ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE LOGICIEL
M. Sc. A.

PAR
Jonathan CLOUTIER

RÉTRO INGÉNIERIE D'APPLICATIONS WEB JAVASCRIPT POUR AIDER À LA COMPRÉHENSION ET À LA DOCUMENTATION

MONTRÉAL, LE 21 AVRIL 2016

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Sègla Kpodjedo, directeur de mémoire
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Abdelouahed Gherbi, président du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

Mme Ghizlane El Boussaidi, membre du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC
LE 12 AVRIL 2016
À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je remercie mon superviseur Sègla Kpodjedo pour avoir accepté de superviser mon mémoire de maitrise. Je le remercie pour m'avoir guidé à travers mon mémoire, pour sa disponibilité et ses conseils.

Je remercie également Anna-Belle Filion pour son support, ses idées et ses commentaires constructifs à propos de mon mémoire.

RÉTRO-INGÉNIERIE D'APPLICATION WEB JAVASCRIPT POUR AIDER À LA COMPRÉHENSION ET À LA DOCUMENTATION

Jonathan CLOUTIER

RÉSUMÉ

Ce mémoire s'intéresse à la rétro-ingénierie comme solution pour aider les développeurs à comprendre, modifier et documenter la structure de leurs applications web. Pour retrouver la structure d'une application, il faut souvent recourir à de l'analyse statique du code source pour retrouver les différents éléments et les différentes relations qui composent l'application. Le développement web présente ici des défis particuliers puisqu'il fait intervenir plusieurs langages. Certains de ces langages, comme HTML et CSS sont relativement simples; d'autres le sont moins. En particulier, JavaScript, un langage clé de la technologie Web, présente des aspects dynamiques importants (p. ex.: typage dynamique, évaluation dynamique de chaines de caractères) qui pourraient rendre très inefficace une analyse statique du code source. En effet, la récupération des éléments constituant l'application et de leurs liens pourrait devoir nécessiter une analyse dite dynamique qui se ferait sur des scénarios d'exécution. Cependant, de telles analyses dynamiques ne garantissent pas une couverture complète de l'application et ne peuvent se faire que si le code est exécutable. Nous avons donc conduit une étude empirique sur la viabilité de l'analyse statique pour la rétro-ingénierie de JavaScript. Forts de ces résultats ainsi que des constats sur les techniques et outils existants, nous proposons nos propres pistes de solutions sous forme d'une nouvelle approche de rétro-ingénierie (Web application Viewer). Cet outil est subséquemment utilisé pour performer des expérimentations de visualisation de structure à l'aide de diagrammes de force dirigée et diagrammes de classes. L'outil de rétro-ingénierie créé permet d'extraire les principaux éléments de la structure d'une application web pour les langages JavaScript, Node.js, HTML et CSS. Les résultats sont satisfaisants et permettent au développeur de documenter leurs applications rapidement à l'aide de diagrammes.

Mots-clés : rétro-ingénierie, développement web, analyse statique, visualisation, diagramme de structure, JavaScript, HTML

REVERSE ENGINEERING OF WEB APPLICATION FOR COMPREHENSION AND DOCUMENTATION

Jonathan CLOUTIER

ABSTRACT

Web developers face some unique challenges when trying to understand, modify and document the structure of their web applications. The heterogeneity and complexity of the underlying technologies and languages heighten comprehension problems. In particular, JavaScript, which is an essential part of the Web ecosystem, is a language that offers a flexibility that can make its code hard to grasp, when it comes to comprehension and documentation tasks. Indeed, its dynamic nature (ex.: dynamically typed, real-time evaluation) seems to complicate things. While the task of analyzing the code seems to be tailor-made for the dynamic analysis, some situations prevent us from using it (eg .: faulty or incomplete code). Moreover, there is no certainty that developers use enough of these dynamic features to render the results of static analysis useless. One of the mandates of this thesis is to empirically assess the use of static analysis for JavaScript as a viable approach for reverse engineering for documentation purpose. Then, using the results of the empirical study and the findings on existing tools and techniques, we offer our own potential solutions. A new reverse engineering approach is presented that takes the form of a tool called WAVI. This tool is subsequently used to perform structural visualization experiments using force directed diagrams and classes. The established reverse engineering tool is used to extract the key elements of the structure of a web application for JavaScript, Node.js, HTML and CSS. The effectiveness of WAVI is evaluated with experiments that demonstrate that it can resolve JavaScript calls better than a recent technique, and its visualisation modules are intuitive and scalable. The results are satisfactory and allow the developer to document their applications quickly using diagrams.

Keywords : reverse engineering, web application, static analysis, JavaScript, class diagram, force-directed diagram.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AJAX	Asynchronous JavaScript and XML
AMD	Asynchronous Module Definition
ASP	Active serveur pages
AST	Abstract syntax tree
CSS	Cascading style sheets
HTML	HyperText mark-up language
HTTP	HyperText transfer protocol
IDE	Integrated Development Environment
JSP	Java serveur pages
NPM	Node Packaged Modules
PHP	HyperText Preprocessor
UML	Unified Modeling Language
W3C	World Wide Web Consortium
XML	Extensible mark-up language

INTRODUCTION

Contexte et problématique

L'une des fonctions les plus populaires des ordinateurs de nos jours est l'utilisation d'Internet. Les développeurs doivent s'affairer à créer des applications web de bonne qualité s'ils souhaitent attirer les visiteurs. Pour garder ces mêmes visiteurs, les applications web sont souvent appelées à évoluer relativement rapidement et ce faisant, voient la complexité de leur structure augmenter. Les applications web présentent la particularité d'un environnement hétérogène où se retrouve du code source provenant de divers langages informatiques. Elles font usage de fichiers de type « HyperText mark-up language » (HTML), JavaScript, « Cascading Style Sheets » (CSS) et de langages serveur comme Node.js, Java ou « HyperText Preprocessor » (PHP). Certains de ces fichiers seront même réutilisés dans plusieurs pages web.

Les fichiers qui composent les applications web ont différents rôles qui ne peuvent être saisis qu'en lisant la documentation, si elle existe et qu'elle est gardée à jour ou bien en analysant le code source. Il existe des outils pour aider les développeurs, mais ils sont souvent limités à un seul langage de programmation. Ils sont moins performants et stables si le projet compte trop de fichiers de grande taille et la plupart sont difficiles à utiliser. Il y a rarement des analyses complètes qui aident les développeurs à comprendre comment rendre le code plus extensible et facile à maintenir. Puisqu'il existe peu de façons efficaces de documenter une application web, plusieurs entreprises n'ont pas les ressources pour garder leur documentation à jour et vont donc abandonner toute cette opération (Amalfitano 2011).

De plus, une application web peut demander des changements substantiels en cours de développement et à la suite de son déploiement (Kienle and Distante 2014). Si plusieurs personnes travaillent sur la même application web, les changements doivent être bien communiqués pour que toutes les parties prenantes restent à jour et que les développeurs évitent d'introduire des bogues dus à une mauvaise compréhension de la structure (Liauw

2012). L'absence de documentation peut causer beaucoup de problèmes et occasionner des pertes d'informations avec le temps lorsqu'il y a changement de personnel. Les nouveaux employés devront passer le temps nécessaire pour comprendre la structure de l'application web ou ils risquent de l'affaiblir, la complexifier et éventuellement de la rendre très difficile à maintenir.

Objectifs

Des outils modernes et performants de rétro-ingénierie seraient donc utiles pour garder une trace de la structure d'une application web, collecter des données sur celle-ci pour la visualiser selon diverses perspectives. S'il était possible de récupérer automatiquement la structure, de visualiser les fichiers et les liens entre eux afin de la comprendre plus rapidement, il serait alors plus facile de respecter la structure lorsque l'on fait des changements pour éviter d'introduire des bogues ou encore augmenter la productivité des employés tout en assurant la qualité du travail accompli. Il serait enfin possible d'utiliser ces outils pour déboguer une application web, la restructurer ou tout simplement pour visualiser la structure avant de développer une extension.

Ce mémoire est consacré à la présentation d'un outil de rétro-ingénierie afin d'aider les développeurs à mieux comprendre la structure de leurs applications web. En général, les efforts de rétro-ingénierie se déclinent en deux principales approches : il y a l'analyse statique qui étudie l'application simplement à l'aide du code source et l'analyse dynamique qui étudie le déroulement lors de l'exécution de l'application (Madsen 2015). Les deux approches comportent des avantages et des inconvénients. Par exemple, l'analyse statique permet d'analyser du code incomplet sans l'exécuter et couvre tout le code du système. Elle demande peu d'aide de l'utilisateur, mais offre une vue globale imprécise du système (faisant abstraction des scénarios d'exécution possible). L'analyse dynamique demande un scénario d'exécution, couvre seulement le code exécuté, mais offre une vue très précise pour ce scénario (faisant abstraction des autres possibilités du système).

Nous avons décidé d'examiner plus à fond la voie de l'analyse statique compte tenu des avantages qu'elle pourrait apporter dans le domaine de la documentation. Nous avons d'abord vérifié la viabilité de cette approche malgré la présence du langage relativement dynamique qu'est JavaScript. À la suite des résultats de cette analyse, nous avons conçu un outil de rétro-ingénierie (WAVI) capable d'extraire les éléments d'une application web et de générer des diagrammes basés sur les travaux existants ainsi que les standards de modélisation (Conallen 2003). En outre, nous avons mis en lumière les obstacles rencontrés, proposé des pistes de solution simples et efficaces pour les surmonter, recueilli et partagé le plus d'information possible sur le processus de la création et de l'utilisation de notre outil de rétro-ingénierie.

Organisation du mémoire

Dans un premier temps, ce mémoire présentera la revue de la littérature qui est composée de la mise en contexte comportant une description des applications web, des langages web ainsi que leurs relations et spécifications. Les particularités du développement web seront abordées avec les problèmes soulevés (éducation, processus, outils, etc.) et des pistes de solution seront rapportées. Ensuite, il y aura un recensement des approches techniques et technologiques de rétro-ingénierie qui utilisent l'analyse statique et dynamique ainsi que la modélisation des applications web. Les outils de rétro-ingénierie d'application web, JavaScript et Node.js seront répertoriés selon leur type d'analyse et leur mission. Dans le chapitre 2, nous présentons une étude empirique qui évaluera les difficultés auxquelles les développeurs doivent faire face lors de l'analyse statique lorsqu'on considère la génération de diagramme de structure d'application web. Dans le chapitre 3, on présente une nouvelle approche pour aider les développeurs à mieux comprendre la structure de leurs applications web en faisant de la rétro-ingénierie ainsi que la méthodologie lors de la création d'un outil d'analyse statique appelé WAVI. Dans le chapitre 4, on présente des expérimentations sur la visualisation des données recueillies par l'analyse statique d'applications web à l'aide de l'outil WAVI. Ces données seront également filtrées et différents profils de visualisation seront proposés aux utilisateurs. Finalement, le chapitre 5 présente la conclusion, nous

revenons sur les éléments de la problématique et nous résumons les faits saillants de cette recherche. Nous présentons également les travaux futurs.

CHAPITRE 1

REVUE DE LA LITTÉRATURE

1.1 Mise en contexte

1.1.1 Les applications web

Contrairement à une application native traditionnelle qui a accès aux fichiers et périphériques de l'ordinateur et qui est disponible hors ligne, les applications web résident sur un ordinateur distant appelé serveur et ne nécessitent pas d'installation. Elles sont limitées dans l'accès aux ressources et requièrent une connexion réseau. L'ordinateur de l'utilisateur qui exécute l'application est nommé le client. Cet utilisateur se sert d'un navigateur web en fournissant une adresse web comme paramètre afin d'initialiser une requête de visualisation de page et navigue à travers les fichiers du serveur en cliquant sur des hyperliens. Une portion du code est exécutée sur le serveur (p. ex.: Node.js, Java, PHP, ASP, Ruby, etc..) et le résultat est envoyé au client sous forme HTML, XML, JSON ou simplement texte. Ensuite, une seconde portion du code (JavaScript client, CSS) est exécutée sur l'ordinateur du client à l'aide du navigateur web. Clients et serveurs communiquent grâce au « HyperText Transport Protocol » (HTTP) (Fielding, Gettys et al. 1999). Ce protocole comprend plusieurs méthodes, dont « POST », qui permet entre autres de soumettre au serveur de l'information recueillie dans un formulaire et « GET » qui permet de réaliser une demande d'information (p. ex.: recevoir le contenu d'une page web).

1.1.2 Langages web

Certains langages sont utilisés du côté client et d'autres du côté serveur pour obtenir une page web dynamique (*Voir* Figure 1.1). Les trois principaux langages utilisés du côté du

client sont HTML, JavaScript[1] et CSS (Mendes 2014). Du côté serveur, il existe plusieurs langages dont Node.js qui est une implémentation JavaScript du côté serveur, PHP, ASP et Java.

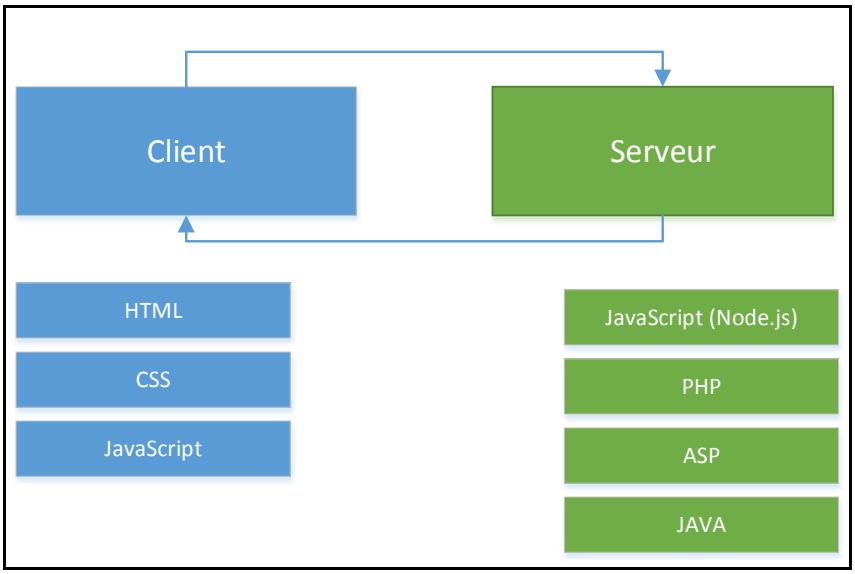

Figure 1.1 Liste des langages reliés aux clients et serveurs.

- **HTML :** HTML ou « HyperText Markup Language » est le langage web de base. Il permet de gérer le contenu des pages avec du texte, des formulaires, des liens et divers éléments multimédias tels que des images et des vidéos. Le standard HTML est rédigé par le « World Wide Web Consortium » (W3C). La version la plus récente est nommée HTML5 et est en vigueur depuis 2014. Elle propose une multitude de nouvelles balises pour aider les développeurs à structurer leurs pages et pallier à des manquements touchant aux fonctions des vidéos, des barres de progrès, des images vectorielles (svg), des dialogues de choix de couleurs, des calendriers, etc. Il retire, en conséquence, les éléments jugés obsolètes tels que les cadres et les applets et plusieurs attributs de style des éléments (p.ex. : « align », « width », « bgcolor ») afin

[1] Un sondage de W3Techs datant de Mars 2016 rapporte que le langage le plus populaire du côté client est JavaScript avec 93,4% d'utilisation.

de favoriser l'usage du CSS. De nouvelles fonctionnalités sont également ajoutées entre autres pour l'entreposage de données et la géolocalisation.

- **CSS :** Bien qu'il soit possible d'intégrer les styles (police, bordure, couleur, etc.) à l'intérieur d'un fichier HTML en utilisant les balises, une bonne pratique est de séparer le code CSS du HTML et d'utiliser des fichiers distincts afin de faciliter la lecture en général. CSS est un langage qui est utilisé pour contenir les attributs de style des éléments HTML. Il permet de regrouper les styles dans un fichier et de les réutiliser à plusieurs endroits en utilisant une classe, en spécifiant un élément précis ou un type d'élément précis. Le standard actuel est CSS3 qui est également régi par le W3C.

- **JavaScript et Node.js :** Créé en 1995, JavaScript est un langage web qui a été créé afin de bonifier l'expérience utilisateur, notamment en dynamisant l'affichage du HTML. Ce langage a su se positionner comme étant incontournable dans le monde du web. JavaScript est un langage qui est interprété différemment par les navigateurs web et machines virtuelles. Par conséquent, le standard EcmaScript créé en 1997 permet de garder une certaine cohérence dans la base de la syntaxe de JavaScript à travers toutes ces différentes interprétations. Il existe plusieurs éditions de ce standard. La version utilisée et implémentée dans les navigateurs actuels est EcmaScript 5.0, qui a été livrée en 2009. Elle corrige certains aspects de la version 3.0 et ajoute des fonctionnalités comme l'utilisation de JSON sans faire trop de changements majeurs. Une autre nouveauté est apparue en 2009. Il s'agit de Node.js qui utilise JavaScript du côté serveur et permet de tirer pleinement profit de la nature asynchrone de JavaScript. Il apporte également comme avantage au développeur d'utiliser le même langage du côté client et serveur et ainsi permettre un développement « isomorphique » où certaines ressources peuvent être partagées entre le client et le serveur. De nos jours, JavaScript est un des langages les plus répandus (Richards, Hammer et al. 2011) et est constamment positionné parmi le palmarès des

10 langages les plus utilisés[2]. EcmaScript 6 (ES2015), qui a été officialisé en juin 2015, ramène plusieurs fonctionnalités qui avaient été proposées pour la version 4.0. Cette nouvelle édition apporte donc de gros changements comme les classes, les générateurs et les modules qui devraient être implémentés dans la prochaine génération de navigateurs et de machines virtuelles. Ces changements seront entre autres très utiles pour programmer des applications modernes, plus robustes et mieux structurées.

1.1.3 Relations entre fichiers

La séparation du code source en plusieurs fichiers est une bonne pratique qui facilite la lisibilité et la réutilisation du code. Il faut cependant être en mesure de gérer les dépendances et importer ou exporter ces fichiers de façon efficace. La gestion de ces opérations est d'autant plus complexe lorsqu'on considère l'agencement de tous les langages différents du Web et les différents choix possibles. Les pages HTML sont très importantes dans cet exercice, car elles peuvent contenir des liens vers des fichiers externes ou bien des balises contenant du code JavaScript et CSS (voir ANNEXE I). JavaScript de son côté permet de communiquer avec n'importe quel type de fichiers avec des requêtes «Asynchronous JavaScript and XML » (AJAX) et ses méthodes « POST », « GET », « PUT » et « DELETE » afin de transmettre des informations ou de faire des demandes d'information au serveur. Pour l'utilisation de module entre les fichiers JavaScript eux-mêmes, il existe deux spécifications populaires différentes. Les modules sont chargés de façon synchrone avec CommonJS (voir ANNEXE II) ou bien ils sont chargés de façon asynchrone avec AMD « asynchronous module definition » (voir ANNEXE III). De façon synchrone, chaque module est chargé l'un à la suite de l'autre selon leur emplacement dans le code. Une fois qu'un module est chargé, il est disponible pour être utilisé. De façon asynchrone, tous les modules doivent être chargés au départ avant leur utilisation. Ces spécifications peuvent ensuite être implémentées de façons différentes selon les besoins. Les deux spécifications

[2] http://spectrum.ieee.org/computing/software/top-10-programminglanguages

(AMD et CommonJS) ne sont pas implémentées de façon native dans JavaScript, mais ES2015 apporte son propre format afin d'uniformiser le chargement et l'exportation des modules. Il ne sera donc plus nécessaire d'utiliser ces spécifications dans le futur afin de gérer les dépendances, car ces fonctionnalités seront désormais intégrées dans le langage de façon native (voir ANNEXE IV).

1.2 Particularités du développement web

1.2.1 Problèmes

Le développement et la maintenance d'applications web sont des tâches complexes qui doivent faire face à un certain nombre de problèmes particuliers. Selon la littérature scientifique, on compte parmi ces problèmes :

- **Le manque de discipline et de constance :** Selon (Welland 2001), les développeurs web ne font pas suffisamment attention aux problèmes de maintenance. Ils manquent de discipline et de constance, ce qui affecte la qualité des applications développées.

- **Le cycle de développement trop court :** Le cycle de développement alloué, par les entreprises, pour les applications web est beaucoup plus court que celui des logiciels conventionnels, ce qui fait qu'il y a davantage de pression sur les développeurs pour terminer le travail (Welland 2001). Ils doivent livrer des applications web rapidement, ce qui fait en sorte que le temps passé sur la conception, le test et la documentation du projet est significativement réduit (Mendes 2014).

- **Les méthodes de développement trop complexes :** Une étude (Jeary, Phalp et al. 2009) explique que 22 participants sur 23 ont abandonné leur méthode de développement parce qu'ils trouvaient qu'elle était trop complexe et difficile à appliquer. Ils ont, entre autres, noté que les méthodologies n'étaient pas suffisamment établies dans le milieu du développement web. De plus, il y a une certaine ambiguïté dans la terminologie utilisée ce qui les rend difficiles à comprendre. Les méthodologies proposées étaient incomplètes et les participants manquaient de directives dans l'application des méthodes, ce qui faisait qu'ils étaient moins confiants pour les utiliser.

- **Les technologies et outils immatures :** Les technologies ainsi que les outils, dans le domaine du développement web, sont actuellement immatures et cela a un impact direct sur les méthodes de développement selon (Jeary, Phalp et al. 2009) et (Liauw 2012).

- **Langage trop flexible :** JavaScript est un langage clé pour les applications web, car il permet de faire le pont entre le côté client et le côté serveur. C'est un langage flexible qui ne possède pas, de façon claire, d'implémenter des classes avec la spécification ES5. Des fois, plusieurs techniques différentes peuvent être utilisées dans le même programme pour créer des classes, ce qui résulte en un code qui est incohérent et difficile à maintenir (Gama, Alalfi et al. 2012).

- **Les pages dynamiques rendent la structure plus complexe :** De nos jours, les applications web font de plus en plus appel aux pages gérées de façon dynamique et créées sur mesure en fonction du besoin de l'utilisateur (Estievenart, Francois et al. 2003). Ces fonctionnalités requièrent souvent l'utilisation de bases de données. Leurs utilisations augmentent les capacités des applications web, mais elles rendent celles-ci toujours un peu plus complexes à gérer.

- **Plusieurs technologies sont utilisées :** La structure des sites web varie également beaucoup selon les technologies utilisées (Mendes 2014). Parmi les technologies utilisées, il y a les combinaisons de langages pour le côté client et pour le côté serveur. Certains se spécialisent dans la gestion de l'interaction utilisateur, d'autres pour personnaliser les styles visuels de l'interface et il en existe également pour faire des requêtes dans les bases de données.

Tous ces problèmes font en sorte qu'il y a un impact direct sur la qualité de la conception et la documentation du code. En conséquence, ce type d'application a une période de vie écourtée puisque les développeurs vont avoir plus de difficulté à maintenir l'application. Cette difficulté augmente le temps nécessaire pour la maintenance, ce qui augmente donc les

coûts. Cette réalité poussera donc rapidement les gestionnaires à prendre la décision de remplacer l'application web au lieu de continuer à la maintenir.

1.2.2 Pistes de solutions

Maintenant que les problèmes sont mieux connus, il est temps de regarder les solutions proposées dans certains articles lors des dernières années.

- **Développer de nouvelles approches et outils :** Afin de réduire les risques dans les projets de développement d'applications web complexe, il est important que l'industrie se concentre sur le développement de meilleures approches, méthodes, guides de développement et outils pour toutes les phases de développement et d'implémentation d'applications web (Ginige and Murugesan 2001). Dans le même sens (Jensen, M et al. 2009), pensent que la création de nouveaux outils est nécessaire pour déboguer et maintenir les applications web.

- **Promouvoir la communication entre équipes de développement :** Les applications web font généralement appel à plusieurs langages de programmation ou technologies. Un bon processus de développement devrait permettre la communication entre les équipes de développement afin d'assurer qu'il y ait une certaine cohérence et qu'il n'y ait pas de duplication entre eux (Welland 2001).

- **Favoriser la documentation des applications :** La documentation joue un rôle important dans la communication. Non seulement elle peut servir de base pour les discussions à propos de ce qui a été développé dans le passé, mais elle permet aussi de planifier l'évolution du logiciel dans le futur. Lorsqu'il y a un roulement de personnel, elle permet de ne pas perdre toutes les connaissances à propos de l'application web. Bref, avec une documentation adéquate, il est possible d'éviter d'éventuels problèmes pour la compréhension d'applications web complexes (Hassan and Holt 2002). Une solution pour mieux comprendre la structure d'une application

web est l'utilisation de diagrammes (Fitzgerald, Counsell et al. 2013). Cela permet de mieux gérer la complexité dans les applications ainsi que d'abstraire et modéliser sa structure (Conallen 1999). Les représentations graphiques documentent les décisions de conception et facilitent la communication à l'intérieur de l'équipe (Schwinger and Koch 2006).

- **Des méthodes de développement plus universelles :** Les développeurs devraient être en mesure d'utiliser des processus qui sont réutilisables de projet en projet. Les méthodes de développements actuelles ne sont pas aussi efficaces d'un projet à un autre. Elles doivent être davantage universelles et être testées dans des cas réels (Jeary, Phalp et al. 2009).

1.3 Pistes de solutions techniques/technologiques explorées

Plusieurs des pistes de solutions rapportées dans la section précédente peuvent être facilitées par des outils efficaces de rétro-ingénierie pour documenter l'application et ainsi améliorer la communication au sein d'une équipe de développement. Le développeur doit comprendre la structure de son application web s'il veut être en mesure d'optimiser ses performances, d'en faire la maintenance et de documenter sa conception afin de garder une trace des changements effectués. La littérature disponible sur la rétro-ingénierie de sites web explore plusieurs questions dont deux cruciales sur lesquelles nous nous penchons dans les sous-sections qui suivent. Il s'agit (1.3.1) du type d'analyses à effectuer pour retrouver la structure d'une application web et (1.3.2) du type de représentations adapté pour sa compréhension et sa communication.

1.3.1 Analyse de structure du code

1.3.1.1 Analyse statique

L'analyse statique est une activité qui nécessite simplement l'accès au code source. Le code source est analysé en profondeur afin de découvrir les aspects qui requièrent l'attention du

développeur. Le code source n'est pas exécuté lors de l'analyse statique, ce qui fait qu'elle n'a pas accès au contexte d'exécution alors que plusieurs éléments de l'analyse peuvent dépendre de ce contexte. L'analyse statique est vulnérable face aux fonctionnalités dynamiques de certains langages (Madsen 2015). En résumé, l'analyse statique peut couvrir tout le code même les parties incomplètes, mais en terme de précision, il faut regarder du côté de l'analyse dynamique qui aide à mieux comprendre le fonctionnement de l'application lors de son exécution.

1.3.1.2 Analyse dynamique

L'analyse dynamique s'appuie sur des scénarios dont l'exécution est suivie de près pour mieux comprendre le fonctionnement d'une application. L'analyse dynamique permet également de mieux comprendre le comportement de l'application et des utilisateurs. En utilisant la journalisation des événements fournis par les navigateurs, il est possible de connaitre les requêtes utilisateurs et d'enregistrer leurs comportements afin de pouvoir les analyser par la suite (Junhua and Baowen 2009). Cependant, ce type d'analyse comporte son lot de problèmes. Par exemple, l'analyse dynamique requiert l'intervention ou les directives de l'humain pour l'élaboration et la sélection de scénarios pertinents. (Hamou-Lhadj, En-Nouaary et al. 2007) (Toma and Islam 2014). Une telle activité non seulement demande des efforts supplémentaires, mais pourrait être carrément impossible en cours de développement si le code est incomplet ou erroné. Enfin, l'analyse dynamique ne permettra pas toujours d'avoir une couverture complète de l'application si le scénario n'utilise pas toutes les fonctionnalités (Feldthaus, Schafer et al. 2013).

1.3.1.3 Analyse de JavaScript

Comme précédemment expliqué, JavaScript est un langage clé dans le développement web. C'est aussi un langage qui offre aux développeurs une flexibilité de codage relativement élevée, que nous détaillons dans les points suivants.

- **Le typage dynamique**

```
var a = 6;
a= ‘’;
a = [];
```

Dans cet exemple, la variable « a », d'abord initialisée avec le type « Number », devient de type « String » et ensuite « Array ». En tout temps, le développeur peut donc changer le type d'une variable simplement en y assignant une nouvelle valeur.

- **Les évaluations en temps réel**

```
eval('alert(”hello world”); ');
new Function( 'alert(“hello world”); ')();
```

Il est possible de faire des évaluations de chaîne de caractères en temps réel. Ces évaluations se font à l'aide des méthodes « eval » et « new Function ».

- **L'accès dynamique des propriétés**

```
o.x = ‘a’;
o[”x”] = ‘b’;
var y = “x”;
o[y] = ‘c’;
```

Dans cet exemple, la propriété « x » de l'objet « o » est accédée de trois façons différentes. Premièrement, directement en utilisant la notation de point «.». Deuxièmement, à l'aide de littéral entre crochets et finalement à l'aide d'une variable entre crochets possédant la valeur « x ». Il existe donc une équivalence tel que o.x = o[”x”]= o[y] afin d'accéder à la propriété « x ».

- **La variadicité des fonctions**

```
function somme(){
   var total = 0;
   for (i = 0; i < arguments.length; i++) {
      total += arguments[i];
   }
   return total;
}
Somme(20,15,5,18);
```

Une fonction peut être appelée avec plus ou moins de paramètres que sa déclaration ne contient. Si l'appel contient moins de paramètres, les paramètres manquants seront considérés comme non définis. En contrepartie, si l'appel contient plus de paramètres, alors ceux-ci seront accessibles à l'aide du tableau nommé « arguments ».

- **Changement dynamique du contexte d'exécution :**

```
var personA = {};
personA.capsName = "Tom";
function greet() {
   console.log("Hi" + this.capsName())
};
greet.apply(personA)
```

Contrairement aux langages orientés objet classiques, une fonction JavaScript peut appartenir à plusieurs objets et être appelée dans différentes instances. Cette fonctionnalité est disponible à l'aide des constructeurs « apply » et « call ». L'appel capsName() dépend de l'objet « greet ». Dans cet exemple, personA est l'objet appelé. De plus, le constructeur « apply » offre de la flexibilité au niveau du nombre de paramètres de fonction, qui peut être déterminé lors de l'exécution.

- **Portée dynamique**

```
var o = {firstName : « john »,lastName : « doe »};
with (o) {
   console.log( firstName + ' ' + lastName);
}
```

JavaScript permet d'avoir une portée dynamique à l'aide de la fonction « with ». Dans cet exemple, les propriétés de l'objet o deviennent des variables locales.

Toute cette flexibilité n'est pas sans conséquence et nuit à la précision et l'efficacité de l'analyse statique de JavaScript. Cela explique en partie la nouvelle version ES2015 qui ajoute des mécanismes pour structurer davantage les programmes et, ce faisant, simplifier l'analyse statique.

