



Stage Cool

GANZA Mykhailo
M2 GEII

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

Remerciements

Introduction

Table des matières

Introduction.....	3
Partie 1 : Présentation.....	5
1 Présentation de l'entreprise.....	5
2 Présentation de contexte de stage.....	5
2.1 IoT.....	5
2.2 Analyse des données.....	6
Partie 2 : Étude de problématique.....	7
1 Objectif initial.....	7
2 Étude initial.....	8
2.1 Séries temporelles.....	8
2.2 Problématique.....	8
2.3 Première palier.....	9
3 Première approche.....	11
3.1 Module unique.....	11
3.2 Problématique.....	12
4 Exploration des données.....	13
5 DC.js.....	15
5.1 Crossfilter.js.....	15
5.2 D3.js.....	18
5.3 DC.js.....	19
5.4 SVG.....	19
5.5 Format des données.....	20
6 Deuxième palier.....	21
7 Fusion des séries temporelles.....	21
8 Nouveau objectif concret.....	22
Partie 3 : Travail réalisé.....	23
1 Deux modules.....	23
2 Module de traitement.....	23
2.1 Choix technologiques.....	24
2.2 Première approche.....	24
2.3 Fonctionnalités réalisées.....	25
2.4 Problèmes rencontrées, futur évolutions.....	26
3 Outil de visualisation.....	27
3.1 Architecture. Choix technologiques.....	27
3.2 Librairies de visualisation.....	28
3.3 Fonctionnalités réalisées.....	29
3.4 Exemples d'utilisation.....	31
3.5 Problèmes, futurs évolutions.....	32
Chapitre 4 : Bilan de stage.....	32
Bibliographie.....	32

Partie 1 : Présentation

1 Présentation de l'entreprise

Présenter Maya

2 Présentation de contexte de stage

2.1 IoT

IoT, Internet of Things, (internet des objets, objets connectées) est une extension d'internet classique (« un réseau des réseaux ») vers le monde physique grâce aux systèmes embarqués. La plupart des temps ces systèmes réalisent l'acquisition des données de leur environnement qui sont ensuite partagées via réseaux. L'exemple d'un objet connecté peut être un moniteur de fréquence cardiaque, ou un capteur de luminosité connecté, smartphone, etc. Peu importe quel équipement électronique ayant une adresse IP, et la possibilité d'échanger des données via réseau, peut être considéré comme une « thing sdf ».

Le concept de connexion à internet des équipements électroniques pour fournir certaines données est apparu pour la première fois à 1982 à l'Université Carnegie-Mellon (Etats-Unis). Il s'agisse d'un bricolage de distributeur de Coca Cola pour sonder à distance (via réseau APRANET) la présence (ou absence) et la température des bouteilles de coca dans la machine¹. Ensuite, depuis les années 1999 le domaine des objets connectés a commencé de prendre d'ampleur avec la présentation par Bill Joy (Unix BSD) d'un protocole de communication D2D (Device to Device) : un réseau des capteurs qui vont « fusionner les systèmes embarqués avec la vie quotidienne »². En même année le terme « Internet of Things » a été inventé par Kevin Ashton³. Un autre événement important dans l'histoire d'IoT est la création en 2005 d'une carte Arduino, qui avait beaucoup d'impact sur les objets connectés⁴, surtout pour les amateurs de domotique. Depuis les années 2013 le marché d'IoT est en pleine expansion avec la croissance grandiose des smartphones, montres connectés, capteurs cardiaques, différentes sortes des objets « smart » ayant une connexion internet. Et de plus en plus des acteurs principales d'industrie numérique (Google, Samsung, etc) participent pour apporter des innovations et gagner une marge de ce secteur.

Les objets connectés ont pénétrés différents domaines remplissant divers tâches. Les plus notables sont la médecine (« santé connectée ») où les objets connectés sont utilisés pour le monitoring en temps réel des indices de santé d'un patient. Un autre domaine important d'IoT est la domotique, avec toute une panoplie des capteurs de luminosité, vidéosurveillance, frigos

¹ https://www.cs.cmu.edu/~coke/history_long.txt

² <https://www.technologyreview.com/s/404694/etc-bill-joys-six-webs/>

³ <http://www.rfidjournal.com/articles/view?4986>

⁴ <http://www.forbes.com/sites/gilpress/2014/06/18/a-very-short-history-of-the-internet-of-things/3/#540eb98143c5>

connectées, etc. Un autre exemple d'application des objets connectés sont les capteurs environnementaux, qui observent l'état de certaines grandeurs physiques (qualité de l'eau, indices de séisme) pour un but préventif et protection d'environnement

Vue le nombre des applications possibles et la quantité des objets connectés actuel (6.4 milliard objets prévus pour 2016⁵) 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 connectés pour être considéré comme des vrais « things » d'internet, doivent être standardisés, et être capables de se parler entre eux. Actuellement la communication des objets connectés 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 autre. **Les protocoles de niveau le plus haut sont différents d'une solution à autre et sont très hétérogènes.** Par exemple, la nature des données échangées par un réseau des capteurs de luminosité et une infrastructure des frigos connectés est complètement différente. Il y a un besoin d'un langage standard pour que toutes les objets connectés, malgré le fournisseur, puissent se parler entre eux. Une avancée dans cette direction est le système EPC⁶, qui en perspective pourra être utilisée comme une base commune pour développer un tel langage universel. Un langage de programmation des scripts JavaScript est vu⁷ aussi comme un langage futur de toutes les IoT car c'est un langage qui évolue très rapidement et se trouve partout sur internet actuellement : côté client, serveur, systèmes embarqués⁸.

Il est important de préciser que l'IoT ne correspond pas à une seule technologie, il s'agit plutôt d'un système des systèmes, une infrastructure de 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 de capteurs miniaturisés (ex : accéléromètre, thermomètre etc), solutions middleware, sécurité des données, etc. Une des autres aspects importants de l'IoT est l'analyse des données acquises.

2.2 Analyse des données

L'aspect d'analyse des données dans l'IoT prend de plus en plus d'ampleur. Les informations acquises ont beaucoup de valeur, car elles peuvent être analysées afin de tirer des indices importants pour la prise des décisions **stratégiques**. Par exemple, clore l'utilisation des certaines cellules dans un réseau des capteurs de luminosité à cause de la puissance négligemment 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

⁵ <http://www.gartner.com/newsroom/id/3165317>

⁶ https://fr.wikipedia.org/wiki/Code_produit_%C3%A9lectronique

⁷ <https://blog.jscrambler.com/javascript-the-perfect-language-for-the-internet-of-things-iot/>

⁸ <https://cylonjs.com/>

cardiaque mesurée par un montre connectée d'un patient.

Si on voudra concevoir une système automatisé d'analyse des données pour l'IoT on sera face aux trois problèmes les plus pertinentes: volume, approche, et l'hétérogénéité des types de stockage et formats. Vu la quantité des objets connectés et haute fréquence des mesures effectuée, une dimension des données acquis peut être assez volumineux. De se fait l'IoT se pose fortement sur les pratiques de Big Data pour la gestion et traitement des énormes massives des données. Une autre point problématique est le choix d'approche. C'est une question de recherche d'un algorithme le plus adaptée pour tirer les conclusions nécessaires à partir d'une ensemble des données acquis. Il s'agit d'un domaine des statistiques, l'utilisation des classificateurs, recherche des trends, étude de similitude, etc. Ces techniques sont très souvent utilisé dans l'IoT pour le but de l'exploration des données.

La dernière problématique, ou si on va se mettre dans un contexte d'un outil automatisée d'analyse des données, une point à considérer, est la diversité des formats et façons de stocker les données. Comme on a vu dans un chapitre précédent, le marché des objets connectés actuel se pose sur les protocoles commun de communication, mais pas jusqu'au point d'utiliser le format des données standard, c'est-à-dire même pour toutes les types des objets connectés. Pour avoir plus des exemples sur les formats des données, cf chapitre suivante. Cela se complexifié aussi par le fait que les domaines d'utilisation d'IoT sont très variées, et pour chaque cas d'utilisation la nature des données métiers sera complètement différente.

