pas cumulatives (par exemple les niveaux de tension hebdomadaire)

Figure 5: exemple réel

cela n'a pas de sens. Et surtout l'utilisation de delta n'a pas passé le premier test d'un exemple réel d'application.

Exemple réel

La continuation sur la voie de développement d'utilisation de la technique de « delta » comme une solution pour synchroniser les séries temporelles a été interrompue face à un exemple réel. Il s'agit de demande de client de MayaNet d'inclure des graphiques dans un IHM pour le monitoring d'état des équipements (Figure 5). Comme le stagiaire est monté en compétence sur l'utilisation des bibliothèques JavaScript graphiques pour visualisation des mesures venant des objets connectées, cet tâche lui a été attribuée. Cette simple mission à relevé plusieurs problèmes qui doivent être traitées impérativement.

Afin de s'assurer de résoudre une problématique réelle, le module a été mis au point et testé avec les données d'un projet en cours de développement, assurant la collecte de mesure de consommation d'éclairages publics. Cet simple test a relevé plusieurs problèmes qui doivent être traitées impérativement.

Le souci le plus simple à résoudre était l'utilisation de *timestamp* avec un format non supportée ISO8601. Cet aspect (multitude des formats de *timestamp*) est à gérer si nous voulons efficacement travailler avec les mesures provenant de différent sources. Ensuite, même si pour les données à visualiser il n'y avait pas de besoin de fusionner les séries temporelles (car il n'y en avait qu'une seule), l'inconsistance de l'approche de « delta » est devenu évidente car elle transforme la nature des données. Et si le but est de visualiser et explorer les données cela n'a pas de sens.

Donc, le contact avec des données d'un système d'IoT réel a permis de se rendre compte d'affiner les spécifications initiales et de pousser la réflexion sur autres façons pour rendre plusieurs séries temporelles homogènes. Finalement, le remplissage des mesures manquantes par interpolation a été choisi. Une autre façon proposé est de ne laisser que des mesures présentes dans toutes les séries temporelles (« intersection » des mesures).

L'aspect positive est qu'on se soit rendu compte, que la séparation logique entre le module de traitement et la couche graphique est une bonne tactique pour distribuer des fonctionnalités de module dans un projet externe. Un autre point positif est que la résolution de tous ces problèmes a permit d'apporter une solution fonctionnelle qui a été introduite dans une application réelle, validée par le client.

2.3 Fonctionnalités réalisées

Le point d'entrée de module *tsproc* est un constructeur qui l'instancie et prend en entrée un ensemble de séries temporelles plus un JSON avec la configuration. La configuration sert à décrire chacune des séries temporelles en entrée (les attributs numériques, *timestamp* et le format de *timestamp*) ainsi que les options de traitement (type d'interpolation à appliquer, est-ce qu'il faut détecter la corrélation ou pas, etc).

Ensuite, le module offre un ensemble des méthodes publiques de traitement des séries temporelles. Leur comportement est configuré à l'instanciation du module via le JSON de configuration. Nous allons lister et expliquer ici quelques méthodes principales:

- **isHomogeneous()**: une méthode qui renvoi vrai si une ensembles de séries temporelles est homogène, et faux sinon ;
- **cut()**: sert à découper la série temporelle et laisser seulement les mesures entre les bornes précisées dans un JSON de configuration ; l'intérêt de cette méthode est de limiter nombre des instances dans un *dataset* et pour rétrécir le domaine d'exploration
- **undersample()**: sert à diminuer le nombre des échantillons dans une série temporelle en remplaçant *n* instances consécutives par une instance, grâce à une opération mathématique spécifiée dans un JSON de configuration
- **interpolate()**: si un ensemble des séries temporelles n'est pas homogène, la méthode réalise l'interpolation de toutes les mesures manquantes ; à l'issue de cette méthode, les séries deviennent homogènes ; cette méthode est utile pour préparer les donnes avant de faire une fusion de séries temporelles
- **intersect()** : même utilité que la méthode *interpolate()*, mais au lieu de faire l'interpolation, cette méthode homogénéise toutes les séries temporelles en ne laissant que des instances présentes dans toute les séries
- **merge()**: réalise la fusion de plusieurs séries temporelles vers une seule, réalisable seulement si elle sont homogènes ; la fusion des plusieurs séries est obligatoire pour explorer les relations entre eux par DC.js