1.3.2 Modélisation d'application web

Le diagramme de classes est l'un des diagrammes UML les plus utilisés pour comprendre la structure d'une application. On y voit les classes, leurs propriétés telles que les méthodes et attributs ainsi que les liens entre les classes. UML est un langage de modélisation très populaire auprès des développeurs d'application standard. Par contre, ce langage ne propose pas, dans sa forme de base, une façon de modéliser tous les éléments des applications web. Il faut donc utiliser des profils UML spécifiques afin d'étendre les fonctions de UML pour tenir compte par exemple des Pages HTML, du CSS et même des particularités de certains langages serveur comme le J2EE (Java 2 Enterprise Edition). Jim Conallen a créé un de ces profils qu'il a appelé « Web Application Extension (WAE) » (Conallen 2003) (Voir Figure 1.2). Il présente « WAE » dans le livre « Building Web Applications with UML» afin d'aider les développeurs à modéliser des applications web et à exprimer les éléments d'applications web de façon cohérente avec les autres éléments du système. Son profil bonifie bien le diagramme de classes UML et est particulièrement efficace pour les analyses statiques. C'est un profil UML composé de stéréotypes, de valeurs étiquetées et de contraintes.

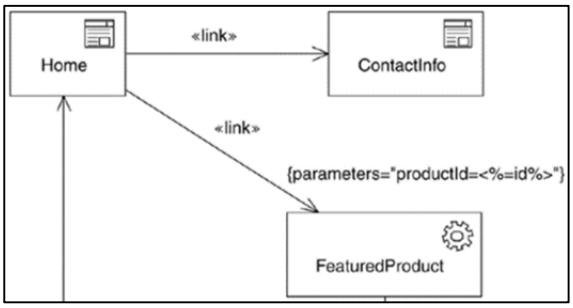

Figure 1.2 Modélisation avec WAE
Tirée de (Conallen 2003)

Les stéréotypes et les icônes permettent de donner un sens sémantique aux éléments et liens. Les textes entre accolades (valeurs étiquetées) donnent de l'information supplémentaire sur un élément ou une propriété. Les contraintes sont des règles à suivre pour s'assurer que le modèle créé est valide. Par exemple, on utilise des contraintes pour réglementer les relations possibles entre les éléments (Conallen 2003).

Pour mieux comprendre les changements proposés par Conallen, un exemple plus complet est disponible (voir ANNEXE V). Les éléments et associations proposés par Conallen sont également présentés (voir ANNEXE VI). Il existe des études empiriques validant l'utilité des stéréotypes de Conallen pour la compréhension d'applications web. Une de ces études (Ricca, Penta et al. 2006) compte 35 étudiants universitaires qui répondent à des questions à l'aide de diagrammes de UML « WAE » et des diagrammes UML standards. Les résultats de l'étude dévoilent que le profil UML créé par Jim Conallen aide significativement à la compréhension d'une application web comparativement aux diagrammes standards UML. Il présente plus d'information sur la structure, les formulaires et leurs soumissions, ce qui réduisait le temps nécessaire passé à étudier le code source. Finalement, « WAE » propose une modélisation simple et efficace pour un outil de rétro-ingénierie, car les éléments et leurs associations peuvent être obtenus facilement à partir du code source. Le livre de Jim Conallen utilise des exemples concrets et sa compatibilité avec le standard UML aide les utilisateurs à comprendre plus rapidement la structure de son diagramme. L'extension WAE est considéré comme une référence pour le processus de rétro-ingénierie d'applications web et la plupart des outils de rétro-ingénierie qui se basent sur l'analyse statique l'utilisent (Tramontana 2005). Par contre, le modèle n'a pas été mis à jour depuis 2003 et ne prend pas en compte les nouveautés technologiques telles qu'AJAX et le langage Node.js.

1.4 Outils et approche de rétro-ingénierie

1.4.1 Rétro-ingénierie d'application web

Le Tableau 1.1 propose une synthèse rapide des principaux outils de rétro-ingénierie d'application web et met en lumière leurs limites à l'aide des informations suivantes :

- Le type d'analyse (statique, dynamique).
- La date de première publication (certains outils ont plus qu'une publication).
- Les types de diagrammes générés (classe, séquence, état, cas d'utilisation, déploiement).

- Le niveau de détail présent dans les diagrammes en général. Faible affiche les fichiers, moyen montre le contenu des fichiers et des relations simples entre eux tandis qu'élevé présente des relations plus complexes comme des connecteurs logiques, des fréquences d'invocations ou des appels de fonction.
- Le besoin d'interaction avec l'utilisateur (fournir un scénario d'exécution).
- La sensibilité face aux fonctionnalités Ajax.
- Une prise en compte des événements du DOM.
- La disponibilité de l'outil au meilleur de nos connaissances (si l'outil possède une page web où il peut être téléchargé).

Tableau 1.1 Comparatif des outils

Outil	**Reweb**	**Ware**	**WebUML**	**Wanda**	**FireDetective**	**DynaRIA**
Date de première publication	2000	2002	2004	2004	2010	2010
Analyse statique	X	X	X	X	-	-
Analyse dynamique	-	X	X	X	X	X
Diagramme de Classe	X	X	X	X	-	-
Diagramme de séquence	-	X	-	X	X	X
Diagramme d'état	-	-	X	-	-	-
Diagramme de cas	-	X	-	-	-	-
Diagramme de déploiement	-	-	-	X	-	-
Niveau de détail	Faible	Moyen	Élevé	Élevé	Élevé	Élevé
Nécessite l'interaction de l'utilisateur	-	X	-	X	X	X
Éléments Ajax	-	-	-	-	X	X
Événements du dom	-	-	-	-	X	X
Disponibilité	Non	Oui[3]	Non	Non	Oui[4]	Oui[5]

[3] http://wpage.unina.it/ptramont/downloads.htm

[4] http://swerl.tudelft.nl/bin/view/Main/FireDetective

[5] http://wpage.unina.it/ptramont/downloads.htm

- **Reweb (2000) :** C'est au début de l'an 2000 que les premières publications d'un outil de rétro-ingénierie pour site web ont été présentées (Ricca and Tonella 2000, Tonella 2000, Ricca and Tonella 2001). Cet outil d'analyse statique permet d'aider les développeurs à maintenir leur site web et de voir la structure de leur site web ainsi que son évolution à l'aide de différents diagrammes. Le modèle pour représenter les différentes vues est basé sur les publications de Conallen (Conallen 2003) ainsi que celles de RMM (Antoniol, Canfora et al. 2000). Le logiciel propose trois types de vue (haut niveau, détaillé, historique et évolutions) afin de voir sous des angles différents le site web (voir Figure 1.3). Il propose également un rapport pour comprendre d'un coup d'œil le contenu du site web. L'outil contient quelques limitations. Pour commencer, il permet de voir des pages statiques, mais il n'offre que des résultats partiels pour les pages dynamiques générées par les serveurs. De plus, il ne considère pas les pages qui sont à l'extérieur du serveur web comme des liens externes vers d'autres sites web. L'outil n'est actuellement plus disponible. Reweb est un outil versatile qui peut aider les développeurs lors de la maintenance et qui permet de voir l'évolution à travers le temps. Cependant, le niveau de détail proposé n'est pas très élevé.

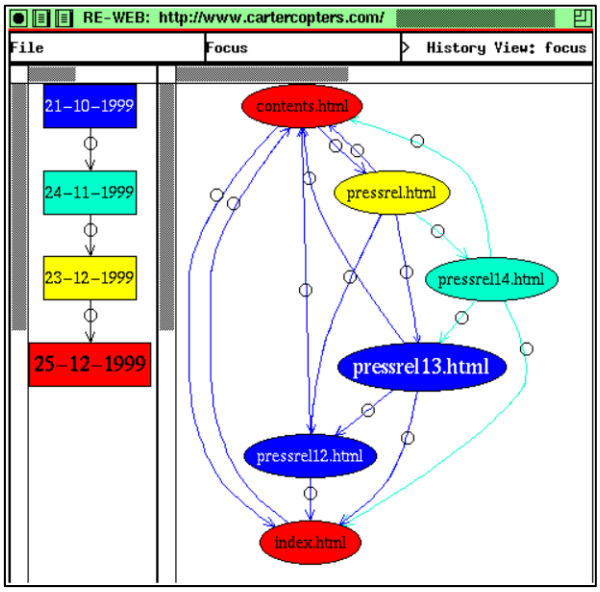

Figure 1.3 Historique et évolution
Tirée de (Ricca and Tonella 2000)

- **Ware (2002)** « Web Application Reverse Engineering » : Deux années après la publication de Reweb, un autre outil de rétro-ingénierie a été présenté. WARE est un outil d'analyse dynamique qui permet d'aider les développeurs à documenter les applications web (Di Lucca, Fasolino et al. 2002, Di Lucca, Fasolino et al. 2004). Il facilite leur maintenance et leur évolution en combinant l'analyse statique et dynamique. Il est possible avec WARE de générer des diagrammes de classe formés d'entités (p. ex.: Page, Formulaire) et de relations qui représentent les actions (p. ex.: « build », « redirect », « submit », etc.) (voir Figure 1.4). De plus, l'outil permet la génération de diagramme de cas (à l'aide d'une heuristique hiérarchique agglomérative) ainsi que de séquence pour voir les interactions entre les éléments de l'application en analysant les données provenant des formulaires web. Les diagrammes de WARE présentent une nette amélioration sur le niveau de détail comparativement à Reweb entre autres avec l'utilisation de stéréotypes et d'associations nommés. L'analyse supporte les langages suivants : ASP, PHP, HTML, JavaScript, VBScript, JScript et utilise l'extension UML « WAE ». Grâce à l'analyse dynamique, WARE peut détecter les pages qui sont générées par le serveur ou qui communiquent avec le serveur.

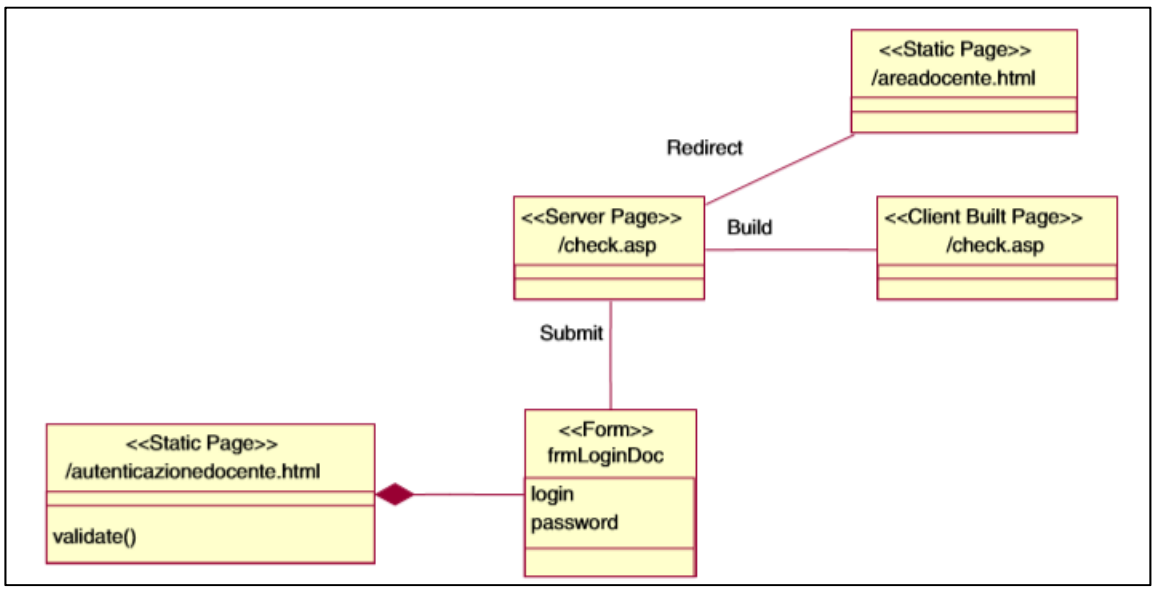

Figure 1.4 WARE, diagramme de classe
Tirée de (Di Lucca, Fasolino et al. 2004)

- **Webuml (2004) :** En 2004, l'outil WebUML qui fait partie d'une suite logicielle de test est proposé par (Bellettini, Marchetto et al. 2004). C'est un outil de rétro-ingénierie qui supporte les langages ASP, JSP et HTML. Contrairement aux autres outils, celui-ci utilise l'analyse statique et dynamique avec comme objectif d'aider le développeur dans la phase de test. Il est particulièrement utile pour les sites web possédant des éléments de logique d'affaire du côté client. Une autre caractéristique qui différencie cet outil est la technique d'analyse par mutation. WebUML génère un ensemble de scripts (mutants) pour chaque fichier afin de simuler la navigation d'un utilisateur. Chaque script va donc effectuer une simulation à l'aide de données choisies au hasard. Les résultats sont sauvegardés et les données redondantes sont éliminées. L'objectif de cette technique est de réduire le nombre d'interactions nécessaires par l'utilisateur afin d'extraire les informations lors de la rétro-ingénierie. L'outil génère ensuite un diagramme de classes (voir Figure 1.5) qui permet de décrire les composantes de la structure de l'application web avec un niveau de détail plus élevé que les outils précédents. Un diagramme d'état permet de visualiser la navigation. Les pages « asp » contenant du code HTML sont décomposées en fragments permettant de voir s'ils sont chargés de façon exclusive ou inclusive (p. ex.: « XOR ») pour mieux comprendre leur utilité.

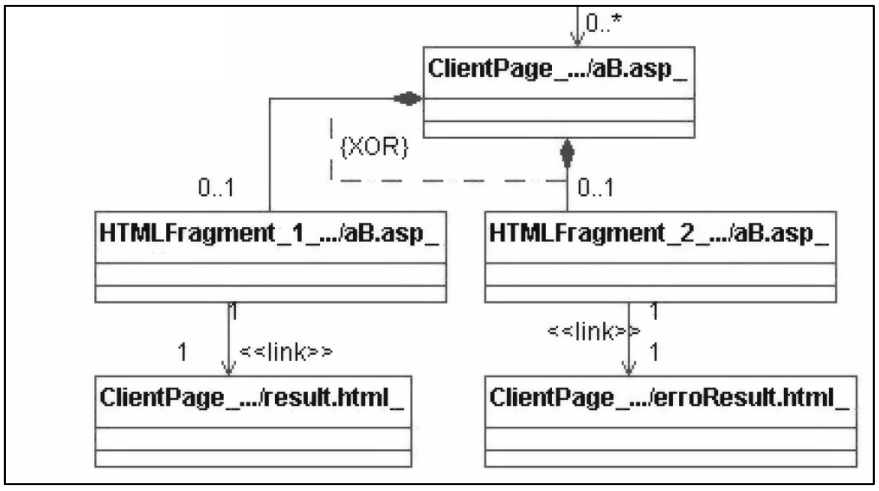

Figure 1.5 Webuml, diagramme de classes
Tirée de (Bellettini, Marchetto et al. 2004)

- **Wanda (juin 2004)** « Web Applications Dynamic Analyzer » : Wanda est un outil qui utilise l'analyse statique et dynamique afin de redécouvrir l'architecture d'applications web. Les analyses se concentrent entre autres sur les flux de données, les sessions, les bases de données, les fichiers et services web (Antoniol, Di Penta et al. 2004). WANDA extrait plusieurs informations utiles à propos de l'architecture de l'application web lors de l'analyse comme des métriques, sessions, "cookies" ainsi que les accès au niveau de la base de données et les fichiers. L'outil comporte quelques limitations au niveau de l'analyse. Par exemple, le contenu des bases de données n'est pas analysé. Une fois les informations récupérées, trois diagrammes UML sont générés soit de déploiement, de classes et de séquences. La Figure 1.6 montre le diagramme de classes avec le type et la fréquence d'invocation des accès entre les entités. Une intégration des outils WARE et WANDA (Di Lucca and Di Penta 2005) est réalisée en 2005. Elle permet de mieux exploiter leurs résultats et ajoute la production d'un diagramme de contrôle de flux des pages.

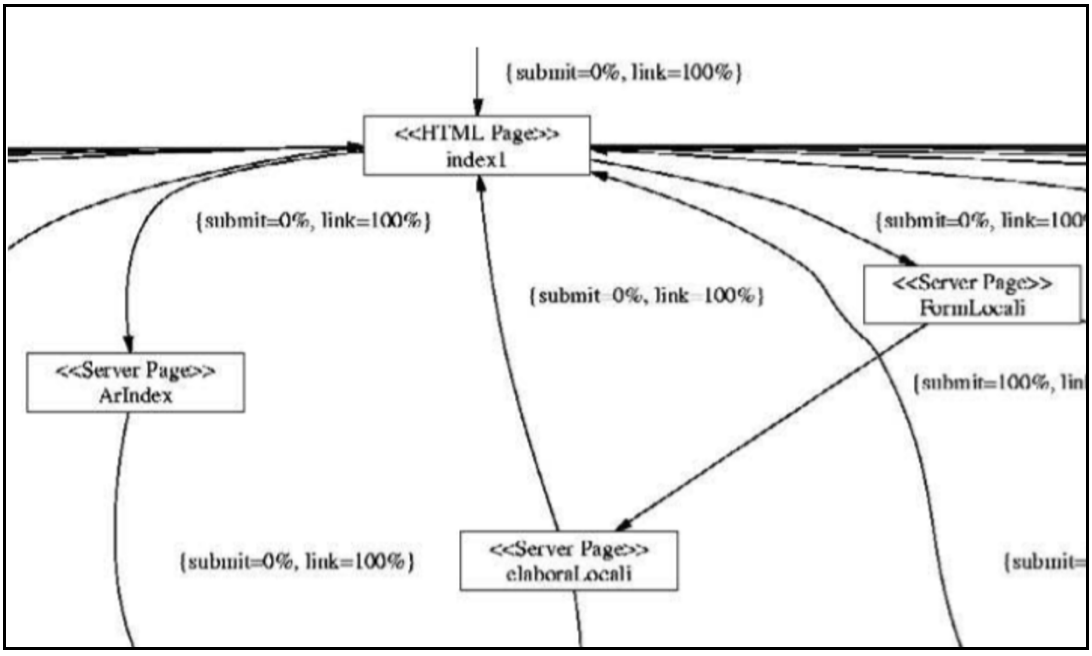

Figure 1.6 Wanda, diagramme de classe
Tirée de (Antoniol, Di Penta et al. 2004)

- **Fire Detective (2010) :** FireDetective se spécialise dans les appels AJAX (Matthijssen, Zaidman et al. 2010, Matthijssen and Zaidman 2011, Zaidman, Matthijssen et al. 2013). Il effectue une rétro-ingénierie à l'aide de l'analyse dynamique et est capable d'analyser les langages Java et JSP. C'est un outil qui permet d'aider à la compréhension des applications web au niveau des fonctions et des requêtes (p. ex.: origines des appels). Il enregistre les traces d'exécution en temps réel. Les niveaux de détail des informations extraites sont entre autres les requêtes pour les pages chargées, les requêtes serveurs AJAX et non-AJAX, les événements du DOM, les temps d'arrêt JavaScript et le chargement de modèle JSP. L'outil est composé de trois modules. Premièrement, une extension Firefox pour voir l'exécution de JavaScript du côté client ; ensuite, un module pour enregistrer les appels serveur (Java EE) et finalement, les données sont envoyées vers le module de visualisation. L'outil est divisé en 4 vues : la première montre une vue de haut niveau en forme d'arbre qui montre les appels. La deuxième vue affiche l'appel sélectionné avec un niveau de détail montrant les traces d'exécution. La troisième vue affiche le code qui a été exécuté pour la trace sélectionnée et la 4^{e} affiche le fichier où se trouve la trace (voir Figure 1.7).

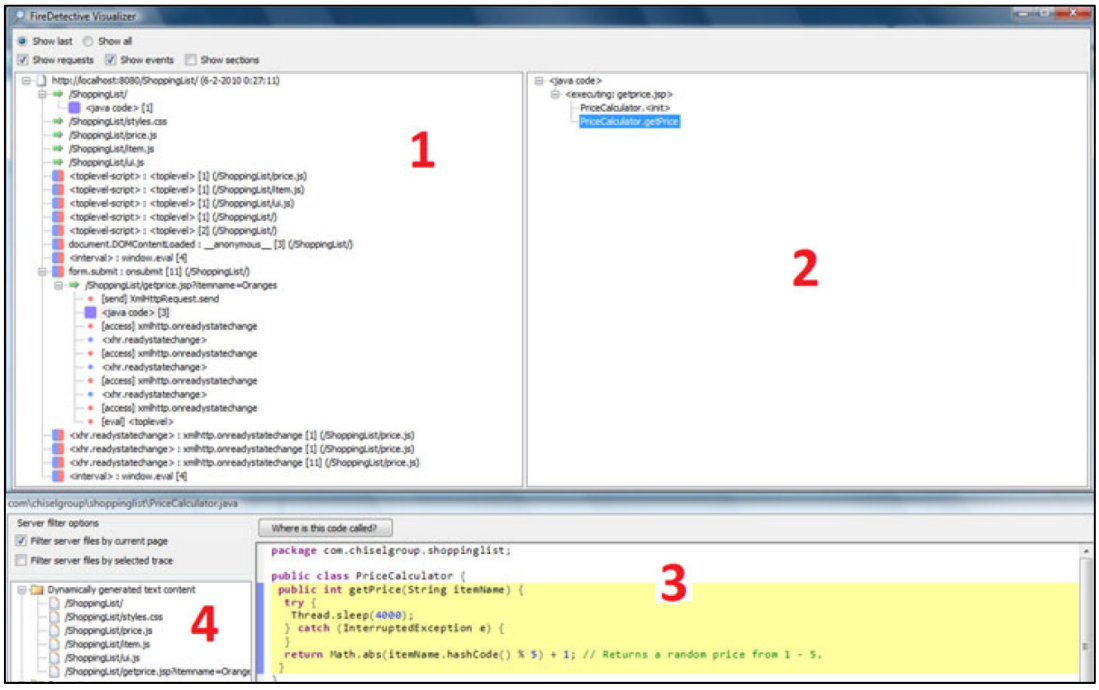

Figure 1.7 FireDetective, capture d'écran présentant les vues

Tirée de (Matthijssen, Zaidman et al. 2010)

- **DynaRIA (2010)** « Dynamic analysis of RIA » : DynaRIA est un outil qui aide à la compréhension d'applications internet riches (Amalfitano, Fasolino et al. 2010, Tramontana, Amalfitano et al. 2013, Amalfitano, Fasolino et al. 2014). Il utilise l'analyse dynamique pour extraire des informations d'une RIA comme les appels et réponses serveur, les événements du DOM ainsi que les changements d'attributs. Il permet de voir la couverture de code (fonctions exécutées), la détection d'erreurs (exception/message d'erreur) ainsi que de générer des diagrammes UML de séquence. Il est possible de voir les diagrammes de séquences avec deux niveaux de détail (voir Figure 1.8). DynaRIA permet d'enregistrer et de rejouer les sessions utilisateurs. Il aide à juger les qualités de maintenabilité et de testabilité des applications web. Pour ce faire, il utilise des métriques de grandeur comme le nombre total de lignes de code, le nombre de modules JavaScript, le nombre de fonctions par module et le nombre de lignes de code par fonction. Il utilise aussi des métriques de couplage comme les appels distincts de fonction, les appels de fichier externe, le couplage entre les modules et le couplage entre le serveur et un module. Finalement, il est possible de voir le nombre de changements survenues aux éléments du DOM en exécutant un module JavaScript.

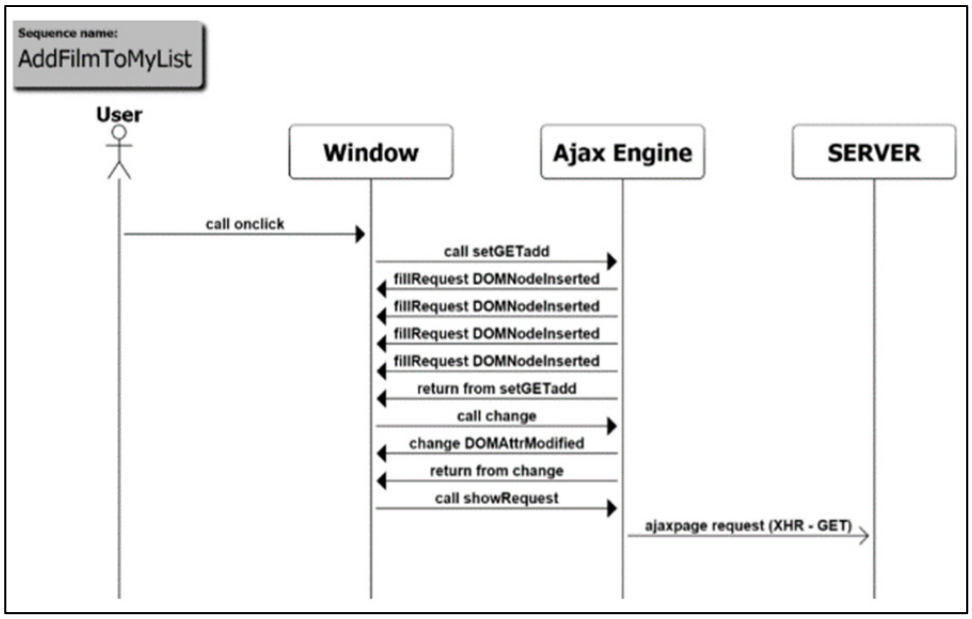

Figure 1.8 DynaRIA, premier niveau de détail
Tirée de (Amalfitano, Fasolino et al. 2014)

Il existe plusieurs outils de rétro-ingénierie d'applications web. Par contre, la plupart datent de plusieurs années ou bien ils se concentrent sur les langages serveur seulement. Un autre problème est que ces outils ne permettent pas de visualiser adéquatement les nouvelles technologies comme Ajax. Les auteurs Di Lucca, Di Penta et al. (2001) croient que le principal problème des outils de rétro-ingénierie est qu'ils ne modélisent pas bien les pages dynamiques et qu'ils n'utilisent pas l'extension UML « WAE ». Puisque les outils ne sont pas adéquats, les développeurs doivent trouver des alternatives. Une étude de Zaidman, Matthijssen et al. (2013) démontre que les développeurs dépendent encore beaucoup de la recherche par texte dans le code. Bref, certaines pistes sont mises de l'avant pour ce problème. Selon Canfora et Di Penta (2007), l'industrie devrait développer des outils de rétro-ingénierie plus robustes et mieux exploiter les recherches qui ont été faites à ce sujet. Pour leur part Ricca et Tonella (2000), expliquent que comme pour le développement des logiciels, la seule façon de produire des applications web fiables et de bonne qualité est que les développeurs utilisent de bonnes méthodologies et techniques de développement. De plus Hassan et Holt (2002), croient qu'il y a un réel besoin pour une nouvelle génération de méthodologies, technologies et outils afin d'aider les développeurs à maintenir et faire évoluer leur application web.

1.4.1.1 Limitations des outils

Certaines observations ont été faites suite à la revue des divers outils de rétro-ingénierie d'application web. Premièrement, il y a peu d'outils de rétro-ingénierie qui utilisent uniquement l'analyse statique. Pour la documentation et visualisation d'application web, il y a l'outil Reweb (Ricca and Tonella 2000, Tonella 2000, Ricca and Tonella 2001) qui date de 2000 et qui n'est plus disponible. Il n'inclue évidemment pas les évolutions des dernières années telles que les nouvelles fonctionnalités proposées par ES2015, l'utilisation d'Ajax et les nouveaux langages comme Node.js. Les outils plus récents tels que DynaRIA (Amalfitano, Fasolino et al. 2010, Tramontana, Amalfitano et al. 2013, Amalfitano, Fasolino et al. 2014) et Fire Detective (Matthijssen, Zaidman et al. 2010, Matthijssen and Zaidman 2011, Zaidman, Matthijssen et al. 2013) se concentrent sur la compréhension lors de l'exécution du code à l'aide de l'analyse dynamique plutôt que la compréhension de la structure de l'application. De plus, ils offrent très peu de solutions efficaces pour se mettre à l'échelle selon la taille de l'application web et pour réduire le nombre d'éléments. Par exemple, Reweb propose simplement une vue peu détaillée de la structure (par dossier) alors que Ware (Di Lucca, Fasolino et al. 2002, Di Lucca, Fasolino et al. 2004) propose quant à lui de regrouper les pages par cas d'utilisation à l'aide d'un expert. Fire Detective permet l'utilisation de filtres pour cacher les appels de librairies et DynaRIA regroupe tous les appels dans 3 catégories, soit « Windows », « Ajax Engine » et « Server » afin de cacher la complexité. Aucun ne propose des fonctions avancées pour réduire efficacement le nombre d'informations selon les désirs de l'utilisateur. Il n'existe pas suffisamment d'outils qui utilisent l'analyse statique à des fins de documentation. Ces outils pourraient servir à des utilisateurs qui désirent en apprendre sur un système dans un court délai sans l'aide d'un expert. Certaines décisions doivent être prises afin que l'approche puisse être mise à l'échelle selon la taille de l'application et le niveau de détail recherché par l'utilisateur. Puisque l'analyse statique permet d'extraire un niveau de détail élevé sur le système, il faut une technique pour garder les informations pertinentes et éliminer celles qui ne le sont pas.

1.4.2 Rétro-ingénierie JavaScript

JavaScript aura bientôt 21 ans et pourtant il existe très peu de publications sur la rétro-ingénierie de ses applications. Une simple recherche sur « Compendex » comportant les termes « JavaScript reverse engineering » retourne 23 résultats alors que Java en retourne 488[6]. Les outils et approches de rétro-ingénierie JavaScript sont donc rares. Une des raisons probables est le fait que JavaScript est un langage dynamique. Effectuer la rétro-ingénierie des langages dynamiques tels que JavaScript est une tâche complexe et incertaine remplie d'ambigüités.

Certaines études se sont concentrées sur des propositions efficaces d'analyse statique ou hybride pour JavaScript avec des préoccupations de compréhension (Feldthaus, Schafer et al. 2013) ou de sécurité (Wei and Ryder 2013). Les techniques proposées peuvent être sensibles à l'ordre des déclarations « flow sensitive », sensibles aux chemins et exécutions des branches conditionnelles ou bien sensibles au contexte (contexte d'appel de la fonction cible). Les analyses par point, qui étudient l'ensemble des objets vers lequel pointe une variable ou une propriété, sont l'une des techniques les plus utilisées pour JavaScript (Jang and Choe 2009) (Wei and Ryder 2014). Des techniques de flux de données ont aussi été proposées, en particulier dans (Feldthaus, Schafer et al. 2013). Les auteurs ont identifié la construction efficace de graphes d'appel comme un problème clé pour la proposition d'outils de développement utiles pour JavaScript. Dans cette perspective, ils ont également souligné la lenteur et le manque de mise à l'échelle de plusieurs techniques avancées d'analyse statique. Ils ont proposé une technique rapide basée sur les champs qui se concentre sur les fonctions et ignore les fonctionnalités dynamiques de JavaScript tels que l'accès dynamique. Plus récemment Humberto Silva, Ramos et al. (2015) ont conduit une étude sur l'utilisation de classes dans les systèmes JavaScript. Il y a donc un intérêt grandissant de la communauté ces dernières années pour l'analyse et la rétro-ingénierie JavaScript.