Cet heterogenite se complexifie encore plus par l'utilisation des différentes 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 des 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 on veut concevoir un outil automatique **et generique** d'analyse 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

1 Objectif initial

Des le début, l'objectif de stage est le suivante :

« Concevoir un outil de l'exploration des données et les relations entre les données ayant une nature différente provenant de sources différentes »

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 les 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.

Les problématiques de « sources différentes » et « nature différente » peuvent se conclure à trois point principaux à gérer si on veut concevoir un outil d'exploration automatisée. Il s'agit tout d'abord de désynchronisation des instants de prise de mesures entre plusieurs sources, les types stockages variées et le format des données différent. L'application doit pouvoir gérer cet nature heterogene des données.

Les concepts « exploration des données » et « relation entre les données » sont à définir par 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 sur le format et la structure des données ne sont pas imposées. Aucune autre contrainte n'est pas présentée.

Cela revient à stagiaire de choisir les technologies les plus adoptées pour résoudre la problématique posée, 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 concret et satisfaisante.

2 Étude initial

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 attendu : le stage se porte dans un domaine d'IoT, et on peut déjà y 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 apprendre le façon dont les observations provenant des objets connectées 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 des valeurs numériques qui évolue suivant le temps. La plupart de temps les séries temporelles sont représentées par une ensemble des associations clef-valeur avec une clef-valeur spécifique d'horodatage qu'on appel timestamp.

Le but d'une majorité des objets connectées 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 le temps 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.

2.2 Problématique

Comme il était dit dans un chapitre précédent, les formats des données et les types de stockage peuvent être différentes, et c'est bien le cas pour les séries temporelles. Par exemple voici une même série temporelle, mais présentée par différentes types de stockage comme les fichiers JSON (à gauche) et CSV (à droite) :

<pre>"year","flows_colorado" "1911",18.11 "1912",21.07 "1913",15.77 "1914",24.17 "1915",14.71</pre>	<pre>[{"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}]</pre>
---	--

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

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

Dans une exemple à droite on peut voir une autre problématique mineur qui se pose et peut être rencontrée lors de gestion des séries temporelles : le format de timestamp est aussi variable. La plupart de temps le timestamp suis un format standard ISO8601, ou souvent 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 un cas d'une série

temporelle avec les mesures mensuelles de niveau de précipitation. Donc, cet aspect variable des séries temporelles doit aussi être géré si on veut concevoir un outil qui pourra travailler avec plusieurs formats et sources des données.

2.3 Première palier

Issue de cet étude initial 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 datasets 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 mesure)
- les timestamps doivent suivre un ordre chronologique

Bien sûr, même si les séries temporelles est une façon très répandue de stockage des observations, il peut y avoir d'autres solutions pour organiser les données. Évidemment 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 répandue
- pour arriver à un outil réel, il faut spécifier ces fonctionnalités

Issue de cet étude initial on a un point d'angle commun pour toutes les types de stockage et formats possibles: les données sont représentées par les séries temporelles.

2.4 Terminologie

Vue qu'on se fixe ne travailler qu'avec les séries temporelles, afin d'éviter la confusion terminologique, on va définir explicitement certaines expressions clés. Sur la Figure 1, 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'un ensemble des « instances ». Une instance, ou échantillon dispose d'un ensemble des attributs. Il y a deux types des attributs : attribut d'horodatage (timestamp) et un attribut numérique (une mesure, valeur mesurée).

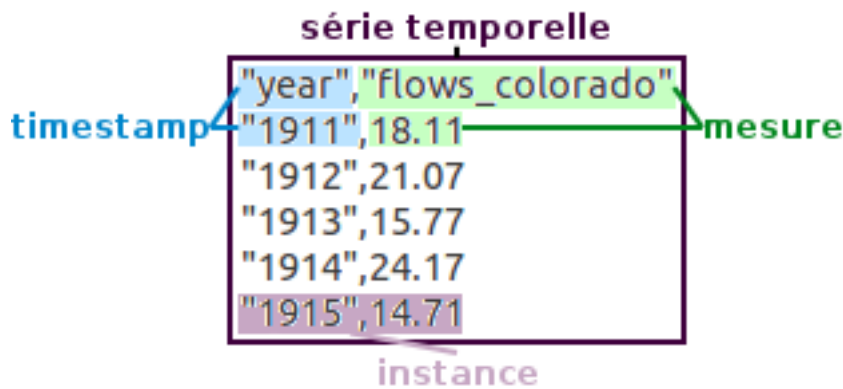


Figure 1: série temporelle

3 Première approche

En se basant sur le premier mini étude sur les séries temporelles, la première application naïve est apparu.

3.1 Module unique

De le début un outil à développer était vu comme un seul unité de traitement, qui peut recevoir en entrée des séries temporelles provenant de différents types de stockages et ayant des formats différent, 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 une problème a était proposée. L'application était écrit sous Java, utilisée Morphia, et travaillée avec MongoDB.

MongoDB

Le choix de MongoDB, comme un type des stockage de base, est argumenté par le fait que c'est une base des données orientée document, et par rapport à les bases de données classiques, 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 formats, parce que cet base de données peut facilement travailler avec les format différentes des données. ~~La raison est que la structure des données stockée dans cet base des données n'est pas fixe (). car elle ne nécessite pas la structure des données rigide.~~

Aussi, MongoDB est une base de données sur laquelle MayaNet vise de migrer en future (encore une fois, à cause de dynamisme de format). Cet base de données est en perspective un seul et unique type de stockage qui sera utilisée par MayaNet.

Vue ces deux arguments, la première application à stockée les séries temporelles dans une base des données MongoDB.

Java

L'utilisation de Java comme une langage de programmation a était aussi

influencé par MayaNet car elle utilise Kura⁹. Kura est un framework Java « basé sur OSGi pour les applications M2M s'exécutant dans les passerelles de services »¹⁰. Il s'agit d'une solution logicielle faisant un intermédiaire entre réseau filaire (ou sans fil) des objets connectés (capteurs, etc) attachés à un concentrateur et un réseau local (ou internet).

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

Morphia

Encore une fois, l'utilisation de Morphia était argumenté pour être compatible avec MayaNet. MayaNet utilise les objets POJO pour définir les données métiers. POJO (Plain Old Java Object) est une simple classe Java, qui par rapport aux JavaBeans, ne suit pas des conventions strictes.

Morphia est un wrapper de driver Java pour MongoDB qui s'en sert des POJO (en ajoutant quelques annotations) pour communiquer avec une base de données. Morphia est une 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, peu importe quel type d'objets connectés que MayaNet pourra servir, il faudra créer des objets métiers (des POJO). Donc, comme les POJO sont créés dans tous les cas, pour toutes les « things », un outil automatisé d'analyse des données pourra utiliser Morphia comme une interface entre peu importe quel format des objets métiers et la base des données.

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

Récapitulatif

La première tentative naïve de répondre à une problématique posée est largement inspirée par les choix technologiques de MayaNet, pour un but d'être compatible avec la dernière. C'est une application qui est capable de travailler avec les formats des données différentes grâce à l'utilisation de Morphia et MongoDB. Le but d'une étape de développement qui devez suivre est d'englober cet l'application par une couche de traitement des données ~~qui sera aussi uniforme pour toutes les formats possibles des données.~~

3.2 Problématique

Le première approche de développer un outil d'analyse des données a permit de se familiariser avec la problématique de multitude des formats et avoir une première initiation à les données réels. Mais la solution proposée

⁹ <http://www.eclipse.org/kura/>

¹⁰ <http://www.electronique-mag.com/article8518.html>

n'était pas du tout satisfaisante à cause des plusieurs points.

