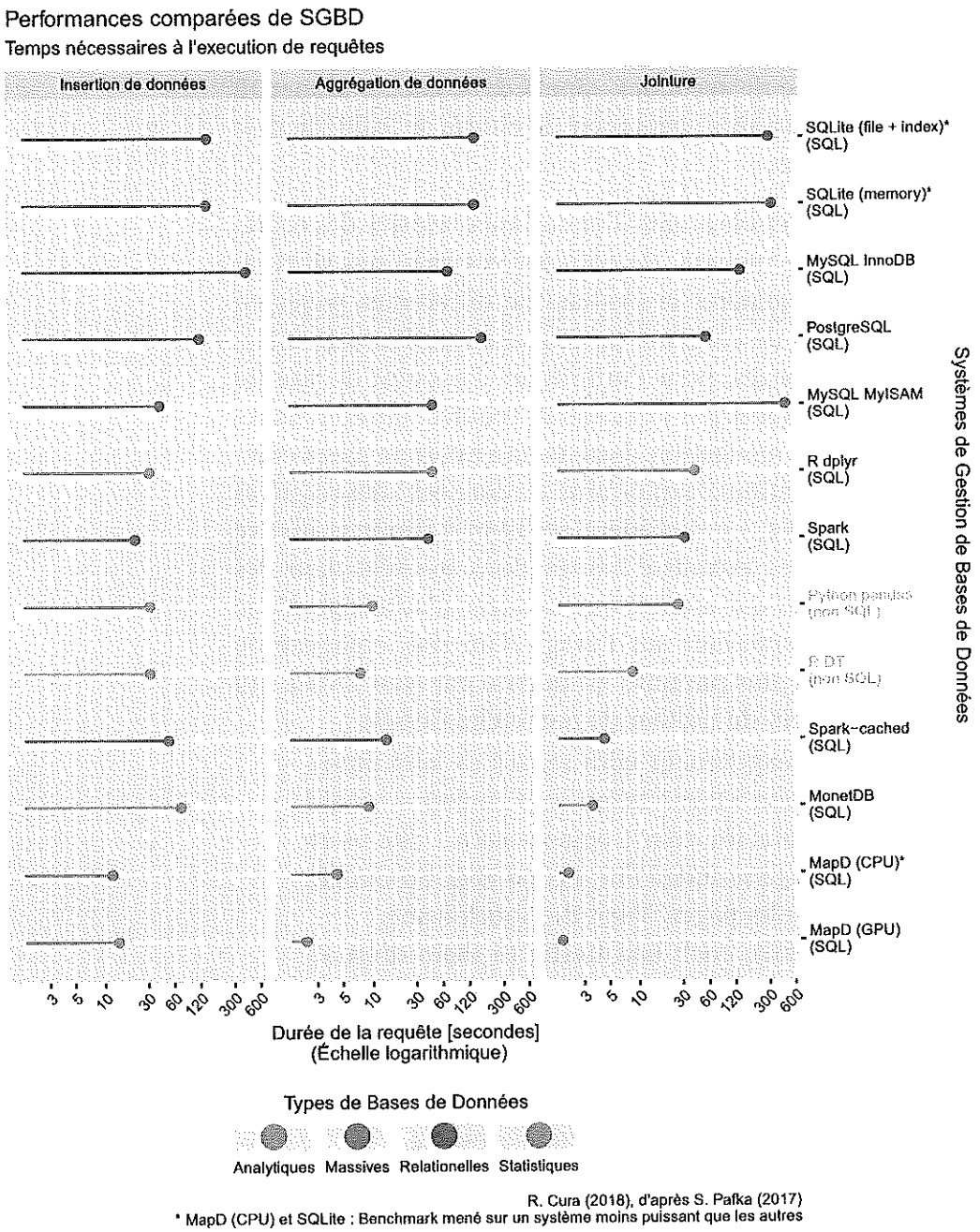

FIGURE 5.9 – Comparaison de la performance de différents SGBD sur un jeu de données test de 100 millions de lignes. Résultats tirés de PAFKA (2017) et complétés par l'auteur.

Les « types de Bases de Données » correspondent aux usages les plus fréquents des SGBD comparés :

- Analytiques : SGBD optimisés pour les traitements de type agrégation, via une architecture orientée colonne plutôt qu'orientée ligne comme dans les SBGD Relationnels. Ils sont optimisés pour la rapidité d'exécution.
- Massives : SGBD pensés pour la gestion et l'interrogation de données massives (big data), permettant notamment une parallélisation des requêtes. Ils sont optimisés pour la capacité à gérer des volumes gigantesques de données.
- Statistiques : SGBD internes aux environnements de traitement de données statistiques, reposant sur une gestion en mémoire vive. Souvent intégrés d'office dans les environnements décrits (R, Python), ce sont les SGBD les plus simples à mettre en place et à manipuler.

De l'intérêt de gagner quelques secondes La figure 5.9 permet d'isoler un sous-ensemble de quatre SGBD ayant, avec le jeu de données testé, des réponses inférieures à une dizaine de secondes : Spark avec cache, MonetDB et MapD sur CPU ou GPU. On pourrait se contenter de choisir le SGBD le plus complet parmi ces quatre solutions.

Pourtant, un autre domaine d'étude appuie l'importance relative des écarts, mêmes faibles, dans les durées de requête. Ce domaine est celui des sites internet, où les requêtes servent à générer le contenu de différentes pages en interrogeant des base de données de contenu. La consultation d'un site internet consiste à charger plusieurs pages, pour l'utilisateur. Du point de vue du serveur, chacune des pages demandées par l'utilisateur requiert différentes requêtes à des bases de données. La navigation dans un site est donc assez comparable à l'utilisation d'une application d'exploration de données : des requêtes hétérogènes, plus ou moins lourdes, s'y succèdent et visent à filtrer et mettre en forme, de manière explicite, des extraits d'informations stockées dans des bases de données. Plusieurs études ont montré que la durée d'affichage d'une page web jouait de manière considérable sur l'usage d'un site, composé de plusieurs de ces pages. L'étude la plus parlante est décrite par Neil PATEL qui relate une expérience vécue au sein du moteur de recherche Google :

> « Google did an interesting experiment with regard to load times. Google Vice President Marissa Mayer asked web surfers – would you rather see 10 or 30 results for your Google search ? The users agreed that 30 results per page sounded like a good idea. So Google implemented it on some results pages. Then the shock came.
> Pages that displayed 30 results each had traffic to them drop an astounding 20%. Google tested the loading difference between the 10 and 30 results pages and found that it was just **half of a second.**
> If half of a second made that much of a difference in how long users were willing to wait, how much of a difference could it make to your site if you carved a second or two off of load time ? »
>
> (PATEL 2011)

Si l'environnement et les conditions décrites ne sont pas directement comparables avec celles de SimEDB, il demeure qu'une différence même faible dans un temps de chargement, ou, pour SimEDB, dans un temps d'affichage d'un indicateur de sortie, pourrait avoir des conséquences négatives pour l'utilisation de la plate-forme.

Un autre exemple appuie ce raisonnement et répond à la dernière interrogation de PATEL, dans un cadre un peu plus proche de SimEDB. Roxana ELLIOTT, employée d'une société qui propose des solutions d'accélération de sites web, a réalisé un rapport sur les perte d'audience des sites webs en fonction du temps de chargement des pages (ELLIOTT 2017). Les résultats de son étude sont présentés dans la figure 5.10, et permettent de quantifier un effet bien connu, qui veut que l'utilisateur quitte plus rapidement un site (et en visite donc moins de pages) quand les pages sont plus longues à charger.

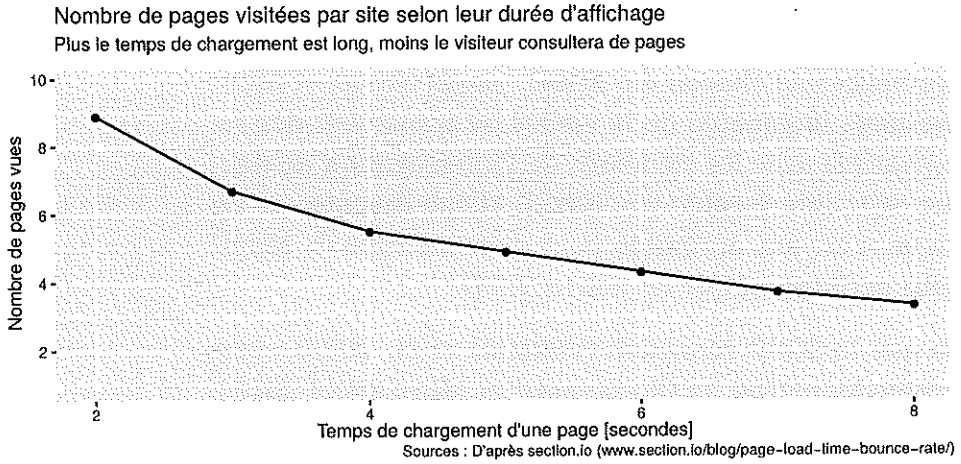

FIGURE 5.10 – Impact du temps de chargement d'un site web sur sa consultation. D'après ELLIOTT (2017).