---

[6] En date du 10 mai 2015

- **TAJS (2010-2011) :** TAJS est un cadre de travail qui fait de l'analyse statique de JavaScript en utilisant le flux de contrôle et l'inférence de type afin d'aider les développeurs à détecter des erreurs potentielles dans leur code (Jensen, M et al. 2010) (Jensen, Madsen et al. 2011). L'outil aide entre autres à trouver les zones de code mortes, les erreurs d'orthographe ainsi qu'a générer des graphes d'appel. La principale technique innovatrice utilisée par l'outil est le « Lazy propagation » pour l'analyse de type afin d'améliorer la rapidité des traitements.

- **Extension WALA (2012) :** Le papier de (Sridharan, Dolby et al. 2012) présente une extension à « Watson Libraries for Analysis » (WALA) et son analyse par point afin de générer des graphes d'appel de JavaScript. L'extension utilise une technique afin d'aider à régler les problèmes reliés à la nature dynamique de JavaScript. Alors que l'analyse standard est lente et offre des résultats imprécis, la technique proposée utilise des heuristiques de corrélation qui aide à améliorer les performances.

- **Call Graph (2013) :** Le papier (Feldthaus, Schafer et al. 2013) propose une approche approximative de type « field-based flow analysis » afin de créer des graphes d'appel. Cette approche a pour but d'offrir un meilleur support de JavaScript dans les IDE (*Integrated Development Environment).* Par exemple, lorsque l'utilisateur veut accéder à la déclaration d'une fonction en cliquant sur l'un de ces appels. Deux variantes sont proposées dans l'article : une optimiste et une pessimiste. Elles offrent des niveaux d'efficacités et de performance variables. Les deux réagissent bien à la mise à l'échelle et fonctionnent avec des programmes de grande taille.

- **Extension de JSAI (2013) :** Un outil d'analyse statique (extension de JSAI) a été développé en utilisant une technique de raffinement de type (Kashyap, Sarracino et al. 2013). Cet outil a pour objectif de détecter de potentielles erreurs d'exception de type dans les systèmes JavaScript. Le raffinement de type analyse les valeurs hypothétiques au cours de l'exécution. Les résultats démontrent que la technique

proposée permet d'observer, dans le meilleur des cas, une augmentation de la précision de 86% tout en ayant un impact faible sur la performance.

- **JSBAF (2013-2014) :** JSBAF est un autre outil basé sur la librairie WALA qui utilise l'analyse par point de JavaScript (Wei and Ryder 2014). Contrairement à l'extension précédente, celle-ci combine des techniques d'analyse statique et dynamique afin de garder la trace des appels et modéliser les changements dynamiques des objets, variables et fonctions. L'outil a également été utilisé dans une étude sur l'identification de failles de sécurité causées par la violation de l'intégrité des données (Wei and Ryder 2013). Les résultats démontrent que l'outil permet de résoudre plus de cas et d'obtenir une meilleure précision en combinant l'analyse statique et dynamique.

1.4.3 Rétro-ingénierie Node.js

Il est maintenant le temps de regarder les modules d'analyse et de visualisation Node.js qui aident à la compréhension de la structure d'une application web. Voici les modules d'analyses et de visualisations existants.

Modules d'analyse

Dans la rétro-ingénierie d'application web, JavaScript est l'un des langages les plus importants à analyser. C'est un langage primordial dans le monde du web qui peut être utilisé tant du côté client que serveur. Pour analyser ce code, l'utilisation d'un analyseur syntaxique peut s'avérer très utile afin d'extraire sa structure sous forme d'un arbre. Il est possible de s'en servir de plusieurs façons. Il peut parfois être plus facile de comprendre le fonctionnement d'un programme ou d'une fonction sous forme d'un arbre. C'est d'ailleurs avec cette structure que fonctionnent plusieurs compilateurs. Cela peut servir pour réaliser des comparaisons entre fichiers et algorithmes, simplifier ou rendre le code plus performant.

Esprima[7] (disponible depuis 2011) est un module qui sert à produire un arbre syntaxique à partir du code source JavaScript (*Voir* Figure 1.9). Il sert de base pour plusieurs autres outils d'analyses de code. C'est un module fiable et performant qui peut s'exécuter même lorsque le code analysé est invalide. Une fois sous forme d'arbre syntaxique (AST), il est possible de performer une multitude d'actions à l'aide de modules supplémentaires, tels que Escodegen[8] qui permet de générer du code JavaScript et Escomplex[9] qui est un module permettant d'extraire des métriques de complexité logicielle. Plusieurs modules permettent également de visualiser des applications Node.js.

```
{
    type: 'Program',
    body: [
        {
            type: 'VariableDeclaration',
            declarations: [
                {
                    type: 'AssignmentExpression',
                    operator: =,
                    left: {
                        type: 'Identifier',
                        name: 'answer'
                    },
                    right: {
                        type: 'Literal',
                        value: 42
                    }
                }
            ]
        }
    ]
}
```

Figure 1.9 Arbre syntaxique du code suivant : « var answer = 42; ».
Tirée de http://esprima.org/doc/index.html

Modules de visualisation

- **D3.js (2010) :** Une des meilleures façons de visualiser les données recueillies lors d'une analyse est à l'aide de la librairie D3.js (D3. 2014). Il s'agit d'une librairie qui

[7] http://esprima.org/

[8] https://github.com/estools/escodegen

[9] https://github.com/philbooth/escomplex-js

permet d'afficher et manipuler des données sous diverses formes de représentations graphes (Murray 2013). Il s'agit d'une librairie fiable et populaire dans le domaine du web, car elle est versatile. Il est possible de créer toutes sortes de représentations et graphes : des diagrammes de force, des histogrammes, des cartes géographiques, des diagrammes de Voronoï, des matrices et autres (*Voir* Figure 1.10).

- **Dependo (2012) :** Dependo est un module qui se spécialise dans la visualisation des dépendances entre fichiers JavaScript à l'aide d'une représentation graphique de D3.js (*Voir* Figure 1.12). Ce module est conçu pour visualiser seulement les fichiers JavaScript utilisant les librairies « CommonJS » ou « AMD ». Le sens des dépendances est affiché à l'aide d'une flèche. Il est possible de déplacer les nœuds dans l'espace. Les fichiers HTML et CSS ne sont pas représentés dans leurs graphes.

  Colony (2012) : Colony est un autre module de visualisation similaire à Dependo. Il s'agit d'un module qui permet de voir les dépendances d'un projet Node.js et qui utilise D3.js pour la visualisation. La Figure 1.11 montre un exemple de représentation graphique de la structure d'une application web et les liens entre les fichiers dans un projet Node.js. Contrairement à Dependo, les modules faisant partie des éléments représentés dans le graphe sont des fichiers et des modules. Colony se limite seulement à la visualisation des éléments JavaScript. Les fichiers HTML et CSS ne sont pas représentés. Le graphe créé est interactif et permet de voir le nom et les relations des fichiers en passant au-dessus d'un nœud.

- **AssetViz (2013) :** Le dernier module Node.js révisé est AssetViz. C'est un module qui utilise les liens trouvés par AssetGraph pour générer une représentation graphique d'une application Node.js avec la libraire D3.js (*Voir* Figure 1.13). Le graphe généré montre les fichiers et les différents liens entre ceux-ci. Il s'agit de la représentation graphique de la structure d'une application qui est la plus réaliste parmi les modules. Par contre, bien qu'AssetViz affiche le type de liens entre les fichiers, il n'affiche pas la direction des relations.

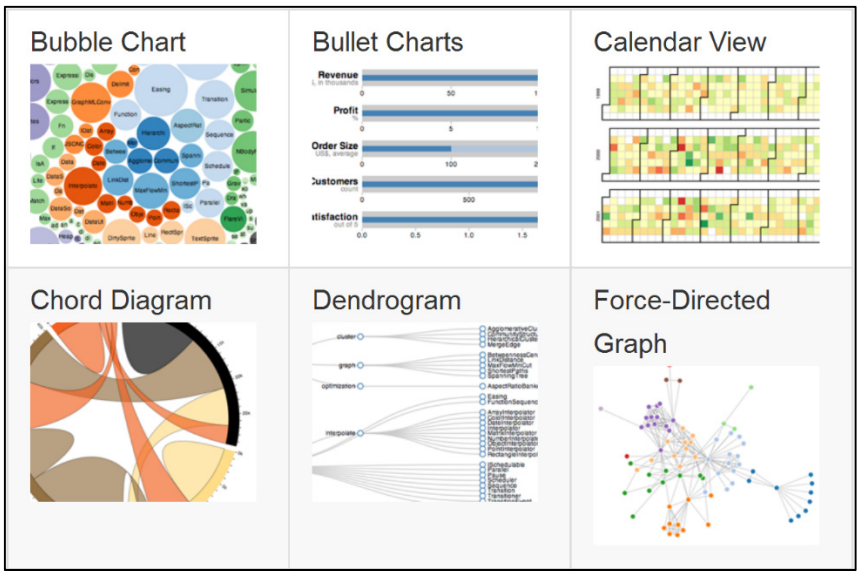

Figure 1.10 Exemples de représentations créées avec D3.js
Tirée de https://github.com/mbostock/d3/wiki/Gallery

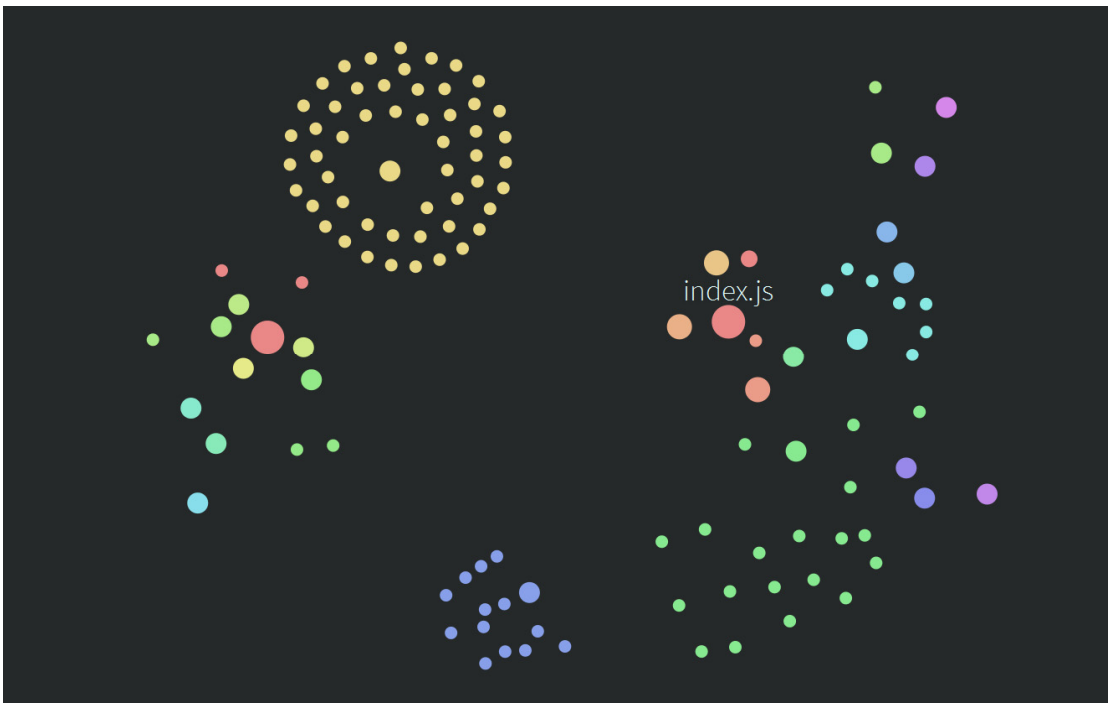

Figure 1.11 Exemple de Colony
Tirée de http://hughsk.io/colony/

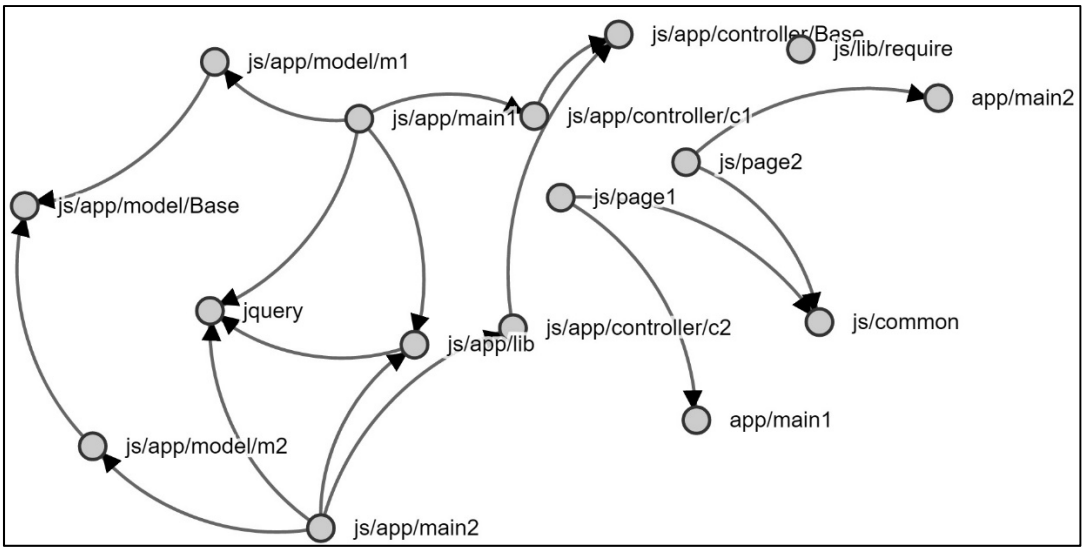

Figure 1.12 Exemple de graphe généré par l'outil Dependo.
Tirée de http://auchenberg.github.io/dependo/example

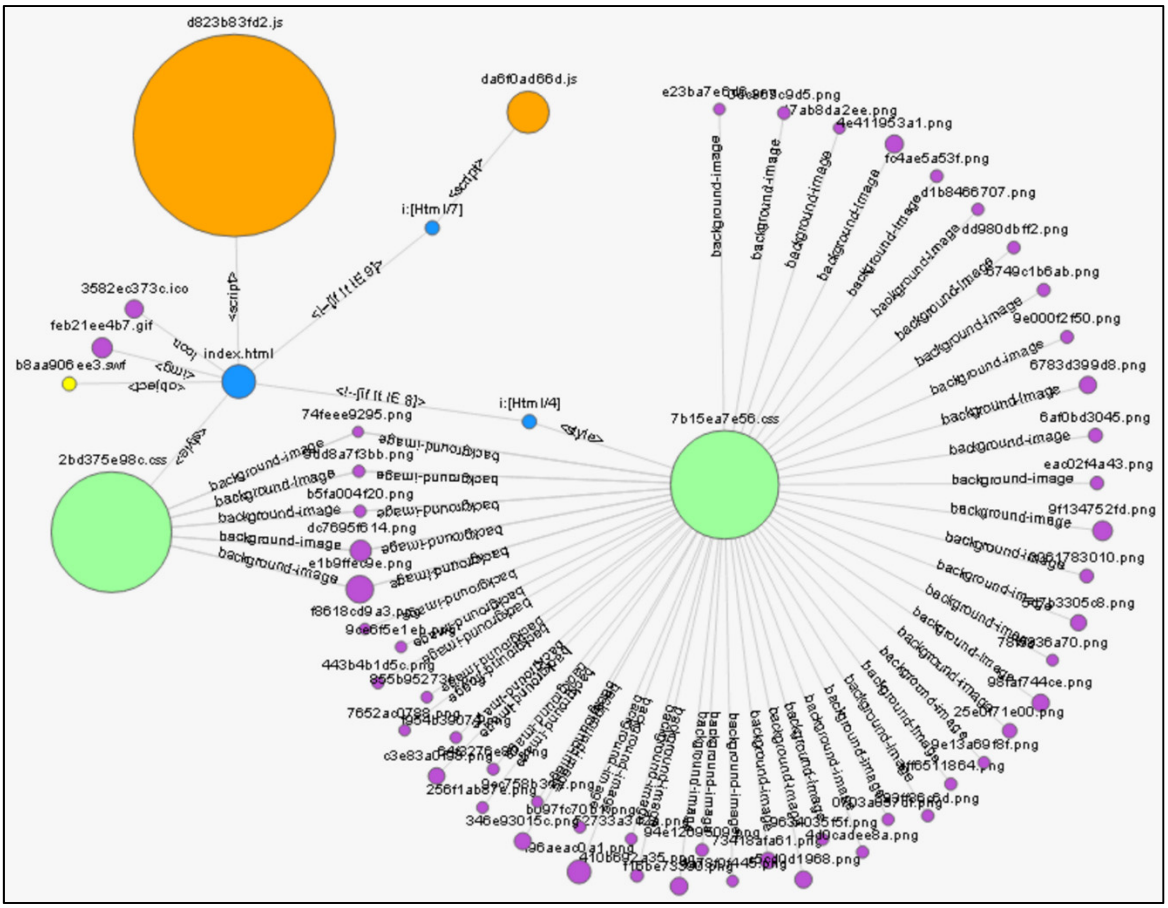

Figure 1.13 Exemple de graphe généré par AssetViz
Tirée de https://www.npmjs.org/package/assetviz

1.4.4 Conclusion sur les outils et approche

En résumé, il existe plusieurs outils de rétro-ingénierie pour applications web. La plupart datent de plusieurs années et certains ne sont plus disponibles. Il n'y a qu'un outil qui se concentre uniquement sur l'analyse statique (Reweb). Les outils récents se concentrent plutôt sur l'analyse dynamique (FireDetective et DynaRIA).

Des études ont été menées sur l'analyse statique JavaScript sensible à l'ordre des déclarations, aux chemins et au contexte, mais la technique la plus populaire est celle de l'analyse par point (Wei and Ryder 2014). L'approche la plus intéressante est proposée par (Feldthaus, Schafer et al. 2013) utilisant une technique sensible aux champs.

Finalement, certains modules de rétro-ingénierie Node.js ont été présentés. Les modules d'analyse « Esprima » permet de transformer le code source en arbre syntaxique pour pouvoir performer des analyses statiques de JavaScript. Il existe également plusieurs modules de visualisation qui présentent des degrés variables d'information. La plupart sont basées sur la librairie d3.js qui permet de générer des diagrammes. Dependo est le module avec le plus de potentiel et le plus plaisant esthétiquement alors qu'AssetViz montre la plus grande variété d'éléments.

CHAPITRE 2

ANALYSE STATIQUE DE JAVASCRIPT : UNE APPROCHE VIABLE ?

JavaScript est un langage clé du web qu'on ne peut ignorer lorsqu'on essaie de faire de la rétro-ingénierie d'applications web. C'est un langage très flexible qui offre plusieurs fonctionnalités dynamiques pouvant a priori rendre une analyse statique impraticable. Cependant, s'il s'avérait que ces fonctionnalités dynamiques sont utilisées de façon modérée, l'analyse statique pourrait donner des résultats satisfaisants et aider effectivement un développeur à comprendre la structure d'un projet.

Plusieurs études ont été menées sur l'utilisation des fonctionnalités dynamiques de JavaScript. Parmi ces études, les auteurs de (Richards, Lebresne et al. 2010) ont enregistré les traces d'interactions significatives avec 100 sites web parmi les plus populaires selon le classement d'Alexa[10], et ce afin d'évaluer empiriquement un certain nombre de suppositions à propos de la nature dynamique de JavaScript. Leurs conclusions sont les suivantes :

- le monomorphisme [11] de site d'appel a été trouvé en moyenne 81% du temps.
- La variadicité des fonctions est relativement faible avec moins de 7% des fonctions appelées possédant un nombre d'arguments différents du nombre d'arguments déclarés.
- L'utilisation du constructeur « eval » se situe entre une douzaine jusqu'à plus de 400 occurrences, divisées entre des situations triviales telles que la notation d'objet (JSON) et des situations plus complexes.

---

[10] Le classement d'Alexa fournit des statistiques sur le traffic des sites web les plus populaires au monde.

[11] On parle de monomorphisme lorsqu'un site d'appel renvoie toujours vers le même corps de fonction, quel que soit le scénario considéré. Ceci est en contraste avec du polymorphisme où plusieurs corps de fonction peuvent être liés au même site d'appel.

Une étude empirique de 2011 (Richards, Hammer et al. 2011) s'est intéressée à l'utilisation des constructeurs JavaScript permettant d'exécuter des chaines de caractères comme si c'était du code : « eval » et « new Function ». Ils ont analysé 10000 des sites les plus populaires et ils ont trouvé que 50% à 82% de ces sites utilisent « eval ». Selon eux, c'est un mal nécessaire, car cela permet des mécanismes qui sont difficiles à émuler sans ces fonctions.

Dans la même veine (Park, Lee et al. 2013) ont évalué la présence du constructeur « with » (portant sur la possibilité de portée dynamique) pour 898 des sites les plus populaires. Ils ont trouvé que 118 sites avaient au total 543 utilisations statiques de « with ». Plus récemment (Humberto Silva, Ramos et al. 2015) ont conduit une étude sur l'utilisation des classes dans 50 systèmes JavaScript. Ils ont trouvé 787 classes dans 37 systèmes sur 50 (74%).

Les études présentées plus haut ont permis d'obtenir des données empiriques quant à la fréquence d'utilisation des possibilités dynamiques offertes par JavaScript. Dans le même esprit et dans le but de vérifier la viabilité d'une analyse purement statique pour la rétro-ingénierie de JavaScript, nous avons conduit une étude empirique pour obtenir des réponses quant à :

1. L'utilisation en général des fonctionnalités dynamiques tels que les « eval », « new Function ».
2. La difficulté de récupération des liens entre fichiers causés par les mécanismes d'importation et exportation.
3. La difficulté de récupération des liens entre méthodes (résolution de sites d'appels) en associant les corps de fonctions (les déclarations) aux sites d'appels.

2.1 Échantillon de systèmes

Afin d'expérimenter les limites de l'analyse statique, 80 systèmes ont été sélectionnés, lesquels sont considérés comme représentatifs du vaste écosystème de JavaScript. À peu près le 1/6^{ième} de ces systèmes viennent d'autres études, entre autres (Feldthaus, Schafer et al.

2013) et (Park, Lee et al. 2013). Ces systèmes ont ensuite été complémentés afin d'avoir six catégories différentes pour pouvoir déceler s'il y a des tendances qui existent dans ces catégories.

Il y a tout d'abord quatorze cadres de travail qui aident à structurer le code tels que Angular, Backbone, Express et Prototype. Ensuite, les systèmes incluent quinze librairies qui offrent des fonctionnalités de bas niveau comme JQuery, React, Underscore et D3. De plus, il y a six plateformes, telles que des sites communautaires, forums et blogues incluant Reddit, Ghost Blog, Strider et NodeBB. On compte également dix-huit utilitaires incluant le compilateur Babel et Eslint. Finalement, les systèmes choisis comptent dix jeux tels que Pacman, Pong, Tic-tac-toe ainsi que dix-sept applications web telles que des éditeurs de code comme Ace, Bracket et des applications de gestion comme Full Calendar. Ces systèmes ont également été sélectionnés afin de représenter différentes tailles. Le nombre moyen de fichiers JavaScript est de 55 (Jasmine) avec un minimum de 1 (Underscore) et un maximum de 785 (Bracket). Le nombre total d'appels est entre 46 (CO) et 41905 (Bracket) alors que le nombre de déclarations de fonction varie entre 21 (Modernizr) et 20862 (Countly). De plus, le nouveau standard ES2015 est représenté par 14 systèmes (p. ex.: Acorn, HandleBars, etc.) et les systèmes sont divisés en 37 applications clients, 20 serveurs et 23 qui sont clients et serveurs (Voir ANNEXE VII).

2.2 Analyses préalables sur la viabilité de JavaScript

Pour répondre à nos questions, nous avons effectué une première série d'analyses qui consistent essentiellement à recenser les situations dans lesquelles les fonctionnalités dynamiques de JavaScript pourraient être problématiques pour une analyse statique.

2.2.1 Utilisation de fonctions dynamiques

Une analyse a été effectuée sur l'utilisation de certains constructeurs dynamiques (eval, apply, call, with et new Function).

Tableau 2.1 Occurrences des constructeurs dynamiques

	Eval	Apply	Call	With	new Function
Somme	48	2376	3592	3	13
Minimum	0	0	0	0	0
Quartile 1	0	2	5.5	0	0
Quartile 2	0	16.5	17	0	0
Quartile 3	2.5	33.5	59	3	1
Max	8	248	465	3	3
Moyenne	0.6	29.7	44.9	0.038	0.16
Std Deviation	1.392839	46.64	72.08	0.33	0.49

Tableau 2.2 Occurrences des constructeurs dynamiques en pourcentage

	Eval %	Apply %	Call %	With %	new Function %
Minimum	0	0	0	0	0
Quartile 1	0	0.49	0.47	0	0
Quartile 2	0	1.36	2.23	0	0
Quartile 3	0.068	2.21	3.75	0.12	0.03
Max	0.408	9.52	28.57	0.12	0.37
Moyenne	0.0255	1.7	2.69	0	0.01
Std Deviation	0.070349	1.77	3.58	0.01	0.05

Le Tableau 2.1 résume les données sur l'occurrence des constructeurs étudiés dans les 80 systèmes. Les données sur le nombre d'occurrences détectées (somme) ainsi que des statistiques descriptives « seven-number sumary » ont été calculés pour tous les constructeurs étudiés. De plus, les pourcentages des occurrences (voir Tableau 2.2) sont comparés au nombre total d'appels excluant les appels aux fonctions natives. Seulement 3 occurrences du constructeur « with » ont été trouvées dans les systèmes, ce qui suggère que leur utilisation est effectivement obsolète. Les résultats suggèrent également que l'utilisation du constructeur « eval » est beaucoup moins importante (22,5% des 80 systèmes utilisent « eval ») que ceux reportés il y a 4 ans (Richards, Hammer et al. 2011) qui avait trouvé qu' « entre 50% et 82% des 10000 sites web les plus populaires utilisent « eval » ». Par contre, lorsque l'on considère les applications web dans nos données, 7 d'entre elles utilisent des « eval ». Cela donne un pourcentage de 41% pour les applications web comparativement à 17% pour le reste des systèmes. Le constructeur « New Function » est très peu utilisé avec seulement 13

occurrences au total, réparties dans 10 systèmes ; il n'est pas relié à une catégorie particulière. Sur une note plus qualitative, près de 15% des utilisations de « eval » étaient reliés au traitement JSON.

```
JSON.decode = function(string, secure){
if (!string || typeOf(string) != 'string') return null;
if (secure == null) secure = JSON.secure;
if (secure){
   if (JSON.parse) return JSON.parse(string);
   if (!JSON.validate(string)) throw new Error('JSON could not decode
   the input; security is enabled and the value is not secure.');
   }
   return eval('(' + string + ')');
};
```

La situation est très différente pour les « call » et « apply », avec un nombre absolu au-dessus de 1000. Ils représentent de 2 à 3% des sites d'appels en moyenne. Ces constructeurs ne devraient pas être ignorés puisqu'ils peuvent être gérés statiquement dans la plupart des cas. En termes d'occurrences, « call » est beaucoup plus utilisé que « apply ». Leur utilisation est cependant répartie équitablement : 85% des systèmes utilisent « call » et 86% utilisent « apply ».

```
var min = Math.min.apply(null, numbers);
```

Pour les études suivantes, la résolution d'appel naïve basée par nom des fonctions « call » et « apply » est modifiée afin de permettre la résolution de ces appels. Dans l'exemple plus haut, le nom de l'appel serait donc « Math.min » au lieu de « Math.min.apply ».

2.2.2 Dépendance entre fichiers

Une analyse a été effectuée pour les 80 systèmes afin de connaitre les habitudes des développeurs sur l'utilisation des constructeurs « require » et « define » qui sont les standards pour lier les fichiers JavaScript ensembles (voir ANNEXE VIII). Plus précisément, nous

voulons savoir quel sont les types de paramètres utilisées par ces constructeurs car certains (tableau, variables et fonction) sont plus difficiles à analyser de façon statique.

Tableau 2.3 Distribution du constructeur « require »

	Littéral	**Variable**	**Littéral et Variable**	**Tableau**	**Fonction**
somme	9324	49	45	5	8
min	0	0	0	0	0
q1	0	0	0	0	0
q2	1.5	0	0	0	0
q3	51.5	2	2	2.5	1
max	2537	10	21	3	4
moyenne	116.55	0.6125	0.5625	0.0625	0.1
std	381.89	1.6	2.43	0.39	0.48

Tableau 2.4 Distribution du constructeur « define »

	Littéral	**Variable**	**Littéral et variable**
somme	1371	0	0
min	0	0	0
q1	0	0	0
q2	0	0	0
q3	30	0	0
max	632	0	0
moyenne	17.13	0	0
std	82.55	0	0

Contrairement aux constructeurs « import » introduits dans ES2015, « require » et « define » permettent l'utilisation d'arguments autres qu'un simple littéral. Par contre, notre analyse suggère que c'est rarement le cas en pratique (voir Tableau 2.3 et Tableau 2.4). Aucune occurrence n'a été trouvée où « define » est utilisé avec une variable, une fonction ou un tableau. Pour ce qui est de « require », ces cas arrivent, mais rarement. Des 80 systèmes, 16 utilisent des variables (20%) et 13 utilisent un mélange de variable et littéral (16,2%). Comme exemple, le système Nodebb implémente ses propres fonctions de dépendance en utilisant des variables. Il utilise également des combinaisons de variables et littéraux.

```
Plugins.requireLibrary = function(pluginID, libraryPath) {
      Plugins.libraries[pluginID] = require(libraryPath);
      Plugins.libraryPaths.push(libraryPath);
};
```

```
function requireModules() {
      …
      modules.forEach(function(module) {
            Namespaces[module] = require('./' + module);
      });
}
```

Il y a 5 systèmes (6,25%) dans lesquels des fonctions sont utilisées comme paramètre au constructeur « require » et 3 systèmes (3,75%) utilisent des tableaux. Un exemple d'utilisation de fonction a été trouvé dans le système Cylon et un exemple d'utilisation d'un tableau dans le système Ace.

```
// convenience function to require modules in lib directory
global.lib = function(module) {
  return require(path.normalize("./../lib/" + module));
};
```

```
function load(url, module, callback) {
    net.loadScript(url, function() {
        require([module], callback);
    });
}
```

Il y a 25% des systèmes qui utilisent des fonctionnalités dynamiques dans « require » dont 5 sont des utilitaires, 5 des librairies, 4 des applications, 3 des cadres de travail, 2 des jeux et une plateforme. Tous ces systèmes contiennent du code côté serveur et 90% contiennent du code côté client. Les systèmes les plus problématiques sont Strider (23 occurrences), Clientsapp (14 occurrences) et Bracket (11 occurrences). Des constructeurs « define » sont présents dans 12 systèmes (15%), soit 3 utilitaires, 3 librairies, 3 jeux, 2 applications et 1 cadre de travail. Tous ces systèmes contiennent du code client et 75% de ces systèmes contiennent du code serveur.