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

Une autre problème évidente de première approche c'est le non respect d'une seule consigne imposée : l'utilisation d'au moins deux types de stockages : 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 deux¹¹. Donc cet approche à résoudre la problématique de multitude des formats et stockages n'est pas valable.

Mais le bien issue de cet première approche, est une réflexion sur le façon comment l'utilisateur pourrai explorer les données, et surtout, les relations entre les données. Cet fonctionnalité n'a pas était rajouté à cet premier application parce que, évidemment, l'application ne correspond vraiment à l'objectif posée, mais encore une fois, elle a permit 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 une acte d'observation et analyse des données acquis, afin de tirer des informations supplémentaires sur les données. En contexte de 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 juste simple visualisation par une table, ~~c'est qui est aussi une manière d'exploration des données sous forme des valeurs numériques.~~ Il va falloir fixer un sens concret pour se terme afin d'apporter une solution réel.

Il y plusieurs façons pour analyser et explorer les données, mais on va étudier seulement les deux les plus communs et convenables. Tout d'abord, l'utilisation des techniques statistiques avancées de machâge des données pour y 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 afin d'être facilement analysables par l'humain. Les deux approches ont ces défauts et avantages.

~~En fait, deux façons sur lesquelles la question est : est-ce que l'outil qu'on vise à développer va posséder des techniques avancées de mâchage des données pour y tirer de connaissances, ou l'outil va visualiser les données de tel sorte, pour mettre en évidence leurs caractéristiques importantes. Chaque approche à ces défauts et avantages.~~

¹¹ <http://hibernate.org/ogm/faq/>

Les techniques statistiques qu'on peut utiliser pour explorer les données, et surtout relation entre les données, peuvent être simples comme recherche de max ou min, filtrage, moyenne glissante. Mais ils peuvent être aussi avancées comme, par exemple, l'étude des trends, l'utilisation des estimateurs, techniques d'extrapolation, classification, etc. L'avantage d'utilisation des méthodes statistiques est qu'ils sont très performantes pour trouver du valeur cachée dans les données. La 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. Surtout ~~pour quelqu'un non-spécialiste de domaine~~. Par contre, rien n'empêche d'appliquer au moins les caractéristiques statistiques simples. **pour tirer plus d'information à partir des données.**

La façon la plus simple et évidente, pour interpréter et explorer les données est l'utilisation des graphiques de différent sorte comme les histogrammes, nuage des points, etc. Les avantages et défauts de cet 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 relation entre eux, n'est pas autant avancée.

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

- question de temps de développement
- compétence de stagiaire
- l'utilité pour un cas d'usage réel
- découverte de DC.js (chapitre suivante)

Vue toutes ces points, et surtout les aptitudes naturelles de stagiaire, l'exploration des données sera réalisée grâce à un outil de visualisation des graphiques dynamiques DC.js (qui en quelque sorte, permet de mettre en évidence la relation entre les données) et quelques statistiques simples à mettre en place pour fournir plus d'information. Cet solution est un mix entre les deux approches exprimées ci-dessous car on utilisera une visualisation de graphiques, mais pas simple et avancée, et on va utiliser certains techniques statistiques, mais pas avancée et simples. Et surtout, c'est une solution le plus réaliste et possible à produire pour fournir un outil fonctionnelle.

5 DC.js

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

5.1 Crossfilter.js

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

Pour expliquer l'intérêt d'utilisation de Crossfilter.js on invite à consulter une page officielle de crossfilter¹² avec une exemple d'utilisation concret qu'on va exposer ici. On précise que le chargement de cet page pourra prendre quelques secondes. Un autre remarque à noter et que les graphes ont été générées via D3.js et sont cliquables (expliquée par la suite).

L'exemple est construit à partir d'un extrait de dataset¹³ 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 étudiées : heure de départ, retard à l'arrivée (en minutes), distance parcouru (en milles) et une date de départ.

Chaque attribut étudiée est représenté par une histogramme. Sur chaque histogramme il est possible de choisir une plage des valeurs, c'est-à-dire appliquer un filtre. 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 2):

- quels sont les journées de la semaine avec les vols les plus longues ?
- quels sont les distances de vols à choisir (imaginons que c'est ce qu'on cherche), pour avoir le moins de retard ?
- quels heures de départ il faut choisir pour avoir le plus de retard (si on aime bien passer du temps en avion ou aeroport, 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 peut importe quels attributs au sein de même instance sont explorables par cet librairie.

La puissance de Crossfilter.js et que les filtres sont appliquées en temps réel, c'est qui implique que lorsque les données en entrée ont été modifiées, les caractéristiques calculées (la sortie) seront modifiées instantanément. Et cela permet d'explorer l'influence de chaque mesure sur le résultat final. Par illustrer cela, prenons un exemple d'une série temporelle, et on va utiliser crossfilter pour compter le nombre d'occurrence de chaque valeur mesurée (Figure 3). A l'entrée on a un dataset avec les données à analyser, en sortie on a un dataset avec le résultat d'analyse. Si en instant t les données en entrée sont modifiées (on appliqué un filtre), le dataset en sortie sera