Cet exemple est plus directement comparables aux contraintes de SimEDB. Chaque consultation d'un indicateur de sortie correspond ainsi à la vue d'une page dans cet exemple. Ces chiffres renforcent l'importance à accorder au temps de chargement des indicateurs, et donc à la durée de l'exécution des requêtes qui les génèrent. Quelques secondes de différence dans le chargement suffisent ainsi à réduire drastiquement le nombre d'indicateurs que l'utilisateur acceptera d'analyser.

On peut toutefois pondérer ces comparaisons et minimiser l'importance d'écarts de l'ordre de grandeur de la seconde. Dans le cas de SimEDB, contrairement à celui d'un site web ou d'un moteur de recherche, l'utilisateur est « captif ». Cela signifie qu'un thématicien souhaitant explorer les résultats produits par SimFeodal n'aura d'autre choix que de passer par SimEDB. De même, sachant que la plate-forme présente pour lui un intérêt professionnel, le thématicien sera bien plus patient que face à un quelconque site de courses en lignes.

Dans le cadre d'environnements de type *visual analysis*, il a été montré que les utilisateurs d'environnement d'exploration étaient toutefois fortement affectés par l'accroissement de délais. Zhicheng Liu et Jeffrey Heer (LIU et HEER 2014) montrent ainsi qu'en introduisant une latence supplémentaire de 500 ms dans une application interactive d'exploration de données spatio-temporelles, le nombre d'interactions chute fortement, quand bien certains utilisateurs de cette application ne remarquent même pas la différence de délai.

Choisir un SGBD adapté à SimEDB L'ensemble de filtres successifs a permis de réduire progressivement la quantité de solutions logicielles appropriées à l'organisation et à l'interrogation des données issues de SimFeodal. Depuis les centaines de solutions disponibles, on parvient ainsi dans un premier temps à isoler les grands types de SGBD correspondant aux besoins identifier : les SGBD relationnels, basés sur une interrogation standardisée en SQL. Ces outils sont ensuite départagés au prisme de leur robustesse, intrinsèque (stabilité) et sur la base de leur niveau d'adoption (pérennité). Un *benchmark* finit de restreindre la liste des possibles à quelques solutions envisageables en fonction des besoins

soulevés par SimEDB.

Au regard des performances de chacun des SGBD, MapD (ROOT et MOSTAK 2016) présente l'avantage indéniable de la vitesse de traitement des requêtes, tout en étant compatible avec les standards de l'interrogation de données (langage de requête SQL, interfaçable via JDBC). Même exécuté sur un infrastructure informatique n'est pas optimisée pour cette solution[38], MapD est incontestablement plus performant que les autres SGBD.

On notera toutefois que le SGBD MonetDB (VERMEIJ et al. 2008), dans son implémentation intégrée MonetDBLite (RAASVELDT et MÜHLEISEN 2018), affiche aussi des performances très compétitives, et aurait pu être choisi pour SimEDB, présentant notamment l'avantage d'être plus utilisé et ancien[39]. Un des ouvrages de référence en *visual analytics* s'interrogeait d'ailleurs sur les nouvelles possibilités et l'adéquation offertes par ce SGBD (FEKETE 2010, p. 105 in KEIM et al. 2010).

Nous avons au final préféré MapD, en particulier parce que les données issues de SimFeodal sont amenées à augmenter, renforçant donc petit à petit l'écart de performance entre MapD et MonetDB. Par ailleurs, par une heureuse coïncidence, MapD a été placé sous licence libre peu avant que nous n'ayons à nous pencher réellement sur les problèmes de performances et de robustesse qui apparaissaient suite à l'augmentation du nombre de simulations effectuées.

5.3.2 Structuration des données de SimFeodal

Le choix d'un SGBD est une étape indispensable à la mise en place d'une base de données, mais il ne concerne que le domaine technique, voire méthodologique, mais aucunement le domaine conceptuel. Un SGBD est un support logiciel qui permet le stockage et l'organisation de données. Il n'est utile qu'une fois que le mode d'organisation des données a été décidé. L'organisation a proprement parler des données est explicitée dans un modèle conceptuel, nommé Modèle Conceptuel de Données (MCD). Un MCD est propre à un ensemble de données d'une part[40], et un ensemble de problématiques d'autres part[41]. Ce MCD décrit donc les « tables », leur composition (attributs) et les liens entre

38. MapD est ainsi un SGBD optimisé pour l'analyse sur processeurs graphiques (les GPU), présents dans les cartes graphiques modernes, contrairement aux SGBD classiques qui s'appuient sur les processeurs (CPU) pour effectuer leurs calculs. Dans le cadre de cette thèse, nous n'avions pas accès à un serveur doté de GPU, et MapD est donc installé sur une infrastructure à base de CPU, bien moins performante.

39. Dans les faits, MonetDBLite a été le SGBD utilisé pendant une large partie de la conception de SimEDB. Il s'est toutefois révélé assez instable dans notre cas, faisant preuves à plusieurs reprises de corruptions de données ayant entraîné l'obligation de recréer entièrement les bases de données depuis les fichiers bruts produits par SimFeodal.

40. Le MCD décrit la manière dont les données sont stockées, organisées et mises en relations. Il ne peut donc être générique, et doit être modifié quand la structure des données évolue.

41. Il y a une infinité de possibilité d'organisation d'un même jeu de données. Le MCD permet d'organiser ces données en vue de répondre à des questions, exprimées sous formes de requêtes particulières. Appliquées au même jeu de données, différents MCD permettront de répondre plus ou moins facilement (et de manière plus ou moins performante) à certaines questions.

tables qui permettent de mener des interrogations croisées. Par exemple, on peut avoir une table élèves, contenant les informations relatives aux élèves d'un établissement, une table enseignants, et une table classe qui permet de faire le lien entre les élèves d'un enseignant, ou au contraire entre un élève et tous ses enseignants.

Les choix de conception d'un MCD sont fortement lié aux types de SGBD dans lesquelles ils doivent être implémentés. On ne peut que difficilement implémenter un MCD très relationnel dans un SGBD pensé collection (NoSQL par exemple). À l'inverse, le stockage d'informations très hétérogènes sur un ensemble d'individus sera complexe à implémenter au sein d'un SGBD relationnel.

Pour décider de la manière la plus efficace d'implémenter les données issues de SimFeodal dans un SGBD, et donc du MCD à suivre, il convient de revenir aux spécificités des données produites par le modèle d'une part, et d'autre part de réfléchir aux modes d'interrogations privilégiés, lesquels orienteront la conception du MCD.

Pré-traitement des données Les données produites par un modèle de simulation sont des données « brutes », c'est-à-dire qu'elles ne sont pas organisées de manière rationnelle, contiennent une quantité non négligeable d'informations incomplètes, superflues ou erronées.

— Par exemple, quand une simulation est arrêtée en cours, soit volontairement, soit en raison d'un *bug*, les données générées par le modèle sont **incomplètes** : elles ne concernent qu'une partie des pas de temps attendus. Elles sont pourtant exportées dans les fichiers bruts, rendant ceux-ci hétérogènes en matière de complétion des informations enregistrées. Pour pouvoir analyser une expérience, il faudra supprimer ces données incomplètes pour qu'elles n'influence pas l'étude des simulations complétées et donc comparables.

— De la même manière, il arrive qu'on exécute, par erreur, plusieurs fois les mêmes simulations. Dans ce cas, le nombre de réplications de chacune des expériences ne sera pas systématiquement le même. Cela pose un problème de comparabilité dû à tailles d'échantillonnage différentes. On fait donc face à un problème de données **superflues** : il faudra supprimer une partie de ces simulations des données avant de pouvoir les traiter.

— On peut enfin voir subvenir des erreurs d'exécution du modèle au niveau des agents, par exemple quand, en raison d'un *bug*, un agent en interroge un autre qui a disparu depuis. Il arrive ainsi fréquemment que des foyers paysans déclarent une appartenance à un agrégat qui a disparu depuis, faute à une mise à jour échouée dans le modèle. Dans ces cas, les données seront aussi inscrites dans les sorties de SimFeodal, quand bien même elles sont **erronées**.

Les données brutes doivent donc nécessairement être vérifiées, filtrées, nettoyées et retravaillées avant de pouvoir les exploiter en vue de générer les indicateurs de sortie.

Organisation des données Même pré-traitées, les données brutes conservent une structure tabulaire assez peu adaptées à un traitement. les attributs de chacun des types d'agent sont enregistré dans des fichier spécifiques. Que ces fichiers aient été nettoyés ou non, ils demeurent fondamentalement isolés les uns des autres. Une partie des indicateurs repose sur des analyses croisant différents types d'agents (dans quels pôles les agrégats s'inscrivent-ils par exemple ?), et il est donc nécessaire de permettre – et de fluidifier – ces requêtes croisées. On a mentionné le choix de SGBD relationnels, il convient donc de concevoir et d'implémenter, dans le SGBD choisi, les relations entre les différentes tables individuelles qui proviennent des sorties brutes d'un modèle.

