5

Explorer visuellement des données de simulation massives pour analyser le comportement d'un modèle.

Version 2019-10-24

— 19/05/2019 : Reprises Lena + Envoi à Hélène
— 24/10/2019 : Reprises Hélène + Envoi à Anne

Sommaire

Introduction **2**
5.1 Capter les sorties de SimFeodal **3**
5.1.1 Masse des données 3
5.1.2 Tenir compte de la stochasticité du modèle - les réplications 7
5.1.3 Des réplications aux expériences 9
5.1.4 Des données aux indicateurs 10
5.2 Comment explorer les sorties de SimFeodal ? **12**
5.2.1 Observer les simulations en direct ou *a posteriori* . . . 12
5.2.2 Générer les indicateurs 16
5.2.3 Organiser les indicateurs en rapports paramétrables . 18
5.2.4 Organiser les rapports : les *dashboards* 23
5.2.5 Interagir avec les rapports : exploration interactive . . 27
5.2.6 Explorer en comparant : la plateforme SimEDB 32
5.3 Organiser les données **35**
5.3.1 Assurer la capacité d'interrogation des données 35
5.3.2 Structuration des données de SimFeodal 49
5.4 Une plate-forme d'exploration de données de simulations : SimEDB **57**
5.4.1 Contraintes 57
5.4.2 Construire une plate-forme interactive pour l'exploration de sorties de simulation 67
Conclusion **83**

Introduction

A écrire une fois que le chapitre aura été découpé.
-> Définir notamment la démarche :

- contraintes générales
- + type explo SimFeodal
- -> choix méthodos
- + spécificités SimFeodal
- -> choix méthodos/techniques

5.1 Capter les sorties de SimFeodal

Pour évaluer un modèle, on s'appuie sur plusieurs indicateurs de sortie de simulation, de types divers (indicateurs numériques, graphiques, cartographiques etc., cf. chapitre 3, partie théorique). Quand le nombre d'indicateurs devient important, comme c'est le cas dans le modèle SimFeodal (chap 3, partie présentation des indicateurs), la consultation des indicateurs pendant le déroulement d'une simulation devient difficile. La complexité de ces indicateurs augmente dans le cas d'un modèle stochastique, où il est nécessaire de multiplier les réplications afin d'avoir une idée fiable des tendances simulées par le modèle. Le travail de paramétrage d'un modèle requiert de plus de mener différentes expériences, c'est-à-dire de faire varier les paramètres (chap 4) du modèle, démultipliant encore la masse des sorties, et avec elle, la complexité de leur analyse.

Nous détaillons ici les contraintes qu'entraînent ces différentes spécificités des données issues de simulations. Ces contraintes sont transversales à plusieurs types de modèles, et on peut noter que certains autres types de modèles peuvent faire face à d'autres contraintes, propres ou génériques.

Dans l'ensemble, les modèles peuvent être amenés à soulever les problèmes génériques à la production de données, quelles qu'en soient la source. Cette recherche n'a pas vocation à dresser le portrait de l'ensemble des contraintes et solutions relatives à l'enregistrement et au stockage de données. Cette partie vise à identifier les plus fortes limites qui rendent difficile l'enregistrement des données issues d'un modèle de simulation à base d'agents, descriptif, fortement stochastique, et exploratoire tel que SimFeodal.

5.1.1 Masse des données

Facteur multiplicatif	**Intitulé**	**Données**		**Indicateurs**		***Type de question***
		Quantité	**Poids**	**Type**	**Quantité**	
	Un agent	1 ligne	≈ 200 octets	Visualisation en direct	Attributs de l'agent	*Le comportement individuel des agents semble-t-il plausible ?*
×5000 [agents]	Un pas de temps	5 000 lignes	≈ 1 Mo	Visualisations en direct	3-4 indicateurs directs	*Le déroulé d'une simulation semble-t-il correct ?*
×20 [pas de temps]	Une simulation	100 000 lignes	≈ 20 Mo	Visualisations en direct	≈ 10 indicateurs	
×20 [réplications]	Une expérience	2 millions de lignes	≈ 400 Mo	Indicateurs de sortie	≈ 30 indicateurs (variabilité des réplications)	*Quel est le comportement agrégé du modèle ?*
×25 [expériences]	Un modèle paramétré ou calibré	100 millions de lignes	≈ 10 Go	Indicateurs de sortie	≈ 3000 indicateurs (à comparer entre les expériences)	*Peut-on obtenir un meilleur ajustement du modèle ?*
×100 [explorations]	Un modèle exploré/connu	10 milliards de lignes	≈ 1 To	Indicateurs numériques synthétiques	—	*Comment le modèle réagit-il aux variations de ses paramètres ?*

TABLEAU 5.1 – Synthèse des questionnements liés à l'exploration de données issues de modèle et de la multiplication des données et indicateurs selon la hiérarchie des simulations.

Dans un premier temps, il convient de noter que l'ensemble des indicateurs observés en sortie de SimFeodal reposent sur des données qu'il est nécessaire de produire et d'enregistrer tout au long de la simulation. Ainsi, pour pouvoir tracer le graphique de l'évolution du nombre d'agrégats au cours du temps, il faut avoir accès à cette information, et dès lors, enregistrer, à chaque pas de temps, cette valeur dans un fichier numérique adapté. Cette information, en tant que telle, est assez faible, aussi bien en valeur sémantique qu'en valeur prise en mémoire. La masse représentée par cette information est toutefois démultipliée par la quantité d'indicateurs de sortie : plus ces derniers sont nombreux, plus la quantité de valeurs à stocker augmente. À chaque pas de temps, il faudra enregistrer les valeurs de plusieurs variables (voir tableau 5.1). Cette pratique est habituelle, et un format de données tabulaire se prête bien à un tel enregistrement : une ligne pour chaque pas de temps, et une colonne pour chaque variable à enregistrer. On obtiendrait ainsi en sortie de simulation un tableau contenant 20 [1] lignes et une cinquantaine de colonnes [2], ce qui serait assez raisonnable pour une unique simulation.

Évolution des indicateurs. Dans le cadre d'un modèle exploratoire (cf. chapitre 1), cette solution doit être écartée car elle ne permet pas de faire face à d'éventuelles évolutions des indicateurs au fur et à mesure du cycle de vie du modèle. Au fur et à mesure que le calibrage d'un modèle progresse, par exemple, on peut chercher à raffiner les indicateurs utilisés. Les indicateurs utilisés au départ peuvent alors ne plus montrer assez de variabilité entre des versions très proches de modèles. En effet, quand l'ajustement des données produites par la simulation aux objectifs empiriques s'améliore, il peut être difficile de départager plusieurs versions concurrentes d'un modèle. On peut alors chercher à créer de nouveaux indicateurs, qui donneront une information plus détaillée sur l'un des résultats de simulation.

Cet exemple illustre le cas des ajouts d'indicateurs de sorties, mais on peut aussi être confronté à des modifications des indicateurs existants. Dans SimFeodal, par exemple, le nombre de paroissiens moyen à chaque pas de temps peut être un indicateur utile au départ, mais on peut être amené à faire évoluer cet indicateur en une étude de la médiane si les nouvelles étapes de paramétrage du modèle en augmentent la variabilité.

Les indicateurs peuvent donc évoluer au cours du temps de vie du modèle (cf. encadré chap 3), ou plus simplement, on peut être amené à réaliser une observation plus fine des sorties du modèle au fur et à mesure de l'analyse de ce dernier. On se trouverait alors dans une situation impossible requérant de ré-exécuter les simulations après avoir adapté ou mis en place l'indicateur

1. Il s'agit ici du nombre de pas de temps de SimFeodal. On notera que ce nombre est particulièrement faible au regard de très nombreux modèles de simulation, en particulier vis-à-vis de ceux qui visent à provoquer l'émergence d'un phénomène. Ces modèles sont en général théoriques, et n'ont qu'une faible correspondance entre pas de temps et durée réelle du phénomène modélisé. Dans le cas de SimFeodal, où le temps est un élément crucial du modèle, la résolution temporelle du modèle ne peut être diminuée artificiellement (voir chap2, section 2.2.2.2), et l'on se satisfait donc de ce nombre d'itérations relativement faible.

2. Ce qui correspondrait par exemple à environ une colonne par indicateur, en plus des quelques colonnes de bases relatives à l'état d'ensemble de la simulation.

voulu.

Données agrégées et individuelles. Un dernier point invalide cette solution d'enregistrement : une part importante des indicateurs s'appuie sur des données individuelles et non agrégées. Ainsi, on peut, à chaque pas de temps, enregistrer le nombre de paroisses, leur superficie moyenne ou encore le nombre moyen de paroissiens que chacune dessert. Mais cela ne permet en aucun cas d'en dresser une cartographie, c'est-à-dire de réaliser une carte de la localisation et des aires d'attraction des paroisses. Cela demanderait, par définition, d'enregistrer la géométrie de chaque paroisse à chaque pas de temps, les configurations spatiales (localisation de chacune et donc distribution spatiale de l'ensemble) variant à chaque simulation.

Pour faire face à cette situation, on a donc fait le choix, dans SimFeodal, d'enregistrer les états des variables à des niveaux d'agrégation multiples, y compris au niveau de l'agent, à chaque pas de simulation. Dans le cas des paroisses, le volume de données résultant reste contenu : on obtient un tableau d'environ 2000 lignes [3] et une dizaine de colonnes [4]. L'enregistrement systématique de l'état de chaque agent à chaque pas de temps est toutefois bien plus gênant dans le cas d'autres agents, par exemple les foyers paysans. Il est en effet nécessaire d'enregistrer les attributs de chacun d'entre eux pour être en mesure d'étudier les liens entre les valeurs de satisfactions et les choix de déplacement, ou encore d'observer la composition précise de la distribution des satisfactions. Avec 4000 foyers paysans au minimum à chaque pas de temps, les données changent d'ordre de grandeur [5] : chaque simulation requiert de générer un fichier contenant des dizaines de milliers de lignes, pour un total, pour cet unique fichier, d'une dizaine de mégaoctets occupés.

Au final, l'enregistrement d'un état représentatif d'une simulation est difficile. Cela requiert de disposer de suffisamment d'éléments numériques pour pouvoir générer les indicateurs de sortie et rendre compte d'une partie de leur évolution. La masse de données produite est de ce fait nécessairement conséquente, comme indiquée dans la ligne « simulation » du tableau 5.1.

Coût temporel des simulations. Un dernier aspect de l'exploration de données issues de simulations, primordial, vient finir invalider l'approche d'enregistrement direct des indicateurs : le coût temporel. L'exécution d'une simulation requiert un certain temps de calcul (3 à 4 minutes pour une exécution du modèle SimFeodal dans la version présentée dans le chapitre 2). Ce temps de calcul ne peut être optimisé que dans des proportions faibles sans avoir à bouleverser l'implémentation des mécanismes, ce qui représenterait un coût temporel encore plus important (voir la succession d'étapes du chapitre 4). En l'état actuel du modèle, la production des données a donc un coût temporel

3. Avec une moyenne de 150 paroisses, cela représente $20_{\text{[pas de temps]}} \times 150_{\text{[paroisses]}} \approx 3000$ lignes pour chaque simulation.

4. Les identifiants de la simulation (nom, graine aléatoire), le pas de temps, l'identifiant de la paroisse, puis les différents attributs et enfin la géométrie stockée dans une colonne textuelle.

5. $20_{\text{[pas de temps]}} \times 4000_{\text{[foyers paysans]}} \approx 80\,000$ lignes pour une exécution du modèle.

élevé.

Qui plus est, ce coût est fortement dépendant du nombre de simulations exécutées : si le modèle est exploratoire, donc sujet à de nombreux changements, et notamment à l'ajout ou à la modification des indicateurs observés, il peut donner lieu à d'encore plus nombreuses simulations. Avec une structure de données fixe et agrégée, on ne pourrait introduire de nouveaux indicateurs, et la mise en comparaison des simulations précédentes impliquerait leur adaptation et re-production, c'est-à-dire la ré-exécution systématique de l'ensemble des simulations précédentes à chaque changement dans les indicateurs. L'introduction de nouveaux indicateurs est pourtant très fréquente, en particulier dans le cas d'un modèle exploratoire où l'on affine au fur et à mesure de l'évolution du modèle ce que l'on peut y observer.

En tenant compte de ces éléments, on a tout intérêt à se prémunir de ré-exécutions du modèle, et donc à enregistrer l'état de variables qui ne seraient pas encore mobilisées pour la production d'indicateurs. Dans l'exemple du nombre de paroissiens, il faudrait en enregistrer au minimum les moyennes, médianes, et sans doute quelques paramètres de dispersions en plus, voir les quantiles, afin d'adapter les indicateurs de sortie de la manière la plus adéquate aux sorties des différentes versions du modèle. Dans le cas contraire, pour chaque changement ou ajout d'indicateur, il faudrait relancer des exécutions du modèle sur l'ensemble des jeux de paramètres précédents afin d'être en mesure d'avoir des indicateurs comparables entre les versions.

Enregistrer l'ensemble des variables d'un modèle est aisé dans le cas d'un modèle théorique simple, par exemple dans le cas d'un modèle comme celui de Schelling (SCHELLING 1971). Cela se complique quand il s'agit d'enregistrer les variables d'un modèle plus complexe comme SimFeodal. Celui-ci requiert en effet bien plus de variables globales (ref chapitre 4, dans distinctions variables, paramètres etc.), pour représenter l'état du système dans son ensemble à chaque instant. Surtout, SimFeodal est un modèle qui voit interagir plusieurs sortes d'entités, chacune relatives à différents niveaux de granularité spatiale et sociale. Afin d'avoir tous les éléments en main une fois la simulation achevée, il est donc nécessaire d'enregistrer l'ensemble des variables non seulement globales, mais aussi afférentes à chacun des types d'agents. D'un unique tableau de données exhaustif en sortie du modèle de Schelling, on passe donc à plusieurs tableaux, dont les variables respectives seront propres à chaque type d'agent.

De plus, pour que les résultats de simulations soient comparables tout au long du développement du modèle, en dépit de l'évolution des indicateurs qui les constituent, il est nécessaire d'enregistrer l'état individuel des agents, et pas seulement des données agrégées. Cela démultiplie la quantité d'informations qu'il est nécessaire de stocker (ligne « simulation » du tableau 5.1), qui plus est quand le nombre d'agents est important [6].

6. Dans les dernières versions de SimFeodal, dont les résultats sont présentés dans le chapitre 6, on est passé de 4 000 foyers paysans à 40 000. On peut donc décupler les valeurs des colonnes « quantité » et « poids »…

5.1.2 Tenir compte de la stochasticité du modèle - les réplications

Un modèle stochastique. Comme on l'a vu dans le chapitre 3, une simulation ne suffit pas à évaluer le modèle. SimFeodal est ainsi un modèle stochastique, c'est-à-dire qu'une large partie des mécanismes qui l'animent sont basés sur des tirages aléatoires. Cet aléa est évident dans les mécanismes faisant appel à un tirage aléatoire explicité, par exemple le choix de déplacement ou non d'un foyer paysan (cf. chap2, mécanisme déplacement). Dans le cas de ce mécanisme, un foyer paysan mobile se déplacera selon une probabilité dépendant de sa satisfaction. Et s'il y a probabilité, il y a donc aléa. Même avec une forte satisfaction — 99% par exemple —, il reste 1% de chance qu'un foyer se déplace, ce qui, sur un grand nombre de tirages (chaque foyer paysan, à chaque pas de temps), aboutit à une probabilité de réalisation non négligeable. Et cette probabilité de réalisation sera encore supérieure pour des foyers paysans ayant des niveaux de satisfaction légèrement moindre mais cependant globalement très élevés, supérieurs à 90% par exemple. En analysant les sortie du modèle, on aura donc la présence d'*outliers*, qu'il sera important d'isoler. Ils présenteront en effet des comportements contre-intuitifs puisque résultant d'une probabilité extrêmement faible. L'aléa a un poids important dans ce type de mécanisme.

Même dans le cas de mécanismes plus anodins, l'aléa est fortement présent, puisqu'il est au cœur de la conception de SimFeodal. L'ordre d'exécution, c'est-à-dire l'ordre aléatoire dans lequel les agent sont appelés pour exécuter leurs mécanismes, aura donc un impact important sur les indicateurs de sortie de simulation, sans que cet impact ne puisse être caractérisé au moyen d'indicateurs agrégés. Par exemple, les seigneurs peuvent créer des châteaux, sous condition de puissance (cf. chapitre 2, section 2.3.12). Pour créer ces châteaux, il faut que des agrégats soient « disponibles », c'est-à-dire ne comportent pas de château pré-existant à une certaine distance. Cette contrainte devient rapidement le facteur principal de la limitation de l'apparition de châteaux. Si un seigneur est plus souvent que les autres « appelé » en premier pour exécuter ce mécanisme, il pourra profiter des nouveaux agrégats disponibles pour créer ses châteaux. A force de création de châteaux, il sera relativement plus puissant, et pourra donc créer d'autant plus de châteaux relativement aux autres seigneurs. Il y aura donc une hiérarchie forte dans le nombre de châteaux possédés par seigneur.
Au contraire, si l'ordre d'appel des mécanismes favorise des seigneurs différents à chaque pas de temps, alors plus de seigneurs seront en mesure de créer des châteaux, et la hiérarchie sera alors plus faible.

Ces mécanismes sont sensibles à l'ordre d'appel, et il est ainsi difficile de discerner ce qui relève d'une tendance simulée et ce qui relève de fines variations dues à l'aléa, dans le comportement du modèle.

Un modèle complexe. On pourrait objecter qu'en considérant les agents de manière agrégée, donc globale, les tirages de probabilité seront appliquées à suffisamment d'individus pour que le résultat statistique soit cohérent et robuste au niveau de la population dans son ensemble. En corollaire, le compor-

tement de chaque agent serait régulé par tant d'états aléatoires qu'on entrerait dans le cadre d'application de la loi forte des grands nombres, les agents adoptant alors en moyenne un comportement proche de l'espérance (moyenne théorique) de chaque tirage. Avec ces considérations, on pourrait justifier la robustesse probable des différentes exécutions de SimFeodal.

SimFeodal n'est toutefois pas simplement un modèle stochastique, mais avant tout, un modèle complexe, c'est-à-dire s'inscrivant dans le champs des systèmes complexes. Sans vouloir ici entrer dans les détails des implications et raisons de ceci, on peut simplement en retenir qu'un modèle tel que SimFeodal est extrêmement sensible aussi bien aux conditions initiales qu'aux différents tirages aléatoires (voir l'analyse de sensibilité, chap 6, section 6.3).

Pour illustrer, on peut s'appuyer sur un exemple, caricatural mais possible : à l'initialisation, tous les foyers paysans, placés aléatoirement dans l'espace, seraient concentrés dans un espace d'étendue restreinte. Seul un énorme agrégat émergerait donc, et aucun pôle ne serait susceptible dès lors de diviser cet agrégat géant. On atteindrait ainsi une situation très éloignée des configurations spatiales observées empiriquement, et très éloignée aussi des réalisations habituelles du modèle. En présence d'un seul agrégat, les possibilités de développement d'attracteurs (châteaux et paroisses) pourraient tout aussi bien être fortes que faibles. À partir d'une telle configuration initiale, on ne peut savoir si la situation convergerait vers un agrégat « paradisiaque », extrêmement développé et doté de pôles satisfaisants, ou au contraire, vers un agrégat « prison », où aucun des foyers paysans ne serait satisfait, mais n'aurait non plus d'alternative.

Cet exemple fictif, volontairement caricatural, ne s'est pas présenté jusqu'ici, mais le cas échéant il faudrait pouvoir repérer ce type de comportement aberrant. Cela serait par exemple utile pour éventuellement les distinguer des autres simulations et ne pas le laisser contrôler l'analyse d'un jeu de paramètres données. De plus, cet exemple concerne uniquement une configuration initiale qui présenterait des caractéristiques tout à fait exceptionnelles. Plus généralement, il existe un grand nombre de situations initiales potentielles éloignées de l'empirique et même du vraisemblable. Les réalisations aberrantes peuvent apparaître à toute étape de la simulation, et déformer les tendances observées dans les indicateurs de sortie du modèle. Au delà de l'initialisation, elles peuvent être issues de tirages aléatoires particulièrement défavorables, ou encore apparaître suite à une succession d'événements improbables qui s'autorenforceraient. Pour distinguer ces réalisations aberrantes de ce que l'on pourrait caractériser d'une tendance normale, il est nécessaire de multiplier les réplications, afin de constituer un contexte suffisant pour isoler ces événements anormaux.

Enregistrement des réplications. On ne peut donc pas raisonner sur une unique simulation pour évaluer un jeu de paramètres (cf. chap 3). On ne peut pas non plus se contenter d'une agrégation des résultats des différentes réplications, sous la forme de moyennes ou d'écarts-types, selon qu'on s'intéresse par exemple à la tendance générale ou qu'on cherche à observer les variations

que peut entraîner l'aléa.

Pour ces raisons, et pour être en mesure d'embrasser l'entière diversité des sorties de simulations issues de variation de la graine aléatoire, il est donc nécessaire de mener plusieurs réplications de chaque simulation, et d'enregistrer l'entièreté des sorties de simulations dans chacun des cas. Le jeu de données produit par une simulation, contenant quelques dizaines de milliers de lignes, est ainsi obligatoirement multiplié par le nombre de réplications. Pour l'exploration de SimFeodal, après différents tests, ce nombre a été fixé à 20 réplications (J'en aurais sans doute parlé dans le chapitre 3 (évaluation), mais à laisser ici jusqu'à ce que ce soit certain.). La vingtaine de mégaoctets issue d'une simulation devient donc approximativement 400 mégaoctets, et le nombre de lignes contenues, par exemple pour les foyers paysans, passe d'environ cent mille à deux millions [7] (voir la ligne « expérience » du tableau 5.1).

5.1.3 Des réplications aux expériences

Comme décrit dans le chapitre 4, le paramétrage de SimFeodal a demandé plusieurs étapes [8]. De plus, chacune de ces étapes représente plusieurs sous-étapes – les expériences – faites d'essais et d'erreurs, en faisant varier à chaque fois les valeurs de paramètres de SimFeodal. Afin de construire le modèle, puis de l'explorer de manière plus systématique, il a été nécessaire de tester des dizaines de configurations de paramètres. Pour comparer, à chaque nouvelle version du modèle, les résultats produits par rapport aux résultats de la version précédente, il est indispensable de conserver, au minimum, l'ensemble des jeux de données de cette version précédente.

Cet archivage des résultats immédiatement précédents n'est pourtant pas suffisant, pour des raisons tenant à la reproductibilité et à traçabilité du modèle obtenu au final. On serait en effet tenté, à chaque nouvelle version « majeure » du modèle, de ne conserver que les indicateurs de sorties des versions précédentes en considérant que le modèle a atteint une phase de maturité supérieure à chaque fois. Les étapes intermédiaires, reléguées au rang de brouillons ou d'esquisses, deviendraient alors inutiles. Le processus de conception et de paramétrage d'un modèle n'est pourtant pas linéaire (cf. chapitre 4), et on peut avoir besoin de comparer une version intermédiaire « actuelle » à une version intermédiaire précédente, quitte à réaliser qu'une modification erronée a été ajoutée au modèle.

La conservation des résultats de chacune des expériences joue donc à nouveau un rôle multiplicatif dans la masse de données à conserver (voir le tableau 5.1). En supposant que le modèle ait compté environ 25 versions [9], on obtient envi-

7. Si cette quantité de données semble tout à fait raisonnable et peut largement être traitée sur un ordinateur classique, on peut toutefois noter qu'elle dépasse déjà le maximum de lignes (limitées à 2^{20}, soit un peu plus d'un million) que les tableurs classiques – LibreOffice ou Microsoft Excel dans leurs dernières versions – sont en capacité de gérer.

8. Y mettre un tableau avec les versions + nombre de sous-version du modèle.

9. Dans SimFeodal, par exemple, on distingue 6 versions « majeures » (cf. tableau dans chap 4, elles-mêmes composées de 4 à 10 sous-versions) à corriger quand tableau finalisé.

ron 500 simulations à stocker, puis à devoir analyser et comparer. Cela représente une somme considérable de données (voir tableau 5.1), qui se chiffrent en dizaines de millions d'enregistrement [10].

En matière de stockage, il ne s'agit jamais que de quelques gigaoctets de données, pourtant à la limite de ce que l'on peut traiter sur un ordinateur individuel [11].

On ne mentionne ici que les expérimentations issues des étapes de paramétrage. Les phases suivantes, visant à l'exploration du comportement du modèle (analyse de sensibilité, calibration…), demandent ainsi d'exécuter, et donc d'enregistrer, une masse bien plus importante de simulations.

5.1.4 Des données aux indicateurs

Dans l'ensemble, l'enregistrement et la sauvegarde des données issues de simulations constituent, pour les modèles de simulations basés sur de nombreux agents et mécanismes, une contrainte importante vis-à-vis de l'exploration du comportement de ces modèles.

C'est particulièrement le cas pour SimFeodal, où l'on ne peut se contenter de produire à la volée les indicateurs pour des raisons de reproductibilité [12].

Analyser une masse de données. La masse de données en sortie est impressionnante et requiert dès lors, d'un point de vue technique, d'utiliser des outils adaptés à la manipulation de grands jeux de données. Cela exclut de fait l'outillage habituel, accessible, de la géographie quantitative, ne laissant par exemple pas la possibilité d'utiliser les outils à interface graphique classiques. Au contraire, face à des données de cet ordre, seules des solutions statistiques, basées sur des analyses en ligne de commande, peuvent être mobilisées. Ces solutions doivent en plus être appuyées par des capacités de calculs importantes, sans toutefois justifier encore l'usage de technologies de calcul intensif [13]. . Cela pose une contrainte dans l'accessibilité aux analyses : le traitement des données requiert des compétences spécifiques en analyse de données volumineuses. Dans un contexte interdisciplinaire caractérisé par une large hétérogénéité en matière de pratiques quantitatives, il n'est pas possible de se contenter d'envoyer les jeux de données produits aux thématiciens – qui ne disposent le

10. $20_{\text{[pas de temps]}} \times 5000_{\text{[agents]}} \times 20_{\text{[réplications]}} \times 25_{\text{[jeux de paramètres]}} \approx 100\ 000\ 000$ de lignes enregistrées pour les agents (4 000 foyers paysans et environ 1 000 autres agents).

11. Selon une approximation courante, on ne peut charger en mémoire des données d'une taille supérieure à la moitié de la mémoire vive. Approximation qui approche du tiers quand on prend en compte les autres processus en cours, et éventuellement des modifications à l'échelle de l'ensemble du jeu de données plutôt que sur des extraits. Pour pouvoir traiter ces 5 Go de données (tableau 5.1), l'ordinateur utilisé doit donc disposer d'au moins 16 gigaoctets de mémoire vive, et encore, au prix d'un traitement potentiellement lent et bloquant.