¹² <http://square.github.io/crossfilter/>

¹³ <http://stat-computing.org/dataexpo/2009/the-data.html>

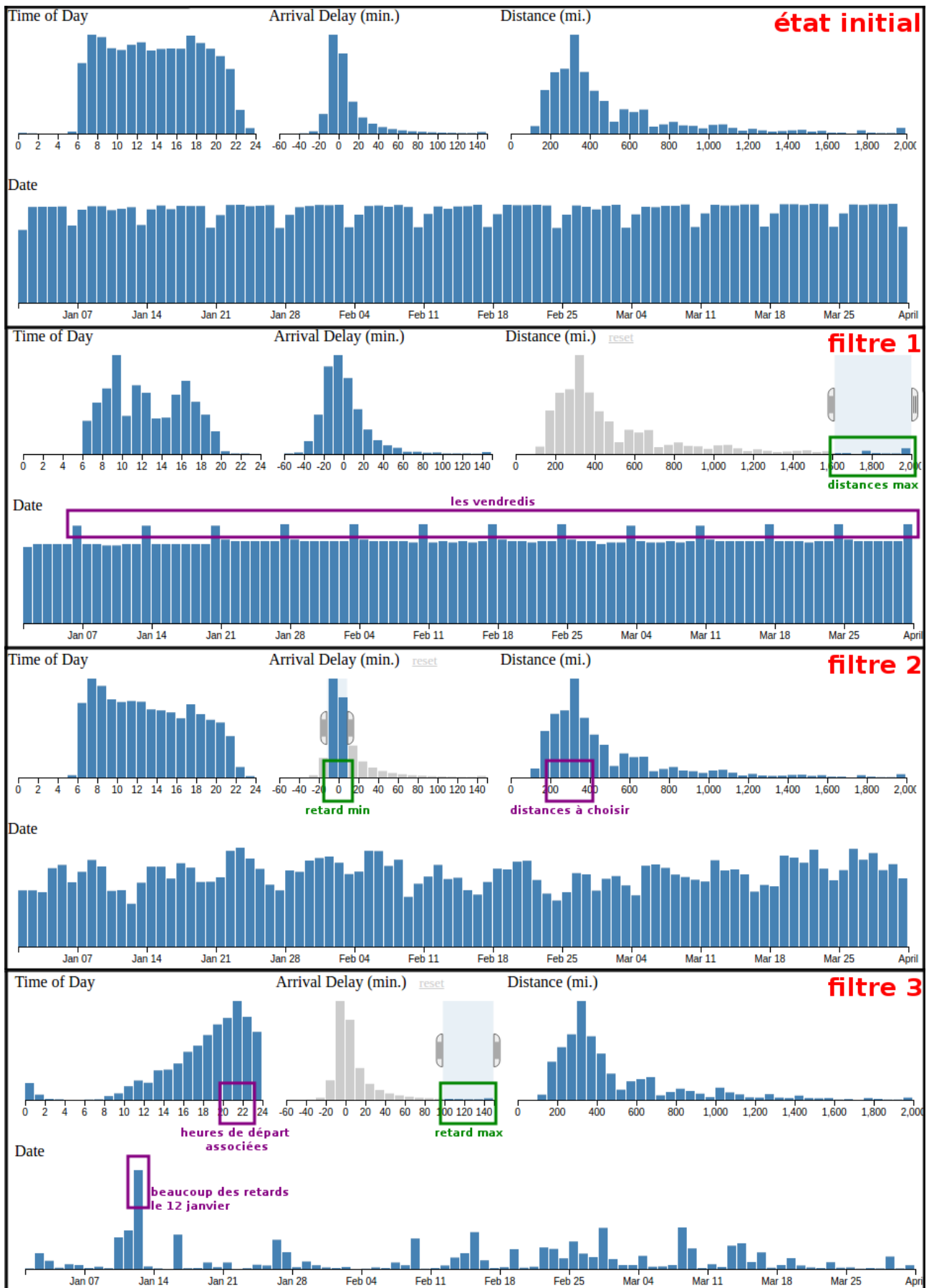


Figure 2: l'exemple de Crossfilter.js rendu avec D3.js

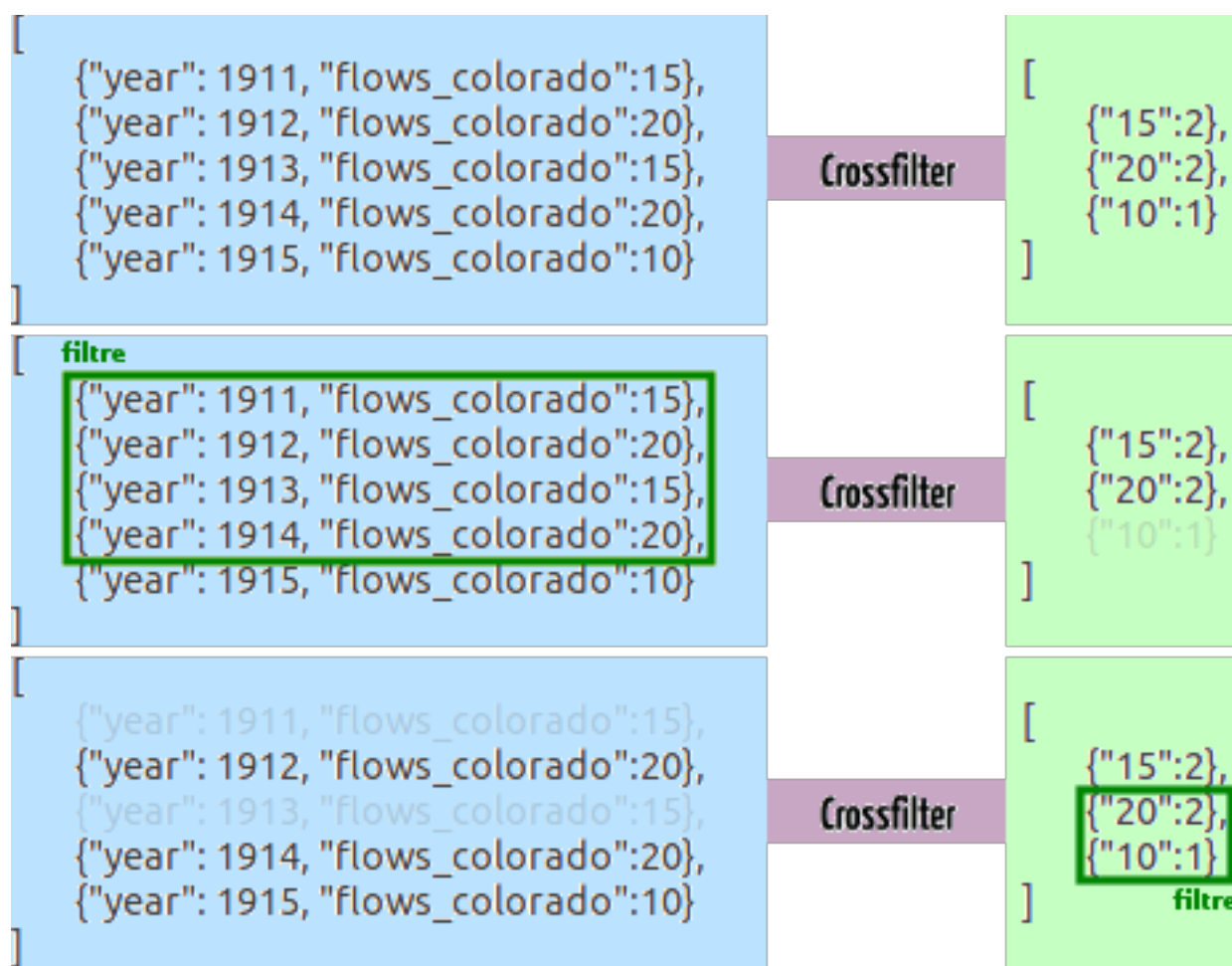


Figure 3: filtres de Crossfilter.js instantanément modifiée.

Comme il était dit avant, crossfilter support les filtres multidirectionnelles, c'est-à-dire dans le cas de cet exemple, on peut appliquer un filtre sur les caracteristiques (les sorties), pour voir comment les données à l'entrée vont être changées.

En conclusion, Crossfilter.js est un outil puissante d'analyse des massives des données, mais il ne s'agit pas d'une bibliothèque graphique, c'est plutôt une machine d'exploration des données. Pour visualiser les relations dynamiques mis en évidence par crossfilter, il faut utiliser une autre librairie JavaScript de graphisme. Il est possible d'utiliser peu importe quel bibliothèque de choix, mais pour tirer le max de Crossfilter.js, il faut considérer deux points importantes: l'interfaçage et dynamisme.

Comme Crossfilter.js travaille directement avec les données (dataset en entrée et en sortie), si on veut l'utiliser avec une librairie de rendu visuel, la dernière doit être capable de générer les graphes directement (ou avec des efforts minimales) à partir des données. Donc, l'interfaçage avec corssfilter se traduit par l'interfaçage avec les données. Un autre point est le dynamisme. Pour se profiter d'utilisation des filtres interactives de crossfilter, la bibliothèque graphique doit supporter de transitions, morphing, passage des objets visuelles d'un état à un autre. En plus, comme l'application des filtres crossfilter 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 un première exemple de ce chapitre), mais il peut être utilisée avec d'autres librairies graphiques comme Chart.js¹⁴ ou NVD3¹⁵.

5.2 D3.js

D3.js (Data-Driven Documents) est une bibliothèque JavaScript graphique qui permet la visualisation des données numériques en forme des graphiques dynamiques. Elle est conforme W3C et utilise les technologies SVG et CSS. Les point forts de cet librairie graphique est dynamisme (les objets graphiques peuvent réaliser des transitions) et le fait qu'elle est « 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 organisées dans une hiérarchie qu'on appel DOM. Les nœuds de la DOM peuvent être modifiées par plusieurs langages de programmation c'est qui va impacter la page html.

D3.js utilise des SVG pour la génération des objets visuelles. Les SVG sont les images vectorielles, et chacun élément d'image fais partie de DOM. C'est qui permet de les modifier facilement et c'est la raison pourquoi 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 de 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 ont était 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 rendering des objets visuelles. On invite à consulter la page principale de D3.js¹⁶ pour se rendre compte de toutes les domaines d'application de cet bibliothèque graphique. Cet flexibilité de génération des objets visuelles 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 importante.

5.3 DC.js

DC.js (Dimensional Charting) et basé sur deux bibliothèques JavaScript : Crossfilter.js pour filtrage multidirectionnelle 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 DC.js.