- **checkSimilarity()**: sert à détecter la similitude entre deux ensembles de mesures par calcul de la corrélation locale
- **quantize()**: sert à réaliser une opération de quantification sur toutes les mesures et *timestamps* ; cela permet de mieux analyser les données, voir quels valeurs des mesures apparaissent le plus souvent
- **toISO()**: passe toutes les *timestamps* de toutes les séries temporelles vers un format ISO8601
- **getAvgPerDay()**: renvoi un nombre moyenne des mesures par jour ; un indicateur important pour caractériser le fréquence de prise des mesures dans un *dataset*

Le module contient une méthode principal **process()**, qui exécute certains des fonctions citées auparavant. A l'issue de cet méthode, on a une seule série temporelle avec un seul *timestamp* 'time' ayant une date en format ISO8601.

2.4 Problèmes rencontrées, futur évolutions

Les problèmes principales rencontrées lors de développement de ce module de traitement des séries temporelles sont la gestion de format de date de *timestamp*, l'interpolation des séries temporelles et la détection de corrélation locale.

Une des problèmes non résolu est le support par certains méthodes de *tsproc* des séries temporelles avec des attributs nominaux. Par exemple, l'interpolation d'une suite des valeurs nominales est une problème assez complexe et nécessite probablement application des techniques de machine learning, ou méthodes avancées d'interpolation[22]. Et de plus, à part la problème avec l'interpolation, il y a d'autres méthodes qui sont concernées et ne pourront pas gérer les attributs nominaux. Donc, actuellement, s'il y a un besoin de travailler avec les séries temporelles nominaux, il faut qu'elles soient homogènes.

Cet problématique des attributs nominaux peut être un objet des évolutions futures de module de traitement.

3 Outil de visualisation

L'outil de visualisation *dataexp* (*Data Exploration*) a pour but d'implémenter les graphiques DC.js pour l'exploration de séries temporelles, et être un exemple de configuration de module de traitement *tsproc*, dont il s'en sert. Il doit supporter le chargement des données depuis plusieurs types de stockages, et avoir un interface simple pour en rajouter des nouvelles.

Aucune contrainte, n'a été imposée, à part le support des deux types de stockage MySQL et MongoDB. Le choix de design et des technologies a été fait par le stagiaire.

[22] http://act.sagepub.com/content/1/2/201.full.pdf+html

3.1 Architecture. Choix technologiques

L'application se décompose de deux tiers principaux (Figure 7) coté serveur (back-end) et client (front-end). Le coté client se charge de la visualisation des graphiques DC.js et Canvas.js (les raisons d'utilisation de cette bibliothèque sont expliquées dans le chapitre suivant), visualisation des