En général, ces résultats suggèrent que la découverte de dépendance entre fichiers ne pose pas de problème majeur à l'analyse statique. Le quart des systèmes présente des cas potentiellement problématiques, mais seulement 1% de tous les appels de dépendance qui n'utilisent pas des littéraux comme argument.

2.2.3 Résolution des sites d'appels

L'une des plus grandes difficultés de l'analyse statique est la résolution des sites d'appels. Cette activité consiste à relier un appel de fonction à l'emplacement où il a été déclaré. Cela sert entre autres pour les IDE (Eclipse) offrant la fonctionnalité de sauter de l'appel de la fonction à la déclaration de cette fonction.

En utilisant l'analyse statique, une façon naïve de résoudre des sites d'appels est en utilisant la technique par nom. Une liste de toutes les déclarations de fonctions à travers le système est assemblée. En utilisant le nom de l'appel, on cherche tous les sites d'appels existants sous le même nom. Chaque appel est relié à une liste de possibilité de site d'appels (candidats). S'il n'y a qu'une possibilité alors il s'agit d'un cas sans ambiguïtés. C'est d'ailleurs l'approche utilisée dans Eclipse Javascript Development Tools JSDT.

Une analyse a été effectuée pour les 80 systèmes sur la résolution des sites d'appels. En moyenne, à travers tous les systèmes, 16,45% des appels invoquent des fonctions natives, 70,09% possèdent au moins un candidat de déclaration et 13,46% des appels ne peuvent pas être résolus (aucun nom de fonction ne correspond à l'appel). Pour tous les appels où il existe au moins un candidat nous avons calculée la moyenne de candidats par site d'appel (voir Tableau 2.5 et Tableau 2.6).

Tableau 2.5 Moyenne de candidats par site d'appel

	Site d'appels
1 candidat	61.50%
2 candidats	16.71%
3 candidats	7.12%
4 candidats	3.93%
5 candidats	2.33%
6+ candidats	8.41%

Tableau 2.6 Statistiques descriptives sur la résolution sans ambiguïtés (moyenne de site d'appel avec 1 candidat trouvé par système).

	Site d'appels
Min	22,05%
q1	47,43%
q2	60,11%
q3	73,20%
Max	100%
moyenne	61,50%
std	0.186

Les appels intra projet sont définitivement difficiles à résoudre. En moyenne, à peu près 61% des appels correspondent à un candidat unique avec une technique naïve basée sur le nom des fonctions. Il y a 3 systèmes qui ne comptent que des sites d'appels monomorphique avec 100% de candidats uniques soit : 3dModel, Gulp et Co. Le système BrowseNpm quant à lui est celui avec le plus d'ambiguïtés pour la résolution des sites d'appels avec seulement 22,05% de candidats uniques.

2.3 Évaluation de l'ambigüité pour la résolution des sites d'appels : la démarche

Les résultats des premières analyses démontrent les limites d'une technique naïve basée sur les noms. Il subsiste clairement des ambigüités pour un pourcentage important des sites d'appel. Cependant, une analyse même superficielle des résultats révèle des possibilités de réduction de cette ambigüité. En effet, pour un site d'appel et sa liste de candidats potentiels associée, on peut se rendre compte que certains candidats peuvent être éliminés avec un très

haut niveau de certitude. Ainsi, il est hautement improbable qu'un appel renvoie vers une fonction définie dans un fichier différent de celui de l'appel et n'y étant aucunement lié. Nous avons donc opté pour une approche permettant d'éliminer ces candidats improbables à l'aide de filtres afin de ne proposer que les candidats les plus plausibles pour un appel. Pour ce faire, il faut décortiquer les informations provenant du code source. Ces informations sont entre autres la structure du code telles que le positionnement des appels et des déclarations. Ensuite, il y a les liens entre les fichiers, les noms des variables, objets et fonctions ainsi que le nombre de paramètre(s) de ces appels et déclarations de fonction. Toute cette information reste vulnérable dans un contexte de langage dynamique tel que JavaScript où la redéfinition est toujours possible. Cependant, chaque fois qu'un candidat échoue à rester dans le champ du possible pour un de ces éléments, son degré de plausibilité est certainement amoindri.

2.3.1 Exemple illustratif

```
1  var Trigo = require("./Trigonometry.js");
2
3  function Calculator_a(){
4    this.square = function(x){
5      return multiplication(x,x);
6    }
7    this.cube = function(x){
8      function multiplication(x,y,z){
9        return x * y * z;
10     }
11     return multiplication(x,x,x);
12   }
13   function multiplication(x,y){
14     return x * y;
15   }
16 }
17 //code...
18 function Calculator_b(){
19   this.advanced = {
20     square: function(x){
21       return multiplication(x,x);
22     },
23     sinus: function(x){
24       return Math.sin(x);
25     }
26   }
27   var multiplication = function(x,y){
28       return Math.abs(x * y);
29   }
30 }
31
32 var my_calculator = new Calculator_b();
33 my_calculator.advanced.square(4);
34 var sine = Trigo.sinus(0.5);
```

Figure 2.1 Exemple de résolution de site d'appel

Considérer l'exemple de la Figure 2.1. L'arbre syntaxique est extrait et deux listes sont générées.

Tableau 2.7 Liste de déclaration

Déclaration	Hiérarchie	Type	Args	Loc
Calculator_a	Calculator_a	Fonction	0	3
square	Calculator_a.square	Fonction	0	4
cube	Calculator_a.cube	Fonction	0	7
multiplication	Calculator_a.cube.multiplication	Fonction	3	8
multiplication	Calculator_a.multiplication	Fonction	2	13
Calculator_b	Calculator_b	Fonction	0	18
advanced	Calculator_b.advanced	Objet	0	19
square	Calculator_b.advanced.square	Fonction	0	20
sinus	Calculator_b.adcanced.sinus	Fonction	0	23
multiplication	Calculator_b.multiplication	Fonction	0	27
my_calculator	my_calculator	Fonction	0	32
sine	sine	Objet	0	34

La première liste (voir Tableau 2.7) est celle des déclarations et comprends les déclarations de variables, de fonctions, d'objets et d'importation. Chaque déclaration de cette liste est ensuite classée selon son type à la déclaration ainsi que ses changements de type au cours de sa vie. De plus, la liste contient des informations supplémentaires telles que le fichier auquel il appartient, sa position dans le code, le nombre de paramètres et si des alias pointe vers la fonction définie.

Tableau 2.8 Liste des sites d'appels

Appel	Hiérarchie	Args	Loc
multiplication	multiplication	2	5
multiplication	multiplication	3	11
multiplication	multiplication	2	21
sin	Math.sin	1	24
abs	Math.abs	1	28
square	My_calculator.advanced.square	1	33
sinus	Trigo.sinus	1	36

La seconde liste (voir Tableau 2.8) comprend les appels de fonction et méthode. Elle contient le nom, la hiérarchie (chaine d'objet pointant vers la fonction), le fichier dans lequel l'appel est effectué. De plus, elle contient son emplacement, les traces de la portée de l'appel (les lignes de codes où la fonction peut être appelée) jusqu'à la portée globale ainsi que ses paramètres.

Une simple analyse basée sur les noms similaires tels que Eclipse JSDT (Feldthaus, Schafer et al. 2013) est appliqué afin d'identifier pour chaque appel, un ensemble de déclarations possibles pouvant contenir :

- 0 élément, indiquant une fonction native ou un échec de résolution (p. ex.: Un appel vers une librairie externe).
- 1 élément indiquant une résolution non ambiguë.
- 2 éléments ou plus, suggérant que l'analyse statique est incapable de résoudre l'ambiguïté. Il est important cependant de noter qu'un site d'appel peut en réalité correspondre à différentes déclarations de fonction durant l'exécution. Selon (Richards, Lebresne et al. 2010), c'est le cas en moyenne pour 19% des sites d'appel.

2.3.2 Première série d'analyses

2.3.2.1 Filtres de résolution des sites d'appels

Dans le cas où il y a des ambiguïtés, on utilise des filtres. Cinq filtres ont été considérés (Dépendance de fichier, Portée, Attachement à un objet, Hiérarchie et Paramètre) afin de réduire le nombre de candidats.

- **Dépendance de fichier :** Lorsqu'un appel est fait à partir d'une variable importée (e.g. à l'aide de « require »), il est possible de se restreindre aux déclarations de fonctions de ce fichier. Par exemple, la déclaration de « sinus » à la ligne 23 sera ignorée comme possibilité pour l'appel « sinus » à la ligne 34 (Trigo.sinus(0.5)) à cause de l'importation à la ligne 1 (var Trigo = require("./Trigonometry.js")).
- **Portée :** Les informations sur la visibilité sont extraites en analysant l'imbrication des déclarations de fonction. Lorsqu'une déclaration candidate est directement visible à partir du site d'appel ou de son conteneur, les autres candidats sont éliminés. Par exemple, en considérant l'appel « multiplication(x,x) » à la ligne 5, la seule déclaration directement visible à partir de son conteneur est à la ligne 13 ; ainsi les lignes 8 et 27 seront éliminées.
- **Attachement à un objet :** Il est possible de diviser les appels et déclarations en deux catégories. Il y a ceux qui sont attachés à des objets et ceux qui ne le sont pas. Dans le premier cas, il y a deux exemples de déclaration de fonctions qui sont attachées à un objet, « square » et « sinus » aux lignes 20 et 23. Ils sont tous les deux attachés à « advanced ». Ces fonctions ne peuvent être appelées qu'en référençant cet objet, car ce sont des méthodes. Cela veut dire que pour résoudre un appel qui est référencé à un objet, la déclaration doit également être rattachée à un objet (autre que « this » ou « windows » qui sont des exceptions). Le deuxième type d'appel de fonction est celui qui n'est pas rattaché à des objets tels que « multiplication(x,x) » à la ligne 5. Si l'appel n'est pas rattaché à un objet, alors la déclaration ne sera pas rattachée à un objet également. Ces informations aident à discriminer les candidats et donnent des résultats plus précis.

- **Hiérarchie :** Les appels et déclarations peuvent être faits dans des objets fortement imbriqués et cette information peut être utilisée pour filtrer certaines déclarations. Par exemple, l'appel « my_calculator.advanced.square(4); » à la ligne 33 peut correspondre aux déclarations « Calculator_a.square » et « Calculator_b.advanced.square ». Lorsque l'on considère un niveau plus haut et considère « advanced », seulement « Calculator_b.advanced.square » sera gardé comme candidat valide.
- **Paramètre :** Les déclarations qui n'ont pas le même nombre d'arguments que leur site d'appel sont filtrées. L'étude empirique de (Richards, Lebresne et al. 2010) rapporte que 94% des appels respectent le nombre d'arguments de la déclaration. Par exemple, les trois paramètres utilisés dans la déclaration « multiplication(x,y,z) » à la ligne 8, la disqualifient de considération pour l'appel « multiplication(x,x) » à la ligne 5.

Note : Tous ces filtres s'appliquent seulement s'ils sont capables de filtrer au moins une déclaration candidate. Étant donnée une liste et un filtre, si le filtre exclut tous les candidats de la liste, il est considéré comme non pertinent et tous les candidats de la liste sont gardés.

2.3.2.2 Combinaison de filtres

Chaque filtre apporte une information sur les candidats. En combinant ces informations, il est possible d'obtenir de meilleurs résultats. Quatre types de combinaisons ont été effectués soit l'union, l'intersection, l'union des filtres pertinents et par majorité des filtres pertinents.

- **Union :** La première combinaison, l'union est la moins discriminative. L'ensemble de tous les candidats non éliminés par les filtres est retenu. Cette combinaison ne rejettera donc pas beaucoup de sites d'appels, mais il y aura moins de chances de se tromper et de rejeter un bon site d'appels.
- **Intersection :** La deuxième combinaison, l'intersection est la plus discriminative. Elle élimine tout candidat rejeté par au moins un des filtres. Cette combinaison

éliminera beaucoup de sites d'appels, mais le risque de rejeter un bon site d'appel sera plus important.

- **Union des filtres pertinents :** La troisième combinaison est une approche plus équilibrée afin de rejeter le plus de candidats possible en minimisant le risque que ce ne soit pas de bons sites d'appels. Cette méthode prend en compte seulement les filtres pertinents qui discriminent au moins un candidat et elle fait une union de tous leurs candidats.
- **Majorité des filtres pertinents :** La quatrième combinaison testée est basée sur le vote. Tout comme la combinaison précédente, elle ne tient en compte que les filtres pertinents qui discriminent au moins un candidat. Par contre, au lieu de faire une union, elle garde seulement les candidats les plus populaires. Chaque filtre qui ne discrimine pas un candidat est considéré comme un vote pour ce candidat, le ou les candidats (s'il y a égalité) ayant la majorité des votes sont gardés, les autres sont filtrés. Cette technique permet d'obtenir une discrimination plus élevée que l'approche précédente.

2.3.3 Deuxième série d'analyses

Une étude similaire à la nôtre été publiée en 2013 (Feldthaus, Schafer et al. 2013). Les auteurs y démontrent que leur proposition d'analyses statiques donne des résultats suffisamment bons pour produire des graphes d'appel relativement précis. Afin de calculer les performances de leur technique, ils ont retrouvé à l'aide de l'analyse dynamique les déclarations vers lesquelles renvoient effectivement les sites d'appels. Leur liste ne couvre pas 100% des appels à l'intérieur des systèmes, mais tente toutefois d'en couvrir le plus possible à l'aide de plusieurs scénarios.

Afin de déterminer si nos filtres et leur combinaison éliminent réellement les bons candidats, nous avons utilisé leurs données. Elles contiennent une liste des bons candidats pour 10 systèmes. Ces systèmes sont : 3dmodel, Beslimed, Coolclock, Flotr, Fullcalendar, Pacman, Pong, Htmledit, Markitup et Pdf. Une comparaison et évaluation de complémentarité a été

effectuée entre nos 2 études et finalement nous avons analysé pourquoi certains sites d'appel ne sont pas résolus.

2.3.3.1 Puissance de discrimination des combinaisons

Nous avons calculé la précision (1.0), le rappel (1.1) et le F-mesure (1.2) pour chaque combinaison. La précision permet de calculer le pourcentage de bons candidats qui ont été retenus par le filtre alors que le rappel permet de calculer le pourcentage de candidats retenus par le filtre qui étaient de bons candidats. Le F-mesure permet d'évaluer la performance du filtre en fonction de la précision et du rappel.

$$\text{Précision} = \frac{|\text{Candidat(s) retenu(s) par le filtre } \cap \text{ Bon(s) candidat(s)}|}{|\text{Bon(s) candidat(s)}|} \quad (1.0)$$

$$\text{Rappel} = \frac{|\text{Candidat(s) retenu(s) par le filtre } \cap \text{ Bon(s) candidat(s)}|}{|\text{Candidat(s)retenu(s) par le filtre}|} \quad (1.1)$$

$$\text{F mesure} = 2 * \frac{\text{Précision} * \text{Rappel}}{(\text{Précision} + \text{Rappel})} \quad (1.2)$$

2.3.3.2 Analyse des échecs de résolution

Finalement, il y a des sites d'appels qui ne possèdent aucun candidat (13,46%). Pour mieux comprendre pourquoi ce phénomène se produit, une analyse du code source a été faite manuellement sur les 10 systèmes de l'article (Feldthaus, Schafer et al. 2013). Chaque cas a été classifié afin de déterminer les causes les plus fréquentes et quelques exemples sont présentés.

2.4 Évaluation de l'ambigüité pour la résolution des sites d'appels : les résultats

2.4.1 Évaluation du pouvoir de discrimination des filtres

Au total, 80 systèmes ont été évalués afin de résoudre la provenance des appels de fonctions.

Lors de l'analyse préalable, nous avions découvert que sans filtre, en moyenne pour tous les systèmes 16,45% des appels invoquaient des fonctions natives, 70,09% possédaient au moins un candidat de déclaration et 13,46% des appels ne pouvait pas être résolus (aucun nom de fonction ne correspond à l'appel).

2.4.1.1 Pouvoir de discrimination de chaque filtre

Une évaluation de la performance des filtres a été effectuée en se concentrant sur les appels qui possèdent au moins un candidat.

Tableau 2.9 Évaluation des filtres

	Aucun filtre	Dépendance de fichier	Portée	Attachement à un objet	Hiérarchie	Paramètre
1 candidat	61%	75%	66%	69%	67%	69%
2 candidats	17%	12%	15%	15%	16%	15%
3 candidats	7%	5%	7%	5%	6%	6%
4 candidats	4%	2%	3%	3%	3%	3%
5 candidats	2%	2%	2%	2%	2%	2%
6+ candidats	8%	5%	7%	6%	6%	5%

Le Tableau 2.9 présente un sommaire de nos découvertes. Le filtre le plus discriminatif est celui se servant de la dépendance des fichiers; il augmente le nombre d'appels à candidat unique, de 61% à 75%, enregistrant des gains de 14%. Les autres filtres se retrouvent entre 66% et 69% de candidats uniques.

2.4.1.2 Pouvoir de discrimination des combinaisons de filtres

Une évaluation de la performance des combinaisons a été faite sur les 80 systèmes pour les appels possédant au moins un candidat (voir Tableau 2.10).

Tableau 2.10 Évaluation des combinaisons

	Aucun	Union	Intersection	Union des filtres pertinents	Majorité des filtres pertinents
1	61%	64%	80%	74%	81%
2	17%	16%	10%	13%	10%
3	7%	7%	4%	5%	4%
4	4%	4%	2%	3%	2%
5	2%	2%	1%	1%	1%
6+	8%	7%	3%	4%	2%

Les combinaisons les plus discriminatives sont celles de l'intersection et de majorité des filtres pertinents qui obtiennent respectivement 80% et 81% d'appels avec un candidat unique. La combinaison majorité des filtres pertinents peut résoudre 91% des appels avec 2 candidats ou moins, ce qui veut dire que 9 fois sur 10, un IDE pourrait orienter un développeur avec une confiance élevée vers une ou deux déclarations de fonction. Les combinaisons d'union des filtres pertinents et d'union obtiennent quant à elles de 6% à 16% moins de candidats uniques.

Tableau 2.11 Discrimination des combinaisons pour les candidats uniques

Système	Union	Intersection	Union des filtres pertinents	Majorité des filtres pertinents
min	22%	50,6%	22%	51,9%
q1	51,7%	70,4%	64,6%	71,9%
q2	61%	80,8%	72,7%	82,1%
q3	75,8%	91,3%	86,2%	92,3%
moyenne	63,8%	80,1%	73,9%	81,4%
max	100%	100%	100%	100%
Std Deviation	0,174	0,125	0,158	0,124

La distribution des appels à candidat unique pour le premier quartile se situe entre 51,7% et 71,9% alors que le troisième quartile est entre 75,8% et 92,3% (voir Tableau 2.11). Ces résultats semblent raisonnables et peuvent être améliorés avec davantage d'information et d'innovation.

2.4.2 Analyses qualitatives

2.4.2.1 Performance des combinaisons

Tableau 2.12 Combinaison union

	Précision	Rappel	F-mesure
Beslimed	40.90%	100.00%	58.06%
coolclock	53.42%	100.00%	69.64%
flotr	55.91%	100.00%	71.72%
fullcalendar	40.87%	100.00%	58.02%
pacman	52.33%	100.00%	68.71%
htmledit	52.55%	100.00%	68.90%
markitup	47.12%	100.00%	64.06%
pdfjs	39.37%	100.00%	56.50%
pong	52.75%	100.00%	69.07%
3dmodel	100.00%	100.00%	100.00%
Moyenne	53.52%	100.00%	68.47%

Tableau 2.13 Combinaison intersection

	Précision	Rappel	F-mesure
Beslimed	50.90%	97.57%	66.90%
coolclock	70.11%	95.31%	80.79%
flotr	69.40%	79.04%	73.91%
fullcalendar	54.50%	83.20%	65.86%
pacman	77.64%	100.00%	87.41%
htmledit	67.83%	98.31%	80.28%
markitup	64.82%	90.43%	75.52%
pdfjs	47.98%	88.02%	62.11%
pong	76.24%	92.31%	83.51%
3dmodel	100.00%	100.00%	100.00%
Moyenne	67.94%	92.42%	77.63%

Tableau 2.14 Combinaison union des filtres pertinents

	Précision	Rappel	F-mesure
Beslimed	50.90%	97.57%	66.90%
coolclock	68.54%	95.31%	79.74%
flotr	69.40%	79.04%	73.91%
fullcalendar	54.75%	87.30%	67.30%
pacman	75.49%	100.00%	86.04%
htmledit	70.56%	98.31%	82.16%
markitup	65.84%	90.43%	76.20%
pdfjs	48.72%	98.00%	65.08%
pong	65.64%	92.64%	76.84%
3dmodel	100.00%	100.00%	100.00%
moyenne	66.99%	93.86%	77.42%

Tableau 2.15 Combinaison majorité des filtres pertinents

	Précision	Rappel	F-mesure
Beslimed	50.90%	97.57%	66.90%
coolclock	70.11%	95.31%	80.79%
flotr	72.66%	79.04%	75.71%
fullcalendar	58.72%	83.20%	68.85%
pacman	77.64%	100.00%	87.41%
htmledit	70.14%	98.31%	81.87%
markitup	65.02%	89.51%	75.32%
pdfjs	48.20%	88.02%	62.29%
pong	76.24%	92.31%	83.51%
3dmodel	100.00%	100.00%	100.00%
moyenne	68.96%	92.33%	78.27%

Afin de calculer la précision, le rappel et le F-mesure de chaque combinaison, une vérification des candidats obtenus a été conduite pour les 10 systèmes de l'article (Feldthaus, Schafer et al. 2013) (voir Tableau 2.12, Tableau 2.13, Tableau 2.14 et Tableau 2.15). Au final, les résultats démontrent que les combinaisons union des filtres pertinents, intersections et majorité des filtres pertinents obtiennent des F-mesure similaires avec un écart de seulement 0,84% entre eux. En contraste, l'union obtient un F-mesure moins élevé de 8,95% par rapport à son plus proche rival et obtient un rappel parfait alors que les trois autres combinaisons ont un rappel se situant entre 92,33% et 93,86%. La précision et le rappel les plus élevés ont été enregistrés pour le système 3dmodel avec un score de 100% dans chacun

des cas. La plus faible précision est obtenue par pdf 46,06% alors que le rappel le plus faible vient de flotr 84,28%.

2.4.2.2 Comparaison et complémentarité entre études

Une étude similaire (Feldthaus, Schafer et al. 2013) a été présentée utilisant une approche approximative afin de résoudre les déclarations candidates pour un appel. Les auteurs utilisent deux algorithmes : pessimiste et optimiste. Dans leur conclusion, ils proposent d'utiliser l'approche pessimiste qui est plus rapide et adaptée pour des applications telles que des IDE. En utilisant nos combinaisons de filtres et les résultats de leur algorithme pessimiste comme s'il serait le résultat d'un filtre indépendant, nous avons comparé les performances et ensuite mesuré la complémentarité entre nos techniques afin de savoir s'il serait possible d'obtenir de meilleurs résultats en combinant les approches. En calculant les résultats de la meilleure approche de (Feldthaus, Schafer et al. 2013) (l'approche pessimiste) utilisant notre méthodologie, on remarque une F-mesure comparable avec 75,75% (voir Tableau 2.16). Il s'agit de résultats supérieurs à l'union 66.15%, mais inférieurs aux trois autres combinaisons: union des filtres pertinents 81,08%, par majorité des filtres pertinents 81,54% et l'intersection 81,87% (voir Tableau 2.17, Tableau 2.18, Tableau 2.19 et Tableau 2.20).

Tableau 2.16 Étude similaire

	Précision	**Rappel**	**F-mesure**
Beslimed	41.70%	97.36%	58.39%
coolclock	57.14%	100.00%	72.73%
flotr	58.59%	80.17%	67.70%
fullcalendar	56.57%	96.12%	71.22%
pacman	100.00%	100.00%	100.00%
htmledit	69.58%	98.31%	81.49%
markitup	56.57%	94.98%	70.91%
pdfjs	43.21%	98.58%	60.27%
pong	62.33%	93.29%	74.73%
3dmodel	100.00%	100.00%	100.00%
moyenne	64.57%	95.88%	75.75%

La précision est moins élevée que la combinaison majorité des filtres pertinents de notre étude par 4,39%. En contrepartie, leur rappel se compare favorablement, étant le plus élevé de tous et de 2,02%, que celui de l'approche union des filtres pertinents. La conjugaison de leur technique avec nos combinaisons (si l'on considère leur technique comme l'un de nos filtres) montre qu'il existe une bonne complémentarité entre les deux approches.

Tableau 2.17 Combinaison « union »

	Précision	**Rappel**	**F-mesure**
Beslimed	40.42%	100.00%	57.57%
coolclock	43.10%	100.00%	60.24%
flotr	52.25%	100.00%	68.63%
fullcalendar	37.40%	100.00%	54.44%
pacman	52.33%	100.00%	68.71%
htmledit	51.85%	100.00%	68.29%
markitup	41.25%	100.00%	58.40%
pdfjs	39.27%	100.00%	56.40%
pong	52.45%	100.00%	68.81%
3dmodel	100.00%	100.00%	100.00%
moyenne	51.03%	100.00%	66.15%

Tableau 2.18 Combinaison « intersection »

	Précision	**Rappel**	**F-mesure**
Beslimed	51.12%	97.36%	67.04%
coolclock	80.54%	93.75%	86.64%
flotr	78.03%	76.49%	77.25%
fullcalendar	67.99%	80.58%	73.75%
pacman	100.00%	100.00%	100.00%
htmledit	77.73%	92.13%	84.32%
markitup	74.43%	87.63%	80.49%
pdfjs	50.60%	87.24%	64.05%
pong	79.02%	92.28%	85.14%
3dmodel	100.00%	100.00%	100.00%
moyenne	75.95%	90.75%	81.87%

Tableau 2.19 Combinaison « union des filtres pertinents »

	Précision	Rappel	F-mesure
Beslimed	51.12%	97.57%	67.09%
coolclock	74.40%	97.66%	84.46%
flotr	71.36%	79.04%	75.00%
fullcalendar	64.90%	87.68%	74.59%
pacman	100.00%	100.00%	100.00%
htmledit	75.11%	98.31%	85.16%
markitup	68.30%	92.98%	78.75%
pdfjs	50.83%	98.45%	67.05%
pong	68.23%	92.95%	78.69%
3dmodel	100.00%	100.00%	100.00%
moyenne	72.42%	94.46%	81.08%

Tableau 2.20 Combinaison « majorité des filtres pertinents »

	Précision	Rappel	F-mesure
Beslimed	51.17%	97.57%	67.13%
coolclock	74.40%	97.66%	84.46%
flotr	74.80%	79.04%	76.86%
fullcalendar	65.99%	85.06%	74.32%
pacman	100.00%	100.00%	100.00%
htmledit	75.11%	98.31%	85.16%
markitup	67.64%	92.98%	78.31%
pdfjs	50.18%	89.05%	64.19%
pong	78.41%	92.62%	84.92%
3dmodel	100.00%	100.00%	100.00%
moyenne	73.77%	93.23%	81.54%

La complémentarité des informations recueillies auprès des appels fait augmenter la F-mesure des quatre combinaisons en moyenne par 2,22%, passant de 75,44% à 77,66%. L'augmentation la plus marquée (4,24%) est celle de l'intersection qui devient maintenant la combinaison la plus efficace. En contraste, l'union perd de la précision, car les candidats des deux études sont jumelés ce qui fait augmenter leur nombre.

2.4.2.3 Les échecs de résolution

Pour mieux comprendre les raisons qui expliquent que certains appels ne possèdent aucun candidat (13,46%), une analyse manuelle des cas non résolus a été menée sur les 10 systèmes de l'article (Feldthaus, Schafer et al. 2013).

Tableau 2.21 Sommaire des appels non résolu

	A	B	C	D
min	18	0	0	0
q1	33	12	1	1
q2	49.5	19	13.5	11
q3	70	27	36	19
moyenne	60.5	30.2	19.7	11.4
Moyenne %	49,7%	24.8%	16.2%	9.4%
max	187	108	51	30
Std Deviation	45.91	33.47	18.79	9.61

Raisons

A – Fonction native non incluse dans notre base de données

B – Appel ou déclaration comporte une formulation trop dynamique

C – Indirections de nom de déclaration

D – La déclaration de fonction vient d'un paramètre (« callback »)

Les résultats (voir Tableau 2.21) démontrent que dans 49.7% des cas, les appels non résolus provenaient de fonctions natives. Par exemple, le système 3dmodel appelle plusieurs méthodes sur des éléments SVG alors que pour le système Pacman, les appels étaient faits sur des méthodes d'objets ActiveX et Flash. Ces méthodes que les navigateurs rendent disponibles aux développeurs afin de manipuler ces objets n'étaient pas prises en compte dans notre base de données de fonctions natives.

Ensuite, 24.8% des appels non résolus étaient dus à des formulations d'appel ou de déclaration qui étaient trop dynamiques pour être récupérées. Voici un exemple tiré du système Beslimed.

```
for (var i = 0, l = this.length; i < l; i++) fn.call(bind, this[i], i, this);
```

Bref, il est difficile de connaitre la fonction appelée avec certitude sans exécuter le code lorsque plusieurs facteurs entrent en compte tels que la valeur de « this » et de la variable « i ».

De plus, une technique basée sur le nom afin de résoudre la correspondance entre les appels et fonctions comporte certains risques. Il y a 16.2% des cas non résolus qui étaient causés par des indirections dans les déclarations de fonction qui peuvent parfois changer plus d'une fois lors de l'exécution. Cet exemple du système Flotr illustre le problème.

```
var m = Math;
var mr = m.round;
…
vmlStr.push('top:', mr(d.y / Z), 'px;left:', mr(d.x / Z), 'px;');
```

La technique propose de chercher une déclaration de fonction au nom de « mr » alors qu'elle devrait reconnaitre qu'il s'agit en fait d'une fonction native correspondant à Math.round. Cet appel est donc classé non résolu.

Et finalement, dans 9.4% des cas, la déclaration vient d'un paramètre.

```
onData: function Promise_onData(callback) {
      if (this._data !== EMPTY_PROMISE) {
        callback(this._data);
      } else {
        this.onDataCallback = callback;
      }
    }
```

La déclaration de « callback » est contenue dans le paramètre de l'appel ce qui rend la résolution par nom complexe. La fonction callback est déclarée lors de son usage et dépend fortement du contexte d'exécution. Dans le cas d'une librairie comme jQuery, il est commun que plusieurs fonctions restent inutilisées et donc qu'il n'existe aucune déclaration de ces fonctions. Pour ces raisons, l'algorithme utilisé ne gère pas les cas de callback.