Tout d'abord, la courbe d'apprentissage de D3.js est exponentielle, il faut

¹⁴ <https://github.com/nsubordin81/Chart.dc.js/tree/master>

¹⁵ <http://stackoverflow.com/questions/19094015/crossfilter-d3-brush-and-nvd3-integration>

¹⁶ <https://d3js.org/>

passer beaucoup de temps pour apprendre comment l'utiliser pour rendre des graphiques simples. De plus, comme D3 travail avec les objets graphiques simples comme rectangle ou cercle, et de plus, les sélecteurs, la mise en place des graphes peu être assez complexe. Bien sûr le temps fourni à cet apprentissage va permettre à développer des visualisations splendides, complexes et spécifiques, comme par exemple celle-ci¹⁷, réalisée en pure D3.js. Mais comme en chapitre 2.3 on a décidé de travailler qu'avec les séries temporelles, pour les analyser il nous suffit d'une 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 et il sera visualisée.

Donc, DC.js est un autre pierre d'angle, un milieu sur lequel on se base pour « explorer les données », et les « relations entre les données ». En langage DC.js cela se traduit par l'exploration des séries temporelles, et des relations entre ces attributs.

5.4 SVG

DC.js est basée sur D3.js, et comme on a dit avant, elle utilise des SVG pour générer les objets visuelles. La puissance de SVG vient de fait que c'est une image vectorielle, et chaque son composant et référençable via DOM. C'est qui permet de mettre à jour facilement certains éléments d'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 milliers des éléments vectorielles, la génération sera ralenti. Donc cela peut engendrer de la latence lorsque nombre des formes à dessiner est importante.

Pour illustrer ceci, on peut comparer¹⁸ SVG avec un autre technologie souvent utilisé pour les images : les canvas. Pour le DOM, canvas représente un trou avec les 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'une 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 visuelles 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 lesquelles travaille Crossfilter.js, et donc DC.js, est une table des JSON. JSON, comme XML, est un format des données textuelles. Il comporte des paires clef-valeur, listes des valeurs. Si on stocke une série temporelle avec une ensemble de JSON, chacun document JSON doit comporter un timestamp et une ensemble attributs numériques.

¹⁷ <http://www.nytimes.com/newsgraphics/2013/09/07/director-star-chart/>

¹⁸ <http://smus.com/canvas-vs-svg-performance/>

Une précision importante à noter à propos de crossfilter 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 (voir figure chepa, en haut la) 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 vaut dire que crossfilter n'accepte qu'une seule table de JSON. Et donc, comme dans un exemple de la Figure 4, il va falloir fusionner plusieurs séries temporelles en une, si on veut pouvoir ces attributs. Plus de détails sur les aspects subtiles de cet fusion sont précisée dans un chapitre 7.

6 Deuxième palier

Le premier palier a figée les données avec lesquelles l'application va travailler. Il s'agit des séries temporelles. La deuxième vise à définir un terme « exploration des données » et la définition de « relation entre » les données.

Et donc, comme on a décidé dans un chapitre 4 de la Partie 2, l'exploration des séries temporelles vaut dire l'exploration visuelle. On va 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 mis en évidence grâce à les filtres des Crossfilter.js qui fais partie de DC.js.

7 Fusion des séries temporelles

Si on se fixe sur l'utilisation de DC.js, et on veut se profiter de ces capacités pour comparer plusieurs séries temporelles, il va falloir les combiner de sorte pour en avoir qu'un seul. Une façon de le faire est de réaliser une fusion horizontale, un peu comme dans une Figure 4.

Cet 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 connectés réalisent les mesures, la période de la prise de mesure peut varier. De plus, il se peut que les timestamps d'une séries temporelles soient complètement désordonnées (c'est-à-dire que les mesures sont prises irrégulièrement dans le temps). Et donc, si on veut comparer les deux suites des mesures provenant de différents sources, on risque de réaliser des fusions fausses si on ne respect pas la correspondance de timestamp entre les deux séries temporelles.

Donc il faut traiter toutes les séries temporelles avant de les fusionner, afin qu'ils soient synchronisées. Les deux séries temporelles sont synchronisées (ou homogènes) s'ils ont le même nombre d'instances, et les timestamps des

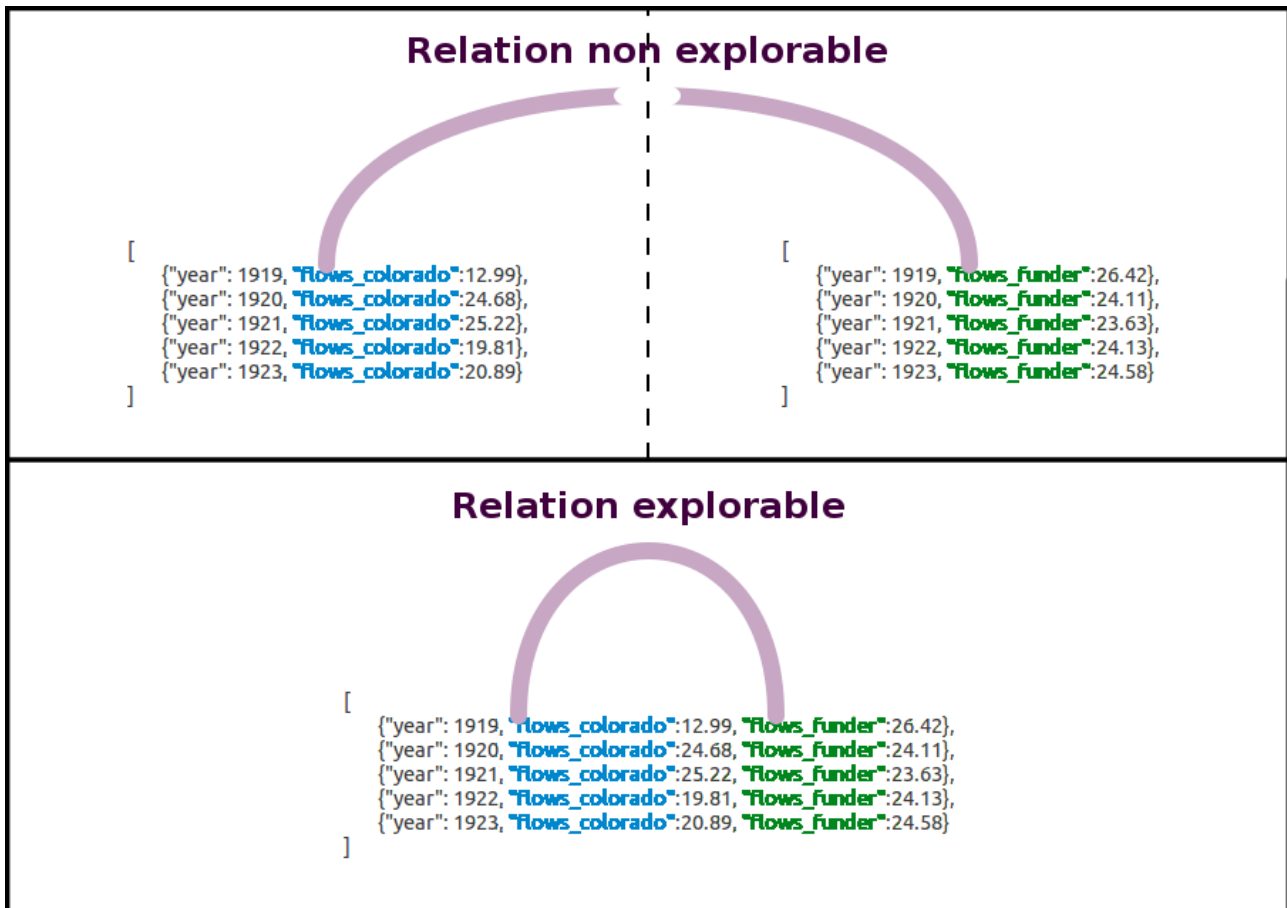


Figure 4: fusion des séries temporelles

instances adjacentes sont égaux. C'est-à-dire avant de fusionner toutes les séries temporelles, il faut s'assurer tout d'abord qu'ils ont tous les mêmes instantes de prise de mesures. Et cela est rarement le cas si les séries proviennent de différents sources. Donc, il va falloir prétraiter les données, pour qu'ils soient homogènes.