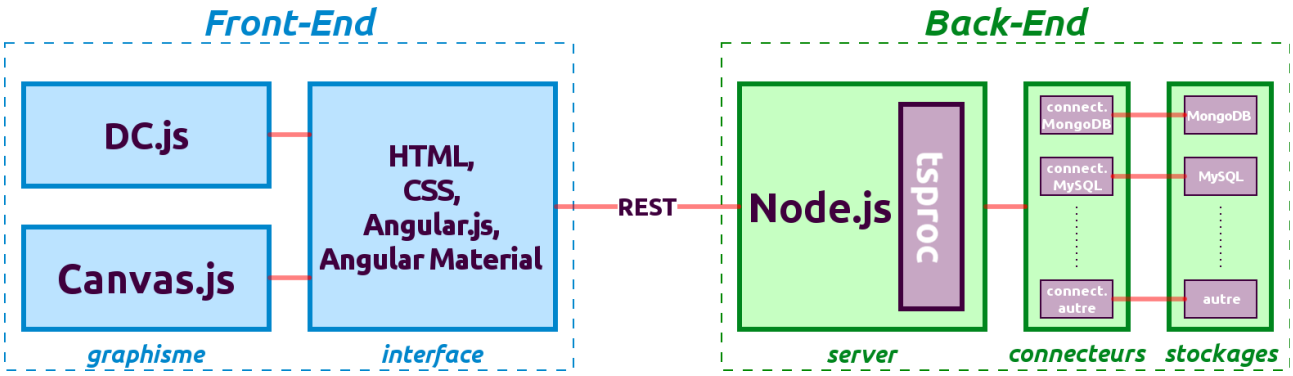


Figure 7: architecture d'application graphique

menus de choix des sources et les options de traitement des séries temporelles.

La vue du client a été développée via les technologies WEB notamment HTML, CSS et Angular.js[23]. La dernière est une *framework* JavaScript qui facilite la création des pages web dynamiques. De plus, nous avons utilisé un module d'angular Angular-Material[24], qui facilite aussi le rendu visuel des éléments d'interface et impose les spécifications de Google Material Design[25]. L'interfaçage avec le back-end se fais grâce à des services angular, qui eux utilisent un API REST[26].

Le but de back-end est de servir les fonctionnalités de module de traitement *tsproc*. C'est la raison pourquoi le back-end est basée sur un serveur Node.js. Pour supporter plusieurs types des stockage, l'accès à chacun doit être réalisé par un connecteur spécifique qui réalise quartes opérations identiques pour toutes les types de stockage:

- lecture de la liste des stores dans un stockage ; en langage MySQL cela correspond à lire les noms de tables dans une base des données, ou les noms des collections dans une base des données MongoDB
- lecture de nombre de documents dans un store (nombre de lignes dans une table pour MySQL, ou nombre des documents dans une collection pour MongoDB)
- lecture des attributs de store (les noms des colonnes dans une table pour MySQL ou les noms des champs dans un document pour MongoDB)
- lecture de *dataset* (lecture de toutes les lignes et colognes pour MySQL ou lecture de tous les documents d'une collection pour MongoDB) ; le format de *dataset* en sortie est toujours fixe : il s'agit d'un table des JSON (un format nativement accepté par *tsproc*)

[23] https://angularjs.org/
[24] https://material.angularjs.org/latest/
[25] https://material.google.com/#introduction-goals
[26] https://fr.wikipedia.org/wiki/Representational_state_transfer

3.2 Librairies de visualisation

L'utilisation de la librairie graphique DC.js est primordiale pour notre application car c'est un outil majeur qui permet d'explorer les données. Mais comme c'était indiquée dans un chapitre 5.4 de la Partie 2, cette bibliothèque se base sur l'utilisation des SVG pour rendre les objets visuelles, et cela peut ralentir le système si le nombre des formes à afficher est important.

Il n'existe pas une librairie graphique capable de faire c'est que DC.js fait, et donc on ne peut pas la remplacer, DC.js dois forcement faire partie de l'outil qu'on est censé réaliser. En gardant cela en tête, il y a deux solutions pour résoudre ce problème, et les deux comportent la réduction du nombre des mesures dans une série temporelle. Cela est inévitable si on veux se profiter des fonctionnalités de DC.js.

La première approche consiste à ne pas autoriser l'utilisateur à visualiser des graphiques tant que la taille d'une série temporelle dépasse un certain seuil. L'utilisateur sera obligé d'appliquer les méthodes offert par un module *tsproc* (choix d'un borne des dates, sous échantillonnage, etc) pour diminuer le nombre des mesures dans une série temporelle. Cette approche garantit que DC.js ne va présenter de latence et que les données seront explorées efficacement. La problème qui s'impose est de l'ordre de la « *user experience* », car nous imposons à l'utilisateur de faire beaucoup des manipulations avant même de commencer de s'initier aux données.