Les librairies utilisées dans les systèmes ont un impact sur les cas non résolus. Certains de ces appels dont les déclarations étaient impossibles à trouver se retrouvent dans les fichiers de librairies (jugé comme système externe) pour les 80 systèmes de l'analyse quantitative. Par exemple, le système WineCellar utilise plusieurs librairies (p.ex. : Backbone, Express).

Tableau 2.22 Appels non résolus excluant les librairies.

	Appel Total	**Appels dans le système**	**Appels dans les librairies**	**Non résolus avec les appels de librairies**	**Non résolus sans les appels de librairies**
3DMODEL*	63	63	0	18	18
BESLIMED	1542	107	1435	209	14
COOLCLOCK	1059	27	1032	117	20
FLOTR	1853	607	1246	122	58
FULLCALENDAR	2959	842	2117	200	50
PACMAN*	332	332	0	21	21
PONG	848	71	777	75	15
HTMLEDIT	801	91	710	102	42
MARKITUP	1207	89	1118	107	21
PDF*	5682	5682	0	707	707

(*) Systèmes ne contenant aucune librairie.

Le Tableau 2.22 démontre l'impact des librairies sur les 10 systèmes étudiés. L'exclusion des appels provenant des librairies (généralement plus complexe) permet de réduire significativement (57.7%) le nombre d'appels non résolu.

2.5 Menaces à la validité

Notre étude ne prétend pas être généralisable, mais essaie de mitiger les menaces. Elle est basée sur 80 systèmes sélectionnés avec soins. La taille de l'échantillon des systèmes choisis se compare favorablement à la taille d'études portant sur l'analyse statique et dynamique ; 10 systèmes pour (Feldthaus, Schafer et al. 2013) et 12 systèmes pour (Wei and Ryder 2014). Elle se compare moins favorablement aux études empiriques basées simplement sur le comptage de constructeurs dynamiques (p. ex.: eval, with) telles que (Richards, Lebresne et al. 2010) qui portait sur 100 sites web et (Richards, Hammer et al. 2011) qui portait sur 10000 sites web. Nous avons tenté de sélectionner un ensemble diversifié de systèmes, mais nous ne pouvons pas prétendre que nos systèmes sont parfaitement représentatifs de tous les systèmes JavaScript. Les filtres proposés pour la résolution d'appel représentent des heuristiques intuitives, ils semblent efficaces et sont supportés par le sens commun, notre analyse qualitative ainsi que quelques études empiriques tel que (Richards, Lebresne et al. 2010) pour la variadicité de fonction, mais ils n'éliminent pas toujours tous les mauvais candidats et peuvent parfois éliminer de bons candidats. En général, cette étude invite la réplication et les extensions (nos données sont disponibles sur Github).

2.6 Améliorations possibles

Les améliorations possibles pour l'étude sur l'analyse statique sont d'étendre la quantité de systèmes dans l'étude afin de mieux représenter l'écosystème. Il y avait très peu de systèmes ES2015 disponibles lors de l'étude empirique. Il serait intéressant de refaire l'exercice, avec un plus grand nombre de ces systèmes dans quelques années. En ce qui concerne la résolution des appels, de nouveaux filtres pourraient être ajoutés qui tireraient avantage de nouvelles caractéristiques sur les appels et déclarations. Il serait également possible de mieux comprendre comment combiner les filtres afin d'améliorer la performance de prédiction des déclarations de fonction pour un site d'appel. De plus, il faudrait tenter d'améliorer notre gestion des appels ou déclarations qui comportent une formulation trop dynamique, les indirections de nom dans les déclarations et finalement gérer les « callback » (déclaration de

fonction venant d'un paramètre). Finalement, il serait intéressant d'enrichir notre technique avec l'analyse dynamique dans un avenir rapproché.

2.7 Conclusion sur l'analyse statique

Suite à l'étude empirique, certains constats sont faits :

- Il y a peu de fonctionnalités strictement dynamiques avec en moyenne 0,03% d'appel de type « eval », « with » et« new Function » combinés. Les constructeurs « apply » et « call » représentent 4,39% des appels au total.
- La dépendance entre les fichiers est rarement compliquée. Il n'y a aucun cas où des « define » utilisent un argument autre qu'un littéral et moins de deux cas en moyenne par système où l'argument de « require » est autre chose qu'un littéral (p.ex. : variable, tableau, fonction).
- La résolution de site d'appel à l'aide de la technique naïve par nom offre des chiffres raisonnables, mais certains sites d'appel ne sont pas résolus. En moyenne, 13,6% des sites d'appel ne possèdent pas de candidat. Les causes les plus fréquentes sont : un manquement à notre base de données de fonction natives (49,7%) et les appels où les déclarations comportaient des formulations trop dynamiques (24.8%). D'autres cas complexes à régler sont les indirections dans les noms de déclaration et les fonctions venant d'un paramètre.
- Lorsqu'il y a au moins un candidat, la technique naïve par nom offre des chiffres raisonnables. 61% des sites d'appels possèdent 1 candidat unique et 85% des sites d'appels comptent 3 candidats ou moins.
- La technique naïve par nom peut être améliorée à l'aide de filtre simples. Les filtres les plus discriminants sont ceux basés sur la dépendance des fichiers (75% de candidats uniques), l'attachement à un objet (69% de candidat unique) et le nombre de paramètres dans l'appel (69% de candidat unique).
- Les filtres peuvent être combinés afin d'améliorer leur pouvoir discriminant. Les combinaisons les plus efficaces sont par majorité des filtres pertinents (81% de candidat unique et 91% avec 2 candidats ou moins) et l'intersection (80% de candidat

unique). Ces résultats sont rassurants, car ils vont dans le même sens que l'étude empirique de (Richards, Lebresne et al. 2010) sur 100 sites web concluant qu'en moyenne 81% des sites d'appels étaient monomorphiques.

- Les combinaisons à l'aide de l'analyse statique offrent des performances comparables à une étude externe. L'étude de (Feldthaus, Schafer et al. 2013) obtient une F-mesure de 75,75%. La F-mesure de la combinaison par majorité des filtres pertinents est de 78,27% alors que celle de la combinaison par intersection est de 77,63%. Il existe une complémentarité entre les deux études. En utilisant la technique de l'étude externe comme une sortie de filtre de notre technique, la F-mesure de la combinaison par intersection augmente à 81,87% alors que celle par majorité des filtres pertinents devient 81.54%

En conclusion, l'analyse statique est viable. Il s'agit d'une technique imparfaite qui peut encore être améliorée, mais les chiffres sont suffisants pour la documentation d'application web. Une des améliorations possibles serait de la combiner avec l'analyse dynamique (qui a ses propres problèmes et manquements).

CHAPITRE 3

CRÉATION DE L'OUTIL WAVI

Les applications web proposent des défis uniques lorsque vient le temps de comprendre et de maintenir leurs structures hétérogènes qui proposent souvent des interactions complexes entre les éléments de divers langages. Une documentation à jour de ces applications est rarement disponible en entreprise et c'est pourquoi il est nécessaire de recourir à de nouvelles approches de rétro-ingénierie pour redécouvrir et représenter ces structures. Notre proposition de rétro-ingénierie pour le web s'appuie sur l'analyse statique (confirmée viable au Chapitre 2) et se concrétise par le développement de l'outil WAVI (Web Application Viewer). Dans les sections suivantes, nous présentons l'outil WAVI, à travers son processus de rétro-ingénierie et les représentations (textuelles et graphiques) qu'il propose en sortie avant de nous intéresser à ses spécifications et son interface utilisateur.

3.1 Processus de rétro-ingénierie de WAVI

3.1.1 Étape 1 : Transformation du code source vers un arbre syntaxique

Le style de programmation (p.ex. : le choix des termes, la formulation et l'organisation du code) peut varier grandement selon les programmeurs, surtout pour un langage dynamique comme JavaScript. Il est donc nécessaire de trouver une façon d'organiser le code d'une manière plus cohérente et générique pour faciliter son analyse. La première étape de l'approche de rétro-ingénierie proposée par WAVI est donc de redécouvrir la structure d'une application web en transformant le code source en arbre syntaxique (AST) ou en Document Object Model (DOM) selon le langage analysé.

Contrairement au code source, les données à l'intérieur de l'AST et du DOM sont simplifiées sous forme de nœuds interconnectés afin qu'ils soient faciles à traverser. De plus, chaque nœud possède une structure prévisible (un type, une valeur et un nom). Il s'agit donc de la

forme idéale pour l'analyse statique. Voici un exemple d'un code source JavaScript transformé en arbre syntaxique à la Figure 3.1.

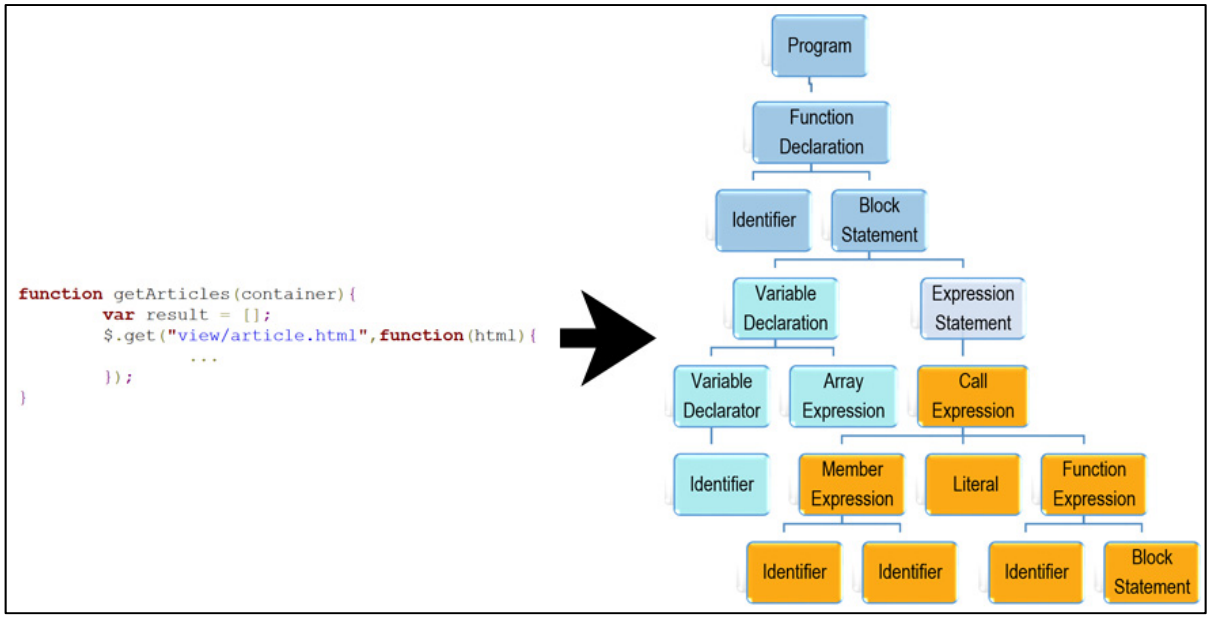

Figure 3.1 Étape 1 : transformation du code vers un arbre syntaxique.

Dans cet exemple, le code source d'une application web est transformé en arbre syntaxique. Le code représente un appel AJAX afin d'acquérir les données d'un fichier HTML. L'extraction des données par WAVI est tout d'abord faite à l'aide de librairies libres de droits telles que Esprima[12] (conversion de code source vers AST), qui sont utilisées pour analyser les fichiers JavaScript et Node.js. L'analyse HTML utilise le paquetage Cheerio[13] (qui permet de faire des recherches de type jQuery dans le code HTML) tandis que l'analyse CSS utilise le paquetage Css[14] (parseur CSS) disponible dans le répertoire de Node Package Manager (NPM).

[12] http://esprima.org

[13] https://github.com/cheeriojs/cheerio

[14] https://github.com/reworkcss/css

3.1.2 Étape 2 : Recherche des éléments dans la structure

La deuxième étape consiste à traverser les nœuds de l'arbre à la recherche des éléments que l'on considère essentiels dans la structure et que l'on désire documenter. Par exemple, la présence d'une variable dans un fichier JavaScript. Les AST contiennent toute l'information nécessaire sous forme de nœuds pour l'analyse statique.

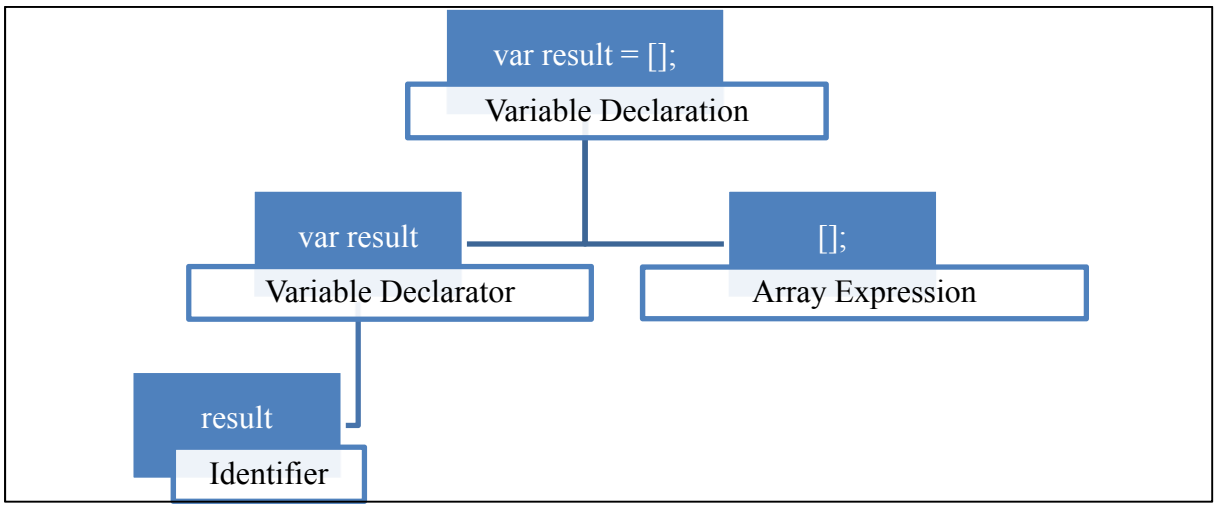

Figure 3.2 Exemple de code représenté par des nœuds d'un arbre syntaxique.

La Figure 3.2 représente le code « var result = []; » à l'aide de 4 nœuds. L'ensemble de cette ligne est une déclaration de variable qui contient un déclarateur de variable « var result ». La variable result est un « Identifier » et la valeur assignée à celle-ci est un tableau « [] ». À chaque fois que l'on traverse l'arbre syntaxique et que l'on retrouve cet ensemble de nœuds nous informant de la présence d'une variable, on ajoute l'élément à notre base de données. Les nœuds ne représentant pas des éléments importants sont quant à eux laissés de côté. À la fin de cette étape, la base de données contient un arbre des éléments importants, qui a grandement été simplifié par rapport à l'arbre de nœuds de l'arbre syntaxique.

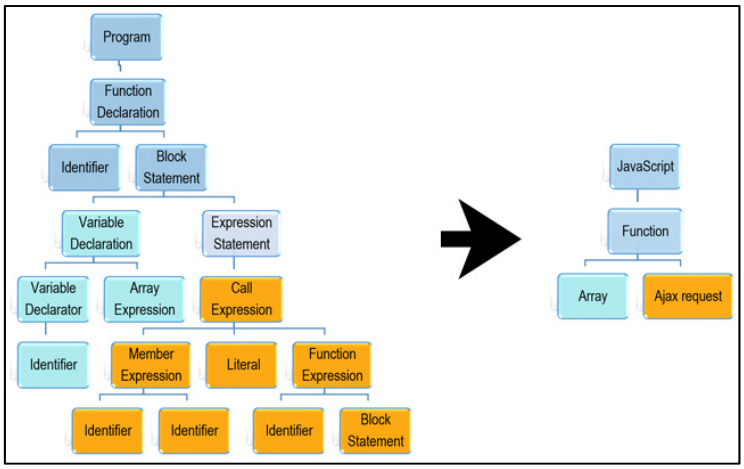

Figure 3.3 Étape 2 : recherche des éléments importants.

La Figure 3.3 présente un arbre syntaxique de code JavaScript transformé en arbre simplifié avec les éléments intéressants. Au final, on se retrouve avec une fonction, un tableau et une requête Ajax. La structure est donc réduite seulement aux éléments intéressants pour la documentation de l'application. Chaque langage contient des éléments spécifiques à considérer.

- **HTML :** Parmi les éléments du langage HTML, il y a tout d'abord les hyperliens qui permettent de naviguer d'une page à une autre, il y a les cadres et les balises « include » qui permettent d'intégrer plusieurs fichiers HTML en un. Ensuite, il y a les éléments des formulaires tels que des boutons, boites de texte, listes, sélecteurs de choix multiple et finalement, les éléments multimédias qui permettent d'afficher des images, des vidéos, du son ou des objets externes tels que du Flash ou bien des Applets.
- **JavaScript et Node.js :** JavaScript et Node.js sont des langages supportant le paradigme orienté-objet, il est donc nécessaire de trouver les classes à l'intérieur des fichiers ainsi que les objets, méthodes, fonctions, variables et tableaux. De plus, pour JavaScript du côté client, il existe plusieurs cadres de travail (« Framework ») disponibles. Par exemple, jQuery qui permet de faciliter l'utilisation de requêtes AJAX. Du côté serveur, il y a Node.js, qui propose plusieurs librairies spécifiques

comme « Express » contenant des éléments intéressants pour la gestion des requêtes HTTP.

- **CSS :** CSS possède une structure simple composée de deux éléments : les sélecteurs et les règles. Un sélecteur est une classe ou un élément spécifique du DOM qui contient des règles. Ces règles sont des paires de propriétés et de valeur.

3.1.3 Étape 3 : Liens entre les fichiers

Lors de l'étape précédente, chaque fichier de l'application web a produit un arbre simplifié. Il faut maintenant trouver les liens entre ces fichiers afin d'intégrer, dans une seule structure, chacun de ces arbres simplifiés et ainsi représenter l'ensemble de l'application web. Pour ce faire, il faut détecter les éléments d'importation des divers fichiers. Les arbres simplifiés de l'étape précédente sont donc analysés afin de trouver toutes les connexions potentielles entre fichiers existants (voir Figure 3.4).

Les liens possibles à l'intérieur des fichiers JavaScript peuvent entre autres se retrouver dans les requêtes AJAX, avec les fonctions « Require », « Define » ou bien dans les redirections de page. Dans les fichiers HTML, ils se retrouvent dans des balises de « link », « script » et « include ». On peut également trouver des liens vers d'autres fichiers dans les cadres et les objets multimédias comme des images, vidéo et audio. L'analyse de liens dans les fichiers CSS, quant à elle, se résume à la recherche de déclarations d'importations.

Dans l'exemple de la Figure 3.4, on peut voir en rouge les liens entre les fichiers. Le fichier HTML possède une relation avec un fichier CSS et deux fichiers JavaScript. Un de ces fichiers JavaScript a une fonction qui exécute une requête AJAX pour récupérer le contenu d'un fichier HTML.

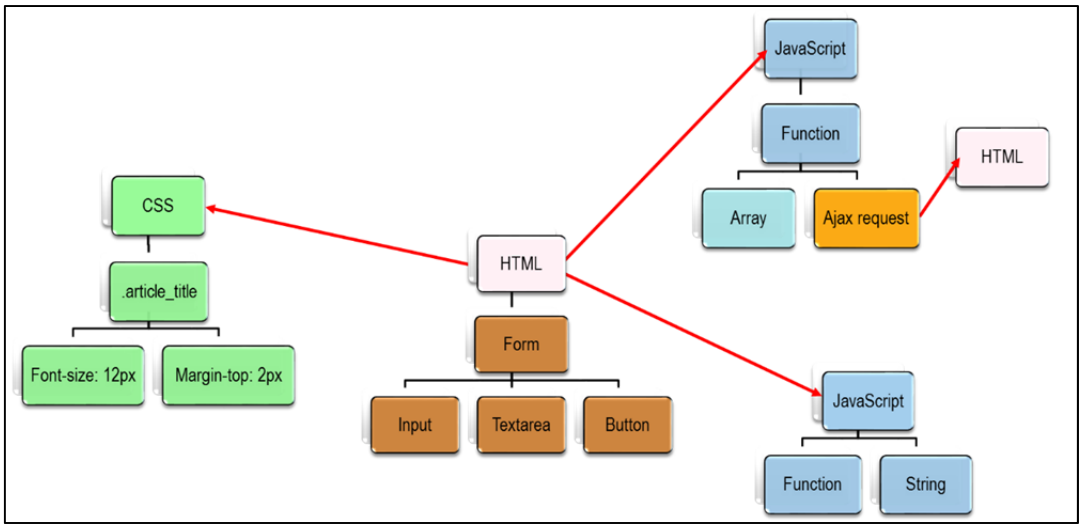

Figure 3.4 Étape 3 : liens entre les fichiers.

3.1.4 Étape 4 : Interaction entre les éléments

Afin que le niveau de détail de l'analyse soit complet, il faut aussi explorer l'interaction des éléments entre eux. Une de ces interactions est de retracer les appels de fonctions afin de les jumeler avec leurs déclarations même s'ils sont dans des fichiers distants. Cette information est importante, car elle permet de mieux évaluer les impacts que le changement de code d'un fichier peut avoir sur les autres fichiers reliés. Elle permet également de mieux comprendre pourquoi et comment les fichiers communiquent ensemble. L'arbre syntaxique peut nous renseigner quand une fonction est déclarée ou bien appelée, mais il ne fait pas le lien entre les deux. Des techniques simples basées sur le nom des appels et déclarations ainsi que des filtres tels que présentés dans le chapitre 2 peuvent aider à déduire les liens entre les appels et déclarations. La combinaison de filtre par intersection qui est jugée la plus efficace (meilleur F-mesure) est utilisée pour cette étape.

3.2 Artéfacts générés par WAVI

3.2.1 Représentation textuelle (JSON)

WAVI produit un fichier de données de type JSON qui contient une liste des éléments, une liste des relations et une liste de tous les groupes d'éléments avec des statistiques (p. ex.: le nombre de boutons ou de liens). Les informations peuvent être utilisées pour le développement d'outils d'analyse dans le futur (évolution, analyse d'impact).

Voici un exemple de code JavaScript:

```
function myFunc(){
   var abc = « mystring »;
   abc = 6;
}
```

Les données recueillies sous format JSON pour la variable « abc » de l'exemple plus bas sont présentées dans le Tableau 3.1.

Tableau 3.1 Analyse textuelle

Champ	Description	Exemple
Id	Identifiant numérique de l'élément (chiffre).	3
Nom	Nom de l'élément.	abc
Hiérarchie	Nom des éléments concaténés depuis la racine du fichier. « :: » sert de séparateur.	Folder/myFile.js ::myFunc ::abc
Groupe	Groupe de l'élément.	variable
Valeur*	Valeur de l'élément.	« mystring »
Type*	Type de l'élément.	String
Visibilité*	Visibilité de l'élément.	Privée
Assignation*	Liste de toutes les assignations de l'élément.	Assignations de type « Number » de la Ligne : 3;3 à 3;7.
Parent	Le parent de l'élément.	myFunc
Emplacement	Ligne et colonne où commence et se termine l'élément dans le fichier.	Ligne : 2;3 à 2;25.
Fichier	Fichier dans lequel se retrouve l'élément.	« Folder/myFile.js »
Dossier	Dossier dans lequel se retrouve le fichier.	« Folder »
Bloc	Bloc de code dans lequel se retrouve l'élément.	Ligne : 2;1 à 2;26

(*) Optionnel, s'applique seulement aux fonctions, variables et classes.

3.2.2 Diagramme de force dirigée

La première représentation graphique proposée par WAVI est celle du diagramme de force dirigée. C'est un diagramme dynamique représenté par des nœuds et des liens. Il s'agit d'un diagramme qui se prête bien à la visualisation des éléments de façon individuelle et leurs liens. Il permet à l'utilisateur de se faire une idée en un coup d'œil du type de structure de

l'application et de voir s'il y a des problèmes apparents. Le diagramme de force dirigée est généré à l'aide de la librairie D3.js. (*Voir* Figure 3.5). Les éléments sont tout d'abord convertis en nœuds et un algorithme est appliqué pour positionner les nœuds. Les liens sont représentés par des flèches entre les éléments. Il s'agit d'un diagramme dynamique où les nœuds se déplacent constamment jusqu'à ce qu'un équilibre soit atteint. Les nœuds sont tous attirés vers le milieu et se repoussent entre eux. Les liens gardent les nœuds reliés rapprochés l'un de l'autre. Graduellement, l'algorithme ajuste la position des éléments et les nœuds les plus connectés se retrouvent au milieu tandis que les nœuds moins utilisés se retrouvent aux extrêmes.

Un problème peut survenir lors de l'utilisation de ces diagrammes. S'il y a trop de fichiers et de liens, on peut facilement se perdre ou bien avoir de la difficulté à comprendre le graphe. De plus, s'il existe une quantité importante de fichiers, il pourrait y avoir des problèmes de performance dans l'affichage du graphe. Pour régler ce problème, deux solutions ont été mises en place. Les paramètres du diagramme de force dirigée et les paramètres additionnels.

Les paramètres de diagrammes de force dirigée :

- La force avec laquelle les nœuds se repoussent.
- La friction qui ralentit les nœuds à chaque itération pour qu'éventuellement les nœuds ne bougent plus. Une friction élevée permet au graphe de ralentir plus rapidement et éventuellement s'immobiliser.
- La distance entre les nœuds.

Les paramètres additionnels :

- Le filtre d'élément par type pour filtrer les éléments selon leur type (p.ex. : classe, méthode, variable).
- La taille de la police de caractère utilisé pour décrire les nœuds.
- La transparence des liens entre les nœuds.
- La recherche des éléments pour trouver un nœud particulier.

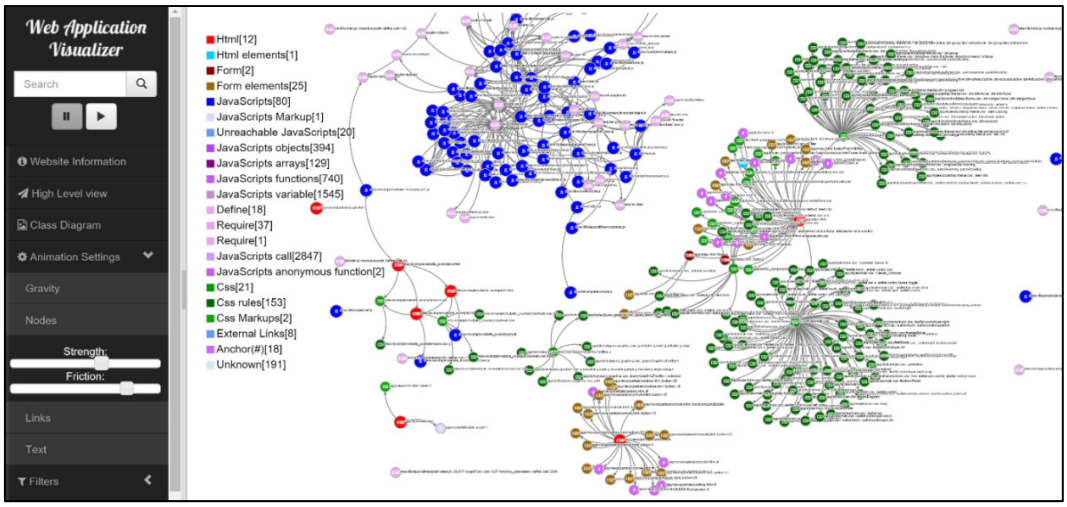

Figure 3.5 Capture d'écran de l'outil WAVI.

Chaque nœud du graphe représente un élément et sa couleur représente son type. Les fichiers HTML sont de couleur rouge, les fichiers JavaScript sont en bleu et les feuilles de style en vert. Si le lien hypertexte pointe vers un fichier existant dans le dossier d'analyse, il s'agit d'un fichier interne. S'il provient d'un autre serveur, il s'agit d'un lien externe.

3.2.3 Diagramme de classe

WAVI offre un second diagramme plus formel : un diagramme de classes adapté pour les applications web, inspiré par WAE[15].Les éléments sont représentés par des classes et des propriétés. Le résultat est un diagramme beaucoup plus compact qui contient moins de liens, ce qui aide à visualiser l'ensemble de la structure d'une application web. C'est également une façon plus intuitive de représenter la structure d'une application puisque les développeurs sont habitués à voir les diagrammes de classes lorsqu'ils programment des applications avec des langages comme Java ou C++.

[15] Nous avons fait des changements mineurs à l'extension WAE tels que l'ajout de la couleur qui distingue les éléments et l'intégration de nouveaux éléments (p.ex. : requêtes Ajax).

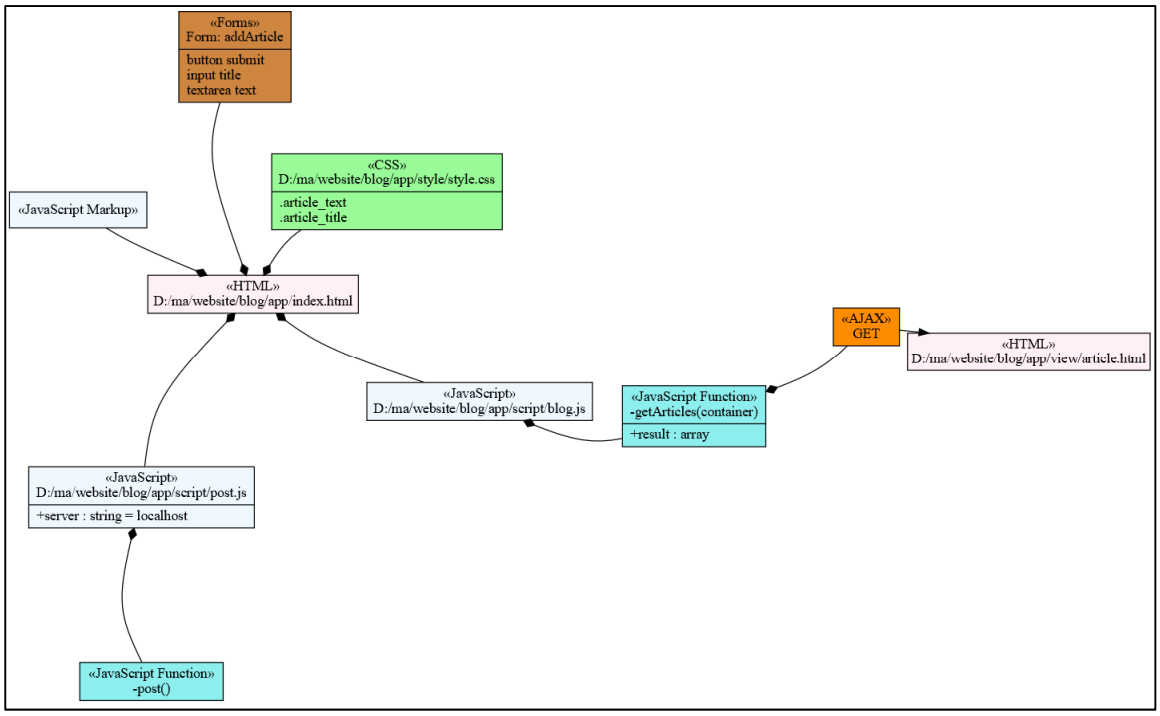

Figure 3.6 Diagramme de classe

Dans la Figure 3.6, les classes sont de différentes couleurs. Par exemple, en bleu pâle il y a les fichiers JavaScript, les fonctions en turquoise et les fichiers CSS en vert contenant les noms de règles. Les formulaires sont en brun et contiennent les éléments comme des boutons ou bien des boites de texte. Les fichiers HTML en rose contiennent les liens et éléments HTML qui ne sont pas dans des formulaires.