Il y a plusieurs façons à le faire. Par exemple, entre une ensemble des séries temporelles, on peut un laisser seulement les mesures qui ont été faite à les instants inclut dans toutes les séries temporelles. Ou on peut passer par l'interpolation des points (mesures) manquant d'une série mais présentes dans une autre. Ou on peut remplacer chaque trou (instance manquante) par des attributs zéro. Ou choisir de ne pas accepter à traiter les suite des mesurer qui ne sont pas compatibles.

Cet question de homogénéité fais partie de problématiques auxquelles le stagiaire doit répondre et trouver la solution la plus approchée de l'objectif initial.

8 Nouveau objectif concret

Un nouveau objectif est une itération d'un objectif initial qui apport plus de concret sur c'est qui est à accomplir. Les termes « données » et « exploration » ont été éclairci. Aussi, l'approfondissement de la problématique a permis d'ajouter certains nouveaux éléments. Voici donc le nouveau objectif détaillée:

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

Partie 3 : Travail réalisé

1 Deux modules

Pour répondre au mieux aux exigences de cahier de charges et maximiser la ré-utilisabilité de l'application, on a décidé de le séparer en deux modules indépendantes. La première module est une boîte à outil de traitement des séries temporelles. Deuxième est une application graphique qui englobe le premier module et offre la possibilité de travailler avec plusieurs types des stockages des séries temporelles et permet de les explorer grâce à les graphiques de DC.js.

Les raisons pour choisir une tel architecture sont nombreux. Tout d'abord, il y a un besoin de traiter plusieurs formats et sources des données. Pour répondre à cet contrainte, une solution logique est de développer un brick principale, qui réalise le traitement d'un certain format des séries temporelles fixée, et ensuite, concevoir des convertisseurs pour chacun format spécifique, autre que supportée par le brick. Cet approche est beaucoup plus optimisée que d'avoir un traitement spécifique pour chacun format des séries temporelles. Et aussi, de point de vue d'évolutivité du code, cet architecture est beaucoup plus accueillante pour les modifications et améliorations, car les algorithmes de traitement des séries temporelles sont détachées de format des données.

Un 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 une 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 un autre librairie graphique que DC.js sera choisi pour le rendu des graphiques, dans ce cas, si l'application qu'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 tout seule ne remplit pas toutes les exigences de cahier des charges, c'est une partie de travail réalisée qui risque d'être utilisée le plus souvent (parce qu'il peut être appliquée pas seulement pour l'exploration des séries temporelles). Et donc, l'application graphique en quelques sorte, joue un rôle d'un exemple de configuration de module de traitement. Et de même, l'application graphique fourni aussi les exemples de mise en place des graphiques de DC.js.

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 une ensemble des séries temporelles, un JSON avec la configuration de cettes séries, et offre plusieurs façons de les traiter.

De ça conception le module était conçu pour pré-traiter les données avant de les servir à DC.js. C'est la raison pourquoi il contient certains fonctionnalités spécifiques comme l'interpolation ou fusion horizontal des datasets, nécessaires pour DC.js. Mais il est fini par être une boîte à outil pour

les séries temporelles. Actuellement l'utilisation de ce module ne se limite pas exclusivement pour servir les graphiques.

2.1 Choix technologiques

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

Les séries temporelles sont stockées par une table des documents JSON, et c'est la seule format acceptée par ce module de traitement. Les autres formats doivent être transformés pour être acceptés par le module. JSON est utilisée car c'est un format accepté par crossfilter. De plus il est natif à JavaScript mais indépendant de langage de programmation et s'utilise partout actuellement.

Même si on fixe la forme sur laquelle les séries temporelles seront stockées (une suite de 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 différents. Il s'agit de la problématique posée de le début de multitude des sources. Pour la résoudre, le format de chaque série temporelle (description de ces attributs) doivent être passé à l'entrée de 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.

Le module est une application Node.js et peut être inclus dans les autres projets grâce à l'injection de dépendance.

2.2 Première approche

Delta

Comment on a vu dans un chapitre 7 de la Partie 2, pour être capable d'explorer plusieurs séries temporelles par DC.js, ils doivent être homogénéisés et fusionnés. 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 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'une delta donnée vont être composées par une certaine opération mathématique (calcul de somme ou moyenne), à une seule mesure. À l'issue de cet algorithme toutes les séries temporelles auront 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 relevés de la pluie annuels (et donc si le delta est égal 5 ans, on somme les relevés et ça va dans le sens), mais pour les séries temporelles où les mesures ne sont pas cumulatives (par exemple les niveaux de tension hebdomadaire) cela n'a pas beaucoup d'intérêt. Et surtout l'utilisation de delta n'a pas passé le premier test d'un exemple réel d'application.

Exemple réel

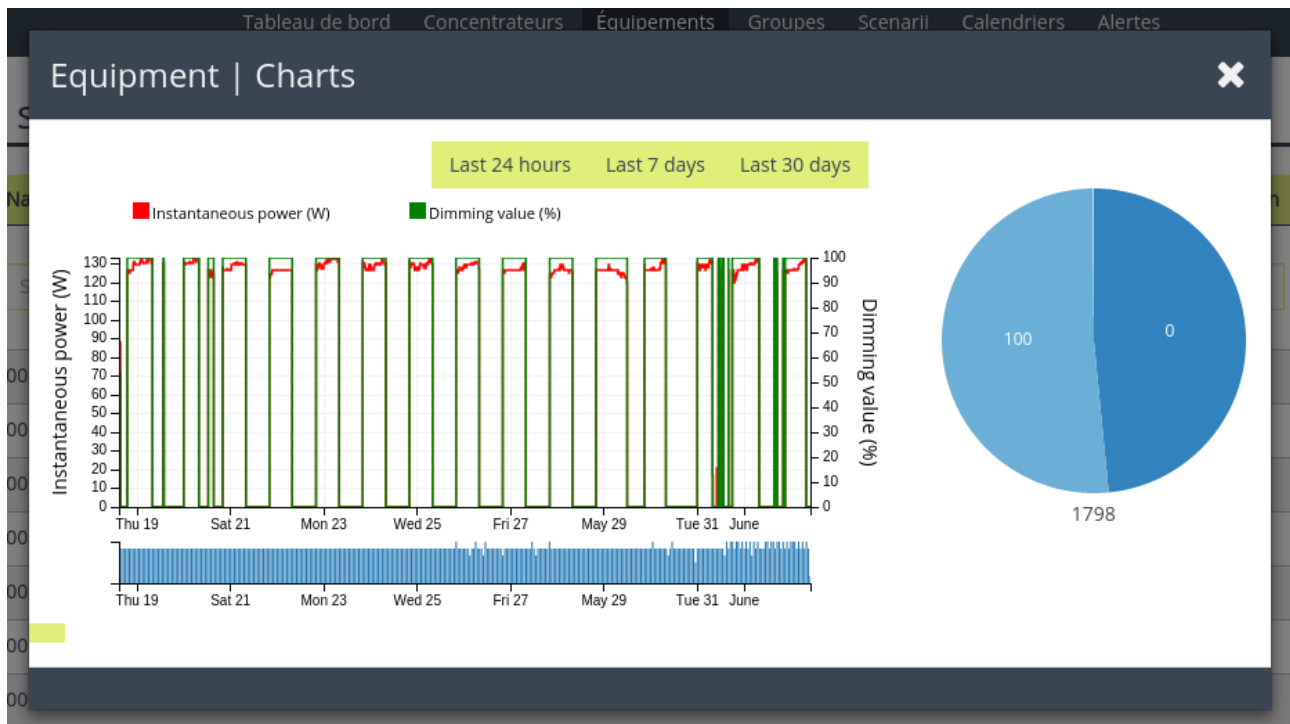


Figure 5: exemple réel

La continuation sur la voie de développement d'utilisation de la technique de « delta » comme une solution pour synchroniser les séries temporelles a été interrompu 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 a montée en compétence sur l'utilisation des bibliothèques JavaScript graphiques pour visualisation des mesures venant des objets connectées, cet tâche lui était attribuée. Cet simple mission a relevé plusieurs problèmes qui doivent être traitées impérativement.