12. La reproductibilité sera abordée dans le chapitre 1 (positionnement).

13. Le « *High-Performance Computing* » (HPC) par exemple, mobilisé pour l'étude de données plus massives, c'est-à-dire trop importantes pour être analysées sur un unique ordinateur ou serveur. Voir REY-COYREHOURCQ (2015, p. 223–283) pour un historique détaillé des usages HPC en SHS, et particulièrement en géographie.

plus souvent pas de ces compétences – : ils seraient alors en difficulté pour en tirer les analyses nécessaire à leur interprétation.

Analyser une masse d'indicateurs. D'un point de vue thématique, et c'est là l'objectif, cette masse de données doit servir à la production d'indicateurs, nombreux et divers aussi bien dans leur forme que dans les caractéristiques des processus qu'ils décrivent (ref. chap. 3, indicateurs). Les mêmes raisonnements, « multiplicatifs », que pour les données s'appliquent ainsi aux indicateurs. On peut rendre compte de la variabilité des réplications directement dans les indicateurs produits (par exemple avec des représentations graphiques de type *box-plot*, utilisés ici pour une large partie des indicateurs).

La production de tels indicateurs au niveau de la variabilité inter-expérience est pourtant difficile, si tant est qu'elle soit souhaitable. De fait, chaque expérience doit pouvoir être comparée aux précédentes sur la base de leurs seules réplications respectives. Dès lors, la raison d'être des indicateurs de sortie est de rendre possible une comparaison, indicateur par indicateur, entre chacune des expériences. Il est donc indispensable de générer, pour chaque expérience, l'ensemble des indicateurs. En ne considérant ici encore que 25 expériences, cela fait donc déjà plusieurs centaines [14] d'indicateurs (tableau 5.1).

Le choix ayant été fait de mener une comparaison visuelle (ref. dans chapitre 3 : indicateurs uniques vs fonctions objectifs), on imagine dès lors que celle-ci va être difficile en présence de tant d'indicateurs.

En sus de la contrainte de l'enregistrement et de la production des indicateurs, le verrou majeur à l'exploration du comportement attendu de SimFeodal est donc la simple capacité à visualiser et à explorer l'ensemble des indicateurs de sortie. Ce qui doit de plus être rendu accessible y compris pour un auditoire non habitué à la manipulation de nombreuses données et sorties quantitatives.

14. En considérant ainsi une trentaine d'indicateurs, on obtient donc $30_{\text{[indicateurs]}} \times 25_{\text{[jeux de paramètres]}} \approx 750$ indicateurs uniques.

5.2 Comment explorer les sorties de SimFeodal ?

Pour évaluer, de manière approfondie, une expérience (voir tableau 5.1) d'un modèle tel que SimFeodal, il est nécessaire de passer en revue de nombreux indicateurs de sortie de simulation. Cette évaluation ne vise pas à produire une « note » unique et synthétique, mais plutôt à tester la capacité de l'expérience à reproduire les dynamiques que le modèle chercher à reproduire. Il ne s'agit pas, à proprement parler, d'une validation du modèle, au sens quantitatif où on pourrait l'entendre.

On vise plutôt à explorer le comportement du modèle en fonction des mécanismes et valeurs de paramètres choisis. Cela aboutit donc sur un jugement qualitatif sur la capacité du modèle à reproduire les dynamiques souhaitées. Pour mener cette exploration, il convient d'utiliser des outils adaptés, c'est-à-dire de disposer de solutions techniques permettant le calcul et l'affichage des indicateurs à partir des données produites par le modèle.

Dans le travail mené autour de SimFeodal, plusieurs solutions ont été utilisées au cours des différentes étapes de construction du modèle. La restitution purement chronologique de ces solutions ne revêt pas d'intérêt propre, mais les contraintes accumulées au cours de la construction du modèle ainsi que les choix devant permettre de les dépasser nous paraissent très largement génériques et généralisables.

La succession de choix d'outils d'explorations se justifie par les verrous dans l'exploration que chacun de ces outils a permis de débloquer. Cela dresse par là-même un portrait des solutions méthodologiques d'exploration de données de simulations dont on peut faire usage selon les contraintes générales des modèles.

5.2.1 Observer les simulations en direct ou *a posteriori*

Classiquement, le premier réflexe d'un modélisateur, du moins pour les modèles à base d'agents, est de définir des sorties graphiques pour accompagner son modèle. Les différentes plate-formes de modélisation agent mettent d'ailleurs régulièrement en avant les possibilités de représentations qu'offrent leurs environnements [15]. Visualiser le déroulement d'un modèle « en direct » (« *online* » dans GRIGNARD et DROGOUL 2017), c'est-à-dire au sein de la plateforme de simulation et au cours l'exécution du modèle, offre ainsi de nombreux avantages : évaluation visuelle du niveau de ségrégation (et de son évolution) dans une implémentation du modèle de Schelling ; visualisation de cohérence du déplacement des nuées d'oiseaux dans un modèle de type « Flocks » (REYNOLDS 1987) ; ou encore suivi d'un indicateur dans le temps – la quantité de ressources collectées – dans un modèle de type « Sugarscape » (EPSTEIN et AXTELL 1996).

15. Voir par exemple les collections de visualisations sur les pages d'accueil de Gama (https ://gama-platform.github.io/), de NetLogo (https ://ccl.northwestern.edu/netlogo/), de GeoMASON (https ://cs.gmu.edu/∼eclab/projects/mason/extensions/geomason/) ou encore de Repast (https ://repast.github.io/screenshots.html).

Visualisation des simulations en direct. Il est à noter que dans le cadre du développement d'un modèle de simulation, l'implémentation du modèle et de son interface graphique sont étroitement liées. D'une part, la plateforme de modélisation choisie contraint fortement le type et la diversité des représentations. Gama et GeoMASON, par exemple, proposent des modes de visualisation de données géographiques bien plus avancés que NetLogo ou Repast. L'interface graphique développée pour chaque modèle est donc largement influencée par la plateforme de simulation dans laquelle il est exprimé. D'autre part, dans la plupart de ces plateformes de simulation, l'interface graphique est implémentée au même niveau que le code-source du modèle en lui-même. Dans Netlogo et Gama par exemple, l'interface graphique est programmée directement dans le modèle en lui-même, en se basant sur les variables qui y sont déclarées. Il n'est donc pas possible de créer une interface graphique générique au sein des plateformes de simulation, laquelle pourrait s'appliquer à plusieurs modèles différents. Il est nécessaire, pour chaque modèle, de reconstruire l'interface depuis les briques de bases proposées par les plateformes, c'est-à-dire un ensemble de primitives graphiques permettant de composer une interface intégrée au modèle.

Dans l'exploration de SimFeodal, la création en direct de quelques graphiques correspondant à des indicateurs étudiés permet d'assurer un rôle de filtrage, grossier, à l'exécution d'une expérience, c'est-à-dire de l'ensemble des réplications nécessaires à son évaluation. Après une modification du code-source du modèle, et avant de lancer de nombreuses simulations, on exécute quelques simulations « manuellement ». On s'assure alors, en direct, que les indicateurs affichés ne présentent pas de caractères aberrants. Cela permet de vérifier, avant de lancer des calculs plus conséquents, que le déroulement de la simulation semble cohérent, c'est-à-dire, le plus souvent pour des modifications mineurs du modèle, qu'un *bug* n'a pas été introduit involontairement.

Le recours à ce type de visualisation en direct des simulations ne peut toutefois être généralisé, c'est-à-dire sorti de son rôle de pré-filtre, en raison de deux principales contraintes.

Visualisation en direct et réplication. La première contrainte, déjà évoquée plus haut, est que le modèle SimFeodal est fortement stochastique. Dès lors, la visualisation des indicateurs d'une simulation particulière n'est pas suffisante pour estimer le comportement du modèle. En conséquence, les indicateurs choisis pour l'évaluation de SimFeodal prennent presque tous en compte la variabilité des résultats induite par l'exécution de réplications. Certaines plateforme de simulation multi-agents [16] permettent toutefois de mener concomitamment plusieurs réplications d'un même modèle et de visualiser directement pendant l'exécution les résultats agrégés des réplications. Cette première contrainte, liée au besoin d'analyser des réplications plutôt que des exécutions isolées, peut donc être dépassée en adaptant l'implémentation du modèle pour faire usage de ces capacités de multi-simulation.

16. Gama dans sa dernière version (1.8) par exemple, voir https ://gama-platform.github.io/wiki/RunSeveralSimulations.html.

Visualisation multi-indicateurs. La seconde contrainte est cruciale dans le cas de SimFeodal et invalide l'usage des méthodes de visualisation en direct. On l'a vu, l'exploration des sorties de simulation du modèle repose sur la consultation systématique de plusieurs indicateurs, dont le nombre peut se révéler important pour une analyse approfondie.

Tout d'abord, il est concrètement difficile de représenter tous ces indicateurs au sein de l'interface graphique d'une plate-forme de simulation agent, comme on peut le remarquer, par exemple en figure 5.1 : l'espace occupé par les quelques indicateurs temporels et numériques affichés est déjà important et rend l'interface d'ensemble complexe. De plus, et c'est sans doute le verrou majeur, la temporalité de l'exécution d'une simulation – ou même des réplications nécessaires – est bien plus courte que celle requise pour la compréhension des résultats produits. Une simulation requiert ainsi au maximum quelques minutes. Pour pouvoir examiner tous les indicateurs pendant cette durée, il serait nécessaire de mettre la simulation en pause régulièrement, presque à chaque pas de temps. On disposerait alors d'un temps suffisant pour observer les indicateurs (organisés en onglets dans l'interface visible dans la figure 5.1). L'analyse des indicateurs de sortie de simulation demandent en effet un examen approfondi, et non simplement superficiel, pour pouvoir juger de l'adéquation de ce que ces indicateurs représentent vis-à-vis des attentes thématiques.

Visualisation différée. Cette contrainte est renforcée par la pratique d'évaluation que la plupart des chercheurs mobilise. L'évaluation n'est pas une étape unique et finie, il est utile de pouvoir revenir sur les résultats à différents moments. Cela est par exemple nécessaire quand il s'agit de comparer de nouveaux résultats produits à ceux générés par des expérimentations antérieures. On ne peut alors se contenter d'évaluations en direct, même en y consacrant un temps important, simplement parce que par nature, ces évaluations seront à reproduire en plusieurs occasions, et qu'il ne serait alors pas rationnel de relancer, à chaque fois, de nouvelles simulations correspondant à des configurations de paramètres et de mécanismes déjà éprouvées.

Visualisations multiples. Un dernier élément contribue à la difficulté de se baser sur une évaluation en direct : en plus du chercheur, amené à revenir de multiples fois sur les résultats d'une expérience, un modèle co-construit peut être évalué par plusieurs chercheurs différents. C'est d'autant plus fréquent en situation d'interdisciplinarité, où les points de vue de chacun des membres sont complémentaires et nécessaires. Sauf à faire preuve d'une discipline exacerbée, par exemple en réalisant l'ensemble du travail d'évaluation uniquement en séances de travail simultanées et collectives, l'évaluation par plusieurs personne demande que chacun puisse mener ces analyses selon ses propres temporalités. En choisissant de baser l'évaluation d'un modèle uniquement sur une analyse en direct, il faudrait alors que chaque chercheur, à chaque fois qu'il souhaite évaluer une même expérience, ré-exécute le modèle de nombreuses fois. Cela serait naturellement possible, mais constituerait clairement un gâchis certain de ressources.

L'évaluation d'un modèle interdisciplinaire et exploratoire ne peut donc que

difficilement être réalisé en direct, qui plus est quand elle demande de faire appel, dans un cadre de co-construction, à plusieurs points de vue hétérogènes. Les modalités mêmes de l'exploration des sorties d'un modèle à évaluation visuelle requièrent donc que les indicateurs soient accessibles et explorables à des temporalités différentes, par des chercheurs différents, depuis des lieux divers.

Il est donc indispensable que les indicateurs soient enregistrés et consultables simplement à tout moment, *a posteriori* des simulations, ce qui élimine de fait la visualisation des indicateurs en direct pendant l'exécution des simulations comme unique méthode d'exploration du comportement des modèles exploratoires et descriptifs.

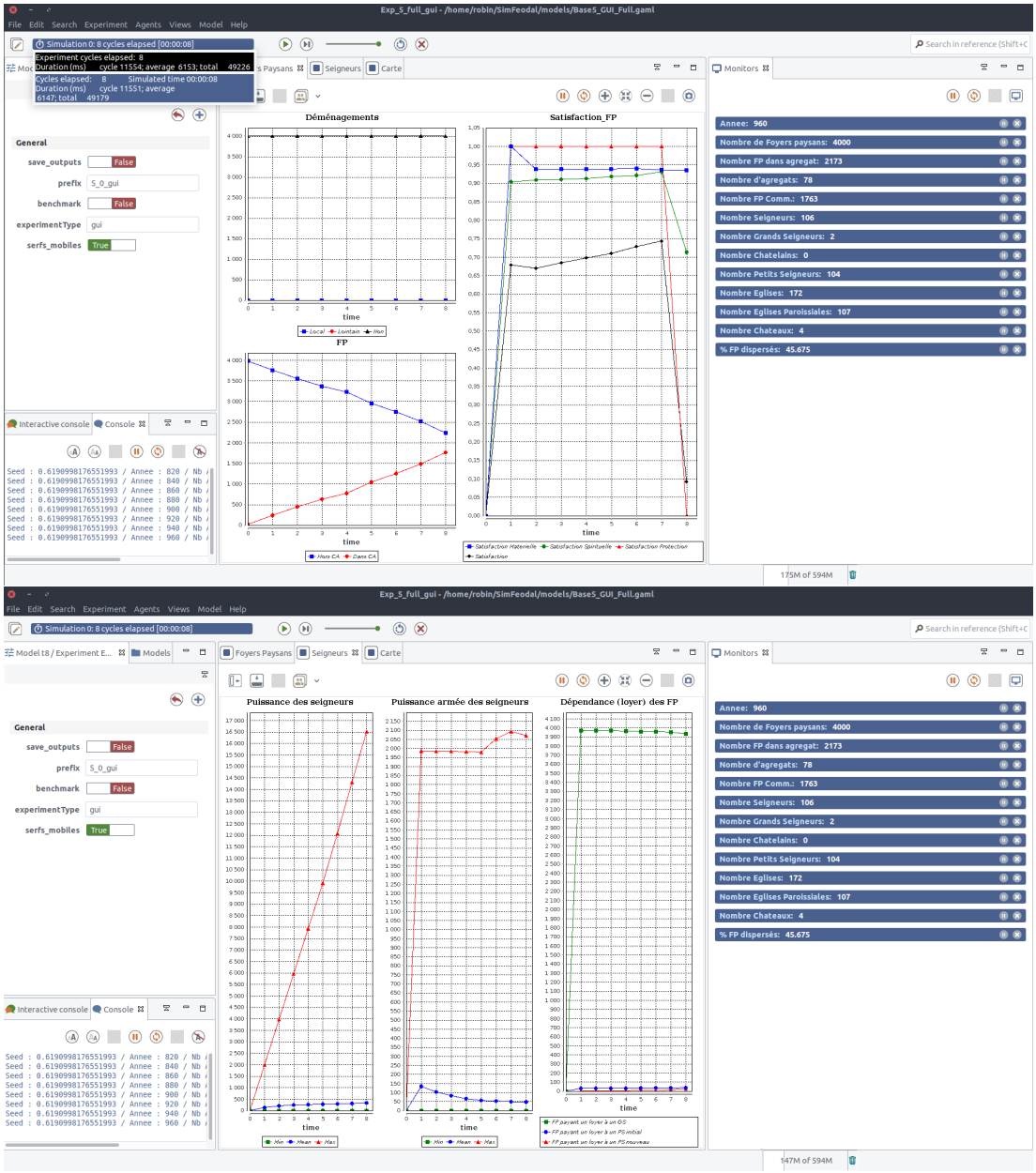


FIGURE 5.1 – Indicateurs intégrés à l'interface graphique interne de SimFeodal. Dans ces captures d'écrans, il s'agit d'indicateurs liés aux foyers paysans et aux seigneurs.

Visualiser en direct pour pré-filtrer. Dans le cadre de la construction et du paramétrage de SimFeodal, nous avons cependant mobilisé ce type de visualisation en direct, comme on peut le constater dans l'interface graphique

du modèle (figures 5.1 et 5.2). La visualisation en direct n'est donc pas mobilisable en tant que méthode d'évaluation principale, mais elle peut tout de même, comme dans un usage très classique, être utilisée comme un outil de validation interne pour tester chaque modification dans les valeurs de paramètres, remplissant alors le rôle de « préfiltre » décrit auparavant. Visualiser une seule simulation, avant d'en exécuter les réplications nécessaires, permet ainsi déjà de vérifier que les modifications apportées dans les valeurs de paramètre ou dans les mécanismes n'ont pas entraîné l'apparition de *bugs* ou d'incohérences immédiatement visibles.

Nous avons donc choisi de développer une interface graphique, très sommaire mais permettant des allers-retours rapides entre l'implémentation et l'exécution, au sein de l'implémentation de SimFeodal. Cette interface n'affiche qu'un nombre réduit d'indicateurs (figure 5.1), ainsi qu'une représentation cartographique (figure 5.2) utile à une analyse rapide du comportement d'ensemble du modèle.

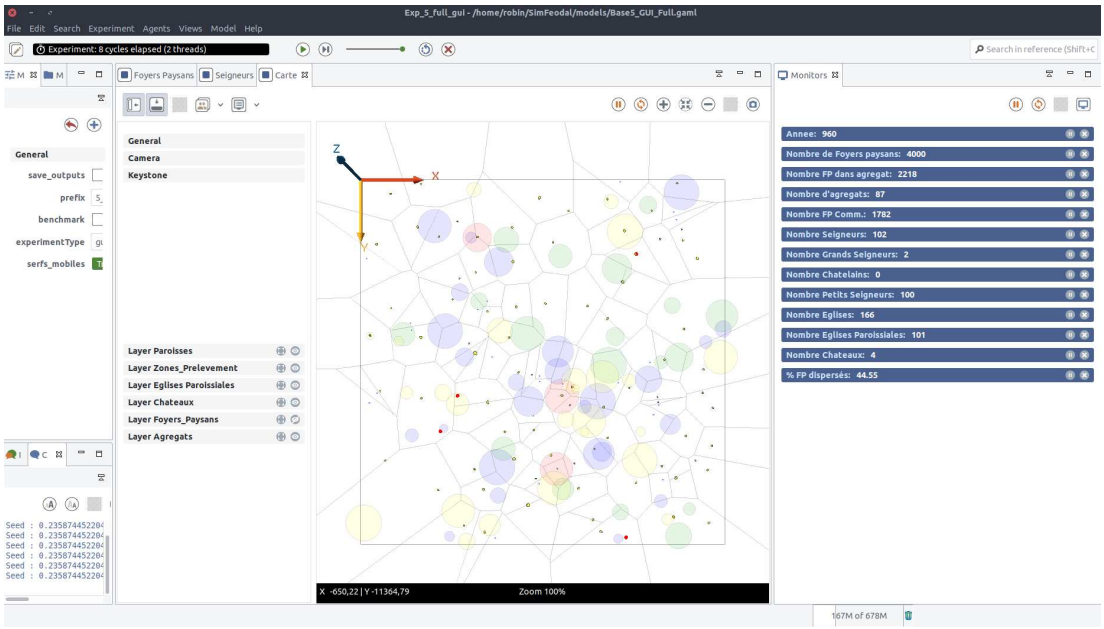


FIGURE 5.2 – Visualisation intégrée à l'interface graphique interne de SimFeodal : cartographie synthétique de l'espace modélisé.

Ces indicateurs intégrés à l'interface graphique interne de SimFeodal permettent de mener une étape préalable à l'évaluation du modèle. Cette étape vise à effectuer un premier filtrage des simulations avant d'exécuter les expériences en elles-mêmes. L'ajout de cette interface graphique vient donc renforcer l'outillage d'évaluation de SimFeodal. A cette étape, il manque encore un outil véritablement adapté à l'analyse conjointe des réplications du modèle, a posteriori de l'exécution des nombreuses réplications.

5.2.2 Générer les indicateurs

La production des indicateurs doit nécessairement être réalisée en aval de l'exécution des simulations – « *offline* » dans GRIGNARD et DROGOUL (2017). Il faut pour cela disposer d'outils adaptés au traitement des données produites, c'est-à-dire répondant aux contraintes identifiées auparavant (sous-section 5.1.4).

La contrainte principale est d'être en mesure de gérer la masse de données produites. On l'a vu, cela élimine d'office les outils de type tableurs, ou encore les outils de manipulation graphique de données les plus courants. Pour les raisons évoquées dans le chapitre 1 (Positionnement : pourquoi utiliser des outils libres ?), seules les solutions techniques libres étaient envisageables.

Interfaces graphiques (GUI) ou en lignes de commande (CLI) Certains outils graphiques (*GUI – Graphical User Interface*), basés sur des logiciels libres en arrière-plan (PSPP, R Commander, Orange), sont extrêmement aisés à prendre en main et auraient pu constituer un bon choix. Pourtant, avec une trentaine d'indicateurs à produire pour chaque expérience, donc de manière régulière, nous avons préféré nous tourner vers des outils plus orientés vers une interface en ligne de commande (*Command Line Interface*, abrégés *CLI*).

L'utilisation de CLI a plusieurs intérêts gravitant autour de la reproductibilité des traitements. En premier lieu, ils permettent une adaptation aisée et rapide aux différents jeux de données. Ainsi, partant du principe que les données générées par les réplications et expérimentations sont de même structures, il suffit généralement de modifier le chemin d'entrée des fichiers résultants pour reproduire à l'identique une analyse sur un nouveau jeu de données.

De manière plus technique, on peut remarquer que les différents indicateurs de sortie de simulation choisis présentent souvent des caractéristiques communes, aussi bien dans le traitement nécessaire que dans les formats (graphiques) produits.

Des outils CLI pour l'analyse de données. Par exemple, la grande majorité des indicateurs reposent sur une première agrégation des données par réplication et pas de temps simulé, puis par une seconde agrégation montrant la variabilité des situations générées, au niveau de l'expérimentation [17]. En terme de manipulation de données, seuls le calcul de la variable à mobiliser, et éventuellement l'agent caractérisé sont ainsi à adapter dans ces nombreux indicateurs de sortie. La variabilité du nombre de foyers paysans et la variabilité du nombre d'agrégats ne diffèrent que par le type d'agent sur lequel le calcul est effectué par exemple. Les variations, en terme de code-sources, sont donc le plus souvent des adaptations minimes (nom de l'agent, type d'agrégation…). Le recours à des traitements en *CLI* permet ainsi un simple copier/coller, voir la création de fonctions dédiées, pour effectuer ces traitements très récurrents.

Des outils CLI pour la visualisation. Au niveau des sorties graphiques, on peut aussi remarquer que la structure des graphiques, en elle-même, est assez largement identique : on représente les pas des temps (les années simulées) en abscisse, un indicateur statistique en ordonnée, et la variabilité sous la

17. On peut considérer ces agrégations comme une succession d'opération imbriquées : pour montrer l'évolution d'un indicateur tel que le taux de foyers paysans dispersés au cours du temps simulé, il faut (1) calculer le ratio entre nombre de foyers paysans dispersés et nombre total de foyers paysans, (2) pour chacun des pas de temps simulé, (3) pour chaque simulation, (4) pour l'ensemble des réplications d'une expérience, (5) éventuellement pour chacune des expériences d'une phase plus large d'expérimentation qui ferait varier des valeurs de paramètres.

forme de *box-plot* minimalistes (« *minimal boxplot* », promus par Edward Tufte pour minimiser le ratio données-encre (TUFTE 2001, p. 123-125)). En disposant d'un environnement de type *CLI*, et qui plus est en faisant usage de solutions graphiques construites sur une syntaxe régulière et générique (voir Fluidifier les étapes de rendu : le « pipeline de visualisation », section 5.4.2.1), il devient très confortable de générer les différents indicateurs de sortie souhaités, puisqu'il suffit d'adapter les graphiques déjà conçus.

Avec ces solutions logicielles d'analyse de données et de visualisation, il est facile de concevoir et d'implémenter les codes informatiques nécessaires à la génération des indicateurs de sortie de simulation. De plus, l'exécution de ces programmes est extrêmement rapide : les différents fichiers de sortie de simulation sont lus et parcourus une unique fois pour en tirer toutes les variables nécessaires à l'établissement des indicateurs.

En ayant choisi de mener une évaluation *a posteriori* – plutôt qu'en direct – basée sur l'observation d'indicateurs générés – par des outils adaptés au traitement de données massives – de manière automatisée, on dispose donc, pour chaque expérience, d'un ensemble de fichiers numériques : chacun des indicateurs de sortie est contenu dans un fichier unique, dans un format facilement exploitable et ré-utilisable.

5.2.3 Organiser les indicateurs en rapports paramétrables

Du point de vue de la manipulation, la création de fichiers informatiques indépendants correspondant aux différents indicateurs de sortie de simulation est extrêmement pratique : on peut facilement les identifier, les transférer et les adapter, par exemple pour en rendre le contenu plus compréhensible par un public différent.

En revanche, du point de vue de la comparaison des résultats, cette forme n'est pas la plus adaptée. Si l'on peut facilement comparer un même indicateur portant sur deux expériences différentes, la tâche se complique quand il s'agit d'avoir une vision globale des différences dans les indicateurs entre deux expériences. Pour cela, la démultiplication des fichiers correspondant aux indicateurs se révèle rapidement être un obstacle : l'utilisateur est en effet amené à jongler entre de très nombreux fichiers.

Les rapports comme instruments de comparaison. Pour faciliter la comparaison d'indicateurs multiples – et d'une forte diversité –, il est nécessaire de les organiser au sein d'une structure englobante. Nous entendons ici par organisation, une présentation structurée, suivant un certain ordre, identique selon les expériences, adaptée à une évaluation des résultats. Pour cela, nous avons choisi d'organiser les indicateurs de sortie de simulation au sein de « rapports ». Cela permet, même en présence de nombreuses expériences, de rassembler l'ensemble des indicateurs de sortie propres à chacune dans un unique fichier, à la structure toujours similaire. Un premier apport, majeur, concerne l'archivage des sorties de simulation. Avec des rapports comprenant l'ensemble des indicateurs de sortie de chaque expérience d'un modèle, il est

ainsi simple de conserver des traces de l'ensemble des versions et sous-version d'un modèle. Cela permet de garantir une certaine pérennité à ce modèle, à sa documentation, et ainsi simplifie le travail rétrospectif de caractérisation de l'évolution d'un modèle.