5.3.2.1 Quel modèle de données ?

Les MCD sont propres à chaque ensemble de données et questionnement associés. Il y a toutefois des grandes tendances dans l'organisation des données. Le MCD peut ainsi être catégorisé, selon sa forme, dans des familles de modèles de données. On nomme ces catégories « modèles logiques » ou « schémas » (*logical schema* en anglais). Ceux-ci décrivent la manière dont les données sont structurées et surtout reliées les unes aux autres, d'une manière générique contrairement aux MCD.

Un modèle « en étoile » Les bases de données relationnelles peuvent s'appuyer sur de nombreux schémas différents. Sans entrer dans le détail, notons que chacun des schémas existant présente des avantages et des inconvénients liés aux types de requêtes qui lui seront adressés. Par exemple, un schéma « en étoile » (*Star Schema* 2018) privilégie l'efficacité de requêtes d'agrégations et de jointures, au détriment de la robustesse des données et de la « liberté » des requêtes. Certains types de requêtes, complexes, seront ainsi difficiles, voire impossibles, à exprimer dans ce type de schéma.

Au contraire, un schéma « en flocons » (*Snowflake Schema* 2018) peut se révéler plus permissif en terme de capacités de requêtes. L'inconvénient est une plus forte complexité des requêtes de bases (exprimées de manière plus verbeuses et tortueuses) et donc d'une expressivité moindre.

Pour choisir un schéma, et donc une manière d'organiser la base de données, il convient donc de savoir – ou de prévoir – le type de requêtes qui lui seront adressées. Dans le cas des données de SimFeodal, les indicateurs avaient été définis avant que le besoin d'une interrogation performante et structurée n'apparaisse. On connaissait déjà les indicateurs nécessaires et le type de requêtes associés. Nous savions ainsi qu'une majorité des requêtes seraient des tâches d'agrégations simples (nombre d'agrégats au cours du temps, taux de foyers paysans dispersés au cours du temps etc.), pour lesquelles il fallait minimiser la complexité des requêtes et calculs.

Il a été choisi de partir d'un schéma en étoile, puisque celui-ci se montre extrêmement efficace pour réduire les besoins en jointures – chronophages – et pour des tâches d'agrégations lourdes.

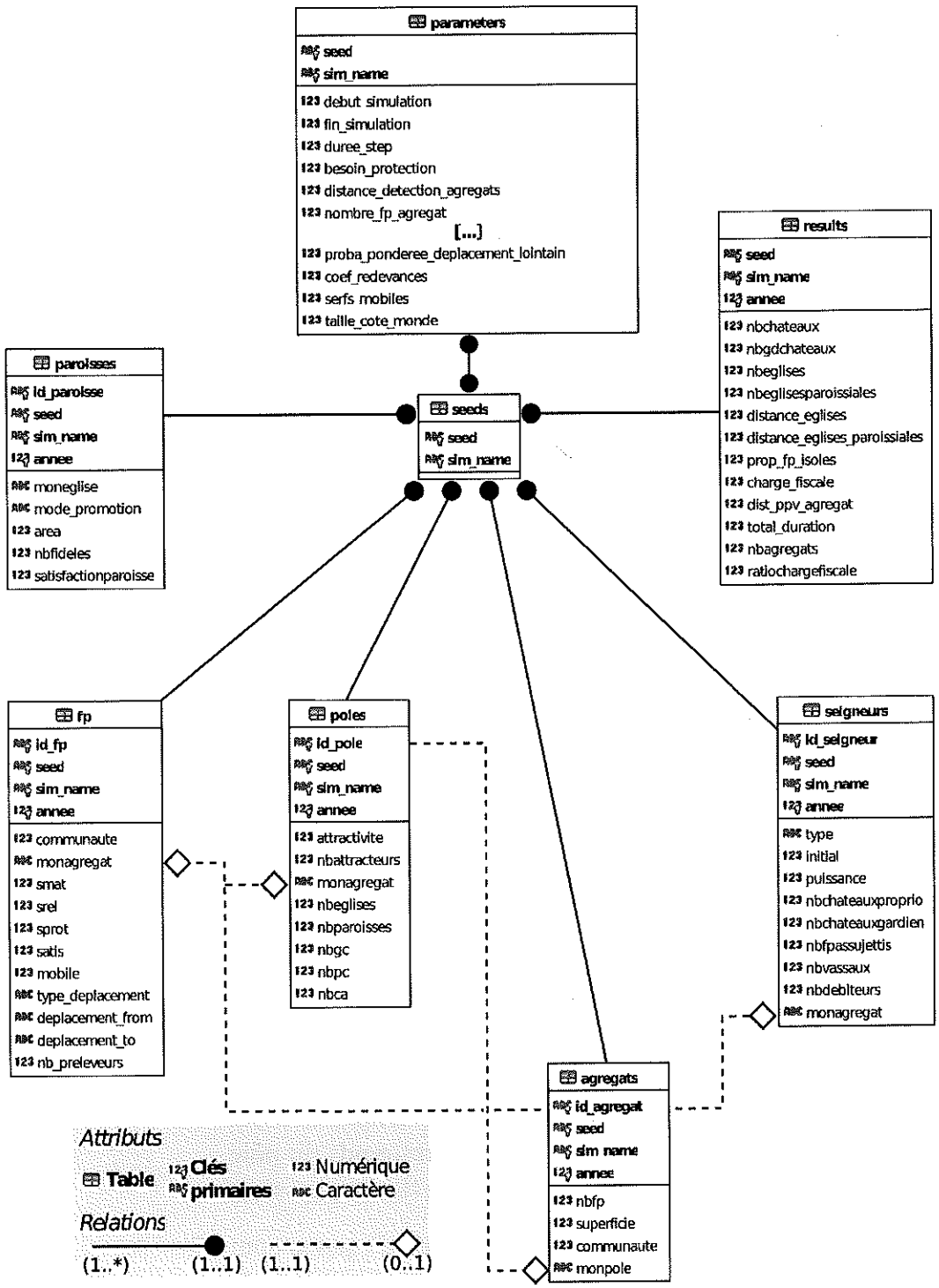

FIGURE 5.11 – Modèle Conceptuel de Données (MCD) des données en sortie de simulation de SimFeodal telles qu'implémentées dans SimEDB.

Au centre de cette étoile (voir figure 5.11), il était donc évident de disposer une table simple, contenant les informations sur lesquelles une majorité des agrégations seraient effectuées : les simulations, identifiées par leur nom (`sim_name`, qui permet de savoir de quelle expérience ces simulations dépendent) et leur identifiant unique, la graine aléatoire utilisée (`seed`) [42].

42. La graine aléatoire ne constitue en tant que tel pas un identifiant unique : comme son nom l'indique, elle est aléatoire et présente donc un risque de répétition. Dans Gama, cette graine aléatoire est une valeur qui varie de 0 à 1 et est composé de 19 décimales. Il y a donc potentiellement 10^{19} graines aléatoires uniques, ce qui est en soi une quasi garantie d'unicité.

Relier les tables Toutes les tables contenant les enregistrements individuels des agents (`fp` pour les foyers paysans, `paroisses` pour les églises paroissiales, etc.) sont donc liées directement à cette table centrale (intitulée `seeds` ici).

En dehors de ces tables liées aux agents, deux autres tables « globales » sont présentes : une table « `results` », qui contient des informations agrégées sur l'état de chaque simulation à chaque pas de temps. Ces informations, par exemple le taux de foyers paysans isolés (champ « `prop_fp_isoles` »), sont redondantes : elles pourraient être calculées directement depuis la table renseignant les foyers paysans, en faisant un ratio entre le nombre de foyers paysans sans agrégat et leur nombre total. Pourtant, pour des raisons d'efficacité autant que de clarté, il a été choisi de dupliquer, en les pré-calculant, ces informations qui sont interrogées extrêmement souvent pour calculer les indicateurs de SimFeodal.

Autre table ne répondant pas au schéma classique, la table « `parameters` » fournit toutes les méta-données sur les simulations. On y retrouve par exemple les valeurs de paramètres de chacune des simulations, identifiées toujours par le couple `sim_name` et `seed`. Cette table est la seule à être reliée de manière bi-directionnelle à la table centrale (`seeds`), en particulier en raison de l'usage qui en est fait interactivement (voir l'encadré 5.2).