Un 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 on veut 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 y avait qu'une seule), l'inconsistance de l'approche de « delta » est devenu évidente car il 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 les données réel d'IoT a permis de se rendre compte de problème de conception initial et poussée la réflexion sur autres façons comment rendre plusieurs séries temporelles homogènes. Un aspect positive est que on se 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 côté positive est que la résolution de toutes ces problèmes a permit d'apporter la solution fonctionnelle qui a été 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 une ensemble des 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 une ensemble des méthodes publiques de traitement des séries temporelles. Leur comportement est configurée à l'instanciation de module via le JSON de configuration. On va lister et expliquer ici quelques méthodes principales:

- **isHomogeneous()**: une méthode qui renvoi vrai si une ensembles de séries temporelles sont homogènes, et faux sinon ; les deux séries des mesures sont considérée homogène si toutes les mesures adjacentes ont était fais aux mêmes instants, c'est-à-dire que les deux séries ont les timestamps identiques
- **cut()**: sert à découper la série temporelle et laisser seulement les mesurées entre les bornes précisées dans le config ; l'intérêt de cet 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 mesures consécutive par une grâce à une opération mathématique spécifiée dans un JSON de configuration
- **interpolate()**: si une ensemble des séries temporelles n'est pas homogène, la méthode réalise l'interpolation des toutes les mesures manquantes ; issue de cet méthodes, les séries deviennent homogènes ; cet méthode est utile pour préparer les donnees avant de faire une fusion des séries temporelles
- **merge()**: réalise le fusion des plusieurs séries temporelles vers une, réalisable seulement s'ils ont 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
- **toISO()**: passe toutes les timestamps des 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 toutes les 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 ISO.

2.4 Problèmes rencontrées, futur évolutions

Les problèmes principales rencontrées lors de développement de cet 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 traitement des séries temporelles avec des attributs nominales. Par exemple l'interpolation d'une suite des valeurs nominaux est une problème assez complexe et nécessite probablement application des techniques de machine learning, ou méthodes avancées d'interpolation¹⁹. Et de plus, à part la problème avec l'interpolation, il y a d'autres méthodes qui sont concernées.

Donc, actuellement, s'il y a un besoin de travailler avec les séries temporelles nominaux, il faut qu'il soient homogènes.

Conclusion ?

3 Outil de visualisation

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

Aucune contrainte, à part le support au moins des deux types de stockage MySQL et MongoDB, n'a pas était imposée. Le choix de design et des technologies était prit par le stagiaire.

3.1 Architecture. Choix technologiques

L'application se décompose de deux tiers principaux (Figure 6) coté serveur (back-end) et client (front-end). Coté client se charge de la visualisation des graphiques DC.js et Canvas.js (les raisons d'utilisation de cet bibliothèque sont expliquée dans un chapitre suivante), visualisation des menus de choix des stockages et les options de traitement des séries temporelles.

La vue de client a été développée via les technologies WEB notamment HTML, CSS et Angular.js²⁰. La dernière est une framework JavaScript qui facilite la création des pages web dynamiques. De plus, on a utilise un module d'angular Angular-Material, qui facilite aussi le rendu visuel des éléments d'interface et impose les spécifications de Google Material Design²¹. L'interfaçage avec le back-end se fais grâce à les services angular, qui eux utilisent un API REST²².

¹⁹ <http://act.sagepub.com/content/1/2/201.full.pdf+html>

²⁰ <https://angularjs.org/>

²¹ <https://material.google.com/#introduction-goals>

²² https://fr.wikipedia.org/wiki/Representational_state_transfer

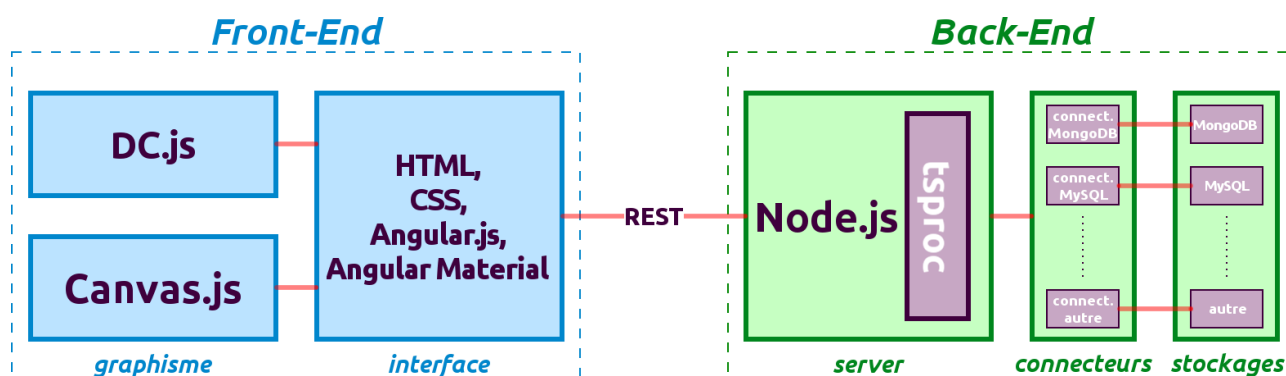


Figure 6: architecture d'application graphique

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 les quarts opérations identiques pour toutes les types de stockage:

- lecture de la liste des stores dans un stockage ; en langage SQL cela correspond de lire noms de tables dans une base des données, ou les noms des collections dans une base des données MongoDB
- lecture de nombre des documents dans un store (nombre des lignes dans un table pour MySQL, ou nombre des documents dans une collection pour MongoDB)
- lecture des attribut 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 toutes 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 acceptée par tsproc)

3.2 Bibliothèques de visualisation

L'utilisation de la librairie graphique DC.js est primordial 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, cet 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 fais, et donc on ne peut pas la remplacer, DC.js dois forcément faire partie de l'outil qu'on était censée de réaliser. En gardant cela en tête, il y a deux solutions à résoudre cet problème, et les deux comportent la réduction de nombre des mesures dans une série temporelle. Cela est inévitable si on veut se profiter des fonctionnalités de DC.js.

La première approche consiste à ne pas autoriser l'utilisateur de visualiser des graphiques tant que la taille d'une série temporelle dépasse un certain seuil. L'utilisateur sera obligée d'appliquer les méthodes offert par un module tsproc (choix d'une fenêtre des dates, sous échantillonnage, etc) pour

diminuer le nombre des mesures dans une série temporelle. Cet approche garantit que DC.js ne va présenter de latence et que les données seront explorées efficacement. La problème qui s'impose est de sphère de « user experience », car on impose à l'utilisateur de faire beaucoup des manipulations avant même de commencer de s'initier à les données.

Donc, une autre alternative, et celle qu'on a choisi, 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, en on a choisi Canvas.js parce-qu'elle contient les figures qu'il nous faut, simple à mettre en place, et fais correctement c'est qu'elle est censé de faire.

A l'inverse de DC.js, Canvas.js²³ 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 référencable 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 nombres 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, on utilise Canvas.js lorsque le nombre des mesurées dans une série temporelle dépasse un seuil acceptable pour une bonne utilisation de DC.js. Et si l'utilisateur veut se 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 on va exposer les éléments de l'interface d'application et ces principaux fonctionnalités offertes. Actuellement l'outil travail avec quatre types des sources (stockages) des séries temporelles : MongoDB, MySQL est les fichiers JSON et CSV.

L'application offre la possibilité d'explorer les données via 5 graphiques : graphique sectoriel, courbe temporelle, nuage des points, histogramme horizontale et verticale. En fonction de taille d'une ou plusieurs séries téléchargée, 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.