L'intérêt majeur de la structuration en rapports est surtout de faciliter la comparaison des expériences, c'est-à-dire des indicateurs de sortie des différentes expériences. On peut ainsi, par exemple, placer côte à côte, visuellement, deux rapports rendant compte de deux expériences différentes, et, en les faisant défiler simultanément, comparer point par point, c'est-à-dire indicateur par indicateur, leurs résultats respectifs, de manière visuelle et intuitive.

Les formes que peuvent prendre des rapports, tout autant que les modalités de leur production, sont multiples et extrêmement diverses. La forme la plus simple et courante consiste à produire manuellement le rapport en insérant les indicateurs adaptés au fur et à mesure, par exemple dans un traitement de texte. A l'opposé, on peut noter les possibilités de créations entièrement automatisées de rapports complets, comprenant par exemple des descriptions et commentaires textuels générés à la volée, en fonction d'expressions conditionnelles [18].

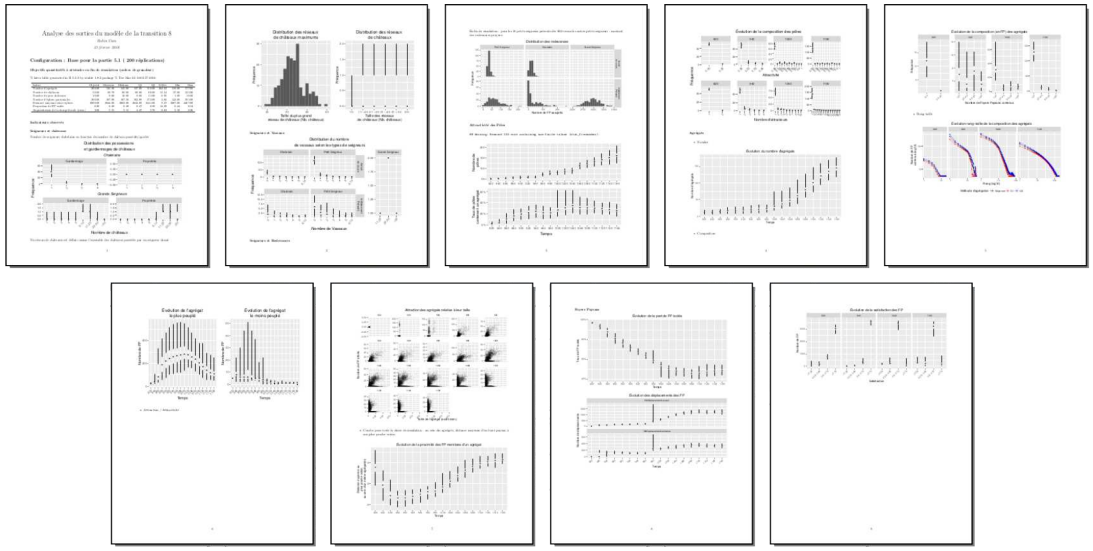


FIGURE 5.3 – Un exemple, miniaturisé, de rapport automatique généré pour une expérimentation (étape 0) de SimFeodal.

Pour SimFeodal, nous avons choisi de restreindre au maximum la manipulation manuelle. On souhaitait générer un rapport entièrement automatique, ne requérant pas d'action spécifique en dehors du choix des données depuis lesquelles créer les indicateurs. Nous n'avons toutefois pas voulu pousser l'automatisation jusqu'à l'ajout de commentaires automatiques des indicateurs de sortie : la richesse – et la difficulté– d'une approche interdisciplinaire telle que la notre est constituée par la multiplication des analyses et points de vue. Il n'y

18. Voir par exemple l'application « SOFIE » du Commissariat général à l'égalité des territoires (CGET), qui génère automatiquement des commentaires relatifs aux inégalités femmes/hommes dans l'accès à l'emploi. Les commentaires propres aux types d'inégalités majeures de chaque EPCI sont produits de manière automatique depuis les données. http ://outils.observatoire-des-territoires.gouv.fr/sofie/

avait donc aucun besoin de générer des annotations standardisés et automatiques, forcément moins aboutis que les analyses de chacun des co-concepteurs du modèle. Le rapport produit (figure 5.3) n'intègre donc que les indicateurs, sous forme de tableaux et de graphiques. Ces indicateurs sont organisés par partie, en l'occurrence en fonction du type d'entités et de comportement qu'ils décrivent.

Structurer des rapports pour aller vers la reproductibilité des analyses. On a donc fait le choix de se baser sur des rapports automatisés et minimalistes, ne contenant que les indicateurs dans une forme structurée. Ce choix s'appuie sur des raisons multiples, qui ont toutes en commun une recherche de reproductibilité des résultats et des analyses menées. Une reproductibilité théorique (encore une ref au positionnement), puisque les résultats de simulation doivent pouvoir être analysés et reproduits par des chercheurs potentiellement intéressés. Mais cette recherche de reproductibilité est aussi pratique, rendue aussi nécessaire par les méthodes de modélisation suivies. Tel qu'explicité auparavant (sections 5.1.2 et 5.1.3), celles-ci s'appuient sur de nombreux allers-retours, ce qui requiert une capacité constante à reproduire et à affiner des résultats déjà observés.

La quantité d'expériences requises pour arriver à un état satisfaisant du modèle est tributaire de ces allers-retours, et le nombre de rapports qu'il faut pouvoir produire est important. La fréquence de production de ces rapports est forte, et le modélisateur a alors tout intérêt à en fluidifier et accélérer le processus de création.

Dans une telle situation, la création d'un rapport automatisé garantit une calcul et une production simplifié et rapide des indicateurs sur les nouvelles données. Cela permet un examen des sorties de simulation presque immédiatement après leur exécution. Cette automatisation permet aussi de mener une seconde évaluation – après le pré-filtrage constitué par l'observation d'indicateurs en direct – du bon déroulement « interne »[19]. Le caractère fixe d'un rapport automatisé se base ainsi sur une structure de données contraignante, par exemple constitués en n fichiers dotés de plusieurs colonnes spécifiquement attendues. Les caractéristiques de ces données sont elles-mêmes contraignantes. Un rapport automatisé ne fonctionne, par exemple, qu'en présence d'un nombre pré-défini de réplications complètes. En l'absence d'un de ces critères dans des données en sortie de simulation, le rapport ne peut être généré et émet une erreur. Par exemple, si le nombre de réplications est plus faible qu'attendu, ou encore si tel attribut d'un agent a changé de type informatique, la création du rapport échoue. La présence ou non de cette erreur constitue donc un nouveau filtre de vérification de la validité du modèle. Cela permet, là encore, de détecter des simulations qui présenteraient des comportements incomplets ou aberrants en terme de production de données.

Un autre intérêt majeur des rapports, déjà pointé en avantage des outils de type *CLI* est leur adaptabilité. On a vu (chap. 3) que les indicateurs à examiner sont

19. Au sens de l'évaluation interne, c'est-à-dire du bon fonctionnement, exempt de *bugs*, du modèle de simulation implémenté. Voir ref chap 3.

nombreux et surtout, évolutifs, dans le sens où ces indicateurs ont fortement été modifiés, remplacés, affinés, au cours des étapes de paramétrage de SimFeodal. L'utilisation de rapports automatiques permet de minimiser le nombre de modifications à effectuer en cas de changements d'indicateurs. Le programme informatique qui génère les rapports s'appuie ainsi sur un code-source unique, générique aux simulations de chaque modèle. Lors d'un changement d'indicateurs, il suffit alors de modifier ce code-source en une seule place, et tous les appels à ce programme seront alors modifiés en conséquence. À partir de là, pour mettre à jour l'ensemble des rapports déjà produits, c'est-à-dire regroupant les indicateurs de chacune des expériences passées, il suffit de ré-exécuter la routine de production des rapports.

Dans le cas de SimFeodal, caractérisé par de fréquents changements dans la forme et le calcul des indicateurs, cela a représenté un gain de temps et d'efficacité très conséquent. Par exemple, lors de certaines phases de paramétrage, on pouvait être amenés à faire évoluer le modèle quotidiennement et à tester, à cette même fréquence, plusieurs jeux de paramètres. Il fallait donc analyser les résultats de plusieurs expériences chaque jour, et régulièrement ajouter des indicateurs graphiques afin d'affiner l'évaluation. Ces indicateurs devaient aussi être ajoutés aux expériences des jours précédents, et au final, on re-générait parfois jusqu'à une plus d'une dizaine de rapports sur des cycles temporels courts de quelques jours.

Par extension, cette même démarche d'automatisation, basée sur l'utilisation d'outils de type *CLI*, devrait pouvoir s'appliquer à l'identique, avec les mêmes avantages, dans le cadre plus large de l'évaluation visuelle de modèles (ref chap 3, section 3.1.4).

Dépasser les limites de la compatibilité d'ensemble. La reproductibilité, de manière générale, est plus une démarche qu'un état final : on peut toujours trouver un nouvel élément à « rendre reproductible » dans un travail (cf. chapitre 1, partie sur reproductibilité recherchée). Dans le cas des rapports automatiques, la reproductibilité recherchée doit permettre de reproduire les analyses menées – le calcul et la représentation des indicateurs de sortie – sur les jeux de données issus des différentes versions du modèle.

Comme explicité dans l'encadré 3.2, chap 3 sur l'incrémentalité des indicateurs, une limite forte empêche d'atteindre une reproductibilité absolue des analyses du comportement des différentes versions de SimFeodal. Les données générées par les différentes versions du modèle ne sont en effet pas systématiquement « compatibles ». On entend par là qu'elles ne présentent pas toute exactement la même structure, à commencer par les variables enregistrées. Quand bien même il aurait été choisi dès le départ d'enregistrer le plus de sorties possibles, la reproductibilité de l'analyse échoue sur les données produites par le modèle et ses nombreuses versions : le modèle évolue, et avec lui, certaines variables apparaissent et d'autres deviennent caduques. La structure contraignante et précise des données nécessaires à la génération des rapports ne peut être entièrement satisfaite. La prise en compte de l'évolution du modèle demande une adaptation régulière – mais aussi rare que possible – des

programmes qui génèrent ces rapports.

On ne peut donc satisfaire globalement à un objectif de reproductibilité, mais il est toutefois possible de limiter la déviance à cette ambition. Pour cela, on peut agencer les différentes versions du modèles au sein de « générations » de modèle, c'est-à-dire d'ensembles de versions présentant des attributs comparables et générant des données de même structure. Plutôt que d'adapter le code-source des rapports à chaque nouvelle version du modèle, ou encore de ne jamais l'adapter et donc d'être tributaire de la structure des toutes premières versions du modèle, cela constitue un choix intermédiaire qui permet de limiter le nombre de variantes de rapports. Cette approche suit les grandes lignes du développement logiciel général. Les itérations successives d'un logiciel sont constituées de versions « majeures » – les générations de modèles dans notre cas –, qui n'assurent pas nécessairement de compatibilité avec les versions majeures précédentes, et de versions « mineures », dans lesquelles la compatibilité est assurée [20].

Pour revenir aux rapports voués à l'évaluation d'un modèle, en inscrivant les différentes versions du modèle – et des programmes générant les rapport correspondant – dans des sous-ensemble de versions, les « générations », les différents rapports peuvent être considérés comme reproductibles et automatiques au sein de ces générations. Pour SimFeodal, cela implique d'organiser les différentes versions du modèle – résultant des étapes de paramétrage, cf. chap4 – au sein de grandes générations, à chaque changements structurels des mécanismes ou données produites par le modèle (voir chapitre 4, tableau des versions et sous-versions).

Les rapports, des instruments suffisants ? A l'issue de la conception et de l'implémentation de ces rapports automatiques, on dispose donc, pour chaque expérience, d'un document aisément partageable et lisible. Ces documents s'enrichissent, au fur et à mesure des générations de modèles, de nouveaux indicateurs, et sont comparables au sein de ces générations. Cela pourrait constituer la dernière étape de la création d'outils d'évaluation d'un modèle, dans la limite d'un nombre de versions ou de génération de modèles assez restreint.

SimFeodal, comme c'est souvent le cas dans les modèles à base d'agent, a toutefois été caractérisé par une forte quantité d'allers-retours entre le modèle et ses résultats, entraînant à chaque fois de nouvelles expérimentations (cf. section 5.1).

On a vu que la manipulation d'un grand nombre d'indicateurs, même pour une quantité restreinte d'expériences, disqualifiait l'usage de fichiers individuels et poussait à l'usage de rapports structurés. Avec un grand nombre d'expériences, les mêmes limites apparaissent pour les rapports : la masse d'expériences rend partiellement caduque l'utilisation unique des rapports automatiques. Il est en

20. Par exemple, un fichier de dessin vectoriel créé avec le logiciel Adobe Illustrator 15.0 ne sera pas lu correctement avec une version 14.0. Ce fichier présentera toutefois un compatibilité parfaite avec les versions 15.1 à 15.*n* du logiciel.

effet aisé de comparer, sur un même écran d'ordinateur, deux ou trois rapports, mais dès lors qu'il faut en comparer un plus grand nombre, la manipulation conjointe des rapports devient complexe, tout autant que d'avoir une vision globale des résultats principaux de chaque expérience.

5.2.4 Organiser les rapports : les *dashboards*

Pour être en mesure de comparer de nombreux éléments, il est nécessaire de passer d'une exploration linéaire, fondée sur le visualisation successive de chacun des indicateurs, à une exploration globale et interactive. En pratique, plutôt que de faire défiler visuellement les nombreuses pages d'indicateurs, mieux vaut une interface présentant les points clefs de l'évaluation et qui permette d'entrer dans le détail de chacun des indicateurs dans un second temps, sur demande. Comme le résume le « mantra » de l'analyse visuelle (« *Visual information-seeking mantra* ») de Ben SHNEIDERMAN :

> « Overview first, zoom and filter, then details on demand »
>
> SHNEIDERMAN 1996, p. 2

5.2.4.1 Les *dashboards*

Cette logique, assez universelle désormais, est celle qui préside à la création des nombreux « tableaux de bord », ou « *dashboards* » que l'ont voit émerger depuis la fin des années 1990. Rob Kitchin et ses co-auteurs définissent ainsi les *dashboards*, en s'appuyant sur les travaux de FEW notamment :

> « For Few [(FEW 2006a, p.34)] a 'dashboard is a visual display of the most important information needed to achieve one or more objectives; consolidated and arranged on a single screen so the information can be monitored at a glance'. Just as a car dashboard provides critical information needed to operate the vehicle at a glance, indicator dashboards provide key information for running companies or cities (~~Dubriwny & Rivard, 2004~~ [RIVARD et COGSWELL 2004]). »
>
> KITCHIN, LAURIAULT et MCARDLE 2015, p. 11

Très répandus dans le monde de l'informatique décisionnelle (*Business Intelligence, BI*), ces outils permettent d'explorer des données d'entreprises, par exemple des résultats financiers. Pour ce faire, ils mettent en avant, dans une interface unique, des indicateurs clés (*Key Performance Indicators, KPI*), qu'il est ensuite possible de filtrer et d'affiner, par exemple par sélection de différents intervalles temporels.

Les *KPI* jouent le rôle d'indicateurs synthétiques, c'est-à-dire qu'ils s'adressent à des gestionnaires, par exemple des *managers*, qui ont une expertise importante sur les résultats produits. Les utilisateurs des *dashboards* ne sont donc pas des analystes, à même d'explorer eux-mêmes les données mobilisées, mais plutôt des thématiciens qui se fondent sur les indicateurs présentés pour prendre des décisions.

La dichotomie « analyste/décisionnaire » s'exprime aussi dans le domaine de la recherche, et notamment dans la recherche en géographie urbaine à visée applicative. Avec l'avènement des données massives et de leur prise en compte pour la gestion des villes (*smart cities*), les géographes se sont aussi penchés sur des outils de ce type (LAURINI 2018). En résulte une utilisation de plus en plus fréquente de *dashboards* en géographie urbaine (« *city-dashboards* », ROUMPANI, O'BRIEN et HUDSON-SMITH 2013 ; KITCHIN, LAURIAULT et MCARDLE 2015 ; BATTY 2015).

Le parallèle avec le monde de l'informatique décisionnelle est en effet présent dans les types d'utilisateurs et de producteurs de ces outils. Il s'agit de mettre à disposition d'experts thématiques (les décideurs publiques) des indicateurs clés, issus de calculs parfois complexes. Cela afin de leur permettre d'évaluer une situation donnée et de prendre les décisions politiques adéquates.

Le constat ayant mené à l'apparition des premiers *dashboards*, tant en informatique décisionnelle qu'en géographie urbaine, est identique. Les informations nécessaires à l'évaluation d'une situation (financière, relative aux politiques publiques…) sont de plus en plus nombreuses et hétérogènes. Les indicateurs permettant de mener ces évaluations, pensés pour les décideurs qui en feront usage (*managers*, acteurs publiques…), se démultiplient et se diversifient aussi par conséquence.

Inspiré autant par l'usage *BI* que par l'usage géographique, nous considérons que ces outils peuvent se révéler utiles dans l'évaluation de modèles de simulations complexes. Les enjeux sont en effet les mêmes : permettre à des thématiciens de comprendre et d'évaluer les sorties d'un modèle, à l'aide d'indicateurs nombreux et complexes présentés de manière transparente relativement à cette complexité.

Ce positionnement méthodologique s'inscrit pleinement dans la démarche de co-construction interdisciplinaire de SimFeodal. On y retrouve ainsi la même logique qui anime les dashboards : une évaluation menée par des thématiciens qui s'appuient pour cela sur des indicateurs clefs (ref. aux indicateurs les plus importants dans le chap 3) et précisent leur analyse à l'aide d'indicateurs secondaires présentés sous forme d'un panel varié de visualisations. Nous avons donc choisi de ré-organiser les rapports initialement produits pour leur donner une forme plus adaptée à ces enjeux, sous forme de *dashboards*.

5.2.4.2 SimVADB

Les *dashboards* font souvent usages de représentations graphiques très métaphoriques des tableaux de bords automobiles. On y retrouve fréquemment une forte mise en valeur d'indicateurs numériques simples au travers de représentations « skeuomorphes », c'est-à-dire qui reprennent l'apparence physique des objets symbolisés [21]. On retrouve communément, par exemple, des indicateurs représentés sous forme de jauges (*gauge charts*), de thermomètres (*thermometer charts*), ou encore de voyants d'alerte et autres témoins lumineux (figure 5.4)).

21. Voir par exemple la page Wikipédia consacrée : https ://fr.wikipedia.org/wiki/Skeuomorphisme

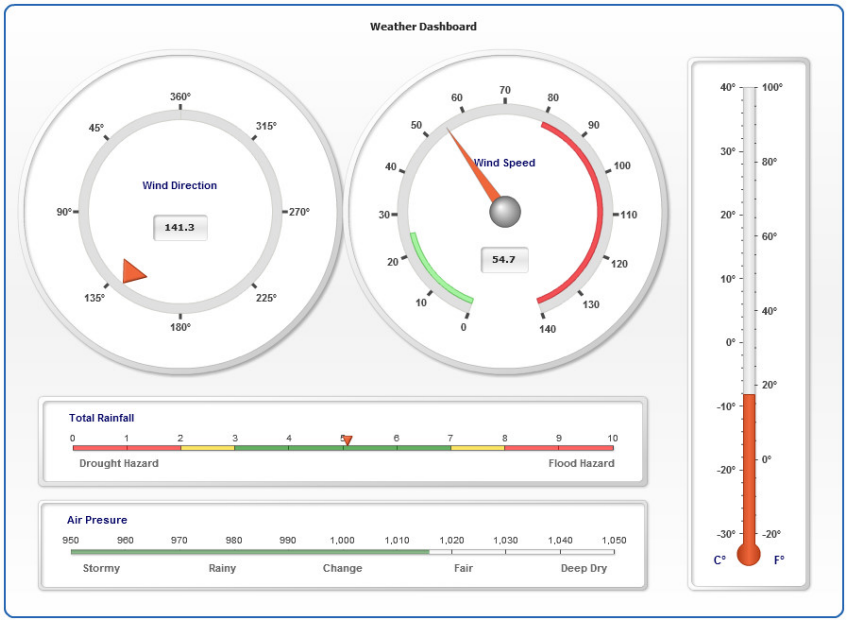


FIGURE 5.4 – Un exemple de représentations visuelles courantes dans les *dashboards*. Tiré de PANDRE (2011)

Pour SimFeodal, les indicateurs étant assez fortement conçus et structurés (chap. 3), nous n'avons pas ressenti le besoin de faire appel à ce type de représentation. Nous avons donc emprunté aux *dashboards* la logique d'organisation visuelle des indicateurs plutôt que les modes de visualisation en eux-mêmes. Pour faciliter la transition pour l'utilisateur, on cherchait ainsi à produire un *dashboard* au plus proche, visuellement, des rapports automatiques qui les précédaient.

On a pour cela développé un *dashboard* adapté à SimFeodal, nommé SimVADB [22]. Dans un premier temps, on souhaitait simplement ré-organiser le code-source produisant les rapports automatiques, afin de convertir ces rapports en *dashboards*. Cela a été effectué au moyen d'outils permettant de générer des applications en ligne, sans changer de langage de programmation. Dans ce cas, on s'est appuyé sur la librairie logicielle Flexdashboard (IANNONE, ALLAIRE et BORGES 2018).

Le passage du rapport automatique au *dashboard* illustre l'un des grands intérêts des outils de type *CLI* : dans le cas de SimVADB, il a suffit de ré-organiser le code, sans modifier à aucun moment les fonctions de calcul et de création des indicateurs de sortie de simulation. Les codes 5.1 et 5.2 illustrent la facilité de cette modification. Il s'agit véritablement de placer les différentes fonctions dans des blocs graphiques. Ces modifications minimes augmentent toutefois considérablement la convivialité et la facilité de l'analyse de résultats de sortie d'un modèle.

22. **S**imulation **V**isual **A**nalysis **D**ash**B**oard.
Cette application a rapidement été remplacée par l'itération suivante (SimEDB, voir section 5.2.5), et n'a donc dans les faits jamais été complètement finalisée. On en trouve une trace, fonctionnelle mais incomplète (les versions ultérieures n'ont pas été enregistrées dans l'outil de versionnement), dans ce dépôt logiciel : https ://github.com/RCura/SimEDB/tree/2cd22c7c

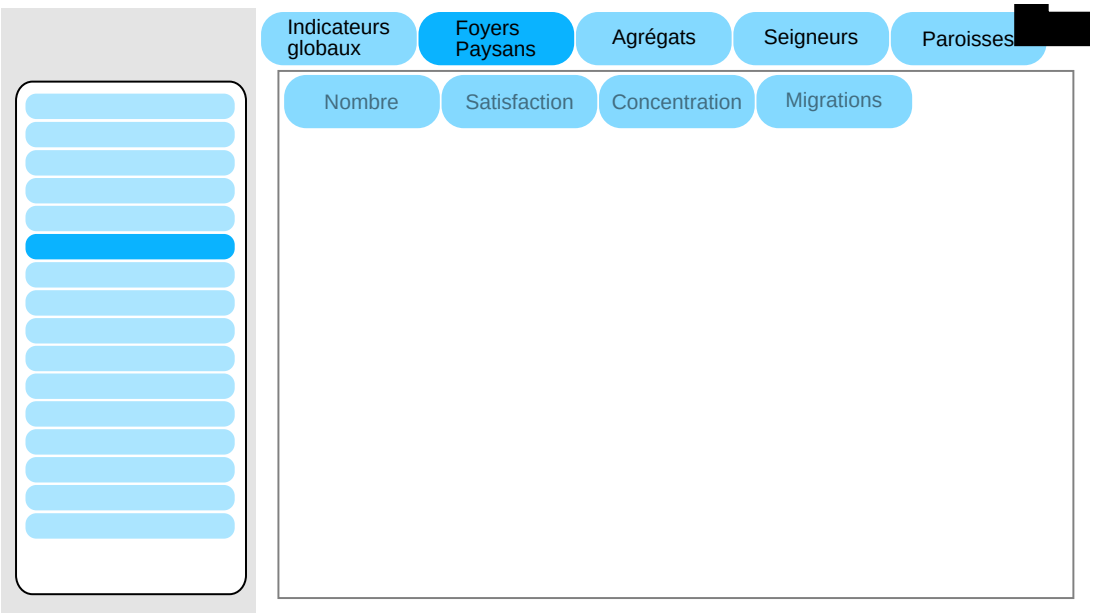
```
# Agent de type A
  afficher('Agent de type A')

  print('Indicateur 1')
  calcul_indicateur_1 {...}
  affichage_indicateur_1 {...}

  afficher('Indicateur 2')
  calcul_indicateur_2 {...}
  affichage_indicateur_2 {...}
# Agent de type B
  afficher('Agent de type B')
  [...]
```

Code 5.1 – Pseudo-code du rapport automatique.

```
onglet{titre = 'Agent de type A',
  sous_onglet{titre = 'Indicateur 1',
    calcul_indicateur_1 {...}
    affichage_indicateur_1 {...}
  },
  sous_onglet{titre = 'Indicateur 2',
    calcul_indicateur_2 {...}
    affichage_indicateur_2 {...}
  }
},
onglet{titre = 'Agent de type B',
  [...]
}
```

Code 5.2 – Pseudo-code du dashboard.

Comme dans l'exemple de code, on a préféré organiser les indicateurs au seins d'onglets plutôt que de les présenter dans des pages successives. Les onglets de premier niveau représentaient les types d'agent, et des onglets de second niveau permettaient de visualiser l'ensemble des indicateurs associés à ces agents (figure 5.5).



donc un ensemble d'expériences qui partagent des mécanismes et un jeu de paramètre par défaut communs.

Une interaction trop simple. Avec le mode de sélection choisi dans SimVADB, basé sur le nom des expériences, il devenait plus difficile de sélectionner rapidement des ensembles d'expériences membres d'une même phase d'expérimentation (cf. tableau versions et sous-version, chap 4). En effet, et comme illustré dans la figure 5.5, les noms d'expériences tendent à s'allonger, et avec leur masse augmentant, il est peu commode d'avoir à parcourir tout un long menu de sélection pour trouver les expériences souhaitées. De plus, malgré des tentatives de nommage régulières et explicites, la multiplication des expériences et expérimentations implique aussi une certaine confusion dans les types de mécanismes et valeurs de paramètres associés. Sans table de correspondance complète entre les noms des expériences et leurs valeurs de paramètres, il devenait impraticable de retrouver les différentes expériences mettant en avant, par exemple, des attractivités fortes par les pôles, une plus forte hiérarchisation des attracteurs ou encore des migrations lointaines facilitées. Le choix méthodologique d'interaction avec la plate-forme d'affichage des indicateurs, basé sur une sélection des expériences depuis leur nom, s'est donc révélé inadapté à la sélection et à l'exploration des sorties de SimFeodal.