Donc, une autre alternative, et celle qu'on a choisie, est d'avertir l'utilisateur que l'utilisation de DC.js n'est pas possible. Mais quand même afficher des graphiques simples, sans le dynamisme et degré d'explorabilité offert par DC.js. Pour réaliser cela, il fallait s'appuyer sur une autre bibliothèque graphique JavaScript, et nous avons choisi Canvas.js parce-qu'elle contient les figures qu'il nous faut, simple à mettre en place, et fait correctement ce qu'elle est censée de faire.

A l'inverse de DC.js, Canvas.js[27] utilise les canvas (dont on a déjà parlée dans un chapitre 5.4 de Partie 2). Pour l'HTML, canvas est un trou avec les pixels, rien est réferençable de l'extérieur. Cela donnée, par rapport à les SVG un gain de vitesse de chargement, et aussi, les canvas sont beaucoup moins sensibles à l'augmentation de nombre des objets visuels à rendre. La raison est que, encore une fois, mille ou milliard rectangles pour canvas correspond toujours à un trous avec les pixels, mais pour SVG il s'agit de mille ou milliard des objets qui ont des métadonnées qui doivent être stockées dans la mémoire.

Donc, nous utilisons Canvas.js lorsque le nombre des mesures dans une série temporelle dépasse un seuil acceptable pour une bonne utilisation de DC.js. Et si l'utilisateur veut profiter des outils d'exploration de DC.js, il pourra utiliser les options de réduction de taille pour en faire.

3.3 Fonctionnalités réalisées

Dans se chapitre nous allons exposer les éléments de l'interface d'application et ces principales fonctionnalités. Actuellement l'outil travail avec

[27] http://canvasjs.com/

quatre types des sources (stockages) de séries temporelles : bases de données MongoDB et MySQL ainsi que les fichiers JSON et CSV.

L'application offre la possibilité d'explorer les données via 5 graphiques :