Notons tout de même que l'on s'éloigne légèrement du classique schéma en étoile en raison des relations que nous avons choisi d'insérer entre les tables des différents agents (relations notées en pointillées dans la figure 5.11). Intégrer ces relations dans la table centrale aurait considérablement complexifié cette dernière, mais pour autant, elles étaient nécessaires : SimFeodal est un modèle complexe, dans lequel des interactions sont présentes à plusieurs niveaux entre différents types d'agents. La base de données résultant de ce modèle complexe l'est donc nécessairement aussi : on doit implémenter, dans la base de données, des relations entre les tables pour chacune des interactions entre les agents du modèle. Ici, ces relations permettent par exemple d'étudier la composition des pôles autour de chaque agrégat, et ainsi d'étudier le lien entre poids du pôle (en nombre d'attracteurs) et poids de l'agrégat (en nombre de foyers paysans).

Ces indicateurs, situés à l'intersection de différents types d'agents, sont toutefois moins utilisé que les indicateurs plus directs (ref à chapitre 3, indicateurs). Les requêtes correspondantes, moins fréquentes, ne perturbent pas les logiques et performances d'ensemble de SimEDB : elles auraient plus facilement exprimées dans un schéma « en flocons », mais leur relative rareté ne remet aucunement en cause l'organisation générale du MCD.

Notons de plus que dans le MCD de SimEDB, la graine aléatoire est systématiquement associée au nom de l'expérience. Même en menant un million de réplications, la probabilité que deux simulations partagent la même graine aléatoire serait largement inférieure à 1%. La graine aléatoire constitue donc un identifiant unique robuste dans notre cas.

5.3.2.2 Un modèle de données pour favoriser l'interrogation et le filtrage conjoint

Le schéma choisi et le Modèle Conceptuel de Données (MCD) associé, permettent une interrogation rapide des données en simplifiant les tâches d'agrégation et en minimisant la quantité de jointures nécessaires à la génération des indicateurs de sortie. Le choix de s'écarter légèrement du schéma en étoile présente un autre avantage, extrêmement utile, dans le cadre d'une exploration interactive des indicateurs de SimFeodal. En effet, comme on l'a vu auparavant (section 5.2.6), dans SimEDB, on compare les simulations en les isolant à partir des valeurs de paramètres qui leur correspondent, via un acte de *brushing* des valeurs de paramètres présentées dans un graphique en coordonnées parallèles interactif. Du côté du MCD, la table correspondante est la table `parameters`. Quand l'utilisateur sélectionne un sous-ensemble de valeurs de paramètres, la table est filtrée, et ne renvoie donc que les simulations correspondantes.

C'est ici que l'intérêt de la table `seeds` et de son lien bidirectionnel avec la table `parameters` apparaît : une fois `parameters` filtrée, cette sélection est renvoyée à la table `seeds`, et se répercute donc directement à toutes les autres tables. Avec une unique requête, qui plus est sur une table de faible dimension (`seeds` ne comporte que deux champs), le filtrage est donc extrêmement véloce, accélérant d'autant le filtrage des autres tables et donc la génération des indicateurs de sortie. Ces étapes de filtrage successifs, optimisées par l'architecture choisie pour les données de SimFeodal, sont présenté dans l'encadré 5.2.

Encadré 5.2 : Un exemple d'interrogation de la base de données de SimEDB.

La figure 5.12 présente l'ensemble des étapes qui permettent de générer un indicateur de sortie. Cette planche montre un exemple de sélection faite dans l'application SimEDB, et décrit la manière dont cette sélection est répercutée à travers le MCD de SimFeodal (figure 5.11). La démarche aboutit par la sélection d'un ensemble de données, qui répondent à un critère sur deux paramètres du modèle. Cette sélection est ensuite utilisée pour générer un indicateur de sortie, ici, l'évolution du nombre d'agrégats au cours du temps.

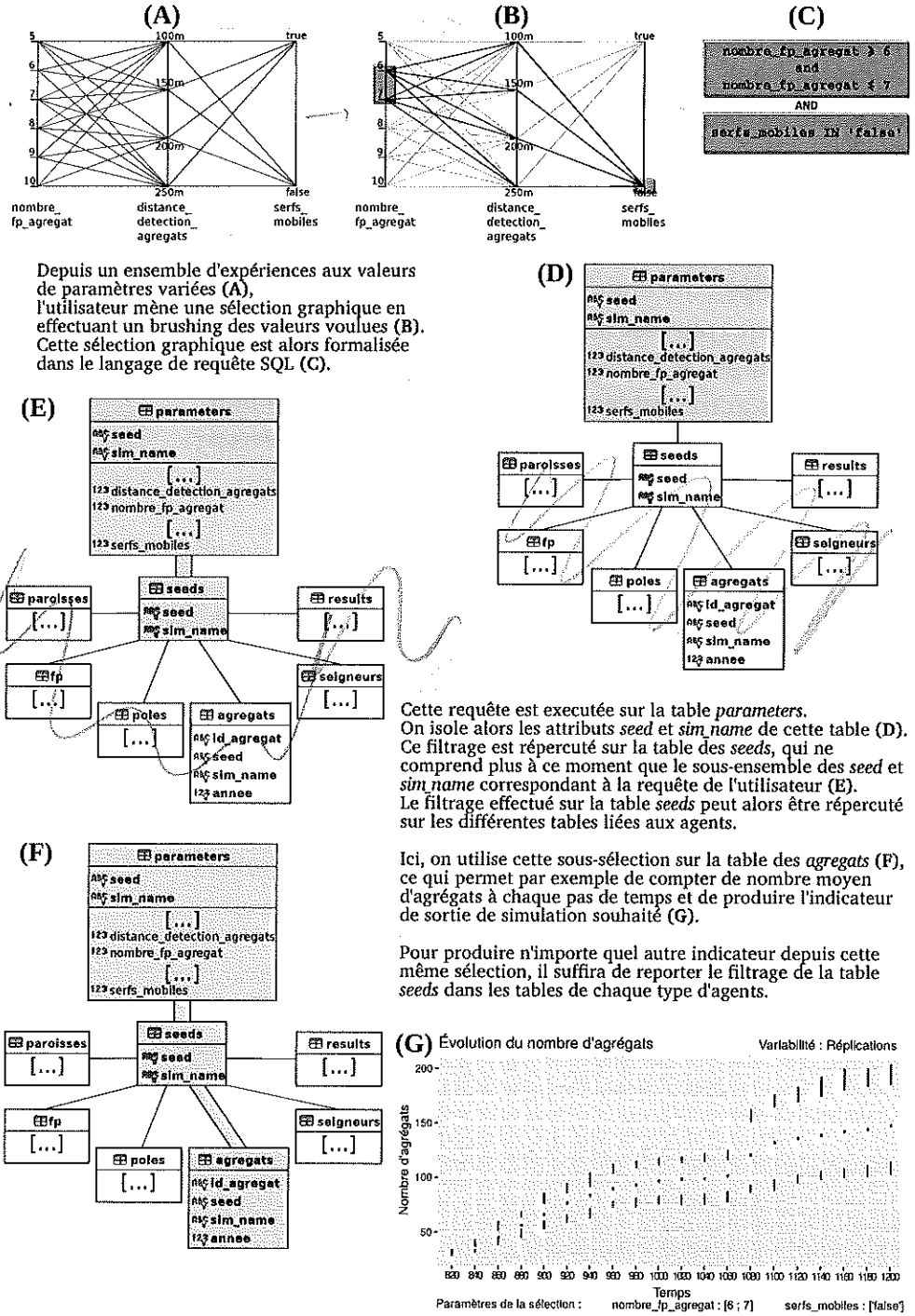

FIGURE 5.12 – De la sélection interactive à l'indicateur de sortie.

Une organisation dédiée à l'exploration interactive

La présentation des choix d'organisation de données témoigne d'une visée résolument applicative, c'est-à-dire visant à penser l'organisation, la structuration et les SGBD d'implémentation, comme au service de la plate-forme d'exploration SimEDB. Le SGBD choisi, MapD, est ainsi un logiciel particulièrement adapté aux besoins identifiés, c'est-à-dire à une efficacité et une robustesse d'interrogation des données générées par SimFeodal. MapD est interrogeable de manière universelle, via des protocoles de connexion standards, au moyen d'un langage qui fait office de *lingua franca* de l'interrogation de données, le SQL. Au sein du SGBD, la structure des données, révélée dans le MCD qui adopte une structure « en étoile », vise aussi à faciliter et à optimiser la vitesse des requêtes visant à générer les indicateurs de sortie de SimFeodal. Cette structure de données est enfin pensée, en amont, pour minimiser le nombre de requêtes nécessaires à l'affichage des indicateurs, dans un cadre interactif, correspondant à des sous-ensembles des nombreuses simulations effectuées au cours de la construction, du paramétrage et de la calibration de SimFeodal.

Il est important de noter qu'en l'absence de ces choix de conception de base de données, de la modélisation conceptuelle jusqu'à l'implémentation technique, la plate-forme d'exploration des données SimEDB, que nous allons maintenant présenter plus en détail, n'aurait pu être conçue, élaborée et bâtie de manière convaincante.