Dépasser le « présentoir » de données. Au delà du souci du mode d'interaction, qui aurait pu être amélioré, un autre problème apparaissait. Pour évaluer visuellement différentes configurations du modèle, on ne pouvait se contenter d'un simple affichage des données, au sein d'un outil de type présentoir interactif tel que SimVADB. Comme dit dans le chapitre 4, le paramétrage de SimFeodal a ainsi reposé sur de nombreuses étapes d'évaluation des différentes version du modèle. L'approche d'analyse principale était donc la comparaison, point par point, entre les résultats des indicateurs de sortie de simulation des versions successives de SimFeodal. Un outil de présentation dynamique de résultats de sorties de simulations est certes plus adapté qu'un rapport statique, mais il ne constitue pas pour autant un outil adapté à la comparaison. S'il suffit pour de la restitution, par exemple dans le cadre du rapport systématique des résultats de SimFeodal, on ne peut s'appuyer uniquement sur une succession d'évaluations visuelles pour appréhender l'étendue des changements apportées par une modification des valeurs de paramètres.

5.2.5 Interagir avec les rapports : exploration interactive

Face à la démultiplication des expérimentations, consécutive aux nombreuses étapes de paramétrage de SimFeodal (cf. chap 4), il a fallu repenser la plate-forme d'évaluation des résultats. Pour cela, considérant que les simulations ne pouvaient être aisément appréhendées et sélectionnées par leur nom, numéro d'étape ou de version, il a été décidé d'adopter une posture plus proche de l'exploration du modèle en elle-même. C'est-à-dire de ne pas caractériser les simulations par un identifiant quelconque, mais plutôt par leur spécificité in-

trinsèque, c'est-à-dire la combinaison de valeurs de paramètres qui les rendent uniques. Ce faisant, au sein de la plate-forme d'exploration SimVADB, l'enjeu devenait plutôt la compréhension des effets des valeurs de paramètres sur les indicateurs que l'évaluation d'une simulation en particulier. Il fallait passer du descriptif, quelle qu'en soit la méthode, à du comparatif.

Du point de vue de l'interface utilisateur, cela implique que la sélection ne se fasse plus par un unique critère (le nom de la simulation), mais au contraire sur du multi-critère. Par une succession de sélections, chaque paramètre pouvait constituer un nouveau filtre dans lequel on avait à choisir les valeurs à interroger (voir la figure 5.6 **(E)** de l'encadré 5.1).

La quantité de paramètres en entrée était importante et pouvait dès lors donner lieu à un mode de sélection complexe et fastidieux – définir une par une les valeurs voulues pour chacun des 45 paramètres –. Nous avons choisi encore une fois de nous appuyer sur l'aspect visuel afin de permettre aux utilisateurs de SimVADB de choisir la ou les expérimentations à analyser.

Visualiser avec des coordonnées parallèles. Pour cela, on a choisi de représenter les combinaisons de paramètres dans un graphique en « coordonnées parallèles » (*parallel coordinates*, d'après INSELBERG et DIMSDALE 1987, voir FEW 2006b par exemple pour une description plus succincte, illustrée et pratique). Ce type de graphique est en effet extrêmement pertinent pour représenter une information multi-dimensionnelle en ce qu'il permet de détecter graphiquement des *clusters* d'individus statistiques[24] (HEINRICH et WEISKOPF 2013, p. 2), c'est-à-dire de faire ressortir visuellement les expériences dont les valeurs de paramètre sont proches. Notons bien que l'on parle ici des valeurs de paramètres, c'est-à-dire des conditions des expériences, et non des indicateurs de sortie. L'approche va ainsi des paramètres aux résultats : depuis des valeurs de paramètres choisies, on analyse la diversité des résultats.

Interagir avec des coordonnées parallèles. De plus, en matière d'interaction, on utilise fréquemment les graphiques en coordonnées parallèles en vue de filtrage. Cette opération est le plus souvent menée par des actions de *brushing* (« brossage »), c'est-à-dire de sélection graphique d'une zone en dessinant son étendue à la souris (voir encadré 5.1). Ce type de sélection se révèle en effet souvent plus efficace et intuitive qu'une sélection textuelle plus systématique :

> « Filtering is an operation that removes signals from its input. A filter reduces the number of lines to be rendered. In this sense, dynamic querying [...] is a filter, if implemented with brushing [...], which reduces clutter by putting the filtered lines in focus using

24. Ici, chaque expérience est un individu statistique. Il est caractérisé par un ensemble de variables, les différents paramètres, et les modalités de ces variables que cet individu emprunte (les valeurs de paramètres). Quand le nombre d'individus est important, les différentes « courbes », qui correspondent au profil des individus sur le graphique en coordonnées parallèles, peuvent se superposer et montrer des tendances similaires. Avec ce type de représentation, il est facile de visualiser les grandes classes d'individus constituées par ces « tendances », et donc de constater des distinctions entre les individus (les expériences) de manière visuelle.

> some highlighting mechanism. Combining simple brushes using logical operators [...] further allows the user to formulate rather complex queries that might even achieve faster and more accurate results using parallel coordinates than using a Structured Query Language (SQL) [...]. »
>
> HEINRICH et WEISKOPF 2013, p. 13

Cette utilisation est courante dans les champs de la visualisation d'information (*InfoVis* et *Visual Analytics*), et en a été reprise en géographie quantitative. On la retrouve par exemple chez l'un des représentants de l'analyse spatiale des années 1990, Stewart Fotheringham. Cet auteur indique même l'usage du graphique en coordonnées parallèles en tant que filtre pour identifier des informations dans une autre dimension, spatiale ici : « the data being displayed in parallel coordinates can be linked to a map and then brushed to highlight the locations of interesting lines displayed in *m*-space on the parallel co-ordinates. » (FOTHERINGHAM 1999).

Encadré 5.1 : Construction et utilisation interactive d'un graphique en coordonnées parallèles

La figure 5.6 illustre les étapes successives de construction d'un graphique en coordonnées parallèles, depuis le tableau statistique (**A**) jusqu'au graphique final (**D**).
Pour cela, on projette les valeurs des variables sur des axes représentant chacune des variables (**B**). En normalisant la taille de ces axes et en les plaçant en parallèle (**C**), on peut alors tracer les « profils » des variables en reliant les positions de chacun des individus statistiques sur chacun des axes (**D**).
La seconde partie de la figure représente le mode d'interaction par *brushing* : on « brosse » sur chaque axe une sélection de valeurs à conserver (**E**). La sélection graphique est convertie en intervalles numériques et formalisée sous une forme classique (**F**) qui permet de filtrer les données sous-jacentes. Au final, cette opération renvoie le seul individu statistique répondant aux deux sélections graphiques (**G**).

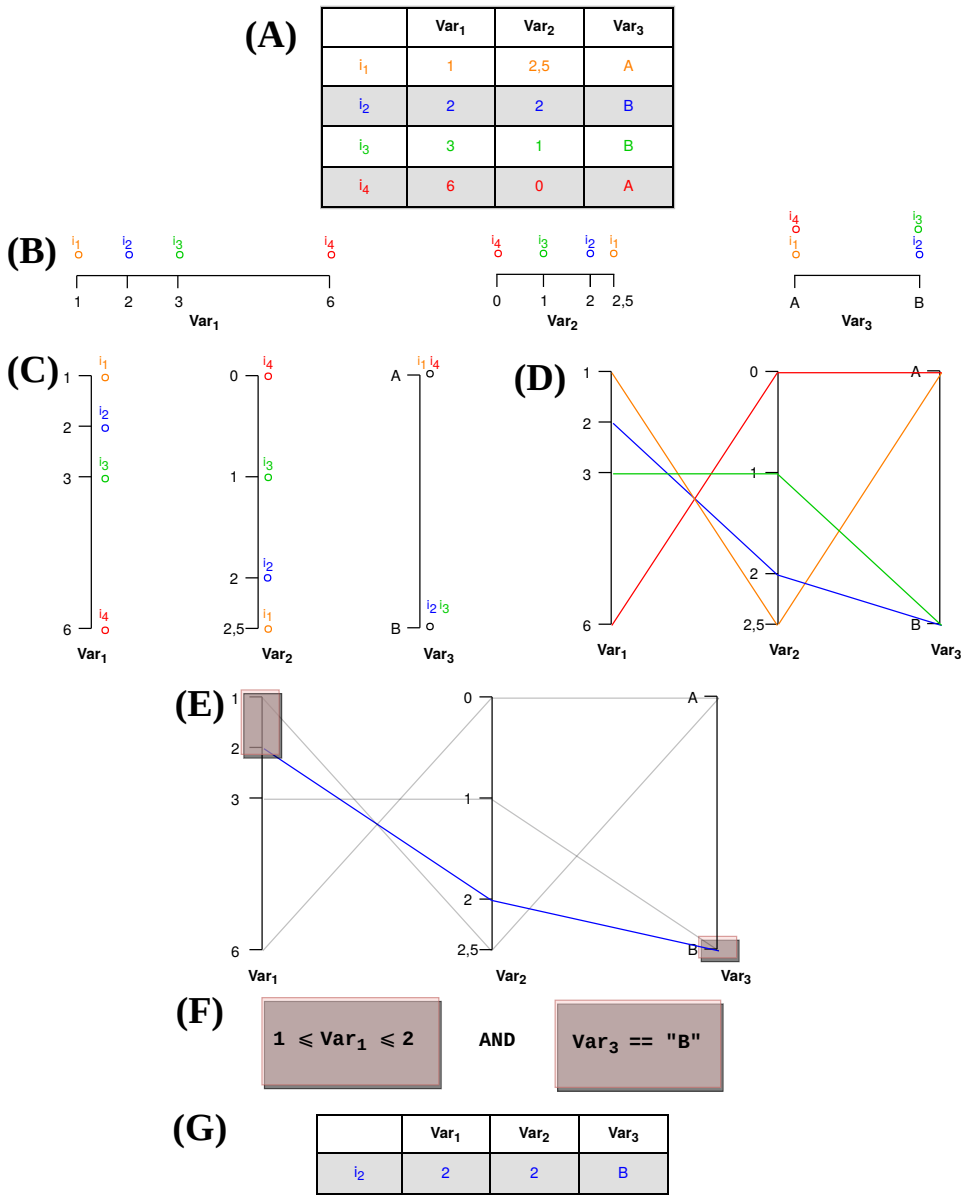


FIGURE 5.6 – Construction d'un graphique en coordonnées parallèles et sélection interactive.

Appliqué aux données de SimFeodal, cette interface (figure 5.7) se révèle particulièrement efficace pour sélectionner les configurations de paramètres à explorer. Ainsi, en « brossant » quelques filtres manuellement (figure 5.7 - **A**), on arrive rapidement à isoler une expérience spécifique.

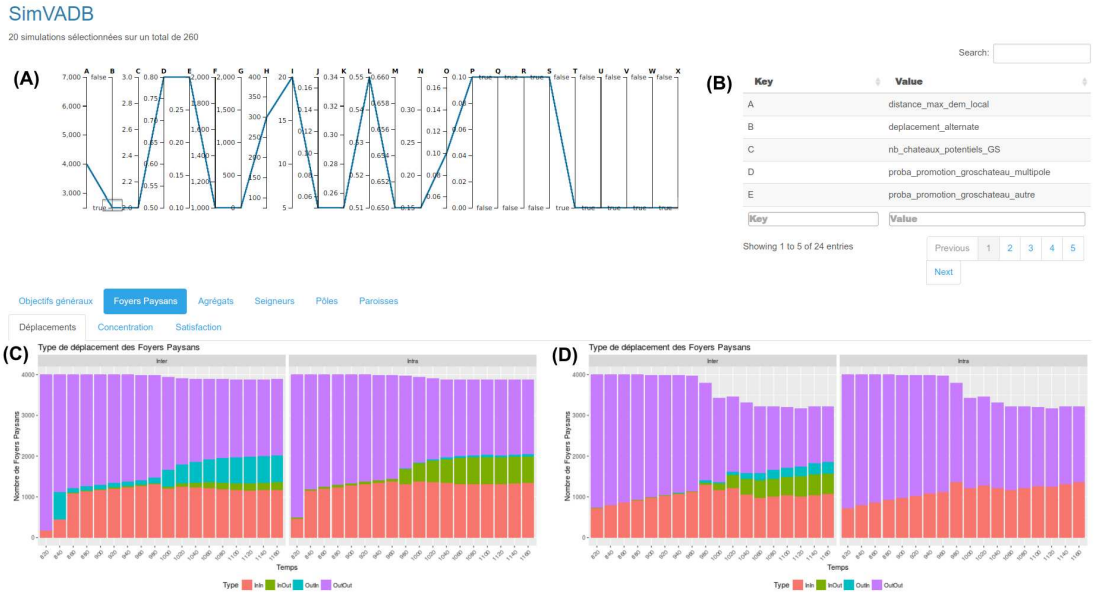


FIGURE 5.7 – SimVADB (Simulation Visual Analysis DashBoard), un *dashboard* d'exploration visuelle des indicateurs de sortie de simulation de SimFeodal. La sélection des simulations à explorer se fait dans le graphique en coordonnées parallèles (**A**), en « brossant » des filtres graphiques sur les « dimensions » du graphique, dimensions dont les intitulés sont explicités dans le tableau **B**. Les graphiques **C** et **D** indiquent l'évolution des types de déplacements des foyers paysans au cours des simulations.
- Le graphique **C** représente, pour cet indicateur, une moyenne de l'ensemble des simulations intégrées dans la base de données (260 ici), recouvrant donc plusieurs valeurs de paramètres.
- Le graphique **D** représente cet indicateur calculé depuis un sous-ensemble de 20 simulations (donc une expérience composée de 20 réplications), pour lesquelles le paramètre « B » (deplacement_alternate) vaut true.

Afin de permettre aux utilisateurs de remarquer les particularités des simulations explorées, nous avons choisi de mettre en emphase les différences entre la tendance générale des indicateurs, en calculant des moyennes de l'ensemble des simulations (figure 5.7 - **C**), et les valeurs spécifiques des indicateurs de l'expérience choisie (figure 5.7 - **D**). Cela permet, visuellement, d'être en mesure d'évaluer les sorties de simulation d'une expérience tout en ayant un référentiel visible. Les différentes expériences produisent des résultats sensiblement similaires [25], et on ne peut alors plus les comprendre sans les confronter à d'autres résultats similaires. Le choix d'une agrégation de l'ensemble des simulations effectuées est discutable, en ce qu'on aurait par exemple pu plutôt isoler des simulations « de référence » afin de diminuer l'effet « d'aplatissement » engendré par l'agrégation de résultats nombreux et hétérogènes. Toutefois, la variabilité des résultats étant encore assez restreinte, au moment de la création et de l'utilisation de SimVADB, ce référentiel agrégé permettait déjà une compréhension plus fine des sorties de simulations, en particulier dans l'analyse de l'impact de variations fines de valeurs de paramètres.

25. Si chaque expérience, et chaque réplication, produisent des résultats uniques, le choix d'une évaluation par des indicateurs visuels peut prêter à confusion si l'on n'a pas de repère précis. Les critères attendus, présentés dans le chapitre 3 sont ainsi assez précis pour départager une simulation très éloignée des attentes et une autre simulation plus conforme. Pour autant, par exemple quand les valeurs de paramètres varient faiblement, les résultats produits peuvent être assez similaires dans les grandes tendances qu'ils font ressortir.

5.2.6 Explorer en comparant : la plateforme SimEDB [26]

Après un travail de paramétrage grossier qui permet de stabiliser les mécanismes, il est souvent nécessaire de passer à une phase plus fine. On vise à ce moment à mieux calibrer un modèle à l'aide de variations de valeurs de paramètres de granularité inférieures. En vue d'évaluer les simulations, et donc de les différencier les unes des autres à l'aune des indicateurs générés, la comparaison d'une expérience spécifique avec un référentiel constitué de toutes les expériences précédentes ne permet plus de mener ce travail de comparaison fine. Les variations entre simulations sont trop fines pour être distinguables les unes des autres par le biais d'une comparaison avec un référentiel unique [27]. Cela s'entend quelque soit la manière dont ce référentiel est constitué, qu'il résulte d'une agrégation de simulations ou encore d'une version « de base » du modèle (par exemple, dans le cas de SimFeodal, les versions principales identifiées dans le chapitre 4).

Pour pouvoir correctement évaluer les apports d'un nouveau jeu de valeurs de paramètres, et donc, dans une démarche itérative, pouvoir différencier deux expériences successives, il est nécessaire d'être en mesure de comparer directement les expériences les unes avec les autres, ou encore avec un référentiel facilement ajustable. On peut énumérer les quelques exemples de cas de figure suivants :

— Comparaison entre une expérience spécifique et une autre expérience spécifique de même « importance ». Par exemple, comparer deux expériences qui font varier légèrement différement une valeur de paramètre.
— Comparaison entre une expérience spécifique et une autre expérience spécifique d'« importance » différente. Cette différence de niveau peut être constituée par exemple par une version « de base » que l'on comparerait à une variante de celle-ci.
— Comparaison entre deux expérimentations, par l'agrégation de leurs résultats. Si l'on a mené des expérimentations faisant varier de manière systématique deux paramètres différents, il peut être intéressant de les comparer en bloc, c'est-à-dire par exemple en prenant les moyennes de chacune des expériences composant ces expérimentations.

Ce faisant, on fait en fait varier la notion de « simulation de référence », qui peut alors revêtir plusieurs formes. Pour cela, il n'est plus possible de mener

26. La plate-forme d'exploration SimVADB (SimFeodal Visual Analysis Dashboard) a été renommée SimEDB ([...] Exploration Dashboard) par soucis de simplicité, le terme « Exploration » nous semblant plus compréhensible et explicite que celui de Visual Analysis. Ce nom apporte de plus une cohérence sémantique entre plusieurs productions de l'auteur – TimeLineEDB - (CURA 2017b) ; RoadTrafficEDB et CitiBikeEDB - (CURA 2017a). Cela inscrit cette plate-forme d'exploration de données issues de simulation dans une « famille » d'outils d'exploration de données spatio-temporelles.

27. On peut prendre l'exemple de SimFeodal. En faisant varier le nombre de foyers paysans de 4000 à 4200 (5% de variation), les résultats du modèle changent peu : de faibles variations de valeurs de paramètres entraînent le plus souvent de faibles variations dans les indicateurs de sorties observés. On peut de plus noter que les répercussions d'un changement de valeur de paramètre peuvent être très différentes de l'ordre de grandeur de ce changement de valeur. Cela s'explique par la non-linéarité de l'influence des paramètres. Dans l'exemple pris, pour ces 5% de variation dans le nombre de foyers paysans, la majorité des indicateurs variera ainsi de moins de 1%.

une comparaison visuelle entre un référentiel commun et une expérience spécifique, mais bien de baser l'évaluation sur la comparaison entre deux ensembles spécifiques qui doivent pouvoir être spécifiés. D'un point de vue méthodologique, cela requiert de pouvoir afficher conjointement les indicateurs de sorties de deux expériences (ou ensembles d'expériences). Cela implique aussi de laisser à l'utilisateur la responsabilité d'un choix supplémentaire puisqu'il faut désormais effectuer deux sélections : une pour chacun des points de comparaison. La sélection d'une expérience via l'usage de *brushing* sur un graphique en coordonnées parallèles des valeurs de paramètres ayant montré son efficacité, il a été choisi d'étendre ce principe d'interactivité au choix du référentiel.

Dans cette version remaniée de la plate-forme d'exploration (voir figure 5.8), renommée SimEDB (**Sim**Feodal **E**xploration **D**ash**B**oard) [28], l'accent est mis sur la comparaison de deux ensembles de résultats, chacun répondant à une sélection propre. L'utilisateur doit ainsi « paramétrer » interactivement, via *brushing*, les expériences à afficher pour le référentiel et pour les expériences à comparer. On dispose pour cela de deux outils de filtrage des simulations (partie de gauche dans la figure 5.8), qui peuvent être utilisés de concert, pour comparaison visuelle, ou par étapes successives [29].

En superposant les graphiques et tableaux des indicateurs, la comparaison visuelle est facilitée. On peut alors comparer deux variations fines d'un mécanisme du modèle, en sélectionnant par exemple une unique différence dans les valeurs de paramètres du modèle (par exemple un paramètre relatif à la promotion des paroisses dans la figure 5.8). De manière générale, ce choix d'outil d'interrogation des données permet de répondre à l'ensemble des cas de figures identifiés dans les paragraphes précédents.

28. Voir la note de bas de page 26.

29. En menant par exemple une première comparaison entre une expérience « A » en haut et « B » en bas, puis en sélectionnant « C » en haut, puis « D » en bas etc. On compare ainsi A avec B, puis B avec C, et enfin C avec D.

FIGURE 5.8 – SimEDB

Nous reviendrons plus précisément et longuement sur la description de SimEDB dans les parties suivantes (section 5.4, p. 57), mais après en avoir décrit les étapes de construction et les besoins auxquelles ces évolutions répondaient, il est maintenant nécessaire de revenir sur les données manipulées par cette plate-forme d'exploration. Le type, la structure et la masse de ces données (section 5.1) sont en effet indissociables des choix méthodologiques effectués pour SimEDB. Il est donc important de présenter les choix et contraintes de ces données avant d'entrer dans une description approfondie de la plate-forme.

5.3 Organiser les données

On a vu dans la première partie de ce chapitre (section 5.1) que les données produites par SimFeodal étaient nombreuses, diverses et massives. La seconde partie (section 5.2) a montré les types de problèmes que de telles données, une fois exploitées pour en tirer les indicateurs de sortie, peuvent poser en matière d'exploration.

Nous souhaitons ici revenir sur une corollaire indispensable à l'utilisation efficace de ces données. De la même manière que la multiplicité des indicateurs, des expériences et des expérimentations requiert des outils d'exploration adaptés, la multiplicité des données requiert des outils de stockage et d'interrogation eux aussi adaptés. Là encore, on peut noter une succession de contraintes liées à ces données et à leur massification, contraintes qui limitent et guident les modes d'organisation de ces données. Sans structuration adéquate, l'acquisition, l'archivage, l'interrogation ou encore la sauvegarde des données générées par le modèle ne peuvent être garantis, et encore moins de manière efficace. Le choix d'une méthode d'organisation des données en sortie de simulations ne relève donc pas d'une quelconque coquetterie technique. Au contraire, conditionne et contraint fortement aussi bien les modalités de création des indicateurs que les possibilités de la plate-forme d'exploration de les afficher de manière interactive.

5.3.1 Assurer la capacité d'interrogation des données

Avant de se soucier du « schéma » de la base de données [30], du choix du Système de Gestion de Base de Données (SGBD), ou encore des performances de ce dernier, il convient de se fixer sur la manière dont on souhaite entreposer les données.

De la myriade de fichiers issus de tableurs organisés dans une multitude de dossiers spécifiques à l'entrepôt de données décentralisé orienté documents, en passant par les traditionnelles bases de données relationnelles, les possibilités de stockage et d'organisation des données issues de simulations sont ainsi innombrables. Plusieurs contraintes successives permettent de limiter le choix à un sous-ensemble de solution adaptées parmi lesquelles on peut alors mener une étude plus approfondie, passant notamment par des comparaisons entre ces candidats.

Une des premières contraintes est constituée par la nécessité d'interroger fréquemment et de manière répétée les données. C'est l'une des contre-parties du passage d'un rapport automatique à un outil d'exploration interactif et dynamique (section 5.2.4). Dans un rapport automatique, on interroge une fois les données pour en tirer les indicateurs de sortie, et ceux-ci ne sont plus amenés à changer, sauf re-calcul par exemple suite à des ajouts d'indicateurs. Au

30. On utilise souvent le terme de « schéma » pour désigner la version implémentée, dans un SGBD spécifique, du Modèle Conceptuel de Données (MCD). Contrairement au MCD, qui donne une version conceptuelle et générique d'une base de données, le schéma est donc tributaire du SGBD dans lequel il est intégré.

contraire, dans un outil dynamique et interactif, les indicateurs sont calculés depuis les données « à la volée », c'est-à-dire à chaque qu'un indicateur doit être affiché. On peut mettre en place un système de cache, pour conserver les calculs déjà effectués, mais avec les dernières itérations de SimEDB où le nombre de possibilités de sélections est extrêmement important, ce n'est plus possible. Il est nécessaire de procéder aux calculs à chaque affichage des graphiques et tableaux de résultats. L'interrogation des données est donc extrêmement fréquente et répétitive. Elle doit alors être aussi simple (en termes de mode d'interrogation) qu'efficace (en termes de rapidité d'interrogation).

Dans le cadre des données issues de SimFeodal, en vue de leur mobilisation dans SimEDB, nous avons ainsi eu à sélectionner quelques SGBD candidats parmi une foule de solution possibles. Afin de guider ce choix, trois critères ont été définies, et forment, selon une hiérarchie propre à leur ordre, un ensemble de filtre ayant permis la réduction des SGBD possibles à un nombre appréhendable. Ces critères, dont l'énonciation guidera cette partie, sont ainsi (1) l'universalité, ou « agnosticité », des SGBD aux outils de requête ; (2) la pérennité et stabilité des solutions disponibles et (3) les performances des SGBD considérés.

5.3.1.1 Interroger de manière universelle et indépendante

Lors de la conception d'un outil faisant appel à des données, qui plus est massives, il convient de se positionner tôt sur la manière d'interroger ces données. Par interrogation, on entend ici, comme souvent dans le domaine des bases de données, la manière de faire appel, concrètement, aux données, pour en tirer les sous-ensembles, agrégations et autres résultats synthétiques résultant du traitement des données brutes. Si l'on considère des données stockées dans un tableur, alors les « formules », les tableaux croisés dynamiques ou encore les graphiques issus du tableur sont des interrogations des données, qui s'expriment dans ce cas via un ensemble de langages, écrits – les formules, qui font appel à des fonctions spécifiques des tableurs – ou visuels – les tableaux croisés dynamiques, construits en faisant glisser des intitulés de colonnes dans un tableau.

Stockage distribué ou centralisé. Avant même de s'intéresser aux spécificités de ces langages, un premier choix réside dans le mode de stockage des données qui doivent être mis à disposition d'une plate-forme. Doit-on laisser l'utilisateur intégrer lui-même les données, et ainsi, en faire un stockage « distribué », dans le sens où chaque utilisateur de l'application possèderait physiquement une copie locale des données ? Ou, au contraire, les données doivent-elles être centralisées, c'est-à-dire enregistrées en une seule copie à laquelle les utilisateurs accèderaient à distance ? Pour reprendre l'exemple des tableurs, doit-on privilégier une solution locale – chacun ayant une copie du fichier tableur et menant ses propres modifications dessus – ou une approche de type centralisée, par exemple en privilégiant des tableurs collaboratifs en ligne (*Google Docs/Sheets* par exemple) ?