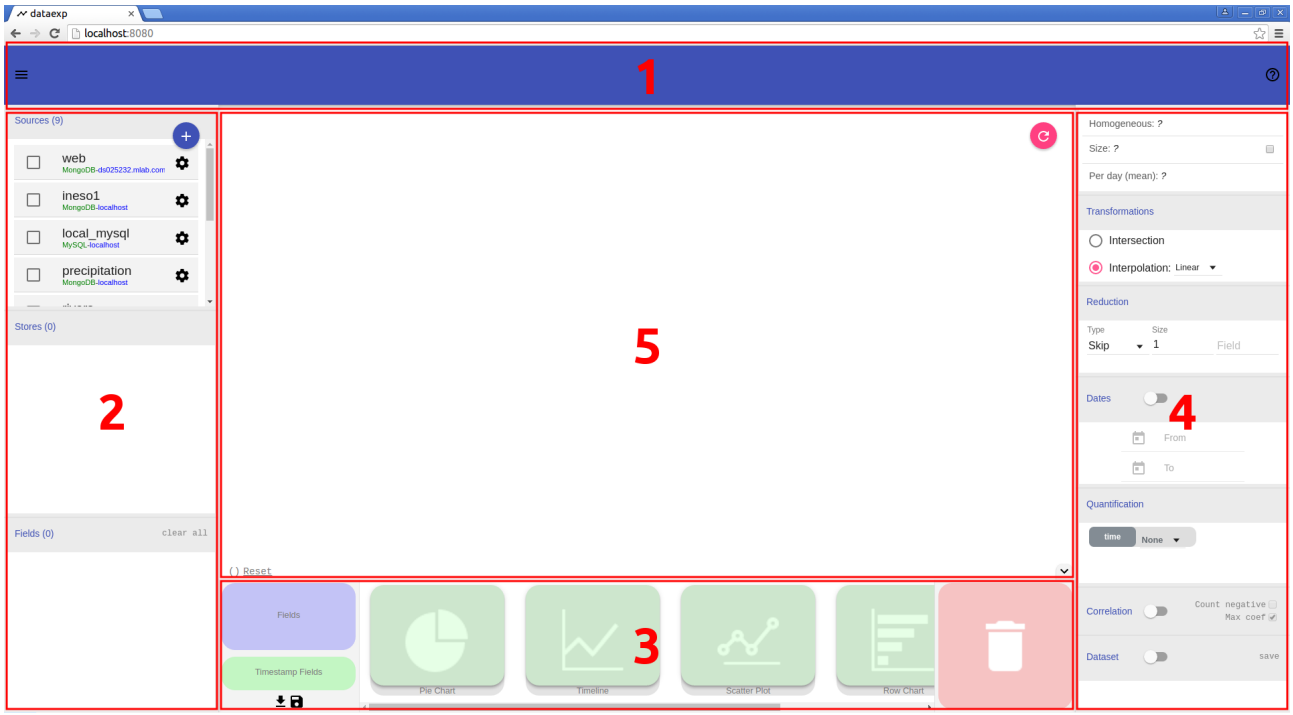


Figure 8: l'interface de l'application

graphique sectoriel, courbe temporelle, nuage des points, histogrammes horizontal et vertical. En fonction de la taille d'une ou plusieurs séries téléchargées, et les options de traitement appliquées, chacun de graphiques est rendu soit en utilisant la librairie Canvas.js soit DC.js. L'utilisation de la dernière, comme c'était dit avant, permet de mieux d'explorer les relations entre les données.

L'interface d'application se compose de 5 zones principales :

1 – **Barre d'outils**. Comporte un bouton pour cacher/montrer les zones 2, 3 et 4 afin de laisser plus de place pour les graphiques dans une zone 5. Contient également un bouton « à propos ».

2 – **Menu de chois des sources**. Il s'agit de menu où l'utilisateur doit tout d'abord choisir une ou plusieurs sources (base des données MySQL, MongoDB, etc). Ensuite, il faut choisir un ou plusieurs stores qui font partie des sources choisi (ex : tables MySQL, collections MongoDB). A la fin l'utilisateur aura un ensemble des attributs (venant de différents sources et stores) où il doit choisir certains pour être visualisés grâce aux graphiques.

3 – **Zone de choix des graphiques**. Pour choisir quels mesures (attributs numériques) doivent être visualisées, il faut les glisser une par une (via *drag&drop*) dans une plate-forme de téléchargement de la zone 3. Cet plate-forme est composée des deux aires où les attributs doivent être jetées. La première est destinée aux attributs numériques (valeurs mesurées), et la deuxième est pour les *timestamps*. La plate-forme contient également deux

boutons : un pour télécharger un *dataset* avec tous les attributs demandées et le second pour sauvegarder sous un format de fichier texte, une configuration appliquée au module *tsproc*. Cela est possible car le *dataset* final, avec toutes les attributs désirées par l'utilisateur, est renvoyée vers le front-end seulement après le passage par le module *tsproc*. Cela implique l'utilisation de configuration spécifique, configurable via la zone 4.

Ensuite, la zone 3 comporte aussi un ensemble des aires, ou les attributs téléchargés doivent être glissés par *drag&drop* depuis la plate-forme de téléchargement. Une fois toutes les mesures sont attachées à un graphique donné, il aussi, doit être glissé vers la zone 5, pour être visualisé.

Finalement, la zone 3 contient une plat-forme de suppression où toutes les objets *drag&drop* doivent être glissées pour être supprimées. En cliquant sur cet plat-forme, les zones 3 et 5 sont vidées.

4 – **Zone des options**. Une zone qui contient les options de traitement à appliquer aux séries temporelles et affiche aussi certains informations supplémentaires. Toute les options et affichages sont directement attachées à le module de traitement *tsproc*, et plus précisément, à son fichier JSON de configuration.

On va lister les traitements et fonctionnalistes possibles. Tout d'abord, les caractéristiques de la taille de série temporelle téléchargée, nombre des mesures par jour, et l'état de homogénéité sont affichées. Ensuite, les quatre traitements applicables sont :

- « Transformation » : choix de comment est-ce que les trous sont résolus dans le cas de multiples séries temporelles, par interpolation ou intersection
- «Reduction » : choix de comment sous-échantillonner les mesures
- « Dates » : permet de couper la série temporelle entre les bornées spécifiées
- « Quantification » : permet de choisir les options de quantification de valeur mesurée ou le temps

Cet zone comporte aussi une option « Corrélation » qui permet d'activer la détection de corrélation entre deux suites de mesures.