²³<http://canvasjs.com/>

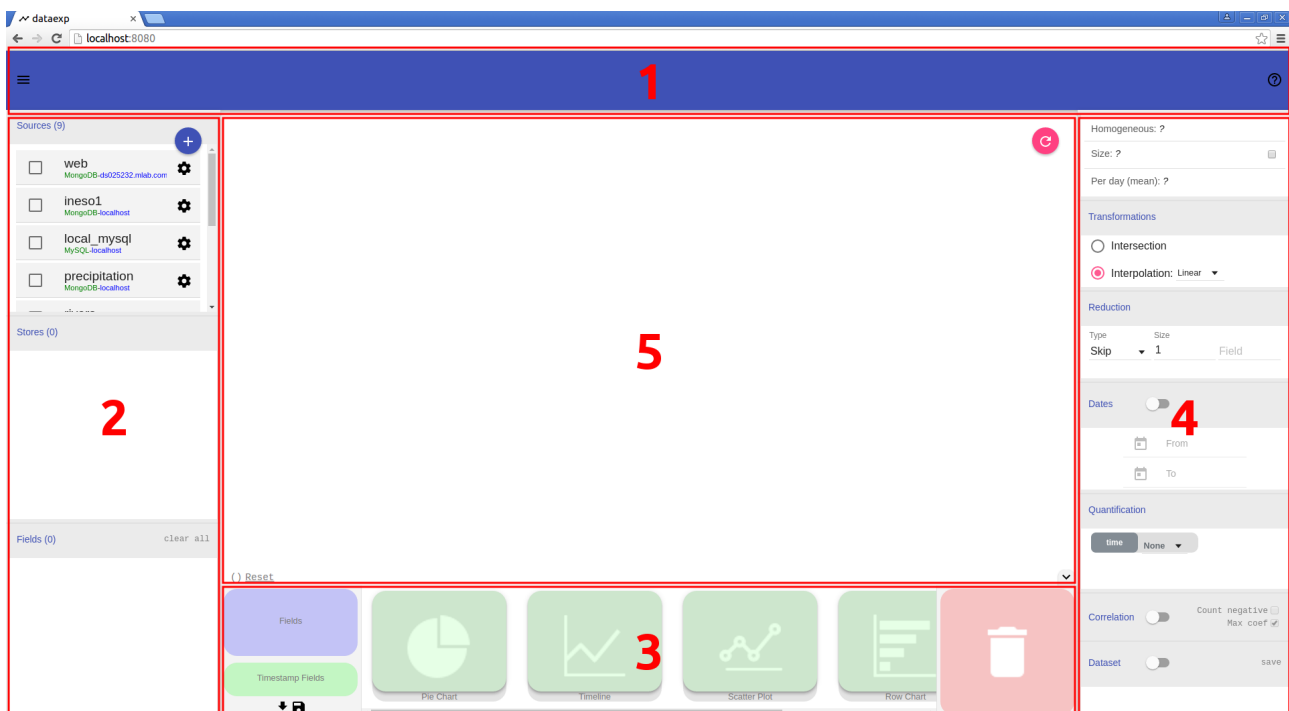


Figure 7: l'interface de l'application

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

1 – **Barre d'outils.** Comporte une 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 une bouton « à propos ».

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

3 – **Zone de choix des graphiques.** Pour choisir quels mesures 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 mesurée), et deuxième est pour les timestamps. La plate-forme contient également deux boutons : une pour télécharger un dataset avec toutes les attributs demandées et seconde pour sauvegarder sous un format de fichier texte, une configuration appliquée à le module tsproc de traitement. La raison est que, les dataset finale, avec toutes les attributs désirées par l'utilisateur, est renvoyée vers le front-end seulement après le passage par le module tsproc. Et cela implique l'utilisation de configuration spécifique, configurable via la zone 4.

Ensuite, la zone 3 comporte aussi une ensemble des aires, où les attributs téléchargées doivent être glissée par drag&drop depuis la plate-forme de téléchargement. Une fois toutes les mesures sont attachées à un graphique spécifiée, il aussi, doit être glissée vers la zone 5, pour être visualisée.

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 vont être vidé.

4 – Zone des options. Une zone qui contient les options des traitement d'appliquer à les 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 de configuration.

On va lister les traitement 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 des comment est-ce que les trous dans multiple séries temporelles doivent être résolu, 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 « Interpolation » qui permet d'activer la détection de corrélation entre deux suites des mesures.

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

5 – Zone des graphiques. Zone des graphiques est un conteneur des toutes les graphiques qui sont utilisées pour l'exploration de nature des données. Pour supprimer un des graphiques, il faut glisser leur titre dans une plat-forme de suppression de la zone 3. La zone comporte aussi une bouton qui serve 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 expliquer les fonctionnalités d'application (Figure 8). Sur la première capture de la Figure chepa, on a chargée deux séries temporelles, une venant de la basé des données MongoDB, l'autre de MySQL.

Les deux graphiques sont utilisées courbe temporelle et un graphique sectorielle. Le dernier contient plusieurs secteurs de différentes surfaces car la quantification de taille 1 a était appliquée. Cela correspond à réaliser une opération d'arrondi. Sans l'utilisation de quantification la graphique sectorielle sera composée secteurs uniforme, parce que chaque mesure apparaisse

qu'une seule fois.

Une 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 de la même couleur et taille indiquent la séquence des mesures corrélées.

Sur la deuxième image de la Figure chepa, on explore les mêmes séries temporelles que dans un premier exemple. La différence est que cet fois-ci, on force l'utilisation de Canvas.js dans tous les cas, même si la taille d'une série temporelle finale est acceptable pour le fonctionnement correct de DC.js. L'allure de la courbe temporelle n'est pas la même que dans un exemple précédent, car on n'affiche que les mesures apparaissant entre 1920 et 1948. De plus, on applique un traitement « réduction » : on ne laisse qu'une mesure sur deux. Ces deux techniques simples doivent être utilisées en but de diminuer la taille d'une série temporelle finale.

Une autre différence par rapport à l'exemple, mais elle est insignifiante est que cet fois on utilise un histogramme vertical au lieu d'un graphique sectoriel, ils sont interchangeables. Aussi, on applique l'interpolation cubique au lieu de linéaire, mais cela ne se voit pas car les deux séries temporelles en question sont homogènes.

Cet fois, sur la troisième image de la Figure chepa, les sources sont différentes : les deux séries temporelles sont stockées dans une base de données MongoDB. Toutes les permutations entre les types de stockages sont possibles. Dans cet exemple les graphiques ont changé aussi. On utilise un nuage de points, histogramme horizontal, et aussi une courbe temporelle. On remarque que sur le premier graphique un filtre a été appliqué, c'est qui a modifié toutes les autres graphiques, en n'affichant que des mesures correspondantes. On peut voir dans une courbe temporelle de quel instant il s'agit, l'histogramme nous montre la distribution des valeurs correspondantes.

En bas du nuage de points, on observe une valeur qui correspond à un indice de corrélation entre les deux grandeurs d'une série temporelle fusionnée. Comme un filtre a été appliqué, la corrélation est forte négative.

Un autre point rajouté dans cet exemple est qu'on visualise la série fusionnée dans une table.

3.5 Problèmes, futurs évolutions

La plus grande problème rencontrée est le temps d'attente. De part de DC.js cela a été résolu par l'utilisation de la librairie graphique Canvas.js dans les cas où DC.js ne sera pas à l'heure.

Il y a un autre problème de latence lorsque l'application fonctionne trop de temps sans la mise à jour d'une page. Ce problème peut être facilement résolu en cliquant sur F5, mais par contre cela va effacer toutes les configurations de choix des stockages et graphiques. Si en futur ce problème devrait être résolu, il va falloir réaliser des tests de mémoire.



Chapitre 4 : Bilan de stage

Bibliographie