Habituellement, c'est-à-dire dans une grande majorité d'applications, les don-

nées sont stockées localement : cela permet, en particulier, de ne pas dépendre d'une connexion internet pour interroger des données qui seraient hébergées sur internet. Dans le cas de SimFeodal, cette solution est rendue difficile, sinon impossible, par la masse de données en sortie de simulations. Si chaque utilisateur de SimEDB devait posséder une copie des données, voir plusieurs en cas d'utilisations depuis différents ordinateurs, cela occuperait plusieurs gigaoctets de données à chaque fois. De plus, en cas de mise à jour des données, c'est-à-dire d'insertions de nouvelles sorties de simulations, il faudrait distribuer à nouveau l'ensemble du jeu de données, à chaque fois.

Pour ces raisons, nous avons fait le choix d'un stockage centralisé, sous forme d'une architecture « client-serveur », hébergé sur un serveur internet dédié, ce qui permet à la plate-forme de travailler à chaque fois sur les données les plus à jour, réduit la taille du stockage physique associé, et dispense d'une configuration sur chaque poste utilisateur : si le lien entre l'application et les données fonctionne correctement pour un utilisateur, il fonctionnera à l'identique pour tous les autres. Ce choix présente un dernier avantage, non négligeable : en stockant les données en un seul lieu, c'est-à-dire sur un serveur informatique, on peut faire en sorte de rendre ce serveur aussi performant que possible, et accélérer ainsi l'interrogation des données pour tous les utilisateurs.

Interrogation spécifique ou générique. De nombreuses solutions intégrées de gestion de données proposent leurs propres modes d'interaction avec les données, c'est-à-dire un langage spécifique permettant d'interroger les données contenues dans le système [31]. Au contraire, les SGBD les plus classiques s'appuient plutôt sur des langages de requêtes aussi standardisés que possibles, afin de faciliter l'adoption de leur propre solution à des utilisateurs d'autres plate-formes. La spécificité présente l'avantage de langages plus adaptés aux données manipulées, et donc souvent plus intuitifs dans l'interrogation des spécificités des données. De plus, la spécificité permet aussi une optimisation des requêtes, et est donc souvent plus performante que les solutions plus génériques.

De manière générale et dans le cas de SimFeodal, nous avons préféré privilégier une approche plus générique, faisant appel à des solutions de SGBD plus standardisées. La raison tient principalement à une volonté de généricité du stockage des données : au cours des différentes étapes de construction de SimFeodal, les besoins en matières d'interrogation des données ont évolué. Cette évolution était prévisible et prévue, et nous avons donc choisi dès le départ d'adopter uniquement des solutions modulaires, garantissant une évolutivité facilitée de la base de données, aussi bien regardant sa structure (enregistrement de nouvelles variables ou de nouveaux agents du modèle) que son contenu (massification des données en sortie au fur et à mesure de l'exploration du modèle).

De plus, dans la perspective de ce travail de thèse, où l'on cherche à rendre

31. Souvent, cette interrogation se fait par appel à des *API* (*Application Programming Interface*, ou Interface de Programmation Applicative en français). Ces interfaces sont propres à chacune des plate-formes, et demandent donc un langage et un formalisme de requête spécifique.

les productions aussi reproductibles et génériques que possible, il était indispensable de disposer d'un SGBD aussi standard que possible pour en faciliter l'adoption et l'adaptation à d'autres modèles de simulations par exemple.

Bases de données relationnelles ou NoSQL. Même une fois arrêté sur le choix de ne faire appel qu'à des outils standards pour stocker les données, le nombre de solutions disponibles demeure très important. Afin de réduire ce nombre, on peut déjà choisir les grands types de SGBD auxquels faire appel. Les SGBD sont ainsi souvent classés selon les grands traits de la méthode dont ils organisent les données. Les deux grands types [32] sont les SGBD « relationnels » et les SGBD « NoSQL ». Si la distinction est sujette à de très nombreux débats, souvent virulents [33] , on se contentera ici de définir les SGBD relationnels comme les SGBD, les plus fréquemment utilisés, où l'information est stockée dans des tables composées de champs – les colonnes, correspondant aux variables – et de lignes – les entités décrites par les variables. Le format des données est donc rectangulaire et n'accepte pas, comme dans un tableau statistique, que les entités possèdent un nombre de variables différent, ou encore des types de valeurs différents de celles des autres entités (une même colonne ne peut donc contenir conjointement un nombre et un texte dans des lignes différentes). On nomme ces bases de données relationnelles via la manière qu'elles ont de faire communiquer des éléments hétérogènes, et donc contenus dans des tables différentes : si les tables ont une colonne en commun, on pourra alors effectuer une opération de jointure permettant de mettre en commun les informations de ces tables dans une unique table résultante.

A l'inverse, les SGBD NoSQL se définissent de manière opposée à ce mode de stockage [34], rompant par exemple la contrainte d'unicité de type des colonnes, ou de nombre identique de colonnes renseignées pour chaque entité. Pour simplifier le discours, on se contentera de caractériser les SGBD NoSQL comme des SGBD non relationnels. Les SGBD NoSQL ont, en général, de bien meilleures performances et une plus grande flexibilité que les SGBD relationnels. Dans le cas de SimEDB/SimFeodal, où l'on est confronté à des données massives, cela présente un avantage non négligeable.

Toutefois, leur flexibilité est associée à une contrainte majeure en termes de généricité : alors que les SGBD relationnels partagent un langage d'interro-

32. Il en existe d'autres, comme les SGBD orientés objets (quasiment disparus aujourd'hui), orientés graphes (Neo4j…), les SGBD pensés pour le stockage et l'interrogation d'ontologies (*Triplestores RDF*, interrogeables en langage SPARQL) ou encore les nouveaux SGBD de type « NewSQL » (Apache Ignite, CockRoachDB…) pensés pour une parallélisation massive des données. Ces types de SGBD ne correspondent toutefois pas du tout aux besoins identifiés pour SimEDB/SimFeodal, et sont en général dédiés à des problèmes et marchés de niches. Nous ne les décrirons donc pas plus en détail ici.

33. À l'instar des violentes querelles qui agitent régulièrement les informaticiens : Vim vs Emacs, Programmation Orientée-Objet vs Programmation Fonctionnelle, R vs Python…

34. À l'origine, c'était le sens fort du nom « NoSQL » : Non SQL, le SQL faisant ici références aux SGBD pré-existants, majoritairement relationnels, dont la mouvance NoSQL, portée par l'apparition des « *big data* » a voulu se distinguer. Sans entrer dans le détail, notons tout de mêmes que de nombreux SGBD NoSQL, qui traduisent désormais cet acronyme par « Not only SQL », sont maintenant relationnels, mais mettent en avant d'autres types d'approches.

gation commun, le SQL (*Structured Query Language*)[35], les SGBD NoSQL font plus souvent appels à des langages spécifiques à chaque SGBD. Pour SimEDB, cela impliquerait une fort dépendance au SGBD choisi : en cas de changement de SGBD, toutes les requêtes seraient à reformuler dans le nouveau langage, parfois même selon des logiques extrêmement différentes les unes des autres (dérivés du SQL, interrogations via des objets JSON, via des langages de parcours de graphes…) etc. Au contraire, avec les SGBD relationnels, le langage de requête étant commun, une fois le code d'interrogation généré, il est très aisé de changer de SGBD cible. Cela garantie une forte capacité d'évolutivité aux outils d'interrogation de données tels que SimEDB. Puisque les fournisseurs de données sont interchangeables, on peut en changer au fur et à mesure de l'apparition de nouveaux besoins.

En raison de la généricité de ces solutions relationnelles, qui vient s'ancrer dans la recherche de reproductibilité et de généricité de notre démarche d'ensemble, nous avons donc choisi de faire reposer le stockage et l'interrogation des données sur des SGBD relationnels. Cette décision s'est montrée d'autant plus heureuse que, au cours de la construction et de l'évolution de SimEDB, le SGBD choisi pour héberger les données a changé plusieurs fois. La généricité des outils choisis a permis de minimiser, voire quasiment d'éviter, les changements à apporter au code-source de SimEDB relatif à l'interrogation de données en vue de produire les indicateurs de sortie depuis les données brutes en sortie de SimFeodal.

Entrepôts de données et interrogation directe. En parallèle des SGBD, des solutions « intermédiaires » permettent de s'abstraire des SGBD en eux-mêmes pour mener les requêtes. Ces solutions, que l'on nomme « Entrepôts de Données » (*Data Warehouses*), se comportent comme une surcouche faisant l'interface entre un ou plusieurs SGBD et la requête émise par le client final. Elles se placent donc comme intermédiaire entre les SGBD mobilisés et les applications qui les interrogent. Les entrepôts de données jouent aussi bien le rôle d'agrégateurs de données[36] que d'environnements de manipulation et de restructuration de données (on les nomme alors « ETL » – *Extract-Transform-Load*).

Le grand intérêt de ces outils d'interface est d'abstraire la complexité de chacune des bases de données manipulées en générant une interface d'interrogation unique et générique, souvent performante grâce à des optimisations spécifiques (pré-calcul des requêtes possibles par exemple).

35. Ce langage d'interrogation est omniprésent dans l'interaction avec les SGBD, mais aussi, avec de légères variantes, au sein de nombreux logiciels reposant des sélections de données, par exemple les logiciels SIG qui se basent sur la syntaxe du SQL (le fameux triptyque `SELECT … FROM … WHERE …`)

36. C'est-à-dire qu'ils permettent d'agréger des sources de données composites, provenant potentiellement de différentes sources (plusieurs bases de données relationnelles) et de différents types de sources (différents SGBD, relationnels ou non par exemple).

Environnements OLAP. Dans le domaine de la visualisation interactive de données, ces outils sont beaucoup utilisés, en particulier dans le monde de l'informatique décisionnelle. Ils se révèlent en effet extrêmement utiles quand les données sources ne peuvent être modifiées (par exemple quand elles sont issues de chaînes de collecte complexes, ou encore quand leur volumétrie et leur débit est important), puisqu'ils permettent de constituer une surcouche rendant l'interrogation et la visualisation de ces données accessibles à des analystes non spécialistes de la manipulation de données. Toujours en informatique décisionnelle, il est courant de faire appel à des entrepôts de données d'un type spécifique, les « traitements analytiques en lignes », ou environnements OLAP (***OnLine Analytical Processing***), qui permettent de structurer, par exemple sous formes de cubes de données, des sources de données hétérogènes présentant de nombreuses dimensions.

Environnements SOLAP. Les environnements OLAP ont été utilisés, promus et adoptés dans le champs scientifique de la géomatique, en ce qu'ils permettent de mettre en place rapidement des environnements d'analyse visuelle de données multi-dimensionnelles spatiales et temporelles. Dans ce cadre, où ces outils sont appelés « SOLAP » (*Spatial OLAP*), les données spatiales s'intègrent extrêmement bien en raison de leur capacité à s'emboîter selon les échelles, ouvrant dès lors la voie à des analyses multi-échelles et multi-dimensionnelles complexes.

Dans la communauté géomatique francophone, les solutions SOLAP sont bien représentées (par exemple autour de Sandro Bimonte et de son travail de visualisation de données spatiales environnementales, (BIMONTE 2007 ; BIMONTE, TCHOUNIKINE et MIQUEL 2005 ; ZAAMOUNE et al. 2013), et sont couramment employées pour répondre à des questionnements méthodologiques proches de ceux développés dans cet ouvrage. En lien avec les besoins de performances identifiés plus haut, on notera que certaines solutions OLAP permettent aussi d'optimiser la vitesse d'interrogation de bases de données, et visent ainsi à garantir une réponse rapide pour des outils d'interrogation de données interactifs (ZENG, AGARWAL et STOICA 2016).

Performance ou généricité. Nous avons cependant choisi de ne pas faire usage de ces outils pour les mêmes raisons que pour les SGBD NoSQL : les avantages qu'ils présentent ne suffisent pas à contre-balancer la perte de généricité qu'ils impliquent. Pour profiter au mieux de ces environnements, il est en effet nécessaire de faire appel à un nouveau langage d'interrogation des données (le « MDX », de « *Multidimensional Expressions* »). Les différentes solutions OLAP/SOLAP, de plus, présentent les mêmes inconvénients que les SGBD NoSQL : chacune interagit de manière propre aux différents SGBD, et ces outils sont donc difficilement interchangeables.

De la même manière, on se restreindra, parmi les SGBD relationnels interrogeables en SQL, à ceux qui disposent d'une méthode d'interrogation standard : si tous ces SGBD acceptent le SQL, certains demandent par exemple des pro-

tocoles spécifiques pour recevoir établir la connexion au SGBD, recevoir la requête et renvoyer les données correspondantes. Pour ce même choix « d'agnosticité » de la plate-forme d'interrogation face à la solution de stockage choisie, on ne conservera que les SGBD acceptant les connexion standardisées (ODBC et JDBC).

SGBD et données spatiales. On a mentionné le fait que les entrepôts de données étaient fortement utilisés, en particulier dans la communauté géomatique, car très appropriés aux données spatiales. De prime abord, ce point peut paraître critique : jusque là, on s'est contenté de mentionner les capacités organisationnelles de SGBD, et non leur aptitude à manipuler des données spatiales. Ce point, dans les SGBD relationnels, constitue un filtre important : sur la centaine de solutions disponibles, seule une poignée est en mesure de stocker efficacement et d'interroger de l'information spatiale [37].

Pourtant, au regard des indicateurs de sortie de simulation sur lesquels on s'appuie pour évaluer le comportement de SimFeodal, une grande majorité est non spatiale, en raison de la difficulté à agréger des données spatiales théoriques. Dans SimFeodal, la plupart des indicateurs de sortie sont non spatiaux, c'est-à-dire qu'ils mobilisent plus la dimension attributaire des données que leur dimension géographique [38].

La gestion de données spatiales ne constitue donc pas une absolue nécessité, contrairement aux points évoqués auparavant. Elle peut toutefois se révéler avantageuse, ne serait-ce que pour permettre l'observation des configurations spatiales simulées. Cela constitue une approche idiographique, visant à exemplifier plus qu'à synthétiser, mais permet tout de même une compréhension rapide des changements de structures spatiales. Toutes choses égales par ailleurs, on privilégiera donc des solutions de stockage ayant une capacité à gérer les données spatiales.

Pour héberger et organiser les données produites par SimFeodal, en vue de leur interrogation dans SimEDB, nous avons choisi de restreindre la myriade de solutions disponibles grâce à plusieurs filtres successifs. En premier lieu, on a choisit de faire appel à des solutions centralisées (Stockage distribué ou centralisé), au sein de Systèmes de Gestion de Base de Données (SGBD). Ces SGBD permettent une interrogation standardisée (Interrogation spécifique ou générique) via un langage de requête universel, le SQL (Bases de données relationnelles ou NoSQL). On a ensuite décidé d'interroger ces SGBD sans passer par l'intermédiaire d'entrepôts de données, et au travers de connexion aussi

37. Les données spatiales peuvent être stockées dans tous les SGBD si l'on attribue une représentation textuelle, en chaînes de caractères, par exemple en utilisant le format *Well-Known Text* (WKT). Pour autant, ce format est lourd, inadapté à une indexation, et ne peut permettre à un SGBD de mener des requêtes spatiales directement depuis ces entités. Il est ainsi, par exemple, impossible de calculer le centroïde d'un polygone directement depuis une représentation WKT, alors que c'est aisé avec un stockage géométrique.

38. Les raisons en sont multiples et on y reviendra largement dans le chapitre 7. Notons tout de même que les indicateurs résultent de l'agrégation des réplications, et que cette agrégation est extrêmement complexe sinon impossible sur des données spatiales majoritairement aléatoires (voir chapitre 2, section 2.2.2.1).

standardisées que possible (Entrepôts de données et interrogation directe). Les SGBD répondant à ces critères sont les SGBD « relationnels », dont certains possèdent qui plus est une capacité intéressante à stocker et interroger des données spatiales (SGBD et données spatiales), ce qui constitue notre dernier filtre.

5.3.1.2 Interroger de manière robuste et performante

En dépit de l'accumulation de critères exposée précédemment, une quantité importante de SGBD demeurent en lice. Afin de les différencier, nous avons choisi d'ajouter des critères qui ne portent plus sur les grands types de SGBD, mais plutôt sur une différenciation des SGBD relationnels existants. Ces deux critères, précisés par la suite, sont d'une part la robustesse des SGBD, et d'autre part leurs performances. Ces critères sont pas des « prétextes » à une hiérarchisation quantifiée des SGBD, mais ont une importance prépondérante dans notre cas d'utilisation.

Robustesse des SGBD. Le premier critère ajouté est celui de la robustesse, c'est-à-dire, ici, de la capacité du SGBD à être interrogé de manière (1) stable et (2) pérenne dans le temps. Une même requête sur les mêmes données doit systématiquement renvoyer le même résultat (stabilité), quelle que soit la durée séparant ces requêtes (pérennité). Si la base de données n'est plus interrogeable quelques mois après sa configuration, ou qu'elle renvoie des résultats différents, alors elle ne peut constituer une solution crédible à l'exploration d'un modèle sur une période longue.

La **stabilité** des bases de données est principalement due à la manière de stocker l'information d'un point de vue informatique.
En premier lieu, l'information peut être contenue « en clair » ou alors de manière archivée. Un stockage « en clair » est plus facilement accessible, puisqu'on peut le consulter avec n'importe quel éditeur de texte. Un stockage archivé est moins universel, mais occupe généralement un espace disque inférieur et comporte des mécanismes de vérification de la cohérence des données. Il est donc plus stable.
Une second différenciation tient à l'emplacement lieu du stockage. Celui-ci peut être effectué dans un unique fichier, ce qui a l'immense avantage de la portabilité des données : pour faire migrer ou sauvegarder la base de données, il suffit de copier le fichier. La plupart des SGBD adoptent toutefois un mode d'organisation en plusieurs fichiers, notamment pour des questions de redondance et de vérification de l'intégrité des données : en multipliant les fichiers, on minimise le risque d'erreur critique sur l'ensemble des fichiers à la fois. On notera enfin un dernier type de SGBD, où l'information n'est pas stockée sur un disque dur, mais est entièrement contenue dans la mémoire vive de l'ordinateur : les SGBD « *in-memory* ». Ces SGBD sont les plus rapides et stables, mais il faut les re-constituer à chaque redémarrage du serveur qui les héberge, ce qui peut prendre un temps important.

L'enjeu du choix est de se prémunir de « corruptions » de la base de données : quand le SGBD ne comporte que pas ou peu de mécanismes de vérification de l'intégrité ou de la cohérence des données, il peut arriver qu'une base de

données se corrompe. On peut prendre l'exemple de l'exécution d'une requête demandant un calcul complexe et long. Cette requête pourrait être interrompue en cours d'exécution par faute d'un *bug* ou d'une expiration de sessions (*timeout*). Dans ce cas, il se peut que la base de données s'arrête dans un état muté – avec une nouvelle table ajoutée pour moitié par exemple – et ne soit donc plus intègre. C'est très fréquent pour les SGBD basés sur un unique fichier, ou encore stockés en clair, puisque les nouvelles informations de la base de données y sont ajoutées au fur et à mesure, plutôt que d'être intégrées dans un fichier annexe que l'on pourrait réinitialiser en cas d'erreur.
Avec la volumétrie des données produites par SimFeodal, les requêtes peuvent s'avérer très longues, et une erreur dans une requête peut fréquemment corrompre la base de données. En termes de stabilité, on se tournera donc plutôt vers des SGBD relationnels stables, basés sur une redondance des données et donc sur des architectures archivées et multi-fichiers.

La **pérennité** des SGBD est un sujet proche, tenant aussi à la capacité à interroger les données contenues dans une base de données, mais cette fois-ci du point de vue de l'interrogation en elle-même plutôt de des données sur lesquelles elle s'applique : si le SQL est un langage standard [39], les types de données intégrées varient cependant d'un SGBD à un autre (champs textuels ou d'entiers « courts » par exemple). SQL étant un langage typé, selon la manière (bas niveau) dont sont intégrées les données, certaines requêtes identiques peuvent renvoyer des résultats différents selon les SGBD. Plus gênant, les normes implémentées peuvent varier d'une version à l'autre d'un SGBD. Un SGBD relationnel respectant strictement la norme SQL pourrait ainsi évoluer pour supporter plus de fonctionnalités, par exemple en ajoutant des fonctions plus récentes (fenêtres glissantes, ajouts en masse etc.), et renverrait dès lors des résultats différents selon les versions. Pour les SGBD les plus employés, le nombre d'utilisateurs garantit une rétro-compatibilité des requêtes. Pour les SGBD de moindre envergure cependant, par exemple les plus performants et récents issus de la recherche en informatique, cette rétro-compatibilité n'est pas du tout garantie.

Comme souvent en matière d'infrastructure informatique, il est donc nécessaire de tenir compte d'un compromis entre l'ancienneté et la forte adoption de certains SGBD d'une part, et les facilités et gains de performances amenées par les plus récents d'autre part. Dans le cas des données de SimFeodal, en tenant compte de cet inévitable compromis, nous avons choisi de privilégier des SGBD reconnus, soient-ils anciens et fortement adoptés ou plus récents mais utilisés par des acteurs d'envergure [40]. Ce faisant, on se coupe immanquablement de solutions extrêmement intéressantes et performantes [41]. Ce choix est toutefois

39. Dans les faits, on notera tout de même qu'il existe plusieurs normes successives, des « révisions » du SQL, qui apportent chacune leur lot de subtilités dans l'usage du langage. Les SGBD interrogeables en SQL ne disposent donc pas toutes des mêmes fonctionnalités, selon la révision du SQL qu'elles respectent.

40. La liste des solutions envisagées, ensuite comparées à l'aube de leurs performances, est visible dans l'axe des ordonnées de la figure 5.9.

41. Par exemple BlinkDB (AGARWAL et al. 2013), qui permet de limiter une requête à un temps maximal d'exécution donné : quand la requête n'est pas complète, le SGBD renvoi une

en la large faveur d'une meilleure garantie de pérennité, et de robustesse en générale, des données de SimFeodal.

Performance des SGBD. Une fois que les solutions disponibles ont été discriminées par leur type, par leur interface avec les requêtes et par leur robustesse, la quantité de SGBD restant demeure de l'ordre de la dizaine. Pour choisir, parmi ceux-là, le SGBD qui sera le plus adapté aux besoins identifiés, il est donc nécessaire d'établir des critères plus précis et quantifiables. Dans le cas d'une application interactive, c'est-à-dire où le nombre de requêtes émises au cours d'une session d'utilisation peut être importante, les performances des SGBD constituent un critère majeur pour départager l'ensemble des SGBD considérés.

Il est difficile de qualifier les « performances » d'un SGBD : on entend en fait par ce terme un vaste ensemble hétérogènes de propriétés. On peut par exemple juger les performances par le filtre de la mémoire occupée par le stockage d'une base de données, ou encore par le nombre de requêtes concurrentes que peut gérer un SGBD, ou encore par la capacité à paralléliser le stockage sur plusieurs serveurs. Dans notre cas, ces points sont assez peu significatifs : en dépit de la quantité de sorties, l'ordre de grandeur – quelques gigaoctets de données – reste largement entreposable sur un environnement classique, sans besoin de parallélisation. De la même manière, SimEDB est un environnement dédié à des utilisateurs experts, en petit nombre : les chercheurs travaillant autour de SimFeodal. La quantité de requêtes simultanées ne peut donc pas dépasser la dizaine, ce qui constitue une trivialité pour l'ensemble des SGBD relationnels classiques. On s'attachera donc à juger les performances en matière de rapidité d'exécution des requêtes. Il ne s'agit pas ici de choisir un SGBD qui ferait gagner quelques millisecondes par rapport à un autre, mais plutôt de discriminer les SGBD présentant une durée de réponse trop importante pour notre usage.

En effet, plus les données sont massives, plus le temps d'exécution d'une requête augmente, souvent sous la forme d'une fonction puissance. Si tous les SGBD présentent des vitesses acceptables et proches sur des bases de données de faible volume, l'écart s'accroît considérablement à mesure que les données s'accumulent. La figure 5.9 [42] montre les différences incontestables qui existent entre les SGBD étudiés. On peut y constater que l'écart est gigantesque, par exemple vis-à-vis du temps nécessaire à une jointure, entre les 4 secondes de MonetDB et les 300 secondes (5 minutes…) de SQLite. Le choix d'un SGBD selon ses performances a donc un impact majeur sur la fluidité d'une application d'exploration de données massives. Pour départager les SGBD, on comparera

estimation du résultat, estimation qui gagne en précision quand on augmente la limite temporelle. Un SGBD de ce type serait extrêmement précieux en *visual analytics*, mais la jeunesse de cet outil ainsi que sa nature de projet de recherche rendent incertain la continuité de son développement dans le temps. En 2018, le projet semble d'ailleurs avoir été abandonné…

42. Dans cette figure, on compare la rapidité de différentes requêtes sur un jeu de données identique selon les SGBD. Ce type de test de performance permettant la comparaison de solutions techniques diverses est appelé *benchmark*. Ce jeu de données, composé de 100 Millions de lignes et de deux colonnes numériques, présente une volumétrie comparable (franchement inférieure en nombre de colonnes toutefois) à celle des données issues de SimFeodal qui sont interrogées dans SimEDB.

leurs performances selon les différents types d'opérations demandées, qu'elles concernent l'écriture dans la base (insertion) ou des types de lecture (agrégation et jointure).

Performances en écriture et en lecture. La figure 5.9 affiche des résultats qui semblent globalement ordonnés (les quatre premiers SGBD sont par exemple quasiment toujours plus lents que les 2 derniers), mais fluctuent cependant à la marge selon les opérations demandées. La première colonne du graphique montre ainsi le temps nécessaire à l'insertion du jeu de données exemple (voir la note de bas de page 42) dans le SGBD depuis un fichier CSV. Les deux colonnes suivantes exposent le temps nécessaire au traitement d'une requête, donc à une interrogation des données une fois archivées dans les bases de données. On peut constater que le classement des SGBD varie faiblement en lecture, et de manière assez faible en insertion : les deuxièmes et troisièmes colonnes respectent un ordre globalement similaire, assez différent de celui de la première colonne. Dans un environnement classique, la performance d'insertion de données est un facteur prépondérant : quand de nouvelles données sont ajoutées constamment, par exemple pour stocker des données issues de capteurs automatiques, l'insertion peut vite constituer le goulot d'étranglement de la solution.