5.4 Une plate-forme d'exploration de données de simulations : SimEDB

La section 5.2 (Comment explorer les sorties de SimFeodal ?) a décrit les étapes successives d'avancement dans l'exploration des données en sortie de SimFeodal, depuis l'observation en direct des simulations (« pré-filtrage ») jusqu'au besoin d'une plate-forme permettant l'exploration et la comparaison interactive des sorties de simulation. La plate-forme proposée en réponse à ce besoin, SimEDB [43], dans un objectif de généricité et d'adéquation, se devait aussi de répondre à de nombreuses contraintes, aussi bien liées aux possibilités offertes qu'à l'usage qui en serait fait. Dans cette partie, nous nous attacherons donc à présenter les contraintes qui ont guidé la conception de SimEDB, ainsi que les choix, méthodologiques et techniques, qui en ont résulté.

5.4.1 Contraintes

5.4.1.1 Adapter la complexité aux utilisateurs

Dans le domaine de l'Interface Homme-Machine (IHM), il est courant de considérer qu'un outil d'analyse et de représentation doit être adapté à un public. La figure 5.13, emblématique de la conception de géovisualisations par Alan MACEACHREN, replace ainsi les types d'usage d'une plate-forme d'exploration selon trois axes : le niveau d'expertise des utilisateurs visés (*users*), le niveau d'interaction souhaité (*interaction*) et l'objectif poursuivi par la (géo)visualisation (*task*). D'après l'auteur, à un niveau d'expertise de l'utilisateur correspond un unique degré d'interaction et un unique objectif : dans le cube, seule une « droite » des usages possibles est présente. L'auteur décompose ces usages en quatre types :

— Pour le grand public (*users* de type *public*), l'objectif est de transmettre une information simple (*info sharing*). Le niveau d'interaction avec la géovisualisation doit donc être faible. Il s'agit d'une tâche de présentation (*present*).

— Pour un public légèrement plus connaisseur, on peut augmenter le niveau d'interaction. On entre alors dans un but de synthèse (*synthesize*).

— En ciblant un niveau encore supérieur d'expertise chez l'utilisateur, et en visant à de la construction de connaissance plus qu'à une transmission de connaissance, on augmente encore le niveau d'interaction. La géovisualisation a alors pour but l'analyse (*analyze*).

— Au plus haut niveau d'interaction, d'expertise et de recherche, la géovisualisation peut servir d'outil d'exploration (*explore*).

43. **Sim**Feodal Exploration **D**ash**B**oard, voir la note de bas de page 23, section 5.2.6.

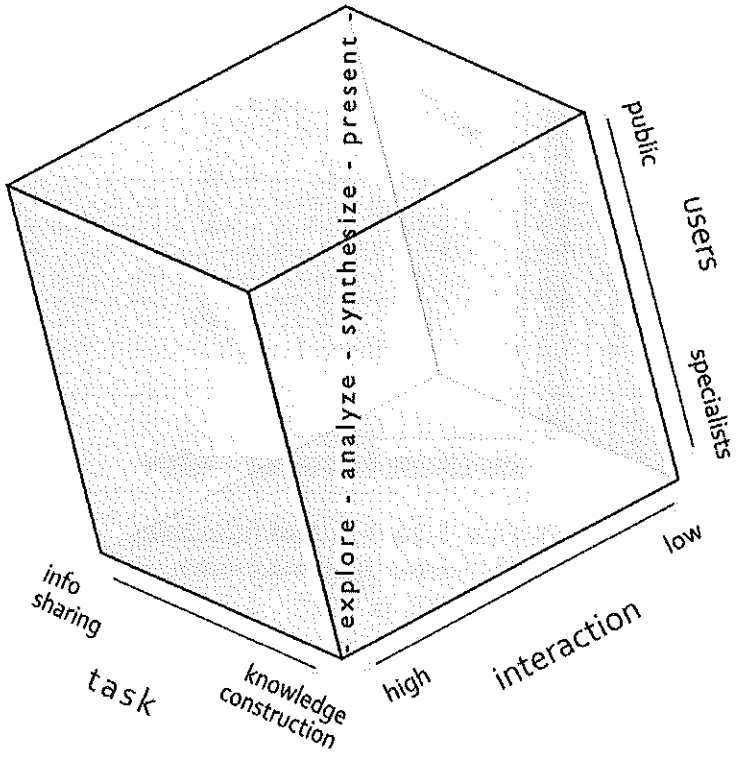

FIGURE 5.13 – « *An update to Cartography³, 10 years after its conception* », par ÇÖLTEKIN, JANETZKO et FABRIKANT 2018, d'après MACEACHREN et al. 2004, p. 10.

ROTH (2015, p. 16) commente cette figure en effectuant une assimilation entre niveau d'interaction et complexité de l'interface de l'outil de géovisualisation : « All participants agreed that user expertise requires increased interface complexity, as suggested by the Cartography³ framework ».

La plateforme SimEDB est conçue pour être utilisée par des experts thématiques (l'équipe de modélisation de SimFeodal), avec un objectif clairement inscrit dans la construction de connaissance. A ce titre, et d'après MACEACHREN, le niveau d'interaction avec l'outil de géovisualisation devrait être élevé (forte complexité de l'interface pour ROTH) et ancrer l'usage dans une dimension exploratoire.

Des utilisateurs hétérogènes mais captifs SimEDB est pourtant pensé à un niveau intermédiaire, entre l'analyse et la synthèse, dans le cube de la figure 5.13. Il ne s'agit ainsi pas d'explorer des données, au sens de MacEachren, qui sous-entend par cette exploration (*explore*) la recherche d'informations dans un jeu de données inconnu de l'utilisateur. Le besoin identifié consiste à permettre aux utilisateurs d'explorer des sorties de simulation à travers des indicateurs déjà pensés et constitués. Il ne s'agit pas de proposer un outil d'exploration de données brutes, permettant de créer à la volée des nouveaux indicateurs, via une approche d'exploration « naïve ». Au contraire, l'exploration est guidée par les indicateurs, et la tâche s'apparente plus à de l'analyse de résultats de simulations, voire à de la synthèse des spécificités des résultats issus d'expériences différentes. L'objectif de SimEDB s'écarte donc du « modèle » de MACEACHREN, puisqu'il ne se situe pas sur la « droite » des usages (voir la figure 5.14).

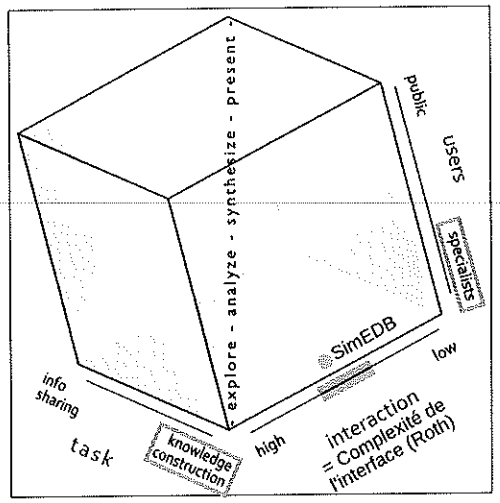

FIGURE 5.14 – Positionnement de SimEDB dans le cube *Cartography*[3] de MACEACHREN.

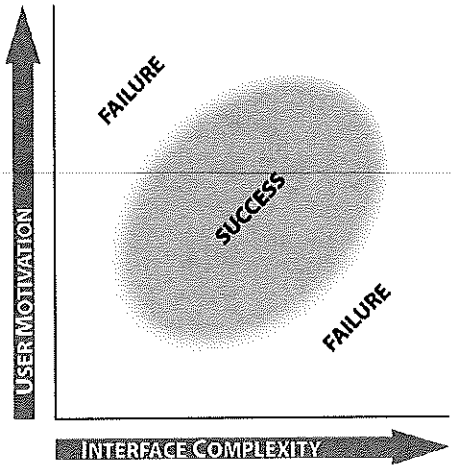

FIGURE 5.15 – « *Interface complexity versus user motivation.* », ROTH 2013, p. 79.

Cet écart au modèle conceptuel de MacEachren s'explique notamment par la diversité des utilisateurs de SimEDB. Il serait absurde de qualifier un niveau d'expertise général des utilisateurs tant les spécificités de cette expertise sont nombreuses. Entre des profils de spécialiste thématiciens, de modélisateurs ou encore de géomaticiens, l'expertise est présente, mais concernant des champs différents, toutefois tous intéressés par l'exploration des sorties de SimFeodal.

Il est dès lors peu évident de se fixer sur un degré de complexité à atteindre dans la plate-forme d'exploration : un niveau faible serait frustrant pour les utilisateurs avancés, et un niveau avancé serait source de confusion et donc de perte de motivation pour les utilisateurs moins expérimentés (figure 5.15).