Toute en bas de panneau il y un afficheur qui visualise une série temporelle sous un forme d'un tableau. C'est une fonctionnalité secondaire, elle n'a pas autant d’intérêt pour l'exploration des données, mais peut être utile dans certains cas. La barre de titre de l'afficheur contient un bouton qui permet de sauvegarder le *dataset* finale sous forme d'un fichier JSON.

5 – **Zone des graphiques**. La zone des graphiques est un conteneur des toutes les graphiques qui sont utilisées pour l'exploration de la nature des données. Pour supprimer un des graphiques, il faut glisser son titre dans une plat-forme de suppression de la zone 3. La zone comporte aussi un bouton qui sert à redessiner toutes les graphiques.

3.4 Exemples d'utilisation

On va montrer 3 captures d'écran en expliquant le contexte et les options utilisées pour mieux exposer les fonctionnalités de l'application (Figure 9). Sur la première capture de la Figure 9, nous avons chargée deux séries temporelles, une venant de la basé des données MongoDB, l'autre de MySQL. Les deux graphiques utilisées sont une courbe temporelle et un graphique sectoriel. Ce dernier contient plusieurs secteurs de différentes surfaces car la quantification de taille 1 a était appliquée, c'est qui correspond à réaliser une opération d'arrondi sur les valeurs mesurées. Sans l'utilisation de quantification le graphique sectoriel risque d'être composée des secteurs uniformes, si chaque mesure n’apparaît qu'une seule fois.

Un autre traitement qui a été appliquée est la détection de corrélation. Les zones fortement corrélées sont visualisées par les barres au dessous de la courbe temporelle. Les barres des la même couleur et taille indiquent la séquence des mesures corrélées.

Sur la deuxième capture de la Figure 9, on explore les mêmes séries temporelles que dans le première exemple. La différence est que cet fois-ci, on force l'utilisation de Canvas.js dans toutes les cas, même si la taille d'une série temporelle finale est acceptable pour le fonctionnement correcte de DC.js. L'allure de la courbe temporelle n'est pas la même que dans l'exemple précédent, car on n'affiche que les mesures apparaissent entre les années 1920 et 1948. De plus, nous avons appliqué un traitement « Reduction » : on ne laisse qu'une instance sur deux. Ces deux techniques simples doivent être utilisées le but de diminuer la taille d'une série temporelle finale.

Une autre différence par rapport à l'exemple précédent, mais insignifiant est que cet fois, nous utilisons un histogramme verticale au lieu du graphique sectoriel, ils ont interchangeables. Aussi, nous appliquons l'interpolation cubique et pas linéaire, mais cela ne se voit pas beaucoup car les deux séries temporelles en question sont homogènes.

Sur la troisième capture de la Figure 9, les sources sont différentes : les deux séries temporelles font partie de la même base des données MongoDB. Toutes les permutations entre les types de stockages sont possibles. Dans cet exemple les graphiques ont changé aussi. On utilise un nuage des points, histogramme horizontal, et aussi une courbe temporelle. On remarque que sur le premier graphique un filtre à été appliqué, ce qui a modifiée toutes les autres graphiques, en n'affichant que les mesures correspondantes. Sur la courbe temporelle il s'agit des instants de mesure pris dans le filtre, et l'histogramme nous affiche la distribution de ces valeurs.

En bas de nuage de points, on observe une valeur qui correspond à une indice de corrélation entre les deux mesures d'une série temporelle fusionnée. Comme un filtre a était appliquée sur les points alignées sur une droite quasiment diagonale, la corrélation calculée est forte négative.

Une autre point rajoutée dans cet exemple est qu'on visualise la série fusionnée dans une table. Cela permet d'étudier les valeurs mesurées.