Pour SimFeodal, l'insertion n'est pas véritablement un enjeu : les données sont ajoutées par bloc, manuellement, une fois que des nouvelles simulations ont été exécutées. C'est donc au pire un acte quotidien, mais dans ce cas, que la requête demande 10 secondes (MapD) ou 10 minutes (MySQL InnoDB), cela n'a que peu d'impact. La première colonne est donc un indicateur de performance peu adapté dans notre cas.

Les deux colonnes suivantes, relatives à l'interrogation de données, se révèlent au contraire extrêmement importantes : à chaque action de l'utilisateur de SimEDB, une nouvelle requête est envoyée pour calculer un nouvel indicateur correspondant au jeu de données filtré manuellement (cf. section 5.2.5). À chaque affichage d'onglet, une nouvelle requête est donc émise et traitée. Même si tous les indicateurs ne sont pas systématiquement mobilisés – et donc calculés – (cf. chap 3), cela signifie tout de même que pour chaque sélection, une bonne dizaine d'indicateurs seront observés, et donc, autant de requêtes. Quand une requête demande 60 secondes (par exemple PostgreSQL en « jointure »), cela implique que chaque indicateur requiert au moins une minute avant de s'afficher. Pour observer une dizaine indicateurs, l'utilisateur devra donc attendre patiemment une dizaine de minutes, sans même tenir compte du temps qu'il passera à les analyser visuellement.

Pour noircir le trait, notons de plus que les résultats communiqués dans la figure 5.9 correspondent à des requêtes simples qui ont valeur d'exemples de base. Dans le cas de SimEDB, le calcul des indicateurs requiert des requêtes plus complexes, faisant appel à des agrégations et à des jointures en même temps, et les délais affichés dans ce *benchmark* sont donc bien inférieures aux durées éprouvées en conditions réelles au sein de SimEDB.

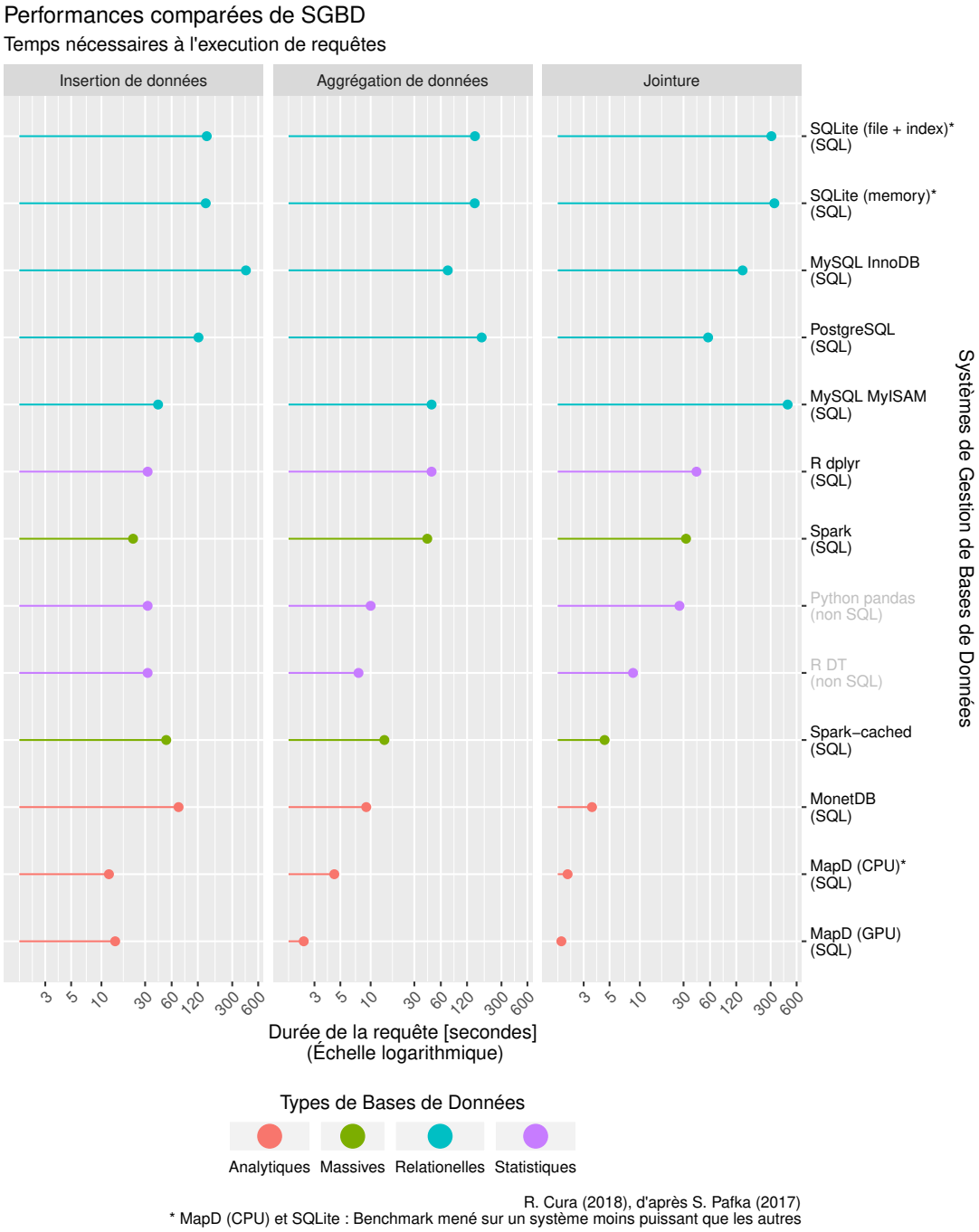


FIGURE 5.9 – Comparaison de la performance de différents SGBD sur un jeu de données test de 100 millions de lignes. Résultats tirés de PAFKA (2017) et complétés par l'auteur.
Les « types de Bases de Données » correspondent aux usages les plus fréquents des SGBD comparés :
- Analytiques : SGBD optimisés 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.

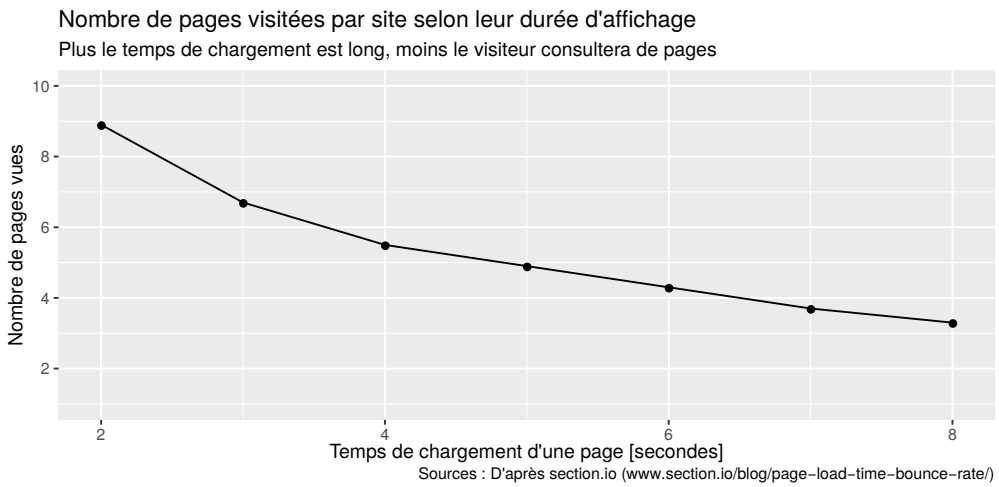


FIGURE 5.10 – Impact du temps de chargement d'un site web sur sa consultation. D'après ELLIOTT (2017).

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 [43], 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 [44]. 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 [45], et un ensemble de problématiques d'autres part [46]. Ce MCD décrit donc les « tables », leur composition (attributs) et les liens entre 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

43. 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.

44. 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.

45. 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.

46. 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.

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 diversité possible 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 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.

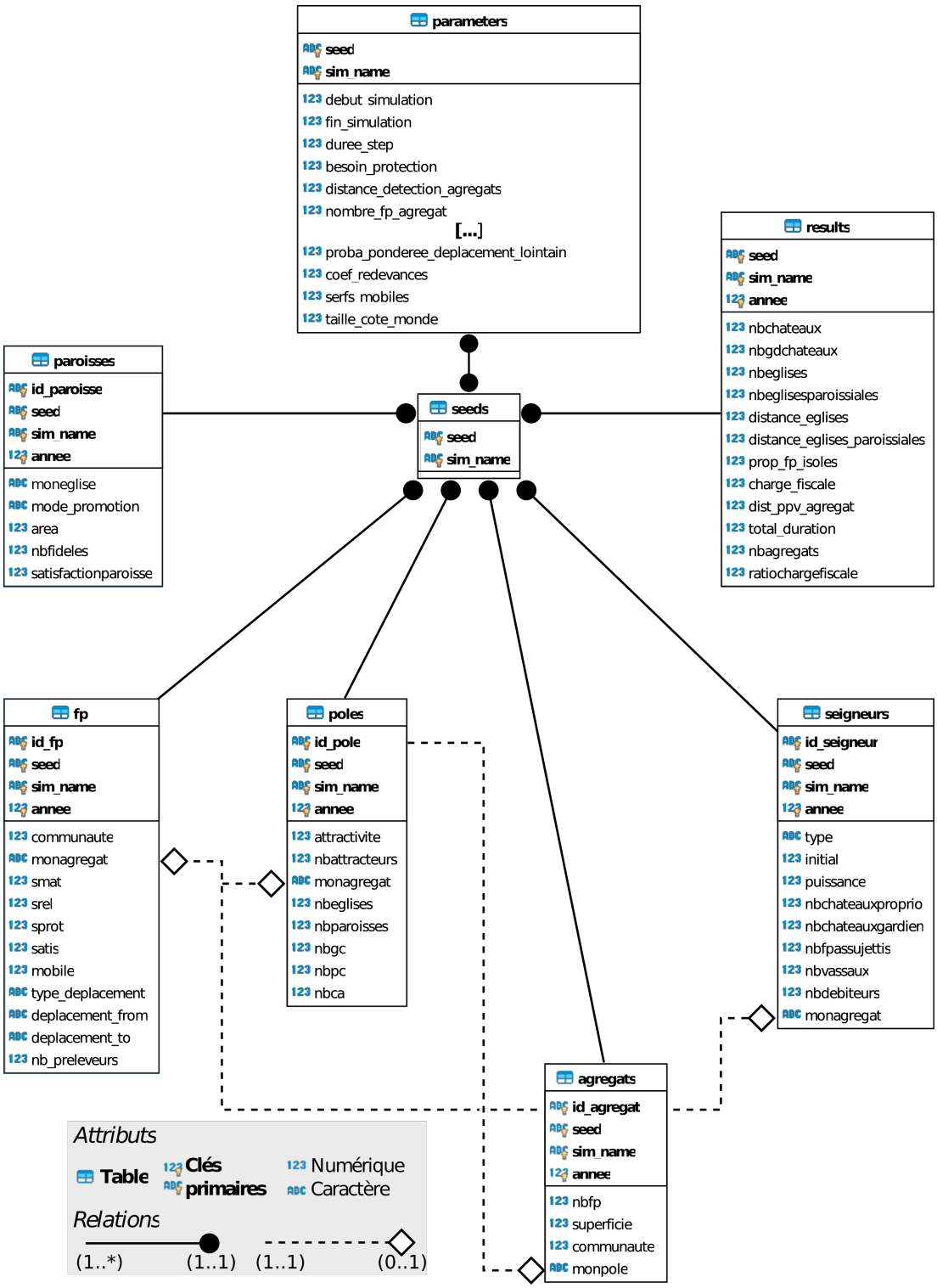


FIGURE 5.11 – Modèle Conceptuel de Données (MCD) des données en sortie de simulation de SimFeodal telles qu'implémentées dans SimEDB.

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`)[47].

47. 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. Notons de plus que dans le MCD de SimEDB, la graine aléatoire est systématiquement associée

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 bidirectionnelle 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.

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'agré-

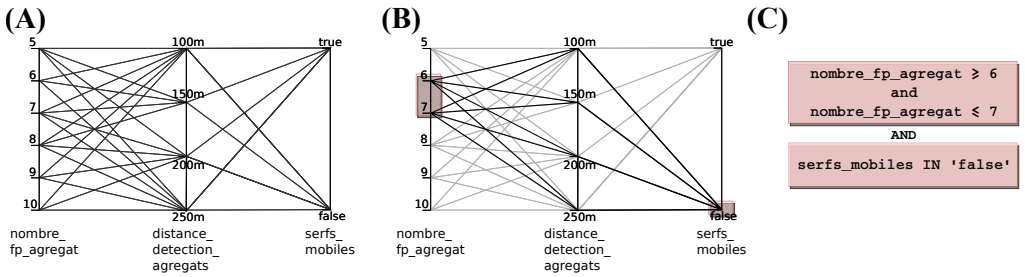
au nom de l'expérience. Même en menant un million de réplications, la probabilité que deux simulations partagent la même graine aléatoire serait largement inférieure à 1%. La graine aléatoire constitue donc un identifiant unique robuste dans notre cas.

gation 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.

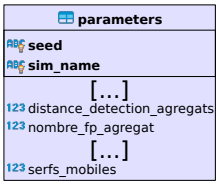


- Depuis un ensemble d'expériences aux valeurs de paramètres variées **(A)**,

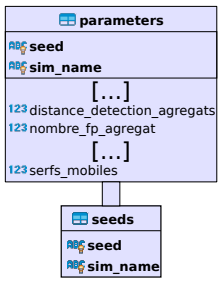
- l'utilisateur mène une sélection graphique en effectuant un brushing des valeurs voulues **(B)**.

- Cette sélection graphique est alors formalisée dans le langage de requête SQL **(C)**.

(D)



(E)



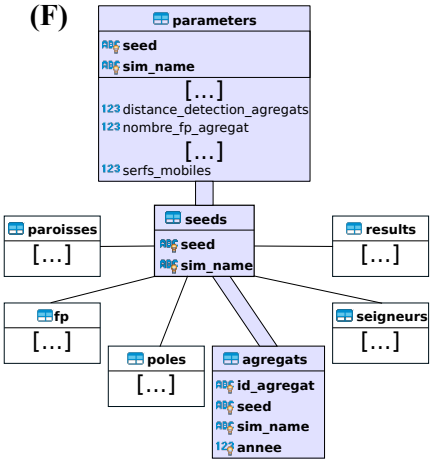
- Cette requête est exécutée sur la table *parameters*.

- On isole alors les attributs *seed* et *sim_name* de cette table **(D)**.

- Ce filtrage est répercuté sur la table des *seeds*, qui ne comprend plus à ce moment que le sous-ensemble des *seed* et *sim_name* correspondant à la requête de l'utilisateur **(E)**.

- Le filtrage effectué sur la table *seeds* peut alors être répercuté sur les différentes tables liées aux agents.

(F)



- Ici, on utilise cette sous-sélection sur la table des *agregats* **(F)**,

- ce qui permet par exemple de compter de nombre moyen d'agrégats à chaque pas de temps et de produire l'indicateur de sortie de simulation souhaité **(G)**.

- Pour produire n'importe quel autre indicateur depuis cette même sélection, il suffira de reporter le filtrage de la table *seeds* dans les tables de chaque type d'agents.

(G)

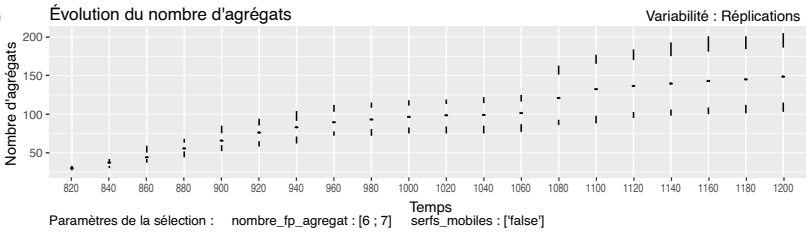


FIGURE 5.12 – De la sélection interactive à l'indicateur de sortie.

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 [48], 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*).

48. **Sim**Feodal **E**xploration **D**ash**B**oard, voir la note de bas de page 26, section 5.2.6.

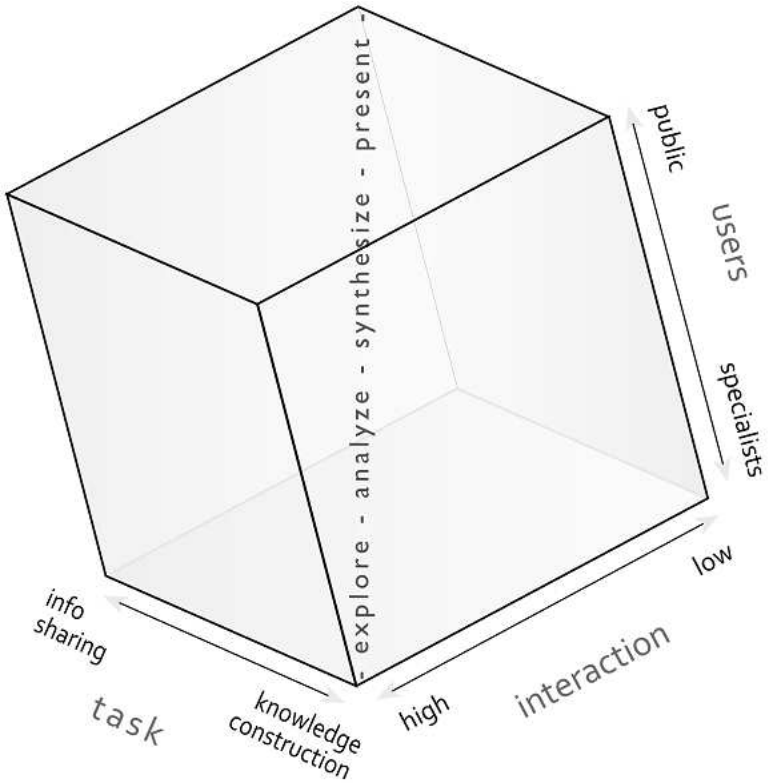


FIGURE 5.13 – « *An update to Cartography³, 10 years after its conception* », par ÇÖLTEKIN, JANETZKO et FABRIKANT 2018, d'après MACEACHREN et al. 2004, p. 10.

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).

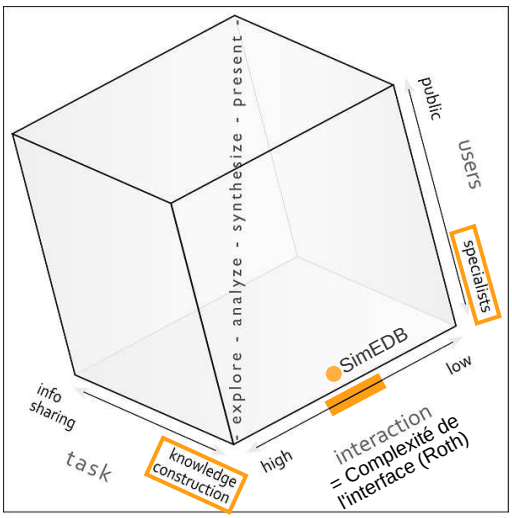


FIGURE 5.14 – Positionnement de SimEDB dans le cube *Cartography*[3] de MACEACHREN.

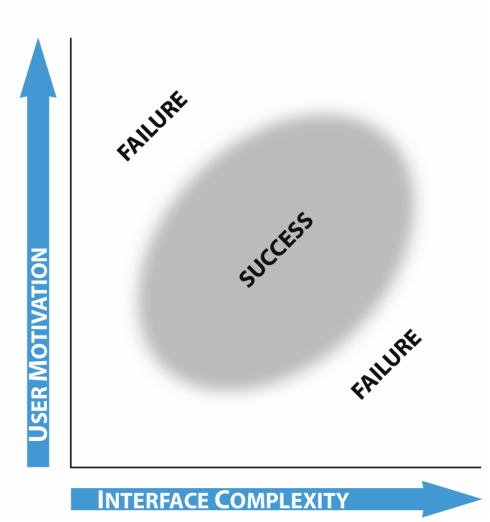


FIGURE 5.15 – « *Interface complexity versus user motivation.* », ROTH 2013, p. 79.

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 plutôt 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 fonc-

tionnalités 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 sur la personnalisation et l'adaptation d'une solution existante, nécessairement générique et donc difficile à spécialiser.

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 [49] (*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 » (ibid., p. 9).

FORCH et al. 2017, pour leur part, étudient la perception du délai de réponse

49. 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.

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 [50].

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 largement de prêter attention à la rapidité de rendu et à son optimisation :

> « 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.

De manière plus précise, on retrouve une quantification, en termes d'ordres de grandeur, des délais acceptables dans un processus d'exploration de données : « However, due to human cognitive constraints, exploration needs highly responsive system response times […] : at 500 ms, users change their querying behavior ; past five or ten seconds, users abandon tasks or lose attention » (FEKETE et al. 2019, p. 2). Ces auteurs font référence à un billet de blog de Jakob NIELSEN qui donne une justification plus poussée en matière de capacité de concentration.

> « When the computer takes more than 0.1 second but less than 1 second to respond to your input, **it feels like the computer is causing the result** to appear. Although users notice the short delay, they **stay focused on their current train of thought** during the one-second interval.
> […]
> After 1 second, users get impatient and notice that they're waiting for a slow computer to respond. The longer the wait, the more this impatience grows ; after about 10 seconds, the average attention span is maxed out. At that point, the user's mind starts wandering and doesn't retain enough information in short-term memory to easily resume the interaction once the computer finally loads the next screen.
>
> More than 10 seconds, and you **break the flow**. Users will often **leave the site** rather than trying to regain the groove once they've started thinking about other things. »
>
> NIELSEN 2009.

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

50. 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.

viser des temps de latence maximale de 10 secondes, 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 [51]. Ce choix a des nombreuses répercussions, aussi bien en matière de possibilité d'accès que de facilité à faire évoluer la plate-forme.

Jusqu'à récemment, le choix le plus classique était 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).

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

51. 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.

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 (« Stockage distribué ou centralisé », 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 [52] 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 vis-à-vis des 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 vis-à-vis du 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).

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

52. 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.

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 plateforme 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 vis-à-vis des 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 [53], mais bien de justifier et présenter les choix relatifs aux technologies employées, en restant à un niveau assez général [54]. 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).

53. 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

54. 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 [55] 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 [56], 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` [57] 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. [58] 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`,

55. Applications en ~~Flash~~, *applets* Java…

56. 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.

57. http ://getbootstrap.com/

58. 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.

ou encore sur le langage Python tels que `Django` ou `Flask`), est extrêmement flexible et performante, au prix d'un développement important. Un *framework* fournit en effet des « briques » logicielles de base – les composants –, très génériques. Ces composants de bases demandent donc une forte personnalisation et un agencement complexe afin d'arriver au résultat souhaité. La communication entre ces composants doit être entièrement prévue et implémentée, et on abouti donc nécessairement sur des projets assez importants, qui demandent une réelle expertise en développement et portent le risque d'être trop complexes pour être facilement adaptés et donc rendus génériques.

À l'autre bout du gradient de développement, on peut aussi choisir de bâtir une application à partir d'un ensemble logiciel intégré, comme `Tableau`, qui permet d'agencer visuellement et graphiquement des composants graphiques et leurs liens. Ces outils, très usités en informatique décisionnelle, sont extrêmement simples à prendre en main, y compris pour des « utilisateurs finaux » – analystes par exemple –. En contre-partie, ils sont moins personnalisables et configurables que des solutions plus bas niveau comme les *frameworks*, et ce sont majoritairement des logiciels propriétaires, donc non modifiables.

Entre ces deux extrêmes, quelques *frameworks* intermédiaires, souvent originaires des outils de manipulation de données plus que du monde de l'informatique décisionnelle, mettent à disposition de l'utilisateur des composants de plus haut-niveau que les « briques élémentaires ». L'interaction entre les composants y est déjà pré-conçue, tout en reposant sur une construction « depuis zéro », donc personnalisables et adaptables.

Généralement, chaque *framework* est associé à un langage de programmation (voir tableau 5.2) : le *framework* Shiny s'appuie sur le langage R, Dash sur le langage Python et Escher sur le langage Julia.

Framework	**Shiny**	**Dash**	**Bokeh**	**Escher**	**Observable**
Référence	CHANG et al. 2015	PLOTLY 2017	BOKEH 2014	GOWDA 2018, d'après BEZANSON et al. 2014	BOSTOCK 2018
Langage	R	Python	Python	Julia	JavaScript / D3.js
Maturité	⊕⊕ (2012)	⊖ (2017)	⊕ (2014)	⇄ (2016)	⊕ (2017) (D3 : 2011)
Communauté	⊕⊕	⇄	⊕	⊖	⊕
Connaissance personnelle	⊕⊕	⊖	⇄	⊖	⊖

TABLEAU 5.2 – Une sélection de différents *frameworks* dédiés à la création d'interfaces d'exploration de données.

Le choix de tel ou tel framework dépend certes de la maturité de chaque projet – Shiny est à ce titre très en avance –, mais surtout du langage informatique que le concepteur de l'application souhaite utiliser. Dans le cas de SimEDB, le créateur de la plate-forme est adepte du langage R (voir COMMENGES et al. 2014) et pratique le *framework* Shiny depuis plusieurs années (voir CURA 2015) : le choix d'utiliser ce *framework*, au sein d'un environnement logiciel

basé sur le langage R, était donc assez évident.

Manipuler les données avec R et dplyr. Les langages de programmation, et en particulier les plus utilisés en analyse de données, reposent souvent sur une architecture logicielle modulaire. Le langage constitue un cœur, autour duquel des bibliothèques logicielles (des *packages* en R) viennent ajouter des fonctionnalités. Parmi ces bibliothèques logicielles, en Python comme en R, certaines sont entièrement dédiées à la manipulation de données tabulaires – on parle alors de « Data Manipulation Language » (DML) – et permettent d'effectuer des traitements avec des approches fonctionnelles, plutôt qu'avec les structures impératives plus fréquemment utilisées en programmation. En R, ces *packages* constituent de véritables écosystèmes, dotés de leur propre DSL (voir p. 65) et donc d'une grammaire de manipulation de données propre.

L'un de ces *packages*, dplyr (WICKHAM et al. 2015), s'inscrit dans un écosystème dénommé tidyverse (WICKHAM 2017), et permet ainsi de chaîner des opérations de manipulation de données en une chaîne de traitement complète, plutôt que de faire appel aux habituelles boucles de parcours de matrices propres aux langages de programmation classiques. Ce faisant, avec des opérations chaînées, qui reposent sur des « verbes » permettant d'effectuer des traitements de restructurations, de modification, de filtrage ou d'enrichissement d'une donnée tabulaire [59], on obtient un ensemble d'instructions qui forment une « phrase » de manipulation de données, exprimées donc dans la « grammaire de traitement de données » fournie par dplyr.