Une spécificité du cas d'usage de SimEDB permet toutefois de miser sur une bonne motivation générale des utilisateurs, et donc sur la possibilité de créer un outil à l'interaction plus complexe qu'un simple présentoir de données. Contrairement à une utilisation grand public, qui ne présente aucun engagement vis-à-vis d'une interface d'exploration de données, ou à l'inverse contrairement à des domaines experts où chaque utilisateur dispose de ses propres outils et méthodes pour explorer un jeu de données, le public cible de SimEDB est « captif ». On entend par là que les utilisateurs concernés par SimEDB ne disposent pas d'autre solution que de passer par cette plate-forme pour explorer les données issues de sortie de simulation, en particulier en raison des contraintes liées aux caractéristiques de ces données (leur masse par exemple, voir la section 5.1.4, Des données aux indicateurs). On peut dès lors se permettre de développer une interface plus complexe que si l'on visait un plus large public.

Intuititivé de l'usage au regard des applications traditionnelles En dépit de cette motivation, les utilisateurs de SimEDB demeurent majoritairement des experts thématiciens, potentiellement peu familiarisés à l'exploration de données interactives. Afin que le temps d'exploration des données issues de SimFeodal soit dévoué à la compréhension et à la synthèse de ces données plu-

tôt qu'à un apprentissage ou amélioration en exploration de données, il a été choisi de créer une application aussi simple que possible au regard des fonctionnalités principales qu'elle devait permettre : observer les indicateurs de sortie de simulation pour des expériences données, et les comparer entre elles aussi efficacement que possible.

Il n'était donc pas question de construire un nouveau « logiciel expert », doté de dizaines de fonctionnalités avancées, mais au contraire, de simplifier au maximum l'interface pour ne pas encombrer et complexifier l'utilisation de ces fonctionnalités principales.

On souhaitait une plate-forme aussi épurée que possible, plutôt que de partir, par exemple, sur la personnalisation et l'adaptation de l'un des outils d'exploration existants et dédiés à offrir une forte possibilité de manipulation[44].

5.4.1.2 Efficacité

Dans la description du choix du SGBD, on a mentionné une première fois l'intérêt de disposer d'une solution d'interrogation de données qui garantisse une certaine rapidité dans l'exécution des requêtes. Sans entrer dans le détail des recherches en IHM, on peut compléter ce besoin de rapidité par deux aspects complémentaires. Une solution interactive qui minimise les latences permet (1) de motiver l'utilisateur, c'est-à-dire de ne pas le décourager d'utiliser l'application, et (2) de lui faire conserver sa concentration[45] (*focus*).

Le premier point a été abordé plus haut (section 5.3.1.2), et surtout, en raison de la « captivité » de l'utilisateur évoquée ci-dessus, ne s'applique que marginalement à notre cas d'étude. Des délais trop importants pourraient décourager l'utilisateur, mais en l'absence d'alternative pour explorer les sorties de simulation, cela n'a pas un impact trop important.

Conserver la concentration Le problème de la concentration de l'utilisateur demeure, lui, critique. Des études ont montré, depuis longtemps (MACKENZIE et WARE 1993), qu'il y avait un lien fort entre la performance d'une interrogation visuelle et le délai nécessaire à son obtention. LIU et HEER 2014, p. 8 montrent ainsi qu'avec un simple délai de 500 millisecondes (ms), la qualité des observations, des généralisations qui peuvent être tirées des données, et des hypothèses émises, décroît nettement chez l'utilisateur. Les auteurs indiquent d'ailleurs que cette diminution est plus importante encore quand l'exploration est effectuée par des actions de *brushing* et de sélections croisées (*linking*), deux méthodes qui sont au cœur de SimEDB : « For example, more aggressive caching or prefetching methods may be employed for operations sensitive to small variations in latency, such as brushing and linking » (LIU et

44. Lena : « Éventuellement dire pourquoi recherche de généricité dans un créneau bien spécifique ?

45. Avant de spécifier ce sujet, notons que quand le délai entre une interaction avec un outil informatique et le retour qu'il doit produire (affichage de graphique par exemple) est important, l'utilisateur perd en capacité d'association entre son action et le retour observé. Typiquement, dans un processus d'exploration de données, plus ce délai est faible, plus l'utilisateur peut mobiliser son intuition pour évaluer, par exemple, les relations entre des variables ou des individus.

HEER 2014, p. 9).

FORCH et al. 2017, pour leur part, étudient la perception du délai de réponse lors d'interactions menées avec une souris d'ordinateur. Ils concluent ainsi que les utilisateurs perçoivent des délais d'attente inférieurs à 100 ms, mais notent que les utilisateurs n'en sont pas pour autant perturbés, en particulier ceux qui ont le moins l'habitude de réactions rapides [46].

Concernant le champ, plus spécifique, des *visual analytics*, nous n'avons pas trouvé d'articles de référence permettant d'établir une comparaison de l'efficacité des résultats trouvés selon la latence de la réponse. Les auteurs de ce champ recommandent de prêter attention à la rapidité de rendu et à son optimisation, mais sans que ne trouvions de résultats plus précis :

> « When simple pattern finding is needed, the importance of having a fast, highly interactive interface cannot be emphasized enough. If a navigation technique is slow, then the cognitive costs can be much greater than just the amount of time lost, because an entire train of thought can become disrupted by the loss of the contents of both visual and nonvisual working memories. »
>
> WARE 2012, tiré de AMIRPOUR AMRAII 2018, p. 12.

Tout au plus pouvons-nous émettre l'idée qu'il serait évident que la latence acceptée dans un environnement graphique de ce type soit largement supérieure à celle des environnements virtuels (réalité augmentée, visualisations immersives...), sans pour autant que nous ne puissions quantifier cet écart. On peut tout de même s'appuyer sur une estimation du temps de concentration, lié à la mémoire à court terme, décrit par Ben SHNEIDERMAN et Catherine PLAISANT :

> « A central issue is the limitation of short-term memory capacity, as outlined in George Miller's (1956) classic paper, "The magical number seven, plus or minus two." Miller identified the limited capacities people have for absorbing information. People can rapidly recognize approximately 7 (this value was contested by later researchers, but it serves as a good estimate) chunks of information at a time and can hold those chunks in short-term memory for 15 to 30 seconds. The size of a chunk of information depends on the person's familiarity with the material. »
>
> SHNEIDERMAN et PLAISANT 2004, p. 459.

Si l'on considère qu'un indicateur contient à peu près ces sept « morceaux d'information » [47] et que l'on se place dans une phase de comparaison entre les résultats de sortie de simulation de deux expériences différentes, il faut alors que le temps de sélection graphique de la seconde expérience et d'affichage

46. Ils remarquent ainsi que les utilisateurs plus habitués à des jeux vidéos rapides (« highly dynamic computer games, such as action games, racing games, or first person shooter games [...] », FORCH et al. 2017, p. 51) sont plus vite affectés par le délai de réponse que les autres.

47. Par exemple, pour un indicateur simple tel que l'évolution du nombre de foyers paysans isolés au cours du temps, il faut retenir le niveau initial, le niveau final, la tendance de sa courbe et éventuellement les dates des deux ou trois inflexions ou décrochages que l'on peut y constater.

de l'indicateur correspondant soit inférieur à quinze secondes. En prenant en compte le temps de sélection graphique, qui peut demander une manipulation de l'interface de l'application (voir section 5.4.2.3 et figure 5.25, p. 79), cela implique qu'un indicateur doit être généré en un maximum de 5 secondes [48].

Selon ces différentes considérations, dans le cadre de SimEDB, on doit donc viser à développer une plate-forme aussi rapide que possible. Celle-ci doit donc viser des temps de latence maximale de l'ordre de quelques secondes maximum, tout en sachant, dès le départ, qu'il sera impossible d'arriver aux délais de 100 ms ou 500 ms évoqués précédemment, ne serait-ce que parce que le temps de requête des données – sans compter le temps de rendu graphique – est déjà supérieur d'un ordre de grandeur.

5.4.1.3 Interopérabilité et évolutivité

Une autre contrainte forte tient cette fois au choix de l'environnement informatique qui accueillera la plate-forme d'exploration. On peut résumer ce choix à deux alternatives : un environnement local, en installant l'application sur l'ordinateur de chaque utilisateur, ou un environnement distant, où l'application serait donc accessible à distance, par exemple via une interface web [49]. Ce choix a des nombreuses répercussions, aussi bien en matière de possibilité d'accès que de facilité à faire évoluer la plate-forme.

Le choix le plus classique est de développer une application installable sur un ordinateur : cela permet de garantir une utilisation à tout moment, sans contrainte d'accès au réseau internet. Cela permet aussi d'obtenir de meilleurs performances, puisque la rapidité de l'application dépend à ce moment uniquement de la puissance de l'ordinateur, plutôt que de devoir souffrir du passage par l'intermédiaire d'un serveur.

Comme pour le choix du type de SGBD, nous avons cependant préféré nous orienter sur une solution de type distante, pour des raisons d'interopérabilité et d'évolutivité que nous allons décrivons ici.