Similaire à ce que l'on a fait avec le diagramme de force dirigée, nous avons proposée des options pour les utilisateurs. En particulier, nous proposons un mécanisme pour accommoder la mise à l'échelle basée sur l'importance relative de chaque élément de la structure.

3.2.3.1 Critères d'importance

Puisqu'il faut parfois être en mesure de mettre à l'échelle les résultats, les éléments doivent pouvoir être filtrés selon un critère d'importance. Lors de la rétro-ingénierie, un niveau d'importance est donc assigné à chaque élément de ce système selon les informations

extraites. Les détails connus sur les éléments sont représentés dans le Tableau 3.2 (voir ANNEXE XII pour un exemple de calcul d'importance).

Tableau 3.2 Détails sur les éléments

Dimension	**Valeur possible**
Le type de l'élément	Classe, méthode, etc.
Le niveau hiérarchique de cet élément	0-infini
La visibilité de l'élément (privé ou publique)	0-1
Si l'élément est anonyme ou non	0-1
Nombre de lignes de code	0-infini
Le nombre de relation avec les autres éléments	0-infini

- **Le type de l'élément** : (p. ex.: fichier, fonction, classe, variable) est l'information qui a le plus d'impact sur la façon dont il sera représenté. Le choix a été fait de toujours représenter les fichiers sous forme de classe, car ils sont les éléments de base du système. Par contre, les fonctions et les variables peuvent parfois être laissées de côté si leurs valeurs sont jugées faibles.
- **Le niveau hiérarchique** : Le niveau hiérarchique de l'élément est la deuxième information ayant le plus d'impact. Dans ce niveau, il y a tout d'abord la hiérarchie du fichier qui est prise en compte (par rapport à l'imbrication des dossiers) et ensuite la hiérarchie de l'élément à l'intérieur du fichier (niveau d'imbrication de l'élément). Bref, une variable est jugée moins importante si elle se retrouve dans plusieurs niveaux d'imbrication de fonction.
- **La visibilité de l'élément** : La visibilité joue également un rôle dans son importance. Si l'élément est une fonction privée, elle aura beaucoup moins d'importance qu'une fonction publique.
- **Éléments anonymes** : Si l'élément est anonyme (p. ex.: une fonction sans nom), il apporte peu de valeur de documentation, son importance est donc amoindrie.

- **Nombre de lignes de code** : le nombre de lignes de code rend un élément plus important (p.ex. : une fonction étendue sur plusieurs lignes est jugée plus importante qu'une petite fonction).
- **Le nombre de relations avec les autres éléments** : Les relations entre éléments permettent de juger l'importance d'un élément. Par exemple, une déclaration de fonction appelée à plusieurs endroits est plus importante qu'une fonction qui n'est jamais appelée.

3.2.3.2 Profils

Chaque utilisateur possède des besoins différents, certains désireront une vue sommaire du système alors que d'autres désireront une vue en détail. Afin de s'adapter au désir de visualisation de tout un chacun, WAVI propose différents profils.

WAVI propose trois façons de représenter un élément :

1. L'élément forme une classe
2. L'élément forme une propriété de classe
3. L'élément est invisible

Le choix final de la représentation pour chaque élément est fait en fonction du critère d'importance de l'élément et du choix de profil. Les profils peuvent être compacts, donc afficher peu de classes et utiliser moins d'espace ou bien éclatés, ce qui permet de voir une plus grande variété d'éléments et leurs imbrications. Ils peuvent également être très détaillés soit posséder beaucoup de propriétés de classes ou bien peu détaillés et ne montrer que les éléments de base comme les fichiers.

WAVI propose 6 profils qui peuvent être représentés selon 2 dimensions (le nombre de classes et le nombre de propriétés) (voir Tableau 3.3).

Tableau 3.3 Profils de WAVI

#Profil	#Classes	# Propriétés	Cx	Py
1	Maximum	*	0	100
2	Medium	Medium	50	25
3	Minimum	Maximum	100	0
4		Medium	100	30
5		Minimum	100	50
6		*	100	100

Les deux dimensions impactent le pourcentage de propriétés qui seront visibles dans le diagramme. Ces profils sont donc représentés par une paire (Cx, Py) ou « x » représente le seuil d'importance pour le nombre de classes et « y » le seuil d'importance pour le nombre de propriétés. Lorsque « x » est au niveau le plus bas, tous les éléments sont représentés sous forme de classes, toutes les relations sont visibles et le niveau de « y » n'a aucun effet. Dans les autres configurations, le niveau de propriété a un impact si les éléments qui ne sont pas représentés en tant que classes peuvent être représentés en tant que propriété ou s'ils disparaissent tout simplement (pour des exemples de calcul, voir ANNEXE XIII).

Pour expliquer comment les profils impactent la visualisation d'une application, utilisons un exemple qui montre trois profils différents.

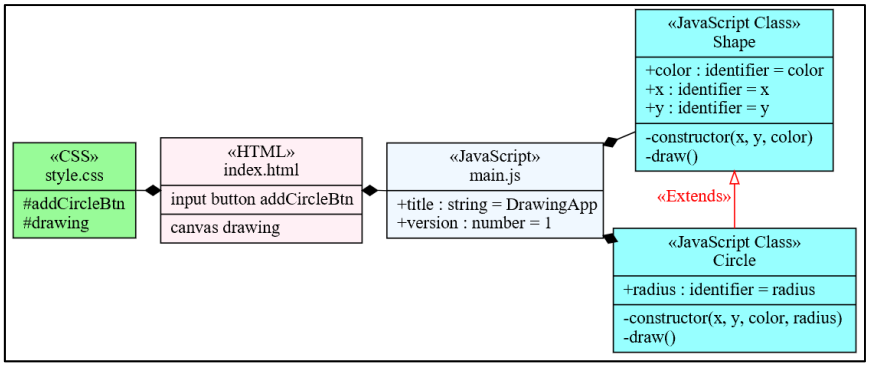

Figure 3.7 Profil « #3 classes minimum, propriétés maximum»

La Figure 3.7 montre un maximum de propriétés dans un minimum de classes. Ce profil offre une vue compacte et détaillée du système ce qui est mieux adapté pour documenter tout un système au complet.

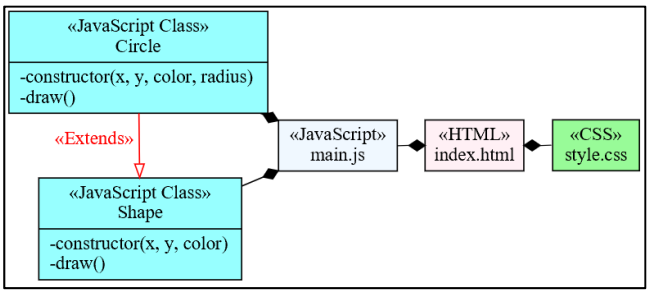

Figure 3.8 Profil « #4 classes minimum,
propriétés médium»

Le deuxième profil (voir Figure 3.8) montre une quantité médium de propriétés dans un minimum de classes. Dans ce profil, les variables, éléments HTML et règles CSS ont été retirés alors que les méthodes des classes sont toujours présentes. Cette vue est plus appropriée pour un système de grande taille qui contient beaucoup d'éléments.

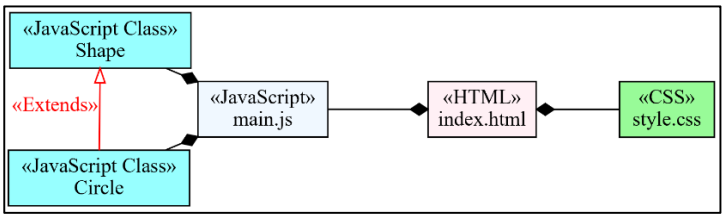

Figure 3.9 Profil « #6 classes minimum et aucune propriété »

Le troisième profil (voir Figure 3.9) ne montre aucune propriété pour un minimum de classes. Il s'agit des mêmes classes que dans l'image précédente, mais plus aucune propriété n'est présentée. Ce profil est plus efficace pour montrer une vue peu détaillée du système.

3.3 Spécification de l'outil

WAVI a été implémenté en Node.js puisqu'il existe d'excellents modules d'analyse de code source pour applications web tel que « Esprima » et « Cheerio ». En utilisant le même langage, cela a permis d'éviter des problèmes d'intégration de module et d'optimiser leurs performances. La gestion efficace des dépendances entre les modules et la qualité des modules de gestion de tâches asynchrones sont également des raisons de l'utilisation de Node.js comme langage. NPM est un autre avantage, il s'agit d'un répertoire qui contient une grande diversité de modules et permet deux choses importantes pour le projet. Premièrement, il permet d'utiliser des modules libres de droits dans la création de l'outil pour gagner du temps précieux. Deuxièmement, il s'agit d'une façon efficace de partager l'outil puisqu'il est disponible dans le répertoire. L'outil peut être utilisé avec une interface graphique ou une ligne de commande. L'utilisation de la ligne de commande permet de simplifier la génération des données pour les applications web à visualiser et d'automatiser leurs exécutions. Par exemple, un utilisateur peut générer les données pendant la nuit ou bien pour faire suite à un déploiement. De plus, l'usage de la ligne de commande facilite l'intégration future de WAVI avec d'autres outils externes. Pour une explication plus détaillée du fonctionnement de l'outil, un guide d'utilisation est disponible (voir ANNEXE X). L'outil peut générer trois sorties différentes, soit sous forme de représentation textuelle dans un fichier JSON, de diagramme de force dirigée ou de diagramme de classe.

3.4 Utilisation de WAVI

WAVI est assez simple d'utilisation pour les débutants. Il existe deux versions, la première possède une interface graphique et la deuxième est une version légère libre de droits disponibles sur NPM.

3.4.1 Application WAVI avec une interface graphique

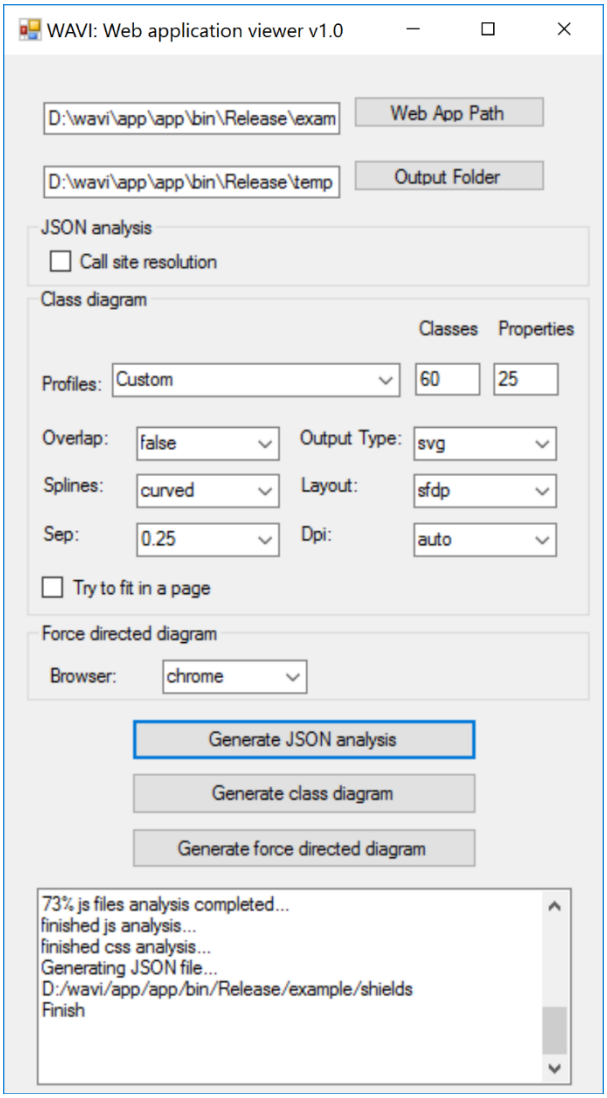

Figure 3.10 Interface graphique de WAVI

L'interface graphique de WAVI (fonctionnant avec le système d'exploitation Windows) demande la sélection de deux dossiers (Voir Figure 3.10). Le chemin vers l'application web à analyser et le chemin où seront générées les sorties. Les trois types d'analyse comportent des options. L'analyse JSON peut se faire avec ou sans la résolution d'appel. Les diagrammes de classes sont générés en fonction d'un profil (expliqué en détail dans la section 3.2.3.2) ainsi que plusieurs options reliées à Graphviz : l'entrelacement des éléments et liens, le type de

liens, le type de fichier généré, l'algorithme pour le positionnement des éléments, la distance entre éléments et le nombre de pixels par pouce. Il y a également l'option de modifier les dimensions du graphique dans la mesure du possible afin que celui-ci entre sur une page. Le diagramme de force dirigée a comme options, le navigateur dans lequel l'utilisateur souhaite voir le diagramme.

3.4.2 Wavi en module NPM

Cette version de l'outil est de type ligne de commande et le résultat généré peut être lu à l'aide d'un navigateur web. Par contre, il comporte plus de difficultés à l'installation. L'utilisateur doit posséder un serveur Node.js installé et configuré sur son ordinateur pour exécuter l'outil d'extraction et d'analyse et générer les données. De plus, la librairie « Graphviz » est requise, ce qui peut demander un certain temps et une expertise pour l'installation. L'outil est fiable, il fonctionne avec les 80 systèmes choisis et a été testé sur plusieurs autres applications web lors de son développement. L'utilisation de « Graphviz » pourrait être un obstacle pour l'exportation de l'outil WAVI sous forme de module. Cette librairie est une dépendance qui demande d'être configurée sur les systèmes utilisant WAVI et peut s'avérer compliquée à installer pour certains. De plus, « Graphviz » est une librairie qui possède des limites. Par exemple, certains caractères spéciaux ne peuvent être affichés et malgré beaucoup de temps consacré à l'optimisation du positionnement des éléments, il y a parfois de l'espace mal utilisé, des liens qui passent au travers des classes et certains textes qui se superposent.

3.5 Conclusion sur l'outil WAVI

Un outil a été créé afin de performer la rétro-ingénierie d'applications web. Le processus est divisé en plusieurs étapes, le code source est transformé en arbre syntaxique afin de faciliter la recherche d'éléments importants. Ces éléments sont ensuite sauvegardés dans une base de données. Les liens entre fichiers sont d'abord retrouvés puis c'est au tour des interactions entre les éléments.

Avec ces informations, WAVI peut générer 3 types d'artéfacts. Le premier est une représentation textuelle contenant 12 champs d'information sur les éléments (p.ex. : nom, groupe, valeur, position). Ensuite, il y a le diagramme de force dirigée qui présente les éléments sous forme de nœuds. Outre les nœuds qui peuvent être déplacés, plusieurs paramètres permettent de modifier la représentation graphique tels que la force, la friction et la distance entre les nœuds.

Finalement, il y a le diagramme de classe. Il s'agit d'un diagramme compact où les éléments sont représentés sous forme de classes et de propriétés. Les classes ont des couleurs différentes selon leur groupe. Les propriétés ont une visibilité, un nom, un type et une valeur. L'outil est distribué soit : avec une interface graphique (Windows) ou bien sous forme de module NPM (Windows, Linux, OSX).

CHAPITRE 4

VISUALISATION DES APPLICATIONS WEB

L'approche proposée par WAVI est de permettre aux développeurs de générer automatiquement une documentation sommaire d'un système et de la garder à jour avec un minimum d'effort. Cette documentation, composée des diagrammes de force et de classe, pourra servir sur demande quand il y a du changement de personnel ou bien simplement à titre de compréhension. Par exemple, lorsque 2 collègues veulent communiquer une idée ou un changement dans le système. Certaines expérimentations ont été faites sur les artéfacts produits par WAVI afin de connaitre les forces et faiblesses de l'outil ainsi que ses limitations.

4.1 Expérimentations de visualisation avec WAVI : la démarche

Dans un premier temps, des expérimentations ont été menée pour illustrer les capacités de la rétro-ingénierie lorsqu'elle est utilisée afin de visualiser la structure d'applications web et de générer de la documentation. Lors de cette expérimentation, WAVI a été utilisé sur 7 systèmes. Un diagramme de force dirigée des principaux éléments reliés a été généré. À l'aide de ce diagramme, l'utilisateur peut voir les principaux éléments du système rapidement. Ce diagramme est interactif et les éléments peuvent être déplacés et filtrés selon leur type (p. ex.: variables, fonctions, classes) en temps réel. Les résultats recueillis par l'outil sont tout d'abord commentés puis comparés à ceux des autres outils de rétro-ingénierie disponibles utilisant des diagrammes de force dirigée.

Une deuxième expérimentation a été menée cette fois-ci avec des diagrammes de classe pour mesurer l'efficacité de WAVI à présenter des structures d'applications web. En outre, les résultats sont passés en revue à l'aide des statistiques sur la quantité d'éléments produits selon les configurations de profil utilisées. WAVI est ensuite comparé aux autres outils générant des diagrammes de classes. Certaines observations ont été faites et une comparaison entre le diagramme produit et ceux des autres outils de rétro-ingénierie a été faite. La taille

des diagrammes générés est ensuite analysée selon les systèmes et les profils utilisés. Finalement, une attention particulière a été portée aux applications ES2015 afin de déterminer si les nouvelles fonctionnalités offertes ont un impact sur la structure des applications.

4.1.1 Échantillon de systèmes

Sept systèmes ont été choisis (voir ANNEXE XI) parmi les 80 de l'expérience précédente : la librairie Passport, le cadre de travail DI ainsi que les applications Screenshot, BrowseNPM, Shields, KiwiIRC et l'utilitaire Babel (voir Tableau 4.1).

Tableau 4.1 Échantillon de systèmes

Système	Type	#Fichier JS	LOC	C/S	ES
DI	Cadre de travail	7	594	S	ES2015
Passport	Librairie	8	455	S	ES5
Screenshot	Application	11	268	C/S	ES5
Shields	Application	15	4797	S	ES5
BrowseNPM	Application	19	533	C/S	ES5
KiwiIRC	Application	91	11281	C/S	ES5
Babel	Utilitaire	208	10390	C/S	ES2015

Certains systèmes ont été choisis, car ils comportent des particularités ; par exemple, le système DI a été choisi, car il utilise activement ES2015, les classes ainsi que le polymorphisme. Le système Shields a été pris en compte car il comprend une portion client avec des fichiers HTML/CSS ainsi que des formulaires pour générer des badges. De plus, la complexité des systèmes choisis varie. À titre d'exemple, ils contiennent de 8 (Passport) à 208 (Babel) fichiers JavaScript et de 268 (Screenshot) à 11281 (KiwiIRC) lignes de codes. Certains systèmes sont populaires, entre autres Passport avec 6900 recommandations sur github et plus de 7 millions de téléchargements depuis le début de l'année 2014. Il y a aussi Shields avec 2000 recommandations sur github et plus de 100 contributeurs.

4.2 Expérimentations avec les diagrammes de force dirigée : les résultats

L'outil WAVI permet de visualiser la structure sous forme de diagramme de force dirigée. Il offre une vue rapide de la structure d'un système et aide à la compréhension avec la visualisation des éléments de l'application. L'outil utilise un algorithme de force pour positionner les éléments dans le graphe. En général, les éléments sont placés de façon à bien voir les liens, mais pas nécessairement de façon optimale. L'utilisateur peut cependant déplacer les éléments comme bon lui semble à l'aide de la souris (p. ex.: pour dégager les éléments pertinents).

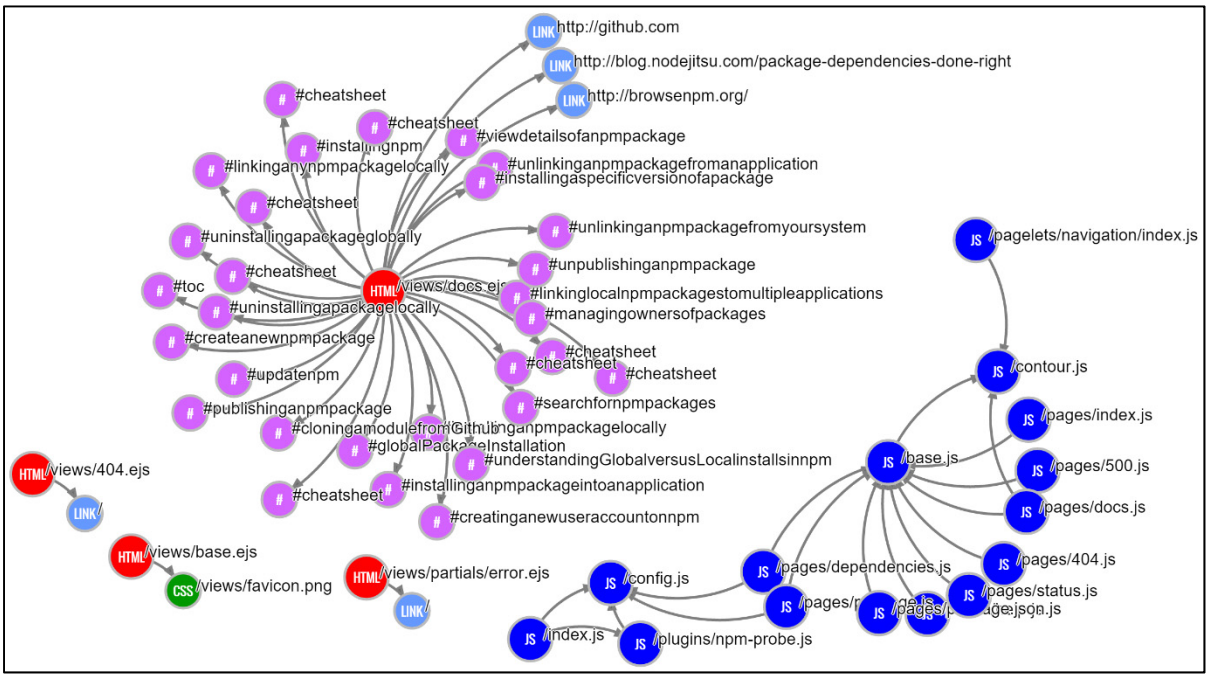

Figure 4.1 Diagramme de force dirigée de l'application BrowseNPM

Il n'est pas recommandé d'utiliser le diagramme de force dirigée pour voir tous les éléments d'un système de taille moyenne sans un minimum de configuration avec les paramètres (p.ex. : distance entre les éléments, filtre d'élément par type), car cela représente trop d'éléments à l'écran. Par contre, en utilisant les paramètres de l'outil, l'utilisateur a beaucoup de flexibilité et peut générer des diagrammes qui montrent seulement des parties spécifiques du système. En se servant des commandes d'agrandissement et en utilisant les divers

contrôles du graphe, il est possible de se concentrer sur un petit groupe d'éléments à la fois et de mieux visualiser les informations (voir Figure 4.1 et Figure 4.2). De plus, l'utilisation de la recherche permet de trouver un fichier particulier dans le graphe. Une autre façon de simplifier la visualisation d'un grand site web est de ne pas inclure tous les répertoires du site web dans l'analyse de l'outil. En spécifiant un seul répertoire, la visualisation peut être grandement améliorée.

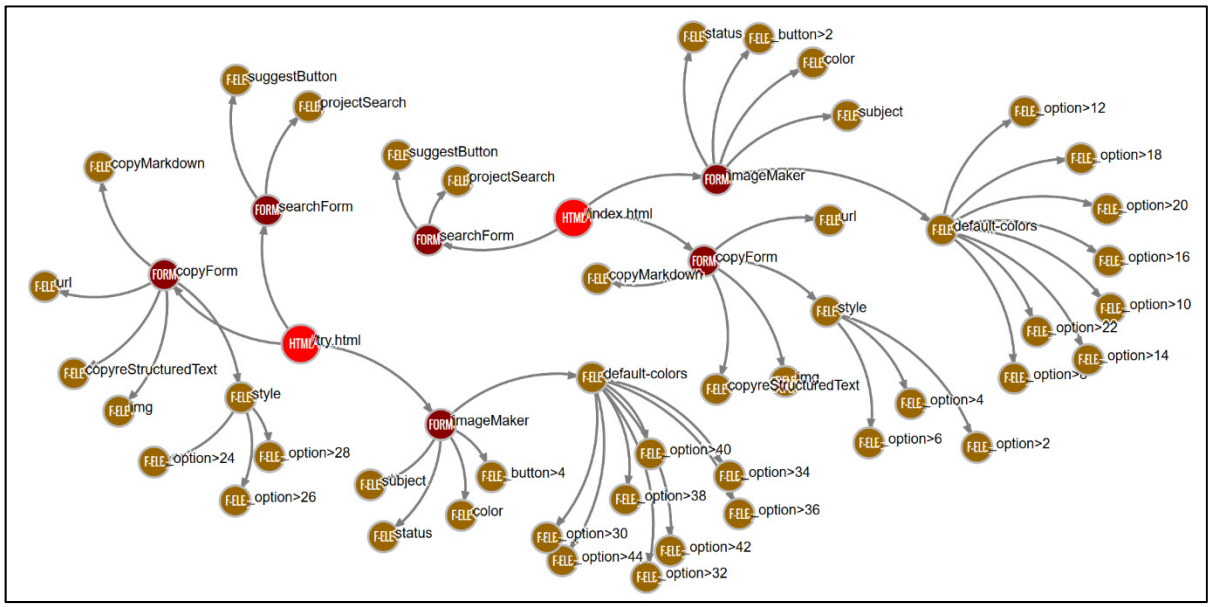

Figure 4.2 Diagramme de force dirigée des formulaires de l'application Shields

Le diagramme de force dirigée généré par WAVI se compare favorablement par rapport aux autres modules Node.js. Bien que le diagramme ressemble à celui de « Dependo », il ne présente pas simplement les fichiers JavaScript, mais aussi ceux de HTML, CSS ainsi que tous les éléments qui se retrouvent à l'intérieur (p. ex.: variables, fonctions, classes). Il y a une plus grande variété d'éléments à visualiser et plus d'options pour personnaliser le graphe. Le diagramme généré est plus facile à comprendre que celui généré par « Colony » qui ne possède pas des liens clairs avec les autres éléments et qui requiert que l'utilisateur survole l'élément avec le curseur de la souris afin d'afficher les informations. Finalement, il montre plus d'information qu'« AssetVis » telle que le sens des liens entre les éléments. Puisque ce

diagramme est interactif, il permet de déplacer librement les éléments afin de faciliter la visualisation.

4.3 Expérimentations avec les diagrammes de Classe : les résultats

Pour cette expérience, les diagrammes de classes des 7 systèmes ont été générés selon 6 profils différents qui sont directement en lien avec deux dimensions (le nombre de classes et de propriétés affichées, voir la section 3.2.3.2). Cela impacte la quantité d'éléments qui sera représentée dans le diagramme.

Tableau 4.2 Représentation des éléments (moyenne pour les 7 systèmes)

#Profil	# Classes	# Propriétés	Éléments visibles	Relations
1	Maximum	*	100% (tous des classes)	100%
2	Medium	Medium	24.35% (51% sont des classes)	10.77%
3	Minimum	Maximum	25.53% (19% sont des classes)	4.55%
4		Medium	16.83% (31% sont des classes)	4.55%
5		Minimum	10.48% (47% sont des classes)	4.55%
6		*	5.36% (tous des classes)	4.55%

Le Tableau 4.2 montre en moyenne comment les éléments ont été représentés selon les divers profils ainsi que les relations visibles. La première ligne du tableau représente le cas où 100% des éléments (incluant les variables, formulaires, boutons HTML et règles CSS, etc.) sont affichés ainsi que 100% des relations entre eux. La dernière ligne représente le profil le plus restrictif; il n'y a quasiment aucune propriété et les classes affichées sont presque toutes des fichiers, des classes JavaScript ou des modules externes. En moyenne, pour les 7 systèmes, cela représente 5.36% des éléments. Les 4 autres profils présentent une visualisation plus nuancée et balancée. Les plus gros systèmes présentés en utilisant le profil le plus restrictif possèdent pour KiwiIRC 111 classes et 64 relations et pour Babel 234 classes et 231 relations. Bien que le nombre de classes soit élevé, l'absence de propriété permet une visualisation relativement confortable.