Cette « grammaire » s'inspire notamment du SQL, bien que beaucoup plus complète, et peut en particulier être « convertie » en SQL (figure 5.16), c'est-à-dire qu'une suite d'instructions exprimées via dplyr en R (figure 5.16a) peut être traduite en SQL (figure 5.16b), et donc envoyée et exécutée sur un SGBD.

En matière de performance, l'approche de dplyr est intéressante : toutes les opérations sont effectuées par le SGBD directement, et seul le résultat final est renvoyé à R (instruction `collect()`). Le traitement de données bénéficie donc de la rapidité d'exécution du SGBD MapD, tout en profitant de la syntaxe expressive de dplyr. De plus, cela permet de minimiser les transferts de données : en exécutant les calculs dans le SGBD, il n'est besoin que d'en renvoyer le résultat à l'utilisateur. Et ce résultat est nécessairement moins lourd que les données dont il provient. On optimise ainsi l'utilisation de bande-passante internet.

59. Les fonctions de base sont donc des « verbes », au sens où elles définissent les opérations qui seront effectuées sur les données. On peut ainsi isoler des colonnes avec le « verbe » `select`, filtrer les lignes avec `filter`, modifier une colonne avec `mutate` etc. La figure 5.16a en donne un exemple commenté et concret.

```r
NombreAgregatParAnnee <- tbl(conMapD, "agregats") %>% # Connexion à la table agregats de la BDD
  filter(sim_name == "5_0") %>% # Filtre de la table en ne conservant que les experiences "5_0"
  group_by(sim_name, seed, annee) %>% # Agrégation sur les identifiants de simulation (seed et sim_name) et par annee
  summarise(NbAgregats = n()) %>% # Calcul du nombre total de lignes pour chaque agrégation
  arrange(sim_name, seed, annee) %>% # Tri de la table selon les trois variables
  collect() # Récupération du résultat de la requête en mémoire
```

(a) Code source R avec le *package* dplyr

```sql
SELECT "sim_name", "seed", "annee", COUNT() AS "NbAgregats"
FROM "agregats"
WHERE ("sim_name" = '5_0')
GROUP BY "sim_name", "seed", "annee"
ORDER BY "sim_name", "seed", "annee"
```

(b) Traduction du code source dplyr en SQL

FIGURE 5.16 – Un exemple de manipulation de données stockées dans un SGBD depuis R. On y interroge la table des agrégats de population pour calculer le nombre moyen d'agrégats par année de simulation.

Création de graphiques avec ggplot2 et la « *grammar of graphics* ». En interrogeant le SGBD avec des outils adaptés, on obtient un jeu de données qui servira de base à la représentation graphique des indicateurs (figure 5.17, étape 1). On peut passer alors passer à l'étape de construction graphique des indicateurs. Il existe pour cela, toujours en restant de l'environnement (et du langage) R, de nombreux *packages* dédiés.

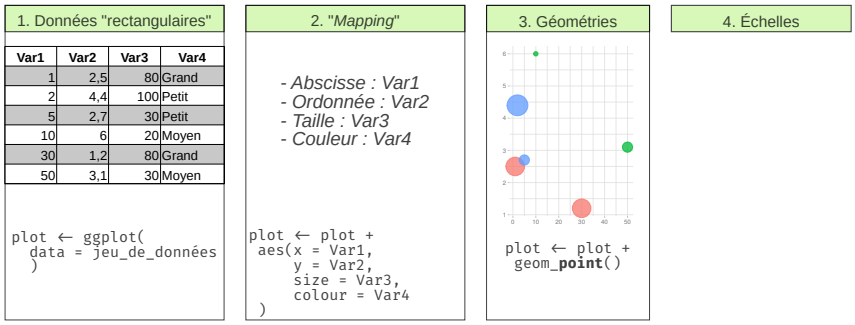


FIGURE 5.17 – Représentation des éléments de grammaire de ggplot2, d'après une idée de HEALY (2018).

L'un des *packages* les plus utilisés, ggplot2 (WICKHAM 2016), met en œuvre une syntaxe assez adaptée à nos contraintes : ce *package* est conceptuellement fondé sur la « *grammar of graphics* », c'est-à-dire une vision modulaire et très structurée de la conception graphique, pensée par Leland Wilkison (WILKINSON 2006). La logique, assez familière pour un utilisateur de Systèmes d'Information Géographique (SIG), consiste à penser une représentation graphique comme un ensemble de couches (*layers*), qui se superposent, se complètent, et sont toutes basées sur une source de données. Les différentes composantes des données (variables par exemple) sont associées à des composantes graphiques de base (abscisse, ordonnée, taille, couleur …), formant ainsi une mise en correspondance (*mapping*) des données avec les composants graphiques (voir figure 5.17, étape 2).

Dans notre cas, cette grammaire est porteuse d'un avantage majeur. Elle est extrêmement structurée et modulaire. Cela permet de ré-utiliser largement les codes-sources écrits pour un indicateur et de les adapter aisément à d'autres

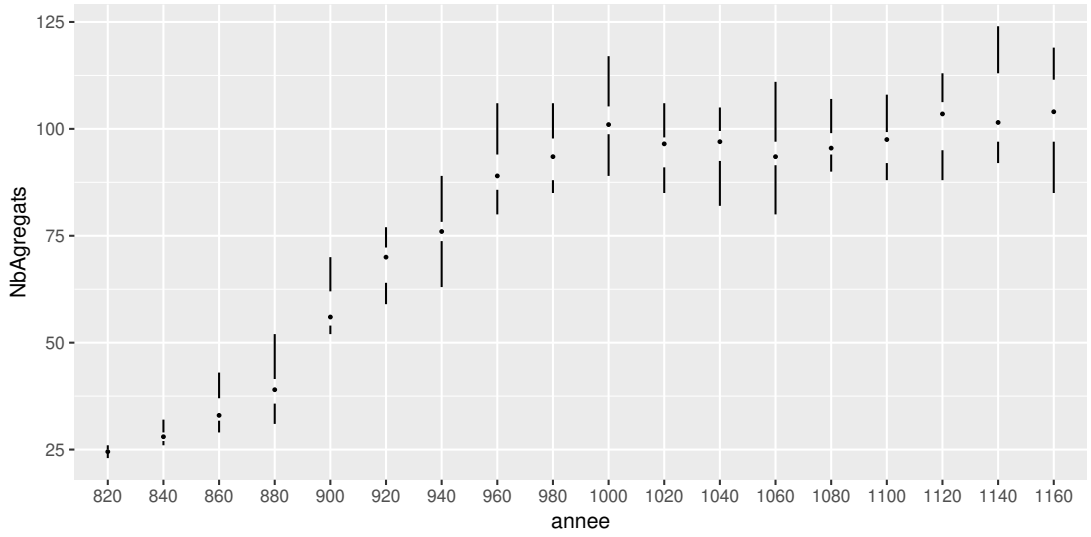
indicateurs. Si la grammaire du graphique est bien définie, elle sera ainsi très indépendante du contenu des données que l'on y insère.

Par exemple, de nombreux indicateurs de sortie de SimFeodal décrivent l'évolution du nombre d'agents au cours des années de simulation (les agrégats dans la figure 5.18). Ce type de graphique est d'une part rapide à produire avec ggplot2 : il ne requiert que quelques lignes de code (figure 5.18a). D'autre part, en changeant le tableau de données en entrée (créé dans figure 5.16a), on reproduit exactement le même type de graphique pour, par exemple, un autre type d'agent (le nombre de foyers paysans, d'églises…).

Le *package* ggplot2 répond tout à fait aux contraintes de modularité exposées plus haut, et permet de factoriser le code-source, ce qui garantit une maintenance plus rapide et une meilleure robustesse de l'application dans son ensemble.

```r
ggplot(NombreAgregatParAnnee) + # Création d'un graphique avec le jeu de données NombreAgregatParAnnee
  aes(x = annee, y = NbAgregats) + # "Mapping" des valeurs : champ "annee" en x, champ "NbAgregats" en y
  geom_tufteboxplot() # Ajout d'une couche graphique de type boxplot minimaliste (Tufte boxplot)
```

(a) Code source R avec le *package* ggplot2



(b) Graphique généré

FIGURE 5.18 – Un exemple de manipulation de données stockées dans un SGBD depuis R.

Fluidifier les étapes de rendu : le « pipeline de visualisation ». DOS SANTOS et BRODLIE (2004) ont conceptualisé et schématisé l'ensemble des étapes nécessaires à la construction d'une visualisation, depuis les données brutes jusqu'à l'image finale, au sein d'un « *pipeline* » de la visualisation (figure 5.19). Ils y décrivent les différents états des données en entrée et en sortie (ligne supérieure), ainsi que les traitements que ces données subissent (ligne inférieure).

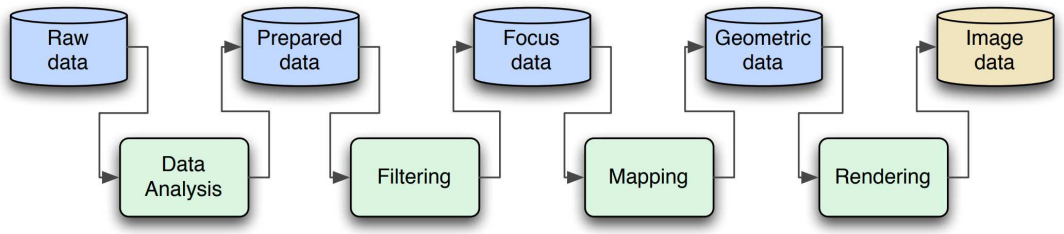


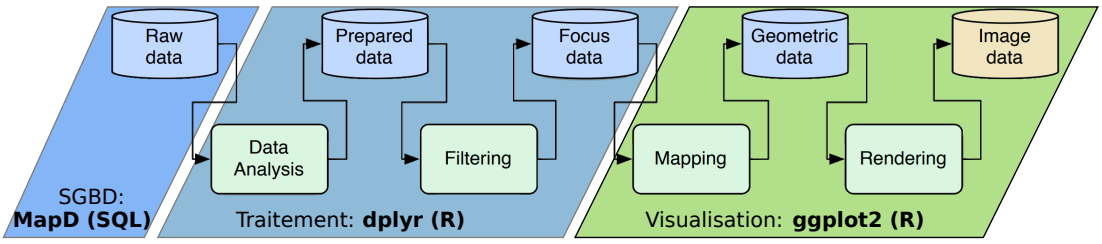
FIGURE 5.19 – « *The Visualisation Pipeline* », de KEIM et al. (2010), p.92, d'après DOS SANTOS et BRODLIE (2004), p. 314.

Ce *pipeline* débute par des données brutes (*raw data*) auxquelles ont fait subir un traitement (*data analysis*, par exemple une agrégation) pour obtenir des données prêtes à l'utilisation (*prepared data*). Il s'agit ensuite de filtrer ces données (choix des expériences à conserver par exemple), dont l'on conserve donc uniquement les éléments nécessaires (*focus data*). Par une étape de mise en correspondance des variables et des primitives graphiques (*mapping*, voir paragraphe précédent), on obtient un jeu de données « géométriques » (*geometric data*). Cette « géométrie » est à entendre au sens de l'espace de la représentation graphique, qui comprend par exemple les coordonnées des points, lignes, la couleur des cercles et autres éléments mobilisés dans la construction d'un graphique. Il n'est donc aucunement question ici de données géographiques ou spatiales. La dernière étape est plus technique : il s'agit du « rendu graphique » (*rendering*), qui convertit un ensemble de spécifications géométriques (textuelles) en une image affichable, faite de pixels (*image data*).

Dans la chaîne de traitement la plus classique, ces étapes s'effectuent au sein de différents logiciels, chacun dédiés à une tâche. Dans le domaine des utilisateurs de SIG, on retrouve par exemple fréquemment une préparation des données dans un tableur, un import dans un logiciel SIG qui va être chargé de la cartographie, puis un export vers un logiciel de dessin vectoriel afin de réaliser la mise en page. À chaque changement de logiciel, il est nécessaire d'exporter les données produites, puis de les ré-importer dans le logiciel suivant.

A contrario, le propre de l'utilisation d'un langage de programmation plutôt que d'un outil graphique est de pouvoir automatiser et intégrer l'ensemble de ces étapes. L'utilisation de R comme langage de développement de SimEDB nous permet ainsi de développer une unique chaîne de traitement, qui ne requiert aucun import/export de données, et peut donc être consolidée, vérifiée et surtout ré-employée *ad libitum*.

L'enchaînement des *packages* employées dans SimEDB est présenté dans la figure 5.20a, et le code-source correspondant à l'exemple développé dans cette sous-partie dans la figure 5.20.



(a) Technologies utilisées dans SimEDB.

```
tbl(conMapD, "agregats") %>% # Connexion à la table agregats de la BDD
  filter(sim_name == "5_0") %>% # Filtre de la table en ne conservant que les experiences "5_0"
  group_by(sim_name, seed, annee) %>% # Agrégation sur les identifiants de simulation (seed et sim_name) et par annee
  summarise(NbAgregats = n()) %>% # Calcul du nombre total de lignes pour chaque agrégation
  arrange(sim_name, seed, annee) %>% # Tri de la table selon les trois variables
  collect() %>% # Récupération du résultat de la requête en mémoire
  mutate(annee = factor(annee)) %>% # Conversion du champ "annee" de numérique à facteur
  ggplot() + # Création d'un graphique avec le jeu de données créé jusque-là
  aes(x = annee, y = NbAgregats) + # "Mapping" des valeurs : champ "annee" en x, champ "NbAgregats" en y
  geom_tufteboxplot() # Ajout d'une couche graphique de type boxplot minimaliste (Tufte boxplot)
```

(b) Implémentation d'un exemple de pipeline de visualisation pour construire un indicateur dans SimEDB

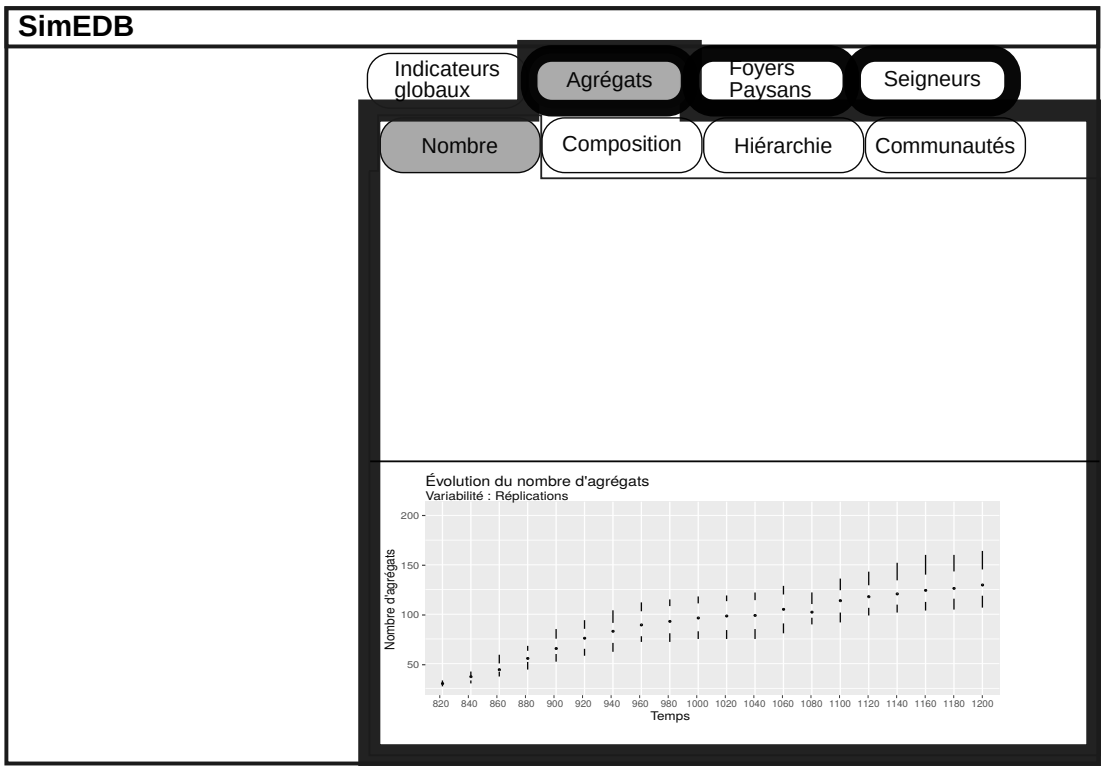
FIGURE 5.20 – Le « pipeline » de visualisation et son implémentation dans SimEDB. Cette implémentation est obtenue en assemblant les codes des figures 5.16a et 5.18a.

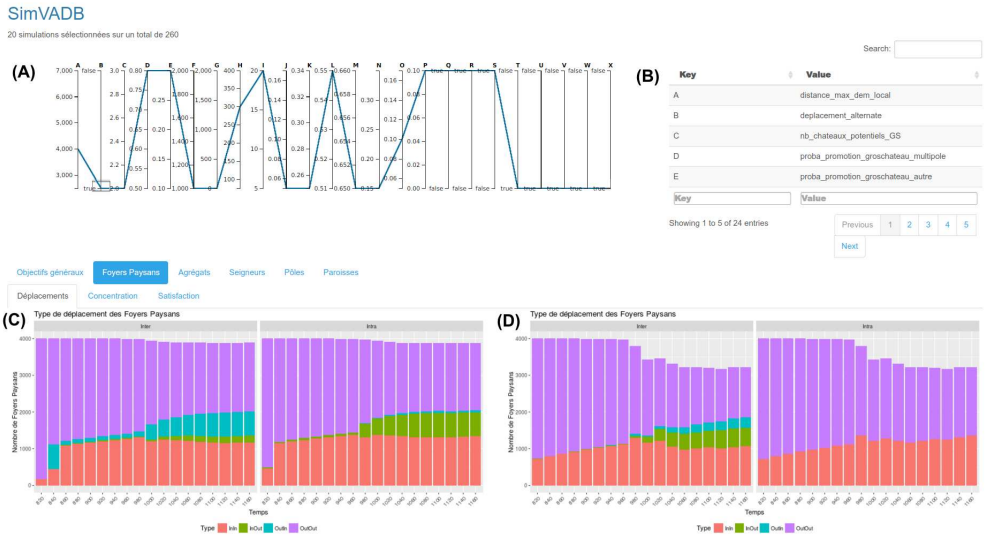
Modulariser les fonctions. Shiny, en tant qu'outil de création d'interface graphique, bénéficie aussi d'un avantage important en matière de conception d'application web : comme ce *package* est basé sur un langage de programmation modulaire, on peut logiquement créer et ré-utiliser des « briques d'interfaces » modulaires. Par l'utilisation de modules [60], il est possible de définir un ensemble d'éléments graphiques adaptatifs et de ré-utiliser tel quel cet ensemble.

Dans l'interface de SimEDB (figure 5.21), par exemple, les indicateurs graphiques sont toujours présentés de la même manière (encadrés oranges) : dans la partie de droite relative aux indicateurs, l'indicateur à proprement parler est à gauche, et des outils de téléchargement (vectoriel et image) et de notation de l'indicateur (les étoiles) sont placés en haut à droite.

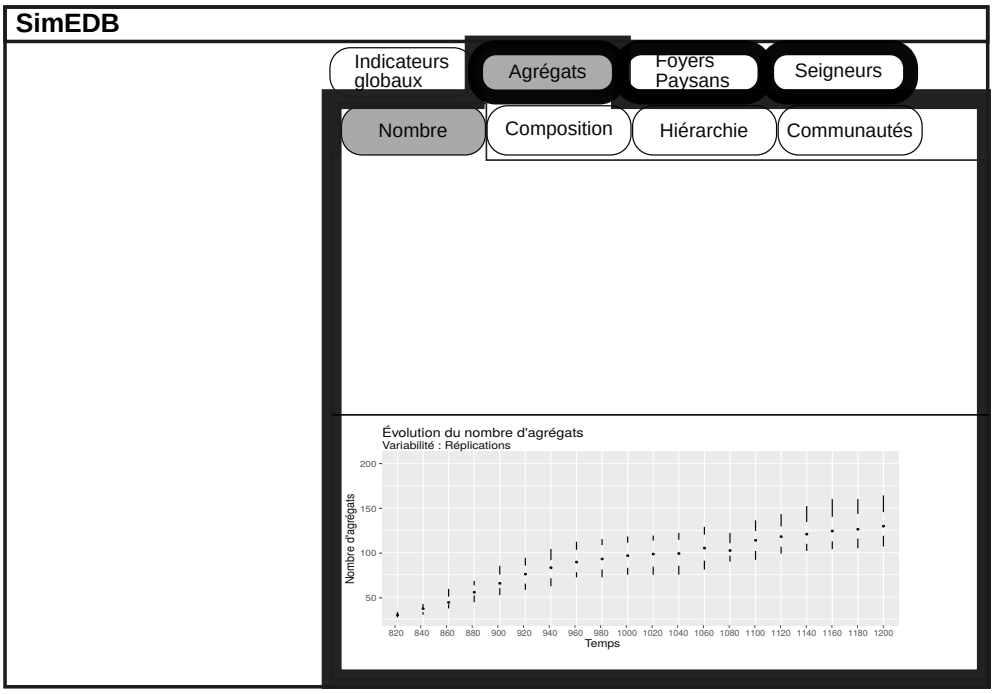
En termes de code-source, la manière de produire les deux indicateurs comparés dans la figure 5.21 est strictement identique : c'est une fonction générique qui prend en entrée des données et un type de graphique à produire. Dans la figure, seul un paramètre varie : le filtre appliqué aux données, qui renvoie ici à différentes expériences. Cela permet donc d'une part de minimiser la taille du code, mais surtout, avec la généricité apportée, de faciliter de manière considérable l'ajout ou la modification d'indicateurs.

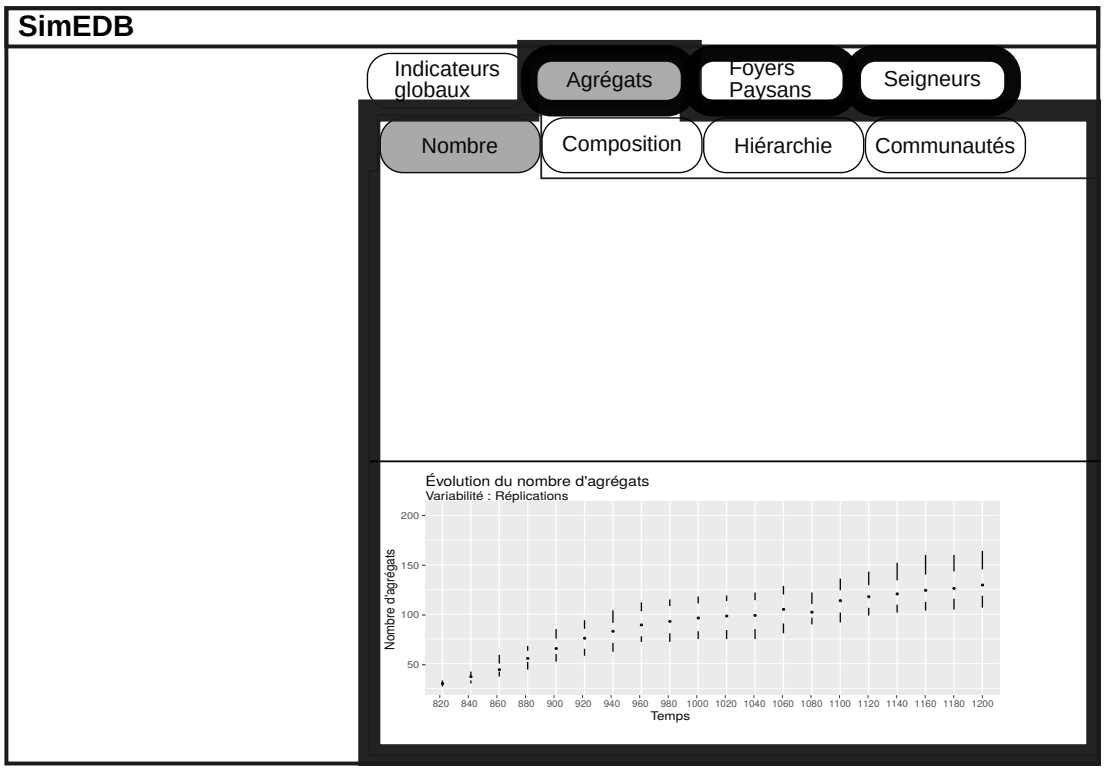
60. https ://shiny.rstudio.com/articles/modules.html


SimEDB
Indicateurs globaux
Agrégats
Foyers Paysans
Seigneurs
Nombre
Composition
Hiérarchie
Communautés
Évolution du nombre d'agrégats
Variabilité : Réplications
Nombre d'agrégats
200
150
100
50
820 840 860 880 900 920 940 960 980 1000 1020 1040 1060 1080 1100 1120 1140 1160 1180 1200
Temps


SimVADB
20 simulations sélectionnées sur un total de 260
(A)
(B)
Search:
Key
Value
A
distance_max_dem_local
B
deplacement_alternate
C
nb_chateaux_potentiels_GS
D
proba_promotion_groschateau_multipole
E
proba_promotion_groschateau_autre
Showing 1 to 5 of 24 entries
Previous
Next
Objectifs généraux
Foyers Paysans
Agrégats
Seigneurs
Pôles
Paroisses
Déplacements
Concentration
Satisfaction
(C)
Type de déplacement des Foyers Paysans
(D)
Type de déplacement des Foyers Paysans
Nombre de Foyers Paysans
Temps

(a) SimVADB


SimEDB
Indicateurs globaux
Agrégats
Foyers Paysans
Seigneurs
Nombre
Composition
Hiérarchie
Communautés
Évolution du nombre d'agrégats
Variabilité : Réplications
Nombre d'agrégats
Temps


SimEDB
Indicateurs globaux
Agrégats
Foyers Paysans
Seigneurs
Nombre
Composition
Hiérarchie
Communautés
Évolution du nombre d'agrégats
Variabilité : Réplications
Nombre d'agrégats
200
150
100
50
820 840 860 880 900 920 940 960 980 1000 1020 1040 1060 1080 1100 1120 1140 1160 1180 1200
Temps

Avec un axe « fixe », il est donc opportun de mener la comparaison visuelle sur cet axe, et donc d'aligner les graphiques sur celui-ci. L'organisation des différents indicateurs est donc verticale plutôt qu'horizontale. Afin que la sélection des simulations à explorer soit intuitive, les contrôleurs doivent être alignés aux indicateurs, et dès lors, verticalisés eux aussi. Pour bien différencier visuellement ce qui relève d'un affichage et ce qui requiert une interaction, les contrôleurs s'inscrivent dans un panneau dédié, grisé (figure 5.23), ce qui constitue presque un standard dans les interfaces modernes d'applications interactives.