Différents supports d'interrogation La performance d'une application locale, par rapport à une application distante, est un atout extrêmement intéressant, comme on vient de le montrer plus haut. Pourtant, cela implique une énorme contrainte : l'application doit être interopérable entre les différents systèmes d'exploitations (*Operating System*, OS) et versions de ceux-ci. Les utilisateurs potentiels de SimEDB, représentation fidèle des acteurs de la recherche, se partagent ainsi entre les trois systèmes d'exploitations majoritaires (Windows, MacOs, Linux).

48. En comptant 10 secondes pour la sélection de l'expérience. On prend ici le cas extrême (faible durée de concentration et manipulation longue), afin de garantir une expérience utilisateur satisfaisante quelque soit l'état de concentration de l'utilisateur.

49. Cette question était également posé pour le choix du SGBD en section 5.3.1.1. L'outil d'exploration et l'architecture de données sont cependant indépendants, et le choix d'un stockage des données sur un serveur distant n'implique aucunement que l'application suive la même logique. On peut ainsi avoir un SGBD distant qui serait interrogé par une application locale.

Pour permettre à chacun d'utiliser SimEDB, il faudrait donc que le développement de cette plate-forme soit compatible avec ces différents OS, ce qui est une contrainte considérable en développement logiciel.

Ne mentionnons même pas les nouveaux OS, centrés autour d'usages tactiles, tels qu'on les retrouve sur les tablettes et autres *smartphones*, qui demandent, eux aussi, de nombreuses spécificités de développement.

En somme, disposer d'une application locale universelle, c'est-à-dire utilisable quelque soit le support informatique, est une quasi-impossibilité technique, et un objectif en soit, que notre travail de recherche ne cherche aucunement à résoudre. Pour garantir la faisabilité d'une plate-forme d'exploration de données locale dédiée aux données de simulation de SimFeodal, il faudrait donc commencer par restreindre son champ d'application à un ou deux supports officiels, par exemple l'OS Windows, abandonnant de fait les utilisateurs potentiels ne disposant pas de cette architecture logicielle.

Gérer les mises à jours et modifications Comme pour les bases de données (« », section 5.3.1.1), la question de l'application locale ou distante pose un contrainte supplémentaire en matière de maintenabilité et d'évolutivité de la plate-forme choisie. Dans le cadre d'une application locale (correspondant au distribué en SGBD), la distribution des différentes mises à jour de l'application entraînent nécessairement l'installation locale, à chaque fois. Le risque est alors que tous les utilisateurs ne disposent pas d'une même version, ce qui peut entraîner, par exemple, des contradictions dans l'évaluation d'expériences, certains utilisateurs ayant accès à une version proposant des différences dans la manière de calculer ou d'afficher les indicateurs.

Sans aller jusqu'à ces extrêmes, notons qu'avec une application locale, le temps de répercussion d'une modification du code de la plate-forme est plus important : il faut en effet réinstaller sur chaque poste le logiciel ainsi modifié. Cela disqualifie de fait des modifications « en direct », par exemple lors d'une session collective d'exploration des résultats où les utilisateurs auraient des propositions de modifications à faire, ne serait-ce que pour des changements aussi infimes que des titres de graphiques ou d'axes.

Le choix d'une application web Au contraire, avec une application distante, donc basée sur l'accès, par un navigateur internet, à une application centralisée, ces problèmes ne se posent pas : des navigateurs sont disponibles pour tous les OS existants (OS dédiés aux ordinateurs ou aux usages mobiles), et interprètent de la même manière une page web, indépendamment de leur support de consultation. De plus, comme pour les SGBD, l'usage d'une plate-forme distante permet une répercussion instantanée des mises à jour et corrections : un utilisateur n'a qu'à rafraichir sa page pour que la dernière version de l'application s'affiche. De la même manière, si un utilisateur souhaite étudier un nouvel indicateur, non prévu auparavant, le temps de déploiement peut être suffisamment court pour que cela soit possible au cours d'une même session d'exploration de données.

Il y a toutefois un désavantage vis-à-vis de solutions entièrement locales, puisque les données permettant l'affichage des indicateurs doivent transiter sur le réseau internet. En cas de connexion lente, l'usage de l'application sera particulièrement difficile, et même impossible en l'absence d'une connexion.

Cette lenteur relative est toutefois compensée par un avantage de la centralisation de l'application : les calculs, parfois lourds, ne reposent pas sur les capacités individuelles des ordinateurs clients. En installant l'application sur un serveur dédié, il suffit donc d'augmenter les caractéristiques de celui-ci pour que les performances soient améliorées pour chacun des utilisateurs de l'application.

Dans le cas de SimEDB, nous disposons de ressources informatiques largement suffisantes [50] pour assurer une rapidité de traitement des données et ainsi permettre à l'application SimEDB de se dégager de ce « goulot d'étranglement » technique qu'aurait sinon éprouvée la plate-forme.

5.4.1.4 Généricité de l'interrogation et indépendance aux données

La dernière contrainte, plus technique, tient au besoin de généricité d'une plate-forme d'exploration de données vis-à-vis des données qu'elle interroge. On a résumé les possibilités et choix effectués en matière de SGBD (section 5.3.1 : « Assurer la capacité d'interrogation des données »), et décidé de ne retenir que des SGBD permettant une interrogation standardisée via des connecteurs génériques et un langage universel (le SQL).

L'infrastructure de stockage et d'organisation des données a ainsi été conçue pour être aussi générique que possible. Encore faut-il que la plate-forme d'exploration de données soit elle aussi aussi générique que possible, et donc en mesure de profiter de l'universalité du SGBD choisi.

Indépendance au support de données Une contrainte forte est donc constituée par la capacité de la plate-forme a être indépendante de la source des données : quelque soit le SGBD choisi, les requêtes émises par la plate-forme doivent être les mêmes, sans requérir d'adaptations spécifiques en dehors de la désignation du lieu de stockage des données et des pilotes du SGBD.

Dans les faits, lors de la construction de SimEDB (cf. section 5.2 : « Comment explorer les sorties de SimFeodal ? »), plusieurs solutions de stockage de données ont été employées successivement, au fur et à mesure des limites rencontrées chez chacune. Depuis les premières implémentations des rapports automatiques jusqu'à l'utilisation de SimEDB dans son état actuellement discuté, les données de sorties de simulation ont ainsi été tour à tour interrogées depuis de simples fichiers CSV au départ jusqu'au SGBD ultra-performant MapD, en passant par des solutions intermédiaires plus classiques (SQLite et MonetDB notamment).

50. En nous appuyant dans un premier temps sur un serveur de calcul interne à l'UMR Géographie-cités, puis sur un serveur de calcul partagé mis à disposition par la « Très Grande Infrastructure de Recherche » Huma-Num ensuite.

Il n'était donc aucunement question d'avoir à adapter le code source des programmes permettant de générer les indicateurs depuis les données, mais au contraire, de s'assurer d'utiliser des bibliothèques logicielles indépendantes des données, c'est-à-dire capables d'exécuter les mêmes chaînes de traitements quelle que soit la provenance des données.

On peut expliciter ce propos à l'aide de l'exemple caricatural des logiciels de type tableurs. Dans ce type d'application, on peut écrire des programmes (les « macros ») qui permettront l'ouverture d'un fichier CSV et effectueront des calculs dessus pour en tirer par exemple des résumés. Dans ce même tableur, on peut aussi faire appel à des sources de données différentes (bases de données Access ou SQL par exemple), mais les programmes (macros) seront alors à réécrire en quasi-totalité pour les adapter aux différences de sources de données.

Dans notre cas, les sources de données ayant très largement évolué au cours du temps, on ne pouvait faire reposer notre application sur une plate-forme qui demande une adaptation forte à la provenance des données, comme c'est le cas des tableurs. Il était donc nécessaire de s'appuyer sur des environnements logiciels (les bibliothèques logicielles) permettant une forte généricité vis-à-vis des sources de données.

Indépendance aux requêtes et modularité de l'implémentation Pour garantir cette généricité, il est donc nécessaire de s'assurer que le mode de communication de la plate-forme vers les données soit bien basé sur un langage universel : le SQL. Il convient donc de choisir un ensemble de technologies permettant de générer des requêtes SQL, quand bien même l'expression de ces requêtes elles-mêmes serait conçue dans un autre langage. Pour les requêtes complexes, le SQL tend en effet a être peu lisible, les opérations s'emboîtant les unes dans les autres de manière très linéaires, et donc, souvent verbeuses. En SQL pur, il est donc peu évident de créer une implémentation modulaire d'une requête, c'est-à-dire permettant une factorisation des commandes et un paramétrage des entrées.