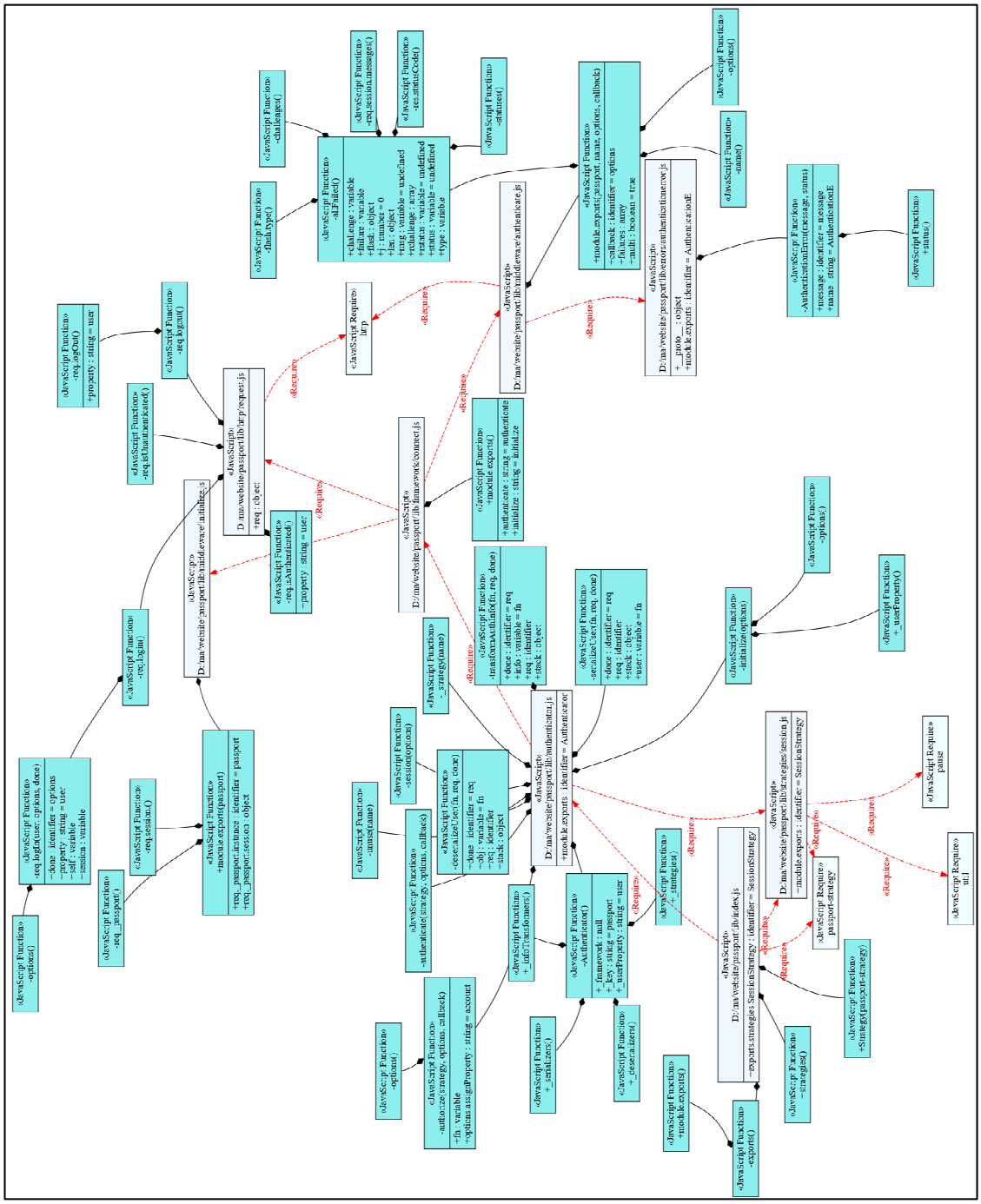

Figure 4.3 Diagramme de Passport avec le profil « #2 classes médium, propriétés médium »

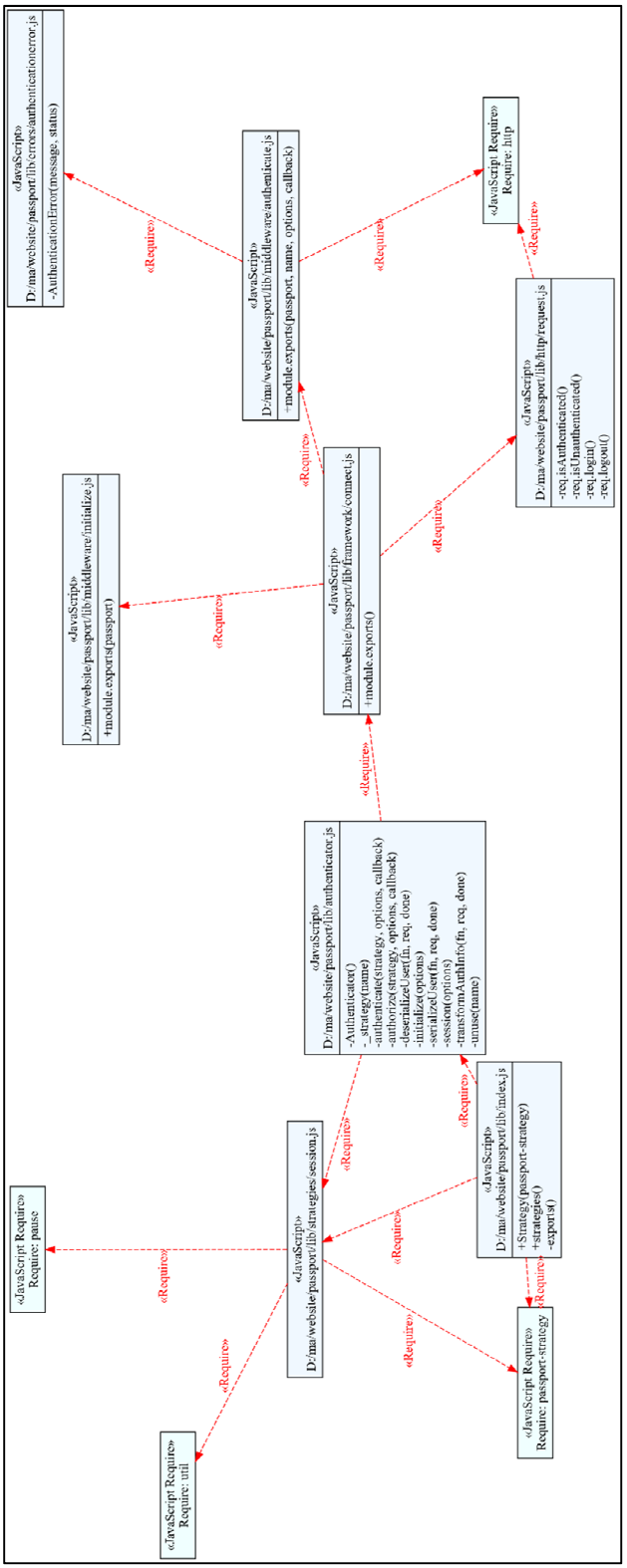

Figure 4.4 Diagramme de Passport avec le profil « #4 classes minimum, propriétés médium »

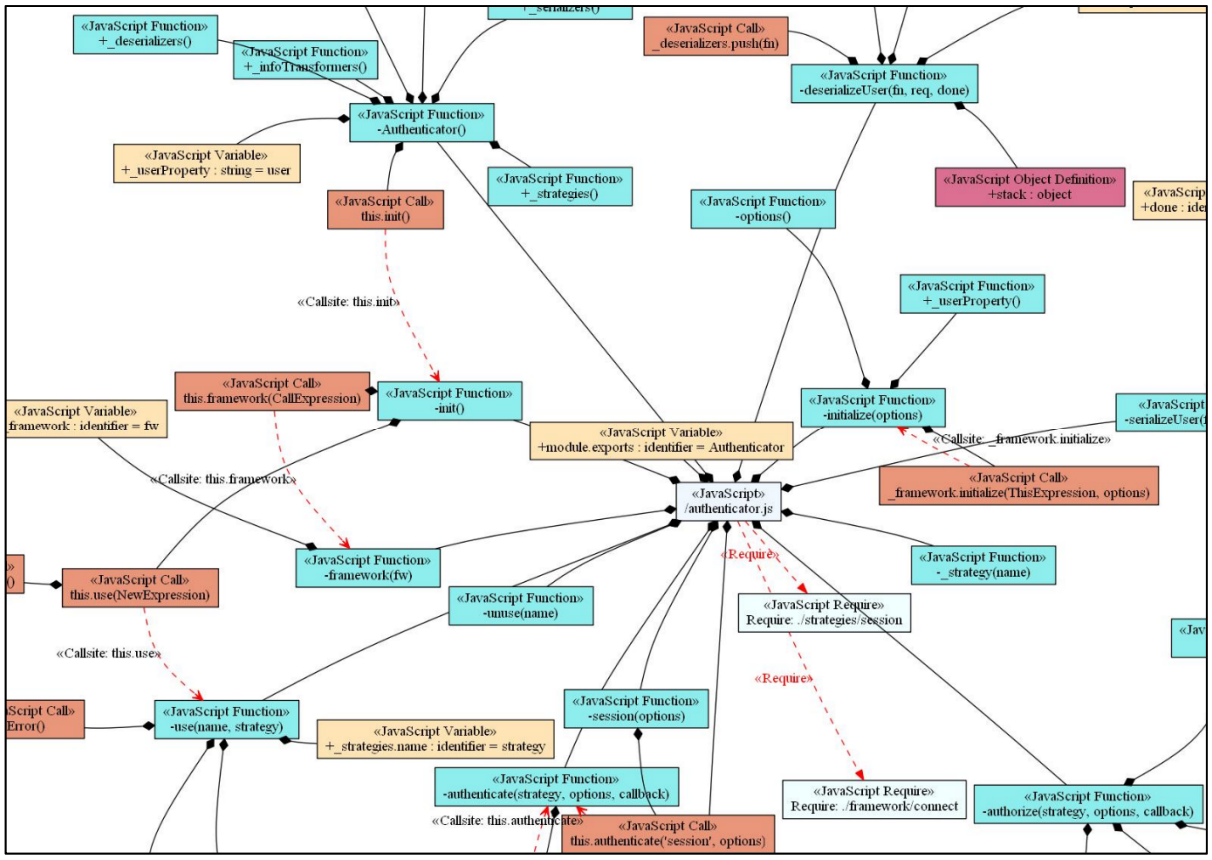

Figure 4.5 Extrait du diagramme de Passport pour le fichier « authenticator.js » avec le profil « #1 classes maximum, aucune propriété »

Les résultats de 3 profils du système Passport sont présentés (voir Figure 4.5, Figure 4.3 et Figure 4.4). Il s'agit d'un système relativement petit possédant 8 fichiers et quelques modules externes de type CommonJS. La plus grande classe générée par Wavi dans ce système est « authenticator.js » contenant une dizaine de fonctions. Il y a 4 librairies externes utilisées, soit « http », « util », « pause » et « passport-strategy ».

Parmi les diagrammes les plus intéressants pour la documentation (voir Tableau 4.2), « #3 classes minimum, propriétés maximum» est efficace si l'utilisateur désire avoir une vue globale du système avec le plus possible d'information dans un espace restreint. Il présente près du quart des éléments (25.53%) tout en ayant le moins possible de classes (19%) et de liens (4.55%). Pour un système de grande taille, le profil «#4 classes minimum, propriétés médium » est plus approprié. Il réduit à 16.83% le nombre d'éléments visibles tout en

gardant le même nombre de classes. Par contre, ce genre de présentation convient beaucoup mieux aux systèmes qui n'utilisent pas beaucoup de niveaux d'imbrication de fonction. Dans le cas de Passport, on peut apercevoir, avec le profil «#2 classes médium, Propriétés médium » de la Figure 4.3, que certaines fonctions sont imbriquées jusqu'à trois reprises dans le fichier « authenticator.js ». Si l'utilisateur désire seulement avoir une vue de haut niveau, le profil «#6 classes minimum, aucune propriété » se concentre seulement sur les fichiers et leur relation. Finalement, si l'utilisateur désire voir tous les éléments incluant les appels de fonctions, le profil «#1 classes maximum, aucune propriété » sera la meilleure option, car elle présente tous les éléments même ceux ayant peu d'importance (voir Figure 4.5). Ce profil peut être efficace si l'utilisateur désire montrer un sous-système ou un algorithme, car il présente beaucoup d'information.

4.3.1 Comparaisons avec les autres outils de rétro-ingénierie

Comparativement aux autres outils de rétro-ingénierie, WAVI est le seul outil avec « Reweb » à être fonctionnel sans nécessiter qu'on lui fournisse des scénarios d'utilisation. Cela lui permet de faire des analyses automatiquement sans l'aide de l'utilisateur et réduit la difficulté d'utilisation de l'outil. Les diagrammes de classes proposent un meilleur niveau de détail que les outils « WARE », « WANDA » et « WEBUML » en affichant la visibilité publique ou privée des éléments ainsi que leur valeur. WAVI propose une vue complète d'une application web moderne avec sa rétro-ingénierie JavaScript qui prend en compte les variables, fonctions, tableaux, objets et classes ES2015. De plus, il affiche les différentes relations entre fichiers et les spécifications de module CommonJS, AMD et ES2015. Par contre, WAVI possède certaines limites ; il ne peut pas présenter les éléments avec des relations de type connecteur logique tel que « WebUML » et il ne peut enregistrer la fréquence d'invocations entre les éléments comme le fait « WANDA ». Ces limitations sont en majorité dues au fait que WAVI ne fait pas d'analyse dynamique. Cela explique aussi qu'il n'y ait pas de génération de diagrammes d'état ou bien de séquences telles que FireDetective et DynaRIA. La plupart des outils de rétro-ingénierie peinent à extraire des informations à la fois du côté client et du côté serveur. Ce n'est pas le cas pour WAVI,

comme on peut le voir pour l'application « Shields », qui est un système de grande taille comportant un côté client avec 3 pages HTML, 2 fichiers CSS et 4 JavaScript Markup ainsi qu'un côté serveur avec 11 fichiers JavaScript.

4.3.2 Observations sur les systèmes étudiés

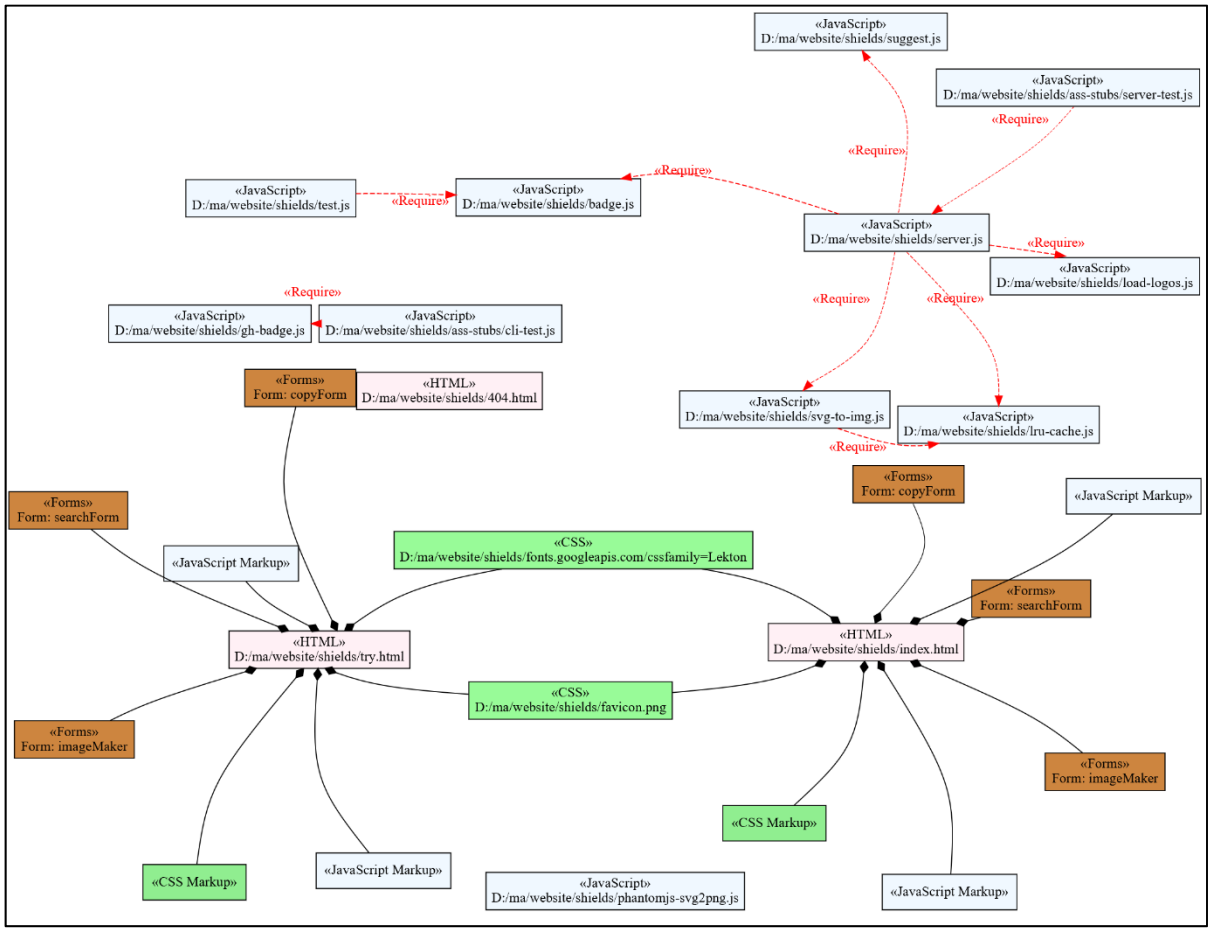

Figure 4.6 Diagramme de classes de Shields avec le profil « #6 classes minimum, aucune propriété »

Certaines observations ont été faites sur les 7 systèmes étudiés. Les systèmes Node.js utilisent beaucoup de modules externes. La facilité d'utilisation de ces modules et leur efficacité permet aux développeurs de sauver du temps lors du développement et permet de simplifier la structure des applications. Il a également été observé que les balises JavaScript (code JavaScript à l'intérieur d'un fichier HTML) ne contiennent pas de déclarations de

fonctions et de variables dans les 7 systèmes étudiés. Ils ne contiennent que des appels de fonctions qui sont déclarés dans d'autres fichiers. Cette observation est rassurante, car l'utilisation de ces balises n'est pas une pratique recommandée dans l'industrie.

4.3.3 Tailles des diagrammes générés

Lors de nos expérimentations, nous avons étudié les diagrammes générés par WAVI. Nous avons classé selon 4 tailles ces diagrammes générés avec les 6 profils (voir Tableau 4.3). Premièrement, il y a la taille d'une feuille de papier donc idéal pour documenter une application à l'intérieur d'un document papier. Deuxièmement, la taille d'un écran d'ordinateur ce qui est acceptable pour documenter sous forme électronique. Troisièmement, plus grand que la taille d'un écran d'ordinateur (1080p) donc forçant l'utilisation des barres de défilement afin de visualiser le système en entier et finalement de très grande taille pouvant représenter des difficultés de lecture (plus de 4 fois la taille d'un écran 1080p).

Tableau 4.3 Taille des diagrammes générés

			# Profils					
Système	**#Fichier JS**	**LOC**	**1**	**2**	**3**	**4**	**5**	**6**
DI	7	594	C	B	A	A	A	A
Passport	8	455	C	B	A	A	A	A
Screenshot	11	268	C	A	A	A	A	A
Shields	15	4797	D	C	C	B	B	A
BrowseNPM	19	533	D	B	B	A	A	A
KiwiIRC	91	11281	D	C	C	C	C	B
Babel	208	10390	D	D	C	C	C	C

Tailles

A – Diagramme de la taille d'une feuille de papier
B – Diagramme de la taille d'un écran d'ordinateur
C – Diagramme plus grand que la taille d'un écran d'ordinateur
D – Diagramme de très grande taille (difficulté de lecture)

Pour le profil le plus compact (#6 classes minimum, aucune propriété), la grande majorité des systèmes (71%) peuvent être présentés sur une feuille de papier. Les systèmes KiwiIRC et Babel ne peuvent être présentés sur une feuille, car ils possèdent beaucoup de fichiers JavaScript (91 et plus) ce qui augmente la taille des diagrammes. Le profil (#4 classes minimum, propriétés médium) et (#5 classes minimum, propriétés minimum) obtiennent des résultats identiques au profil (#6) sauf pour les systèmes Shields (+ de 65 classes) et KiwiIRC (+ de 188 classes) qui augmentent d'une catégorie en taille.

La grande majorité des systèmes (71%) peuvent être présentés sur une feuille de papier avec le profil le plus compact (#6 classes minimum, aucune propriété). Les systèmes KiwiIRC et Babel ne peuvent être présentés sur une feuille, car ils possèdent beaucoup de fichiers JavaScript (91 et plus) ce qui augmente la taille des diagrammes. Lorsque le nombre de classes est au minimum (Profil #3,#4,#5,#6), aucun diagramme n'est de très grande taille. Ce sont donc les profils les plus efficaces pour la documentation d'un système complet sans trop de problèmes. Le système Babel est le premier à générer un diagramme de grande taille et présenter des difficultés de lecture (plus de 1000 relations entre les éléments) avec le profil (#2 Classe medium, propriétés médium). La plus grande difficulté de lecture se situe au niveau des multiples relations qui se superposent et qui deviennent difficiles à lire. Avec ce profil, les fonctions les plus importantes forment des classes ce qui prend plus de place. Le profil (#1 classes maximum, aucune propriété) fait pour représenter des algorithmes ou des sous-systèmes a de la difficulté à représenter un système complet. Avec ce profil, un total de 4 systèmes sur 7 génèrent des diagrammes de très grande taille avec des problèmes de lisibilité.

4.3.4 Les systèmes ES2015

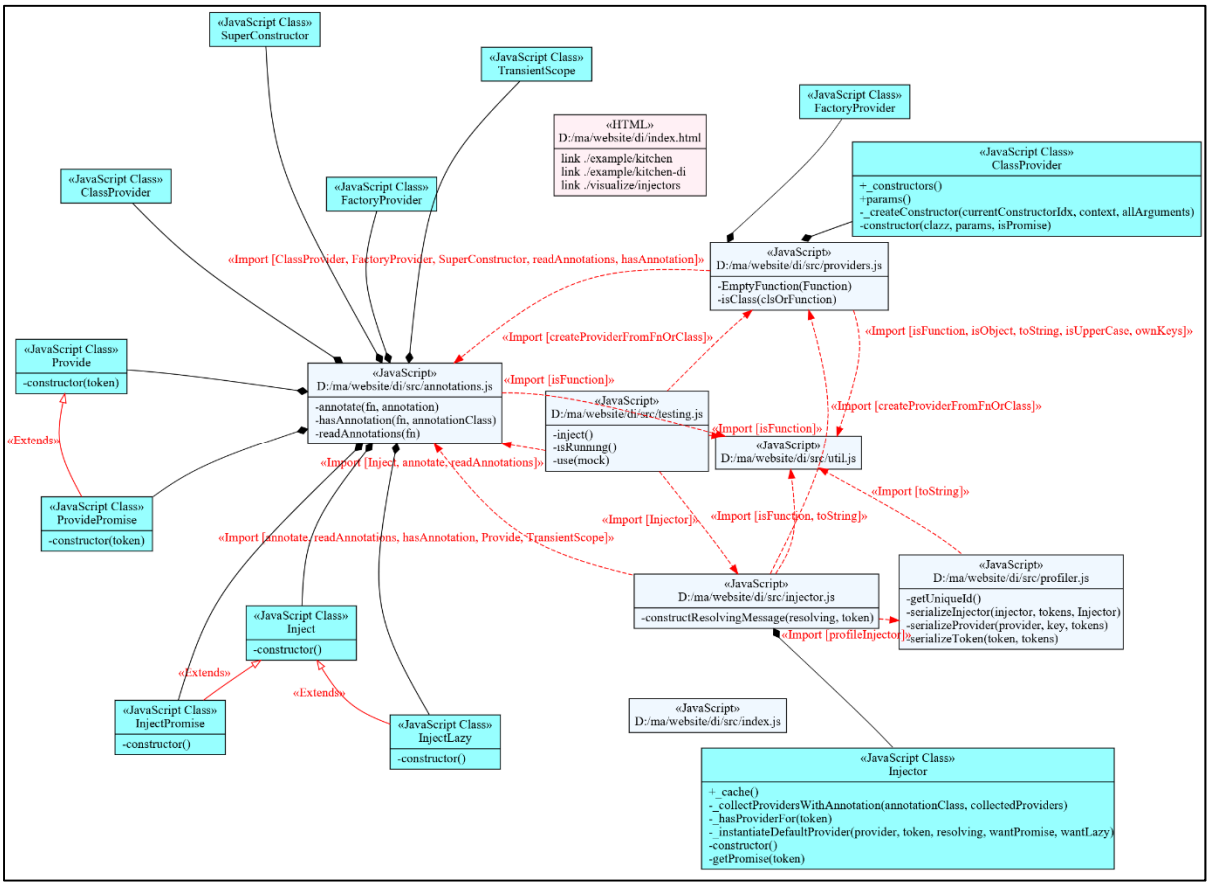

Figure 4.7 Diagramme de classes avec le profil « #5 classes minimum, propriétés minimum » du système DI

Parmi les 7 systèmes évalués, « DI » semble être l'un des plus faciles à documenter à l'aide de diagrammes, et ce pour plusieurs raisons (voir Figure 4.7). Premièrement, il s'agit d'un système qui utilise ES2015 qui respecte mieux le paradigme de la programmation orientée-objet que ES5. Le système « DI » utilise un environnement plus structuré et n'a pas besoin d'utiliser des imbrications de fonction. Comme exemple, le système « DI » ne possède aucune fonction imbriquée à l'intérieur des classes JavaScript. Bref, il n'y a plus d'ambiguïté avec l'usage des fonctions, car le développeur n'a pas besoin d'utiliser les fonctions pour émuler la structure d'une classe.

Deuxièmement, en plus de voir réellement des classes, on voit également l'utilisation de polymorphisme avec le constructeur « extends ». Cet ajout d'ES2015 permet aux développeurs de mieux structurer leurs applications et, par le fait même, rend les systèmes plus faciles à documenter. Finalement, ES2015 se démarque également par son système d'importation qui permet de spécifier exactement quelle fonction le développeur désire importer et ainsi mieux comprendre pourquoi les fichiers dépendent les uns des autres. Cela peut, par exemple, servir à mieux évaluer l'impact de la maintenance de code d'un fichier selon ses dépendances.

4.4 Améliorations possibles de l'outil WAVI

La création de l'outil de rétro-ingénierie WAVI s'intéresse au problème d'analyse de la structure d'applications web. Il propose une solution simple et facile d'utilisation. Cependant, certaines améliorations permettraient de rendre l'outil plus attrayant pour les développeurs. Voici une liste d'améliorations possibles.

- **Le support de nouveaux langages :** La possibilité de visualiser de nouveaux langages client ou serveur est probablement l'amélioration qui est la plus intéressante. Elle permettrait de voir une plus grande variété de structures d'application web, de mieux comprendre les liens qui existent entre le côté client et le côté serveur. Parmi ces langages, il y a entre autres PHP, Java, Python et Ruby qui sont des choix intéressants. Du côté client, des langages qui se compilent en CSS, HTML et JavaScript tel que « Less » « Stylus », « Jade » et« CoffeeScript » pourraient être ajoutés afin d'augmenter la compatibilité de l'outil.

- **La perspective de la vue :** La vue actuelle prend la perspective des fichiers, mais elle pourrait prendre d'autres formes. Par exemple, en prenant un niveau plus haut, il serait possible de voir la structure en fonction des dossiers. Cela permettrait de proposer davantage de niveaux de détail en fonction du site web et des besoins du développeur.

- **L'évolution de la structure :** Une des améliorations possibles serait de permettre la mise à jour du résultat d'une analyse en tenant compte seulement des fichiers qui ont changé depuis la dernière analyse. De cette façon, le temps de calcul serait grandement réduit. Cela ouvre aussi la porte à de nouvelles fonctionnalités comme la possibilité de voir l'évolution de la structure à travers le temps et de choisir quelle version de la structure on désire afficher. L'évolution des résultats des analyses permettrait aussi de voir si les modifications apportées auront des effets positifs ou négatifs sur les qualités du site web comme la performance et la maintenabilité.

- **La modification de la structure :** Une fonctionnalité intéressante qui pourrait être ajoutée est la possibilité de modifier le graphique vectoriel (svg) du diagramme de classes de façon dynamique. Une fois le graphe généré, l'utilisateur pourrait repositionner les éléments qu'il désire et supprimer ceux qui ne l'intéressent pas. Pour l'instant, il ne peut le faire sans l'utilisation d'un logiciel vectoriel externe tel que Adobe Illustrator ou Corel Draw.

- **Profils de visualisation :** L'ajout de nouveaux profils pourrait être intéressant afin d'avoir de nouvelles façons de visualiser les données. Par exemple, un profil avec les éléments du côté client seulement ou bien les éléments du côté serveur.

4.5 Conclusion sur la visualisation des applications web

Deux diagrammes sont générés par l'outil WAVI, un utilisant la force dirigée et l'autre utilisant des classes et propriétés. Le diagramme de force dirigée offre une vue des éléments sous forme de nœuds interconnectés. L'outil propose plusieurs paramètres afin de positionner et filtrer les nœuds. Il offre un meilleur niveau de détail que les autres modules de visualisation similaires. De plus, le diagramme de force dirigée gère mieux l'imbrication des éléments que le diagramme de classes (p.ex. : une variable à l'intérieur d'une fonction qui est à l'intérieur d'une autre fonction) (voir un exemple de graphique d'appel ANNEXE XIV).

En général, le diagramme de classes est plus compact et lisible que celui de force dirigée. Il offre plusieurs profils différents selon les désirs de visualisation de l'utilisateur. Il se compare bien aux autres outils car il n'a pas besoin d'exécuter le code et ne nécessite pas l'interaction de l'utilisateur pour générer les diagrammes. La taille des diagrammes générés pour les profils 2 à 6 ne présentent pas de difficultés de vision pour 6 des 7 systèmes. Le profil 1 quant à lui est plus pertinent pour visualiser des algorithmes que des systèmes en entier. Les systèmes ES2015 sont plus faciles à lire, l'utilisation de nouvelles fonctionnalités telles que les classes réduit l'ambiguïté sur le rôle des fonctions. Les relations entre les modules sont également plus simples à suivre et à documenter avec les nouveaux paramètres du constructeur d'importation ES2015.

CONCLUSION

Le développement d'applications web est une discipline complexe qui évolue constamment et qui fait usage de plusieurs langages et technologies. Rien n'indique que dans un futur rapproché que l'utilité et la popularité des applications web s'affaibliront. Pourtant, les outils de développement web proposés sont rares et immatures, notamment à cause de la nature dynamique de JavaScript. Il est primordial de développer de nouveaux outils de rétro-ingénierie afin de mieux comprendre la structure des applications web et ainsi pouvoir les optimiser, les documenter et les modifier. Certains outils de rétro-ingénierie devraient produire des résultats nonobstant le fait que le code source visé soit complet ou non (p.ex. : en utilisant l'analyse statique). Afin de contribuer à la résolution de ce problème, nous avons tout d'abord vérifié que l'analyse statique est une approche viable. Dans une étude empirique, nous avons trouvé que l'utilisation des constructeurs dynamiques telles qu'« eval », « new Function », « with », « call » et « apply » ainsi que les constructeurs pour la dépendance entre fichiers ne causent pas de problème significatif. De plus, une heuristique basée seulement sur le nom, telle qu'elle est utilisée par Eclipse JSDT pour la résolution statique d'appel de fonction, n'est pas suffisante. Elle permettrait seulement la résolution de la moitié des cas. Nous avons donc proposé un ensemble de filtres intuitifs pour réduire l'ambiguïté. Ces filtres basés sur la dépendance de fichier, la hiérarchie, la portée, l'attachement à un objet et le nombre de paramètres ont ensuite été combinés pour améliorer la résolution d'appel. Ils ont démontré des résultats meilleurs que ceux d'une approche récente et plus complexe (Feldthaus, Schafer et al. 2013).

Suite à cette analyse, nous avons créé un outil de rétro-ingénierie basé sur l'analyse statique appelé WAVI. Cet outil permet d'extraire la structure d'une application web, de la représenter sous forme de représentation textuelle JSON ou bien de la visualiser à l'aide de deux types de diagrammes (force dirigée et de classe). Le diagramme de force dirigée est interactif et permet de filtrer les éléments par type et de repositionner les nœuds selon les désirs de l'utilisateur pour des vues personnalisées. Le diagramme de classes propose quant à lui des profils de visualisation différents afin de pouvoir adapter les résultats à la taille des

systèmes. L'outil créé est actuellement limité sur certains aspects comme le nombre de langages pris en charge, mais les résultats démontrent que l'outil est efficace. Les nouvelles avenues pour WAVI sont entre autres le support de nouveaux langages, que ce soit du côté client ou bien du côté serveur. Ensuite, il y a la possibilité de voir l'évolution de la structure à travers le temps, de visualiser la structure à l'aide de nouvelles perspectives par exemple, en fonction des dossiers. De plus, il serait possible d'utiliser WAVI afin de réaliser de nouvelles analyses et études empiriques sur la structure des applications web. Bref, l'outil développé démontre du potentiel et offre des possibilités d'extension dans le futur.