Onglets et sous-onglets. Comme dans SimVADB (figure 5.22a), on a choisi de conserver une navigation entre indicateurs par un systèmes d'onglets imbriqués : un premier niveau d'onglets permet d'accéder au type d'agents concernés par les indicateurs, et un second niveau permet de sélectionner spécifiquement l'indicateur choisi [61].

En terme de disposition, cela force l'utilisateur à interagir avec l'application régulièrement puisque chaque indicateur doit être sur un onglet dédié. La majorité des utilisateurs potentiels de SimEDB consultent toutefois l'application sur des ordinateurs portables, dotés d'écran réduits et d'une résolution faible. L'encombrement visuel est alors atteint rapidement, et mieux vaut présenter un indicateur à la fois plutôt que de présenter l'ensemble des indicateurs sur une unique page : la démarche d'étude visuelle sera plus longue, mais ne sera pas gênée ou faussée par des graphiques de dimension trop réduites qui peuvent induire des erreurs de lecture.

L'organisation des onglets en eux-mêmes pose aussi une question importante : vaut-il mieux organiser la consultation par type d'agent, ou plutôt, hiérarchiquement, selon la catégorie de processus examinée (à corriger une fois fixé sur le terme, faire ref au chap3), par exemple en respectant l'ordre de consultation des indicateurs déterminé ?

Les deux approches présentent des avantages, mais nous avons choisi de rendre l'utilisation de SimEDB plus intuitive à tous, c'est-à-dire en organisant les indicateurs par type d'agents, plutôt qu'efficace, pour les utilisateurs habitués qui auraient bénéficié d'une organisation structurée hiérarchiquement.

5.4.2.3 Choix des modes d'interactions

Avant même la conception de SimEDB, avec la plate-forme SimVADB, nous avions décidé de baser la sélection des simulations sur des graphiques en coordonnées parallèles interactifs (section 5.2.5 : « Interagir avec les rapports : exploration interactive »). La logique d'ensemble du filtrage de simulations restant la même, il n'était pas nécessaire de modifier ce choix pour SimEDB.

L'accumulation d'expériences, reposant sur les variations de paramètres diffé-

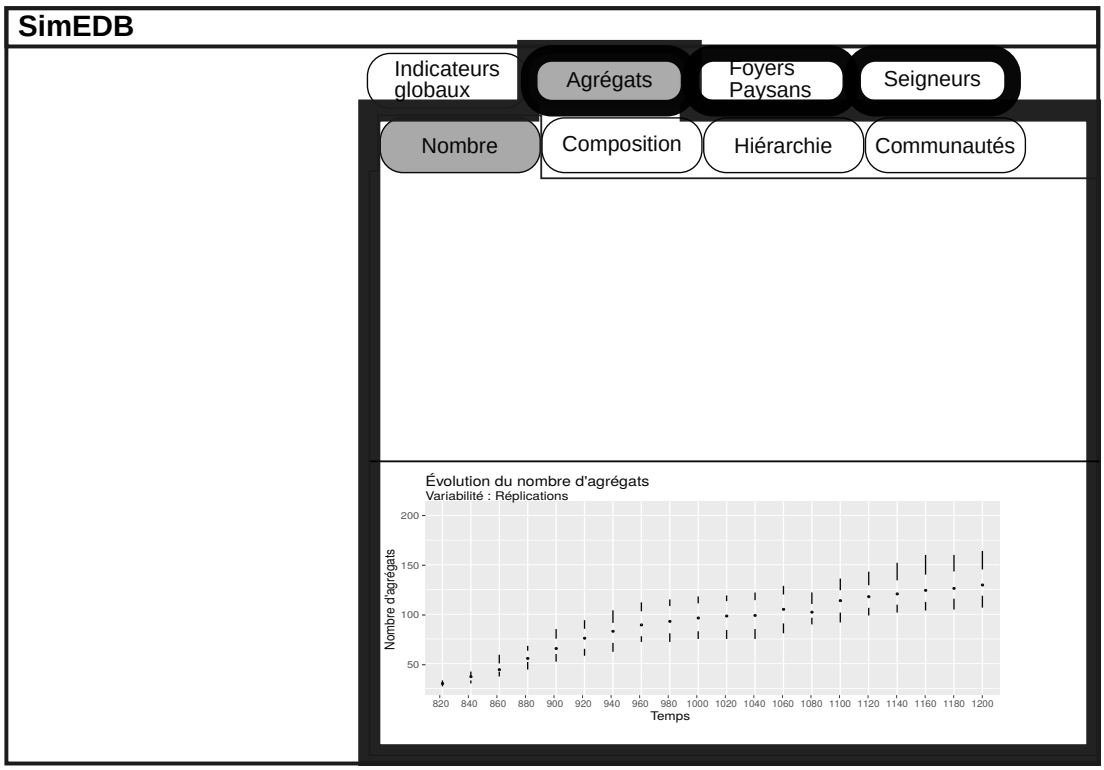
61. Notons que cette question revêt une importance réelle en matière d'ergonomie de l'application, mais que l'aspect technique en est pourtant assez simple. Pour changer le mode d'organisation des onglets et sous-onglets, il suffirait de ré-organiser les appels aux composants dans le code-source de l'application.

rents, ainsi que la démultiplication des paramètres du modèle SimFeodal ayant accompagné son paramétrage, ont pourtant demandé de reconsidérer l'usage de ces graphiques interactifs. Là où seuls quelques paramètres étaient mobilisés auparavant, les graphiques en coordonnées parallèles reposaient sur peu d'axes. Avec l'augmentation du nombre d'axes, le graphique en coordonnées parallèle est rapidement devenu illisibleen raison de la surcharge graphique due au recouvrement des axes.

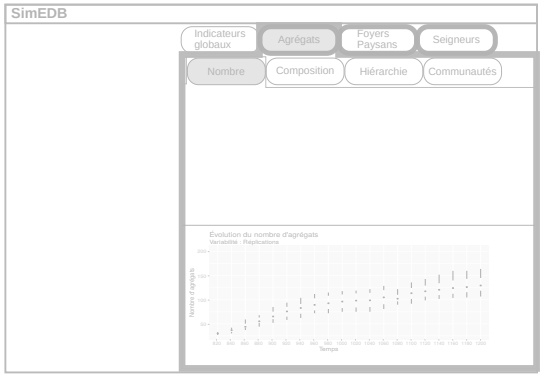
Réduire la surcharge visuelle des graphiques en coordonnées parallèles. La première mesure pour y remédier a été de filtrer les paramètres affichés : nul besoin d'afficher un axe correspondant à un paramètre qui n'est jamais manipulé dans les expériences. Plutôt que de définir les paramètres « utiles », et donc d'avoir à les redéfinir dans l'application à chaque ajout d'expérience qui reposerait sur la variation d'un paramètre différent, nous avons fait en sorte que cette discrimination des paramètres « actifs » soit exécutée de manière automatique : quand SimEDB est lancé, une requête est exécutée sur la table des paramètres pour identifier ceux qui présentent plusieurs modalités et ceux qui n'en ont qu'une. Seuls sont alors affichés les paramètres de la première catégorie, car eux-seuls présent un intérêt à être discriminés.

Ce faisant, le nombre de paramètre affichés est réduit, et permet d'afficher leurs intitulés plutôt que de faire appel à une table de correspondance comme dans SimVADB (figure 5.22a, partie **(B)**). L'automatisation de ce traitement permet de plus de ne pas avoir à changer quoi que ce soit à la plate-forme lors d'ajouts ou de suppressions de simulations de la base de données, ce qui concoure à l'objectif d'indépendance aux données de la plate-forme d'exploration.

Pré-filtrer les simulations. Au fur et à mesure du paramétrage puis de la calibration de SimFeodal, les expériences ont tout de même continué à mobiliser de plus en plus de paramètres différents. Pour réduire la quantité d'information représentée et améliorer en conséquence « l'expérience utilisateur », nous avons ajouté un filtre, moins visuel que les graphiques en coordonnées parallèle, qui permet toutefois de restreindre le nombre de simulations affichées à partir de leur dénomination. Plutôt que de cibler des valeurs spécifiques de paramètres, l'idée est donc de soustraire des choix possibles des expériences entières. Pour SimEDB, on a donc ajouté un pré-filtrage, sous forme de « boîte de sélection » (*select input*, figure 5.24), qui interroge la base de données directement pour connaître les différents intitulés de simulations et agit comme un premier filtre réduisant donc les simulations interrogées dans les graphiques en coordonnées parallèles.


SimEDB
Indicateurs globaux
Agrégats
Foyers Paysans
Seigneurs
Nombre
Composition
Hiérarchie
Communautés
Évolution du nombre d'agrégats
Variabilité : Réplications
Nombre d'agrégats
200
150
100
50
820 840 860 880 900 920 940 960 980 1000 1020 1040 1060 1080 1100 1120 1140 1160 1180 1200
Temps

et donc la taille de ces derniers.



Noter les simulations. Un dernier point d'interaction avec l'application a été prévu, sans pouvoir toutefois être mobilisé jusque là : il s'agissait d'aller vers une semi-automatisation de l'évaluation des simulations, par l'intermédiaire d'un outil graphique permettant de « noter » les simulations sélectionnées. Pour ce faire, et parce que, on l'a vu, l'évaluation d'un ensemble de simulations ne peut se faire de manière unique, on a choisi de donner la possibilité aux utilisateurs experts de noter chacun des indicateurs de sortie, pour chacun des ensembles de simulations qu'ils exploreraient. L'évaluation se fait au moyen d'un outil simple, composé de 5 « étoiles », et est enregistré à chaque nouvelle note.

Une piste d'utilisation serait de mobiliser les données ainsi créées, composées d'une note donnée à un indicateur pour un ensemble d'identifiants uniques de simulations, afin de réaliser des analyses quantitatives des notes attribuées : est-ce que certaines simulations sont systématiquement bien notées avec chacun des indicateurs affichés ? Certains indicateurs ne sont-ils jamais observés ou ne donnent-ils jamais lieu à évaluation ?

Cette fonctionnalité, bien qu'implémentée, n'est pas encore utilisée, mais devrait à terme permettre d'aller vers une meilleure connaissance des résultats de simulation, tout autant que vers une mesure de l'efficacité des indicateurs de sortie choisis pour évaluer un ensemble de simulations.

5.4.2.4 Présentation générale

SimEDB est une application interactive. Il nous semblait dommage d'en présenter l'utilisation d'ensemble par une succession de captures d'écran commentées. Nous avons donc réalisé une vidéo qui en montre l'usage et est disponible à cette adresse : A faire...

Conclusion

A reprendre après redécoupage du chapitre !
Au terme de la construction de la plate-forme d'exploration, nous disposons donc d'une application, SimEDB, conçue et développée spécifiquement pour les problématiques propres à l'exploration des données de SimFeodal.

Elle s'inscrit dans les méthodes des Interactions Homme-Machine, ou même dans ce que certains nomment désormais les « Interactions Homme-Données » (« *Human-Data Interaction* », ELMQVIST 2011 ; MORTIER et al. 2014) et s'efforce de suivre les préceptes identifiés dans ce champs (AMIRPOUR AMRAII 2018, p. 167-170 par exemple).

Le développement a été fortement guidé par les contraintes et besoins identifiées, aussi bien en terme d'approches méthodologiques que de choix technologiques. SimEDB est donc un outil *ad-hoc*, toutefois pensé de manière modulaire. Tous les composants logiciels de SimEDB sont indépendants et communiquent de manière standardisée, ouvrant la voie à leur remplacement ou « interchangeabilité » : l'architecture logicielle et les choix techologiques le permettent. La plate-forme SimEDB est donc intrinsèquement pensée comme une réponse à des besoins spécifiques, mais cette réponse a été conçue comme générique et en mesure d'être adaptée aisément à d'autres types de données et/ou sorties de modèles de simulation.

Plus généralement, l'ensemble de ce chapitre montre une démarche similaire, pensée pour répondre à des besoins spécifiques avec des solutions génériques et généralisables. Le passage, depuis une succession de rapports jusqu'à une application d'exploration de ces rapports, ou encore les différents éléments relatifs au choix d'un système de gestion de base de données ou au dessin d'un modèle conceptuel de données s'inscrivent en effet dans cette même démarche qui s'ancre profondément dans une logique de recherche reproductible, aussi bien d'un point de vue technique que de celui du concept et de la méthodologie.

Références

AGARWAL, Sameer et al. (2013). « BlinkDB : Queries with Bounded Errors and Bounded Response Times on Very Large Data ». In : *Proceedings of the 8th ACM European Conference on Computer Systems*. EuroSys '13. 00474. New York, NY, USA : ACM, p. 29-42. DOI : 10/bwrd. URL : http://doi.acm.org/10.1145/2465351.2465355.

AMIRPOUR AMRAII, Saman (2018). « Human-Data Interaction in Large and High-Dimensional Data ». PhD Thesis. University of Pittsburgh.

BATTY, Michael (2015). « A Perspective on City Dashboards ». In : *Regional Studies, Regional Science* 2.1. 00016, p. 29-32. DOI : 10/gfw9mr. URL : https://doi.org/10.1080/21681376.2014.987540.

BEZANSON, Jeff et al. (2014). « Julia : A Fresh Approach to Numerical Computing ». In : URL : http://arxiv.org/abs/1411.1607.

BIMONTE, S., A. TCHOUNIKINE et M. MIQUEL (2005). « Towards a Spatial Multidimensional Model ». In : *Proceedings of the 8th ACM International Workshop on Data Warehousing and OLAP*. DOLAP '05. 00075. New York, NY, USA : ACM, p. 39-46. DOI : 10/cfn8w7. URL : http://doi.acm.org/10.1145/1097002.1097009.

BIMONTE, Sandro (2007). *Intégration de l'information Géographique Dans Les Entrepôts de Données et l'analyse En Ligne : De La Modélisation à La Visualisation*. 00029. Lyon, INSA. URL : http://www.theses.fr/2007ISAL0105.

BOKEH, Development Team (2014). *Bokeh : Python Library for Interactive Visualization*. URL : https://docs.bokeh.org/en/latest/index.html.

BOSTOCK, Mike (2018). *A Better Way to Code*. URL : https://medium.com/@mbostock/a-better-way-to-code-2b1d2876a3a0.

CHANG, Winston et al. (2015). « Shiny : Web Application Framework for R ». In : *R package version 0.11* 1.4. 00553, p. 106.

COMMENGES, Hadrien et al. (2014). *R et espace : Traitement de l'information géographique*. Groupe ElementR. Lyon : Framabook.

CURA, Robin (2015). « Créer des documents reproductibles et des applications web interactives d'analyse de données avec R : Knitr & Shiny ». http ://umr5600.ish-lyon.cnrs.fr/20150610_EVS-ISIG_CafeMethodo. URL : https://github.com/RCura/CafeMethodo.

CURA, Robin (2017a). « Making Large Spatio-Temporal Data Analysis Easier. Illustrated Plea for Using (Geo)Visual Analytics. » cites : cura_making_2017. URL : http://www.geog.leeds.ac.uk/ectqg17/home.html.

— (2017b). « « TimeLineEDB », application web d'exploration interactive de données de géolocalisation ». In : *M@ppemonde* 120.2015/4. URL : https://halshs.archives-ouvertes.fr/halshs-01935702/document.

DOS SANTOS, Selan et Ken BRODLIE (2004). « Gaining Understanding of Multivariate and Multidimensional Data through Visualization ». In : *Computers & Graphics* 28.3, p. 311-325. DOI : 10/cttwpw. URL : http://www.sciencedirect.com/science/article/pii/S0097849304000251.

ELLIOTT, Roxana (2017). *How Page Load Time Affects Bounce Rate and Page Views*. Avec la coll. de SECTION.IO. URL : https://www.section.io/blog/page-load-time-bounce-rate/.

ELMQVIST, Niklas (2011). « Embodied Human-Data Interaction ». In : *ACM CHI 2011 Workshop "Embodied Interaction : Theory and Practice in HCI*, p. 104-107.

EPSTEIN, Joshua M. et Robert L. AXTELL (1996). *Growing Artificial Societies : Social Science from the Bottom Up*. 05176. Washington, D.C : MIT Press. 228 p.

FEKETE, Jean-Daniel (2010). « Infrastructure ». In : *Mastering the Information Age - Solving Problems with Visual Analytics*. Sous la dir. d'EUROGRAPHICS ASSOCIATION. Eurographics Association, p. 87-108. URL : https://hal.inria.fr/hal-00696814.

FEKETE, Jean-Daniel et al. (2019). « Progressive Data Analysis and Visualization ». In : Dagstuhl Seminar 18411. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. URL : https://www.dagstuhl.de/en/program/calendar/semhp/?semnr=18411.

FEW, Stephen (2006a). *Information Dashboard Design : The Effective Visual Communication of Data*. O'Reilly Media, Inc.

— (2006b). « Multivariate Analysis Using Parallel Coordinates ». In : *Perceptual edge*. 00039, p. 1-9.

FORCH, Valentin et al. (2017). « Are 100 Ms Fast Enough ? Characterizing Latency Perception Thresholds in Mouse-Based Interaction ». In : *Engineering Psychology and Cognitive Ergonomics : Cognition and Design*. Sous la dir. de Don HARRIS. Lecture Notes in Computer Science. 00001. Springer International Publishing, p. 45-56.

FOTHERINGHAM, A. Stewart (1999). « Trends in Quantitative Methods III : Stressing the Visual ». In : *Progress in Human Geography* 23.4. 00038, p. 597-606. DOI : 10/c549jd. URL : https://doi.org/10.1191/030913299667756016.

GOWDA, Shashi (2018). *Escher - Composable Web UIs in Julia*. JuliaGizmos. URL : https://github.com/JuliaGizmos/Escher.jl.

GRIGNARD, Arnaud et Alexis DROGOUL (2017). « Agent-Based Visualization : A Real-Time Visualization Tool Applied Both to Data and Simulation Outputs ». In : *The AAAI-17 Workshop on Human-Machine Collaborative Learning*. Association for the Advancement of Artificial Intelligence 17. 00002.

HEALY, Kieran (2018). *Data Visualization : A Practical Introduction*. 00000. S.l. : Princeton University Press. 304 p. URL : http://socviz.co/.

HEINRICH, Julian et Daniel WEISKOPF (2013). « State of the Art of Parallel Coordinates ». In : Eurographics (STARs), p. 95-116. DOI : 10/gd87qm. URL : https://diglib.eg.org:443/handle/10.2312/conf.EG2013.stars.095-116.

IANNONE, Richard, Joseph J. ALLAIRE et Barbara BORGES (2018). *Flexdashboard : R Markdown Format for Flexible Dashboards*. 00000 R package version 0.5.1.1. URL : https://CRAN.R-project.org/package=flexdashboard.

INSELBERG, Alfred et Bernard DIMSDALE (1987). « Parallel Coordinates for Visualizing Multi-Dimensional Geometry ». In : *Computer Graphics 1987*. 01538. Springer, p. 25-44.

KEIM, Daniel et al. (2010). *Mastering the Information Age Solving Problems with Visual Analytics*. 00019. Eurographics Association.

KITCHIN, Rob, Tracey P. LAURIAULT et Gavin MCARDLE (2015). « Knowing and Governing Cities through Urban Indicators, City Benchmarking and Real-Time Dashboards ». In : *Regional Studies, Regional Science* 2.1. 00169, p. 6-28. DOI : 10/gc92g7. URL : https://doi.org/10.1080/21681376.2014.983149.

LAURINI, Robert (2018). « From GIS to Smart Cities : Key-Concept Landmarks ». In : *AIAP'18*. International Conference on Artificiel Intelligence and Its Application. Sous la dir. de Brahim LEJDEL, Mohammed-Khireddine KHOLLADI et Mohammed Charaf Eddine MEFTAH. El Oued, Algeria.

LIU, Z. et J. HEER (2014). « The Effects of Interactive Latency on Exploratory Visual Analysis ». In : *IEEE Transactions on Visualization and Computer Graphics* 20.12. 00125, p. 2122-2131. DOI : 10/f3tvrw.

MACEACHREN, Alan M. et al. (2004). « Geovisualization for Knowledge Construction and Decision Support ». In : *IEEE computer graphics and applications* 24.1. 00256, p. 13-17. pmid : 15384662. URL : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3181162/.

MACKENZIE, I. Scott et Colin WARE (1993). « Lag As a Determinant of Human Performance in Interactive Systems ». In : *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*. CHI '93. 00442. New York, NY, USA : ACM, p. 488-493. DOI : 10/dr7rj6. URL : http://doi.acm.org/10.1145/169059.169431.

MORTIER, Richard et al. (2014). *Human-Data Interaction : The Human Face of the Data-Driven Society*. SSRN Scholarly Paper ID 2508051. Rochester, NY : Social Science Research Network. URL : https://papers.ssrn.com/abstract=2508051.

NIELSEN, Jakob (2009). *Powers of 10 : Time Scales in User Experience*. URL : https://www.nngroup.com/articles/powers-of-10-time-scales-in-ux/.

PAFKA, Szilard (2017). *Benchm-Databases : A Minimal Benchmark of Various Tools (Statistical Software, Databases Etc.) for Working with Tabular Data of Moderately Large Sizes (Interactive Data Analysis)*. URL : https://github.com/szilard/benchm-databases.

PANDRE, Andrew (2011). *Charts and Their Dimensionality*. URL : https://apandre.wordpress.com/dataviews/dimensionality/.

PATEL, Neil (2011). *Speed Is A Killer*. 00000. URL : https://neilpatel.com/blog/speed-is-a-killer/.

PLOTLY (2017). *Introducing Dash*. URL : https://medium.com/@plotlygraphs/introducing-dash-5ecf7191b503.

RAASVELDT, Mark et Hannes MÜHLEISEN (2018). « MonetDBLite : An Embedded Analytical Database ». In : *Proceedings of the 2018 International Conference on Management of Data*. SIGMOD '18. New York, NY, USA : ACM, p. 1837-1838. DOI : 10/gfw9mm. URL : http://arxiv.org/abs/1805.08520.

REY-COYREHOURCQ, Sébastien (2015). « Une plateforme intégrée pour la construction et l'évaluation de modèles de simulation en géographie ». Thèse de doctorat en Géographie. Paris : Université Paris I - Panthéon-Sorbonne.

REYNOLDS, Craig W. (1987). « Flocks, Herds and Schools : A Distributed Behavioral Model ». In : *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. 10133. New York, NY, USA : ACM, p. 25-34. DOI : 10/chhdjr. URL : http://doi.acm.org/10.1145/37401.37406.

RIVARD, Kurt et Doug COGSWELL (2004). « Are You Drowning in BI Reports ? Using Analytical Dashboards to Cut through the Clutter ». In : *DM Review, http ://goo. gl/hle9Wc*. 00000.

ROOT, Christopher et Todd MOSTAK (2016). « MapD : A GPU-Powered Big Data Analytics and Visualization Platform ». In : *ACM SIGGRAPH 2016 Talks*. SIGGRAPH '16. New York, NY, USA : ACM, 73 :1-73 :2. DOI : 10/gd7hg8. URL : http://doi.acm.org/10.1145/2897839.2927468.

ROTH, Robert E. (2013). « Interactive Maps : What We Know and What We Need to Know ». In : *Journal of Spatial Information Science* 6. 00099. DOI : 10/gfw9mq. URL : http://www.josis.org/index.php/josis/article/view/105.

— (2015). « Interactivity and Cartography : A Contemporary Perspective on User Interface and User Experience Design from Geospatial Professionals ». In : *Cartographica : The International Journal for Geographic Information and Geovisualization*. 00016. DOI : 10/gfw9mp. URL : https://www.utpjournals.press/doi/abs/10.3138/cart.50.2.2427.

ROUMPANI, F., O. O'BRIEN et A. HUDSON-SMITH (2013). « Creating, Visualizing and Modelling the Realtime City ». In : *Proceedings of Hybrid City II 'Subtle rEvolutions' Conference*. 00002.

SCHELLING, Thomas C (1971). « Dynamic Models of Segregation ». In : *Journal of mathematical sociology* 1.2. 04241, p. 143-186.

SHNEIDERMAN, Ben (1996). « The Eyes Have It : A Task by Data Type Taxonomy for Information Visualizations ». In : *Proceedings 1996 IEEE Symposium on Visual Languages*. IEEE, p. 336-343. DOI : 10/fwdq26.

Snowflake Schema (2018). In : *Wikipedia*. Page Version ID : 856363739. URL : https://en.wikipedia.org/w/index.php?title=Snowflake_schema&oldid=856363739.

Star Schema (2018). In : *Wikipedia*. Page Version ID : 853084099. URL : https://en.wikipedia.org/w/index.php?title=Star_schema&oldid=853084099.

TUFTE, Edward R. (2001). *The Visual Display of Quantitative Information*. 2nd edition. 11580. Cheshire, Conn : Graphics Press USA. 190 p.

VERMEIJ, Maarten et al. (2008). « MonetDB, a Novel Spatial Columnstore Dbms ». In : *Academic Proceedings of the 2008 Free and Open Source for Geospatial (FOSS4G) Conference, OSGeo*. 00015, p. 193-199.

WARE, Colin (2012). *Information Visualization : Perception for Design*. 05270. Elsevier.

WICKHAM, Hadley (2016). *Ggplot2 : Elegant Graphics for Data Analysis*. 00082. Springer.

— (2017). « Tidyverse : Easily Install and Load 'tidyverse' Packages ». In : *R package version* 1.1. 00113.

WICKHAM, Hadley et al. (2015). « Dplyr : A Grammar of Data Manipulation ». In : *R package version 0.4* 3. 00547.

WILKINSON, Leland (2006). *The Grammar of Graphics*. 01037. Springer Science & Business Media.

ZAAMOUNE, Mehdi et al. (2013). « A New Relational Spatial OLAP Approach for Multi-Resolution and Spatio-Multidimensional Analysis of Incomplete Field Data ». In : *ICEIS 2013 INSTICC International Conference on Enterprise Information Systems*. 00010, p.

ZENG, Kai, Sameer AGARWAL et Ion STOICA (2016). « iOLAP : Managing Uncertainty for Efficient Incremental OLAP ». In : *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD '16. 00012. New York, NY, USA : ACM, p. 1347-1361. DOI : 10/gfw9mn. URL : http://doi.acm.org/10.1145/2882903.2915240.

ÇÖLTEKIN, Arzu, Halldór JANETZKO et Sara FABRIKANT (2018). « Geovisualization ». In : *Geographic Information Science & Technology Body of Knowledge* 2018.Q2. DOI : 10.22224/gistbok/2018.2.6. URL : https://gistbok.ucgis.org/bok-topics/geovisualization.