Les indicateurs de sortie de SimFeodal sont, on l'a vu, assez fréquemment basés sur le même type d'opération : on observe par exemple souvent l'évolution du nombre d'agents d'un certain type (agrégats, foyers paysans...)au cours du temps. Dans le cas de cet exemple, en SQL, pour spécifier une requête permettant de récupérer le nombre de foyers paysans au cours du temps, groupés par année et avec un filtre sur certaines simulations, il ne faut que quelques lignes de code. Pour que cette requête devienne générique, c'est-à-dire indépendante du type d'agent qui en deviendrait un argument, il est nécessaire d'y ajouter de nombreuses lignes de code. Cela revient potentiellement à doubler, pour chacun des indicateurs, la longueur du code-source requis pour l'expression des requêtes, et bien sûr à les rendre plus complexe à modifier et corriger. De plus, les modes d'expression qui permettent de modulariser du code SQL peuvent varier fortement selon les SGBD choisis, n'étant pas strictement décrits dans les normes SQL. Par exemple, la déclaration d'une variable, par exemple pour paramétrer le nom de la table contenant les agents, est très différente dans les deux SGBD les plus utilisés (MySQL et PostgreSQL).

Faire appel à un langage intermédiaire, générant du SQL en sortie depuis une entrée sous forme d'un « *Domain Specific Language* » (DSL) permet ainsi de bénéficier d'une part de l'universalité du SQL, et d'autre part, d'une syntaxe plus expressive que celle du SQL. En utilisant un DSL, plus adapté à la manipulation de données qu'à la sélection de sous-ensembles, on gagne donc en modularité d'implémentation , et donc en ré-utilisation de fonctions plus génériques, ce qui permet de disposer d'un code-source plus robuste, ré-utilisable et évolutif.

Conclusion : Vers une plate-forme web générique et intuitive

Dans cette sous-partie, nous avons présenté les principales contraintes qui ont orienté le choix des cadres méthodologiques et techniques utilisables pour concevoir une plateforme telle que SimEDB.

En premier lieu, on fait le choix de se tourner vers une plate-forme implémentée sous forme d'application web, utilisable depuis un simple navigateur – donc inter-opérable entre les différents supports technologiques –, ce qui exclue de fait quantités d'outils, de logiciels et de bibliothèques logicielles pensées pour l'exploration interactive de données.

On souhaite de plus que la plate-forme utilisée dispose d'une interface aussi épurée que possible, donc nécessairement très adaptée au cas particulier des données issues de simulation que l'on manipule. Là encore, l'étendue des possibles est restreinte, éliminant l'ensemble de solutions « clefs-en-main », par exemple conçues autour des « webSIG » ou de bibliothèques logicielles de visualisations interactives intégrées.

L'utilisation de la plate-forme doit être aussi efficace que possible, en cherchant à minimiser les temps de latence entre sélection interactive et affichage des indicateurs en résultant. On devra donc privilégier des ensembles technologiques récents et performants, intrinsèquement dédiés à l'interactivité, au détriment de *frameworks* plus génériques.

Enfin, il faut que cette solution, dans la mesure du possible, soit en mesure de proposer une syntaxes d'interrogation de données modulaire, factorisée, et plus expressive que le SQL sur lequel elle doit toutefois s'appuyer.

Ces contraintes sont des éléments génériques à prendre en compte dans la conception d'un outil d'exploration de données, et elles dépassent largement notre seul cas d'utilisation. Nous n'avons pour autant pas tenté de brosser un paysage complet des contraintes potentielles, liées aux différents usages possibles, qui peuvent guider les choix techniques et méthodologiques de la conception d'un outil. La relative spécificité de SimEDB tient à la combinaison des contraintes identifiées et à la combinaison des choix effectuées pour les dépasser, que nous allons maintenant expliciter.

5.4.2 Construire une plate-forme interactive pour l'exploration de sorties de simulation

Dans cette dernière sous-partie, nous allons donc présenter les choix – techniques, esthétiques et interactifs – qui ont été adoptés dans la conception et l'implémentation de SimEDB. Nous les présentons ici de manière linéaire, dans l'ordre quasi-chronologique du développement, mais il est important de garder en considération que ces éléments sont intimement intriqués. Un choix technique, par exemple, peut conditionner les types d'interactions possibles, parce que l'utilisation de telle méthode d'interaction peut n'être proposée que dans tels et tels environnements logiciels.

Notons enfin que l'application SimEDB présentée ici, aussi bien dans son usage que dans sa conception, représente un instantané de développement, qui correspond à la période de rédaction du présent chapitre : à l'instar d'un modèle, une plate-forme peut et doit évoluer pour s'adapter aux besoins de ses utilisateurs tant qu'elle est utilisée. Les technologies et choix esthétiques introduits n'ont pas toujours été présents, et auront sans doute à évoluer dans la suite de la « durée de vie » de SimEDB. Pour les raisons évoquées en termes de facilité de mise à jour d'une solution distante, cela ne pose toutefois aucun problème vis-à-vis de l'utilisation de la plate-forme, largement indépendante, en matière de temporalités, du présent ouvrage.

5.4.2.1 Choix des technologies

Nous présentons ici les technologies mobilisées dans le cadre du développement de SimEDB. Le but n'est pas d'entrer dans les détails de l'implémentation[51], mais bien de justifier et présenter les choix relatifs aux technologies employées, en restant à un niveau assez général[52]. Il nous paraît important d'entrer dans ces choix qui relèvent plus de la technique que de la méthodologie en ce qu'ils concourent de la volonté de reproductibilité de la thèse, et particulièrement de la reproductibilité de la démarche, conceptuelle et méthodologique, mise en place. Nous portons la conviction que l'ensemble de technologies assemblées ici dans notre « chaîne de traitement » est très largement ré-utilisable, dans le cadre d'adaptations à d'autres cas d'études, mais aussi et surtout, pour une multitude de problématiques requérant une analyse visuelle de données massives (on y reviendra dans le chap 7).

Technologies webs « natives » et adaptativité Au cours de la dernière décennie, les interfaces physiques de consultation de médias informatiques se sont largement diversifiées. Cela a provoqué une hétérogénéisation importante aussi bien des modes d'interaction (dispositifs « tactiles ») que des modes d'affichages (les tailles et résolutions des écrans n'ont jamais été aussi diverses et imprévisibles).

51. Le code source de SimEDB – et l'historique de son versionnement – sont, pour cela, disponibles en ligne sous licence libre, sur la plate-forme Github : github.com/RCura/SimEDB

52. À ce titre, les quelques lignes de codes présentes par la suite servent un but illustratif et descriptif, et nous semblent remplir ce rôle bien plus efficacement que n'importe quel schéma structurel ne le pourrait.

En conséquence, les normes de présentations graphiques ont évolué, vers plus d'« adaptativité », en particulier avec l'avènement du « responsive web design » (« conception de sites web adaptatifs ») qui permet de prévoir efficacement l'agencement d'une page web quelque soit le support de consultation.

Les technologies qui prédominaient dans la réalisation d'applications web interactives il y a quelques années [53] ont largement disparu suite à un manque d'adaptation à ces nouveaux support.

Pendant ce temps, de nouveaux standards du développement web (HTML5 entre autre) ont émergé et atteint un niveau de maturité suffisant pour remplacer l'ensemble des possibilités (et les étendre) proposées par ces anciens environnements trop monolithiques.

Ces technologies, aujourd'hui indispensables, reposent sur des codes standardisés, verbeux et peu explicites [54], mais toutefois assez universellement interprétables par les navigateurs. Pour pallier à leur faible expressivité, on peut faire appel à des *frameworks* graphiques qui en simplifient l'usage : comme les DSL évoqués plus haut, ce sont des ensembles de bibliothèques logicielles qui génèrent à l'aide d'instructions courtes et simples les centaines de lignes de codes nécessaires à l'affichage interopérable, universel et constant d'un site ou d'une application web.

Nous avons donc fait le choix de nous concentrer sur des environnements standardisés, capables de générer du `HTML` (« *HyperText Markup Language* »), lui-même mise en forme à l'aide de styles `CSS` (« *Cascading Style Sheets* ») et rendu interactif par du code `JavaScript`.

À ce titre, le framework `Bootstrap` [55] s'est révélé extrêmement utile dans le *design* de l'interface de SimEDB (et des versions précédentes), tant il simplifie l'expressivité d'une mise en page à l'aide d'une grille graphique et de composants interactifs ré-utilisables.

Le choix d'environnements de développement intermédiaires [56] Pour construire des applications interactives en lignes, de multiples choix sont possibles, et on peut les catégoriser selon le niveau de développement qu'ils demandent. Par exemple, il est tout à fait possible de s'appuyer sur des briques logicielles de bas niveau (ce que l'on appelle communément *framework*), et de développer à partir de celles-ci toute l'interface et le fonctionnement d'une application.

Cette approche, majoritaire dans la construction d'applications actuelles (avec des *frameworks* basés sur le langage JavaScript tels que `ReactJS` ou `AngularJS`,

53. Applications en FLASH, *applets* Java...

54. Il suffit de consulter le code-source d'une page web contenant des visualisation interactives pour le constater. Les assemblages de langages SVG, CSS et JavaScript sont ainsi assez largement indéchiffrables pour qui n'en est pas un spécialiste.

55. http ://getbootstrap.com/

56. Ne pas oublier, dans le positionnement (chap1) de consacrer au moins un paragraphe (ou encadré) au choix « militant » de ne se tourner QUE vers des outils libres, sans exception.