Figure 9: les cas d'usage

3.5 Problèmes rencontrées, futurs évolutions

La plus grand problème rencontrée est le temps d'attente. Pour de DC.js cela est résolu par l'utilisation de librairie graphique Canvas.js dans les cas ou DC.js ne sera pas à la hauteur.

Une latence de fonctionnement peut être observée lorsque l'application est utilisée longtemps sans mise à jour d'une page web. Cet problème n'est pas si pénalisant et peut être résolue juste en cliquant sur une touche F5. Par contre, le mise à jour d'une page web va effacer toutes les manipulations d'utilisateur.

Une des perspectives visible est de rajouter d'autres types de stockage et d'autres graphiques. Une autre voie d'évolution est la conception d'un algorithme de détection automatique de format de *timestamp*.

Partie 4 : Bilan de stage

1 Outil réalisé

L'application réalisée est composée des deux parties logiques indépendantes. Première partie, développée en Node.js, est un module réutilisable qui offre diverses méthodes pour le traitement des séries temporelles. La seconde partie est une application web qui tourne sous un serveur Node.js ayant un interface dynamique grâce à Angular.js. Le but de ce second module logiciel est de pouvoir charger plusieurs types des stockages des séries temporelles et offrir les graphiques interactifs de DC.js pour les explorer. L'application a été testé pour les navigateurs Mozilla (45.0.1) et Chrome (50.0.2661.75).

La plus grande obstacle à franchir était le passage d'un objectif générique vers une solution spécifique et applicable dans un monde réel. Pour le faire, pendant le chemin de développement il fallait faire plusieurs « sacrifices » : trouver des approches pratiques et réalisables plutôt que des solutions purement génériques et « idéales ». Nous nous sommes fixés de ne traiter que des séries temporelles car c'est une structure des données le plus utilisée pour le domaine d'IoT. On n'est pas parti dans le développement d'un algorithme tout puissant qui pourrai détecter le format d'un *timestamp*, mais plutôt nous imposons à l'utilisateur de le spécifier. La plus grande simplification, est de ne pas développer un algorithme générique de traitement des séries temporelles des toutes les sources possibles (types de stockage), mais plutôt de figer le traitement pour un certain format, et de transformer par la suite les autres formats vers celui l'on a fixée.

Toutes ces simplifications nous ont permit néanmoins d'arriver à un outil qui répond aux exigences initiales. L'application travaille avec les données souvent utilisées en IoT (séries temporelles), provenant de différents sources (tant qu'ils sont sauvegardées dans un type de stockage supporté par l'application). Différents formats des séries temporelles sont acceptées, il suffit juste de préciser leur structure d'organisation. L'exploration des données, et de relations entre eux, se fait grâce à une bibliothèque des graphiques interactives DC.js.

	Module de traitement (*tsproc*)	Application graphique (*dataexp*)
Node.js	954	867
Angular.js		1439
HTML		638
Documentation	234	513
Totale:	4645	

Figure 11: nombre des lignes de code par module

2 Plan personnel

Ce stage a été très bénéfique pour moi car il m'a permis de découvrir plus en détails le secteur de l'IoT, ces acteurs et contraintes. En réalisant l'application d'exploration des séries temporelles, j'ai pu développer mes compétences sur les technologies JavaScript intéressantes tel que Angular.js et Node.js. Ainsi que les librairies utiles pour l'analyse de données comme D3.js et Crossfilter.js. Ce stage m'a aussi permis de mieux comprendre mes compétences globales. Même si rester à jour de plusieurs domaines et technologies est quelque chose d'important pour moi, naturellement je m'oriente vers le développement logiciel. Et c'est une conclusion importante pour moi parce qu'elle me permet de correctement construire ma carrière future en se basant sur mes aptitudes.

Je suis très fier d'avoir pu participer à la problématique d'analyse des données dans l'IoT. J’espère que mes recherches permettrons de faire évoluer MayaNet afin d'être une solution majeure pour déploiement d'un réseaux des objets connectés.