ANNEXE I

RELATIONS ENTRE FICHIERS JAVASCRIPT, CSS ET HTML

Il est possible de relier du code CSS avec un fichier HTML de deux façons. Premièrement, utiliser une balise <LINK> reliée à un fichier externe ou bien utiliser une balise <STYLE> à l'intérieur du fichier HTML et y inclure directement le code CSS à l'intérieur.

```
<LINK REL="StyleSheet" HREF="mainStyle.css" TYPE="text/css"
MEDIA="screen">
<STYLE TYPE="text/css" MEDIA="screen">
   p { text-align:right;
   }
</STYLE>
```

De plus, il est possible d'importer du code CSS d'un fichier externe à l'intérieur du code CSS en utilisant la fonction « @import ». Maintenant, si l'on regarde les relations possibles entre les fichiers HTML et les fichiers JavaScript, il y a deux façons de créer un lien. Utiliser la balise <SCRIPT> à l'intérieur d'un fichier HTML ou inclure le code JavaScript directement dans le fichier HTML en utilisant une balise <SCRIPT>.

```
<STYLE type="text/css" MEDIA="screen">
   @import url("mainStyle.css");
</STYLE>
```

```
<SCRIPT type="text/JavaScript" src="fichier.js"></SCRIPT>
…
<SCRIPT type="text/JavaScript">
   document.write("Hello World!");
</SCRIPT>
```

ANNEXE II

COMMONJS

Voici un exemple d'un module effectuant de simples addition et soustraction mathématiques.

```
// Fichier MonModuleDeMath.js
module.exports = {
    addition: function(a,b) {
      return a + b;
    },
    soustraction: function(a,b) {
      return a - b;
    }
};
```

Il y a tout d'abord le fichier utilisant le constructeur module.exports pour déclarer les méthodes d'opérations mathématiques rendues disponibles par le module.

```
// Fichier MonProgramme.js
var moduleMath = require("./MonModuleDeMath");

var a = 4;
var b = 1;

moduleMath.addition(a,b); // retourne 5
moduleMath.soustraction(a,b); // retourne 3
```

Le fichier « MonProgramme.js » utilise le module de mathématique. Le constructeur « require » est utilisé afin d'assigner une instance du module « MonModuleDeMath.js » dans la variable « moduleMath ». Chaque module est chargé un après l'autre avec la fonction « require » et l'utilisateur peut ensuite utiliser ces modules qui se retrouvent encapsulés dans une variable.

Il existe une différence fondamentale entre l'exécution de code JavaScript avec Node.js et son exécution dans les navigateurs web. Il s'agit de la portée des variables dans les fichiers. Cette syntaxe de commonJS fonctionne bien avec Node.js, car la portée des variables est limitée au contenu des fichiers. Bref, les variables « moduleMath », « a » et« b » sont inaccessibles si elles ne sont pas exportées alors que lorsque le code est exécuté dans un navigateur, la portée des variables se transporte au-delà des fichiers et se répand à tous les fichiers importés. Avec cette différence, l'utilisation de cette implémentation de CommonJS pourrait rapidement mener à des conflits si les modules utilisent des noms de variable communs. Il existe donc des variantes d'implémentation de CommonJS pour les navigateurs ou bien il est possible d'utiliser la spécification AMD.

ANNEXE III

AMD

AMD permet de charger les scripts justes à temps, donc seulement lorsque nécessaire. Voici un exemple de définition de module AMD.

```
// Fichier MonModuleDeMath.js
define("MonModuleDeMath",function () {
    return {
    addition: function(a,b) {
      return a + b;
    },
    soustraction: function(a,b) {
      return a - b;
    }
};
});
```

On peut voir dans cet exemple que le premier argument de la fonction « define » est le nom du module et le deuxième argument est une fonction qui retourne les variables et méthodes du module.

Il faut noter que la portée des variables se limite à l'intérieur de la fonction, ce qui rend le code plus résistant aux failles de sécurité de même que plus robuste aux erreurs qui pourrait arriver lorsque l'on charge plusieurs scripts et que les variables entrent en conflit. Voici un exemple d'utilisation du module AMD avec la fonction « require ».

```
// Fichier MonProgramme.js
require(["MonModuleDeMath"], function (moduleMath) {
    var a = 4;
var b = 1;
    document.write(moduleMath.addition(a,b));
// retourne 5
    document.write("<br>");
    document.write(moduleMath.soustraction(a,b));
// retourne 3
});
```

Dans cet exemple, « moduleMath », n'est pas accessible à l'extérieur de la fonction « require ». Le premier paramètre est un tableau des modules nécessaire pour l'exécution du code. Le second paramètre de « require » est une fonction contenant comme paramètre la variable récipiendaire des fonctionnalités du module requis dans le premier paramètre.

ANNEXE IV

ES2015

Voici un exemple de l'implémentation de module en ES2015.

```
// fichier MonModuleDeMath.js
var addition = function(a,b){
return a + c;
}
var soustraction = function(a,b){
return a - b;
}
export { addition, soustraction };
```

Le fichier « MonModuleDeMath.js » contient des variables et méthodes que le constructeur « export » rend disponibles publiquement aux modules qui désirent les utiliser. Et maintenant, voici un exemple d'utilisation de module en ES2015.

```
// Fichier MonProgramme.js
import {addition,soustraction} from "./MonModuleDeMath.js";

var a = 4;
var b = 1;

console.log(addition(a,b));
console.log(soustraction(a,b));
```

Le constructeur « import » prend en paramètre entre accolades les fonctionnalités du module « MonModuleDeMath » qu'il désire utiliser. Il pourrait également remplacer ces paramètres par l'étoile « * » pour importer toutes les fonctionnalités automatiquement.

Si le développeur désire utiliser le module de façon asynchrone du côté client, comme il était possible avec AMD, il peut utiliser l'objet global « System.import ».

```
// Fichier MonProgramme.js
System.import("./MonModuleDeMath.js").then(function(moduleMath) { var
a = 4;
      var b = 1;
      console.log(moduleMath.addition(a,b));
      console.log(moduleMath.soustraction(a,b));
});
```

ANNEXE V

EXEMPLE EXTENSION WAE UML

Voici une application web standard utilisant UML sans l'extension WAE:

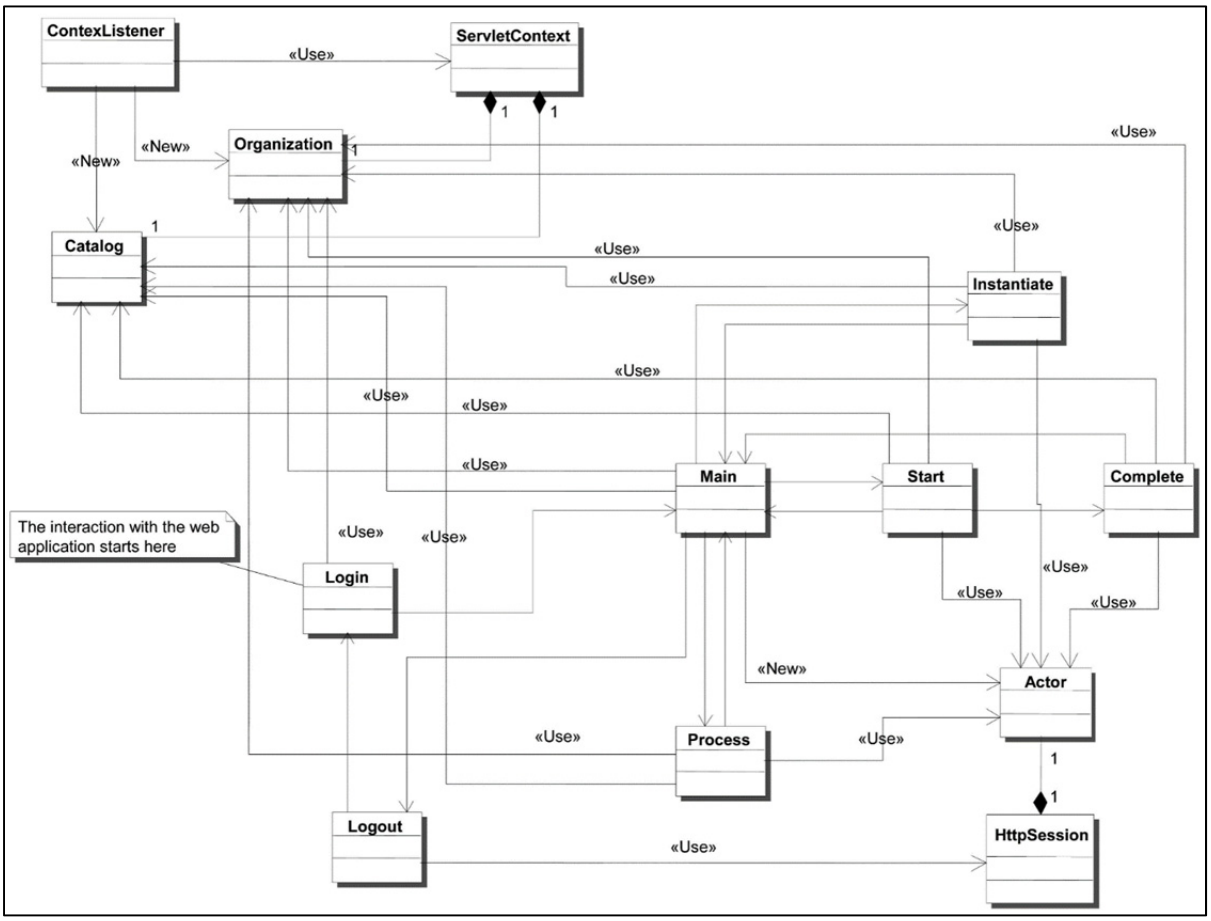

Figure-A V-1 diagramme UML standard
Tirée de (Ricca, Di Penta et al. 2010)

Il existe plusieurs éléments, mais il est difficile de savoir s'il s'agit de pages situées du côté client ou bien s'il s'agit de pages du côté serveur. De plus, il n'existe pas un bon niveau de détail pour la description des relations. Les relations possibles entre éléments sont « use » ou « new ». Les éléments ne contiennent aucune information puisqu'il n'y a pas de standard défini pour le faire.

Maintenant, voici un exemple de la même application web en utilisant l'extension WAE.

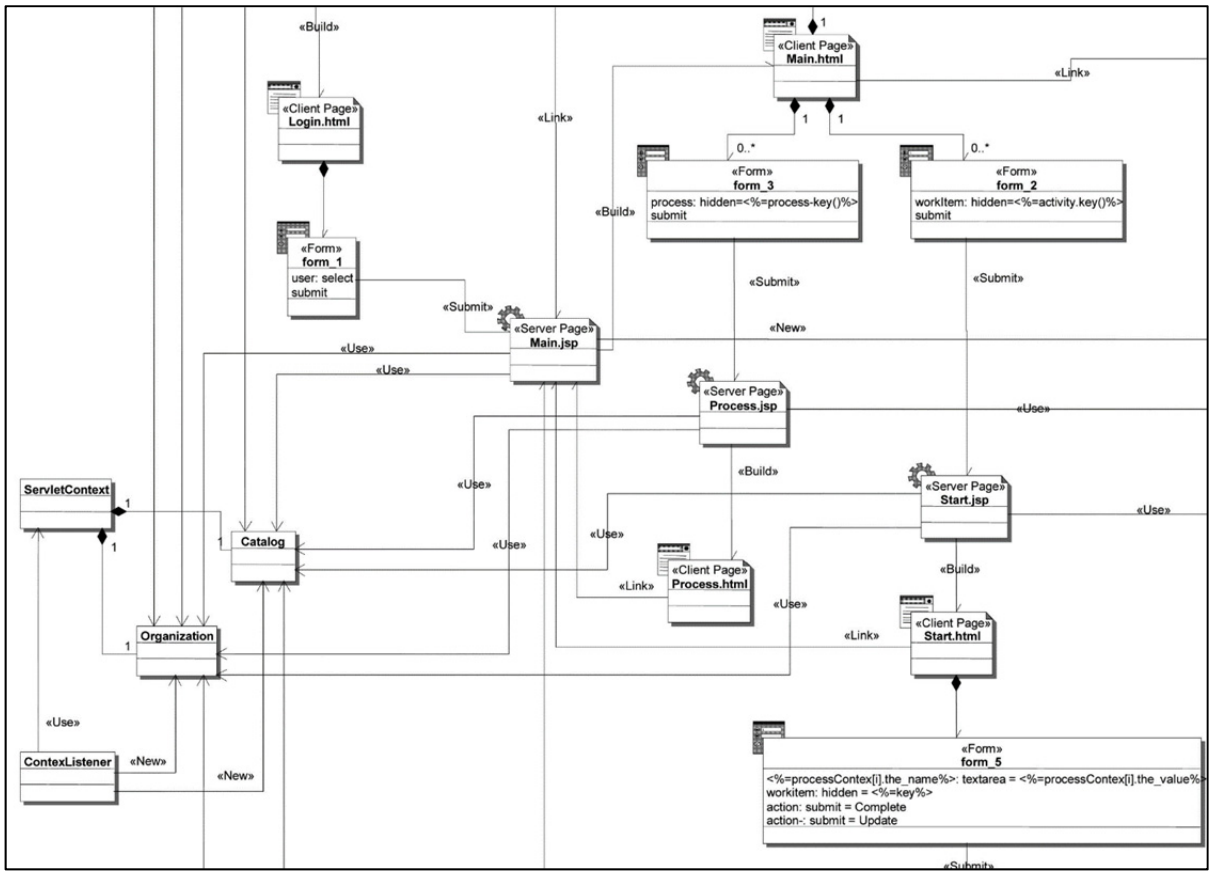

Figure-A V-2 extrait du diagramme utilisant l'extension WAE
Tirée de (Ricca, Di Penta et al. 2010)

Une fois tous les ajouts d'information de Conallen au diagramme UML standard effectués, la modélisation d'application web se fait de façon beaucoup plus naturelle. Il est maintenant plus facile de distinguer le type de page, savoir ce que les éléments du diagramme représentent et avoir les informations pertinentes à l'intérieur de ceux-ci.

ANNEXE VI

ÉLÉMENTS ET ASSOTIATIONS DE L'EXTENSION WAE UML

Il existe plusieurs éléments de base qui utilisent le modèle de classe:

Tableau A VI-1 Éléments WAE

Éléments	Description
Les pages serveur	Un script ou une classe qui est exécuté sur le serveur. Par exemple un servlet, une classe java, une classe PHP.
Les pages clientes	Une page HTML ou le résultat d'une page dynamique généré par le serveur.
Les pages JavaScripts	Un script ou une classe JavaScript exécuté sur l'ordinateur client
Les formulaires	Un formulaire situé à l'intérieur d'une page HTML.
Les « Frameset »	Un ensemble de cadres.
Les « Target »	Les cibles de liens.

Ensuite, il y a les associations entre ces éléments :

Tableau A VI-2 associations WAE

Association	**Description**
Les liens hypertextes standard	Un hyperlien d'une page cliente à une page serveur ou cliente.
Les liens ciblés	Un hyperlien d'une page standard, mais redirigé dans un cadre ou une autre fenêtre.
Les liens avec bouton	Il s'agit d'une action « submit » qui relie le bouton d'envoi d'un formulaire vers une page serveur.
L'association «Build»	C'est une association « Build » entre une page cliente et une page serveur.
Rediriger	C'est une page client ou une page serveur qui est redirigée vers une autre page.
Objet	C'est une relation entre une page cliente et un objet qui pourrait être un flash, un applet, un activeX ou bien une vidéo.
Include	C'est une association directe entre une page serveur et une autre page serveur ou cliente dont le contenu est utilisé par le parent.

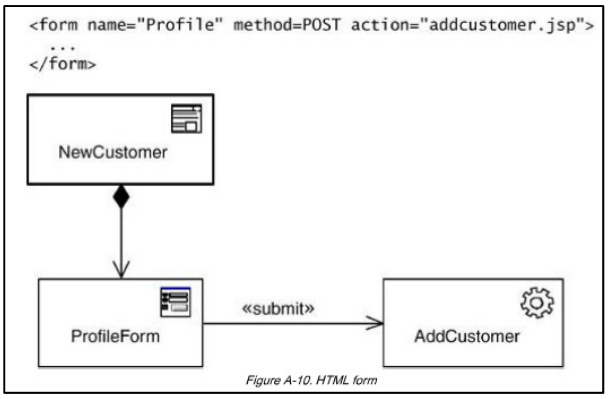

Figure-A VI-1 action « submit »
Tirée de (Conallen 2003)

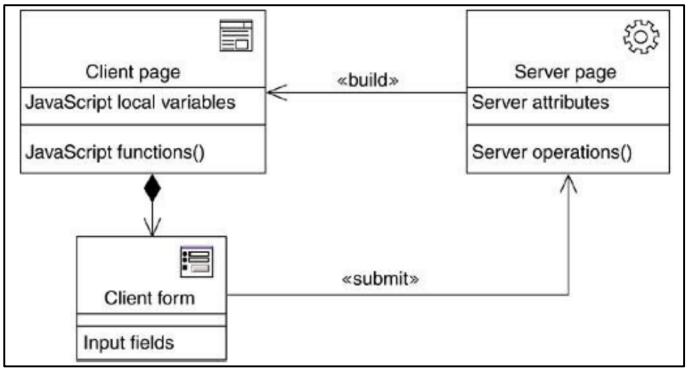

Figure-A VI-2 association « build »
Tirée de (Conallen 2003)

ANNEXE VII

ANALYSE PRÉALABLE SUR 80 SYSTÈMES

Tableau A VII-1 ANALYSE PRÉALABLE SUR 80 SYSTÈMES (1/3)

Système	Type	Env	Nb Fichier	SLOC
Ace	A	C	620	159540
Aditti	A	C	38	2478
Bracket	A	C	785	98496
Browsenpm	A	C/S	19	533
Clientsapp	A	C/S	51	16967
Countly	A	C/S	215	43918
Fullcalendar	A	C	9	12643
Iloveopensource	A	C/S	74	4908
Kiwiirc	A	C/S	91	11281
Map	A	C	10	9376
Mediacenterjs	A	C/S	68	4728
Paintweb	A	C	21	8175
Screenshot	A	C/S	11	268
Shields	A	C/S	15	4797
Slate	A	C/S	20	446
Todoes6*	A	C/S	5	1434
Winecellar	A	C/S	15	676
Angular	C	C	102	14986
Aurelia*	C	C	5	420
Backbone	C	C	1	1067
Cylon	C	S	42	3666
Di*	C	S	7	594
Express	C	S	11	1676
Hapi	C	S	16	3401
Jasmine	C	S	55	3087
Koa	C	S	4	565
Mocha	C	S	45	3050

Tableau A VII-2 ANALYSE PRÉALABLE SUR 80 SYSTÈME (2/3)

Système	Type	Env	Nb Fichier	SLOC
Mootools	C	C	1	4044
Prototype	C	C	1	5572
Socketio	C	C/S	8	2724
Spry	C	C	19	11488
Clumsybird	J	C	15	847
Cobaltcalibur3	J	S	9	708
Hextris	J	C	18	1805
Mkjs	J	C/S	5	2074
Nodegames	J	C/S	22	10715
hooter				
Pacman	J	C	2	3513
Pandajs	J	C/S	14	14654
Pong	J	C	3	3670
Tank	J	C	9	799
Tictacnode	J	C/S	17	713
Chartjs	L	C	6	2645
D3*	L	C	290	11697
Flotr	L	C	23	5024
Jquery	L	C	84	5514
Modernizr	L	C	1	612
Passport	L	S	8	455
Pixi	L	C	87	11563
Raphael	L	C	1	5745
React	L	C/S	57	4418
Router*	L	S	14	1239
Rsvp*	L	S	32	1212
Scriptaculous	L	C	9	8541
Three	L	C	165	18671

Tableau A VII-3 ANALYSE PRÉALABLE SUR 80 SYSTÈME (3/3)

Système	Type	Env	Nb Fichier	SLOC
Underscore	L	C/S	1	1070
Videojs*	L	C	74	5619
Devkit	P	S	91	10646
Ghost	P	C/S	145	11233
Nodebb	P	C/S	162	23408
Pdf	P	C	2	31694
Reddit	P	C	61	9862
Strider	P	C/S	100	7444
3dmodel	U	C	3	4893
Acorn*	U	C/S	25	3582
Async	U	C/S	1	1039
Babel*	U	C/S	208	10390
Beslimed	U	C	2	4755
Co*	U	S	1	100
Coolclock	U	C	4	6933
Eslint	U	S	3	21383
Fly*	U	S	12	310
Gitbook	U	S	42	3519
Grunt	U	S	14	1526
Gulp	U	S	4	240
Handlebars*	U	C	19	2536
Htmledit	U	C	10	3607
Markitup	U	C	4	6499
Momentjs*	U	C	87	2690
Mustache	U	C	1	355
Watchtower*	U	S	9	1719

Type U – Utilitaire, A – Application, C – Cadre de travail, L – Librairie, J – Jeux, P – Plateforme

Env S – Serveur C – Client

Les systèmes avec « * » sont des systèmes utilisant ES2015.

ANNEXE VIII

DÉPENDANCE ENTRE FICHIERS

Tableau A VIII-1 Utilisation de « require » et « define » 1/3

	Constructeur « require »					**Constructeur « define »**		
Système	Litt.	Var.	Litt. Var.	Tableau	Fonction	Litt.	Var.	Litt. Et var.
Strider	339	2	21	0	0	0	0	0
Clientsapp	180	10	1	3	0	0	0	0
Bracket	2537	6	2	3	0	306	0	0
Ace	1975	4	3	2	0	632	0	0
Nodebb	1174	1	3	0	4	0	0	0
Devkit	277	3	2	0	0	0	0	0
Kiwiirc	142	5	0	0	0	0	0	0
Ghost	689	4	0	0	0	260	0	0
Countly	283	0	4	0	0	20	0	0
Mediacenterjs	197	2	2	0	0	0	0	0
Cylon	74	2	1	0	1	0	0	0
Mocha	100	3	1	0	0	0	0	0
Gitbook	171	2	0	0	1	41	0	0
Acorn	8	0	2	0	0	0	0	0
Grunt	41	0	1	0	1	0	0	0
Babel	77	2	0	0	0	0	0	0
Gulp	20	2	0	0	0	0	0	0
Iloveopensource	258	1	0	0	0	40	0	0
Momentjs	0	0	1	0	0	0	0	0
Express	77	0	0	0	1	0	0	0
Jquery	0	0	0	0	0	66	0	0
Raphael	1	0	0	0	0	2	0	0
Backbone	1	0	0	0	0	1	0	0
Mustache	0	0	0	0	0	1	0	0
Rsvp	18	0	0	0	0	1	0	0
Todoes6	3	0	0	0	0	1	0	0
Pdf	8	0	0	0	0	0	0	0
3dmodel	0	0	0	0	0	0	0	0
Angular	0	0	0	0	0	0	0	0
Async	0	0	0	0	0	0	0	0
Aurelia-core	0	0	0	0	0	0	0	0

Tableau A VIII-2 Utilisation de « require » et « define » 2/3

	Constructeur « require »					Constructeur « define »		
Système	Litt.	Var.	Litt. Var.	Tableau	Fonction	Litt.	Var.	Litt. Et var.
Beslimed	0	0	0	0	0	0	0	0
Browsenpm	57	0	0	0	0	0	0	0
Chartjs	0	0	0	0	0	0	0	0
Clumsybird	0	0	0	0	0	0	0	0
Co	0	0	0	0	0	0	0	0
Cobaltcalibur3	31	0	0	0	0	0	0	0
Coolclock	0	0	0	0	0	0	0	0
D3	0	0	0	0	0	0	0	0
Di	0	0	0	0	0	0	0	0
Ecommerce	0	0	0	0	0	0	0	0
Eslint	332	0	0	0	0	0	0	0
Flotr	0	0	0	0	0	0	0	0
Fly	0	0	0	0	0	0	0	0
Fullcalendar	0	0	0	0	0	0	0	0
Handlebars	4	0	0	0	0	0	0	0
Hapi	98	0	0	0	0	0	0	0
Hextris	0	0	0	0	0	0	0	0
Htmledit	0	0	0	0	0	0	0	0
Jasmine	0	0	0	0	0	0	0	0
Koa	9	0	0	0	0	0	0	0
Markitup	0	0	0	0	0	0	0	0
Mkjs	4	0	0	0	0	0	0	0
Modernizr	0	0	0	0	0	0	0	0
Mootools	0	0	0	0	0	0	0	0
Nodegame-shooter	15	0	0	0	0	0	0	0
Pacman	0	0	0	0	0	0	0	0
Paintweb	0	0	0	0	0	0	0	0
Pandajs	2	0	0	0	0	0	0	0
Passport	14	0	0	0	0	0	0	0
Pixi	0	0	0	0	0	0	0	0
Pong	0	0	0	0	0	0	0	0
Prototype	0	0	0	0	0	0	0	0
React	145	0	0	0	0	0	0	0
Reddit	0	0	0	0	0	0	0	0

Tableau A VIII-3 Utilisation de « require » et « define » 3/3

	Constructeur « require »					**Constructeur « define »**		
Système	Litt.	Var.	Litt. Var.	Tableau	Fonction	Litt.	Var.	Litt. Et var.
Router	0	0	0	0	0	0	0	0
Screenshot	21	0	0	0	0	0	0	0
Scriptaculous	0	0	0	0	0	0	0	0
Shields	54	0	0	0	0	0	0	0
Slate	32	0	0	0	0	0	0	0
Socketio	45	0	0	0	0	0	0	0
map	1	0	0	0	0	0	0	0
Spry	0	0	0	0	0	0	0	0
Tank	0	0	0	0	0	0	0	0
Three	0	0	0	0	0	0	0	0
Tictacnode	49	0	0	0	0	0	0	0
Underscore	0	0	0	0	0	0	0	0
Videojs	0	0	0	0	0	0	0	0
Watchtower	6	0	0	0	0	0	0	0
Winecellar	10	0	0	0	0	0	0	0

(Litt = Littéral, Var = Variable)

ANNEXE IX

SITE D'APPEL NON RÉSOLU

Tableau A IX-1 site d'appel non résolu

Système	Appels totaux	Appel n. résolu	A	B	C	D
3DMODEL	118	18	18 (100%)	0 (0%)	0 (0%)	0 (0%)
BESLIMED	1989	209	33 (15.8%)	108 (51.7%)	49 (23.4%)	19 (9.1%)
COOLCLOCK	1731	117	58 (49.6%)	15(12.8%)	36 (30.8%)	8 (6.8%)
FLOTR	2656	122	70 (57.4%)	27 (22.1%)	10 (8.2%)	15 (12.3%)
FULL-CALENDAR	4058	200	49 (24.5%)	79 (39.5%)	51 (25.5%)	21 (10.5%)
PACMAN	484	21	21 (100%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
PONG	1314	75	50 (66.7%)	12 (16.0%)	7 (9.3%)	6 (8.0%)
HTMLEDIT	1255	102	77 (75.5%)	23 (22.6%)	1 (1%)	1 (1%)
MARKITUP	1831	107	42 (39.3%)	25 (23.4%)	26 (24.3%)	14 (13.1%)
PDF	7070	247	187 (75.7%)	13 (5.3%)	17 (6.9%)	30 (12.2%)

Raisons

A – Fonction native non incluse dans notre base de données

B – Appel ou déclaration comporte une formulation trop dynamique

C – Indirections de nom de déclaration

D – La déclaration de fonction vient d'un paramètre (« callback »)

ANNEXE X

GUIDE D'UTILISATION DE L'OUTIL WAVI EN LIGNE DE COMMANDE

Téléchargement

Une version open source de l'outil est disponible sur github à l'adresse suivante :
https://github.com/bakunin95/wavi

Prérequis

Node.js
Graphviz

Utilisation

Taper la commande suivante dans votre console Node.js à l'intérieur du dossier où se trouve WAVI :

```
Node wavi –svg dossier/Cible dossier/Résultat.svg
```

ANNEXE XI

SYSTÈMES ÉTUDIÉES LORS DE LA VISUALISATION

Passport : Passport est une librairie (intergiciel) très populaire avec 500 000 téléchargements par mois sur npm.js. Elle gère l'authentification d'usager et propose plus de 300 stratégies d'authentification. Par exemple, avec une base de données, un compte local ou bien à l'aide d'un compte de réseau social tel que google, Yahoo, Facebook ou twitter. Elle permet de garder une implémentation cohérente, flexible et sécuritaire pour n'importe quel système nécessitant une authentification.

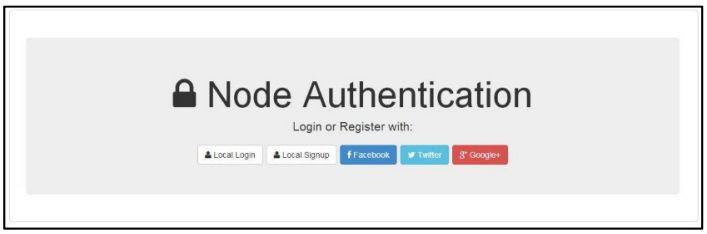

Figure-A XI-1 Système Passport. Image
Tirée d'un tutoriel sur passport[16]

DI : DI est un cadre de travail qui gère les injections de dépendances. L'injection est une méthode de gestion des dépendances. Elle aide à réduire la responsabilité des classes (responsabilité de gérer la complexité des dépendances) afin de rendre ceux-ci plus faciles à tester et réutiliser. DI contient 7 fichiers, 108 déclarations de fonctions et utilise la nouvelle version de JavaScript (ES2015) incluant les classes (12) ainsi que les importations.

[16] https://scotch.io/tutorials/easy-node-authentication-setup-and-local

```
1  import {Inject} from 'di';
2
3  import {Heater} from './heater';
4  import {Pump} from './pump';
5
6  @Inject(Heater, Pump)
7  export class CoffeeMaker {
8    constructor(heater, pump) {
9      this.heater = heater;
10     this.pump = pump;
11   }
12
13   brew() {
14     this.pump.pump();
15     this.heater.on();
16     // console.log('Brewing...')
17   }
18 }
```

Figure-A XI -2 Système DI

BrowseNpm : L'application BrowseNpm permet d'explorer les nombreux paquetages publics de Node.js. Les informations qu'elle rend disponibles sont les directives d'installation du paquetage, les dépendances, le nom des mainteneurs, les versions disponibles, le nombre de téléchargements par jour et plus. BrowseNpm est le système choisi qui contient le plus de fichiers (19) provenant du client et du serveur ainsi que 122 déclarations de fonction.

Figure-A XI -3 Système BrowseNpm. Capture d'écran de BrowseNpm (disponible à browsenpm.org).

Screenshot : Screenshot est une application de type ligne de commande qui permet de prendre des captures d'écran de page web. Elle utilise le « webkit » PhantomJS, le standard JavaScript ES5 et contient 11 fichiers ainsi que 28 déclarations de fonctions.

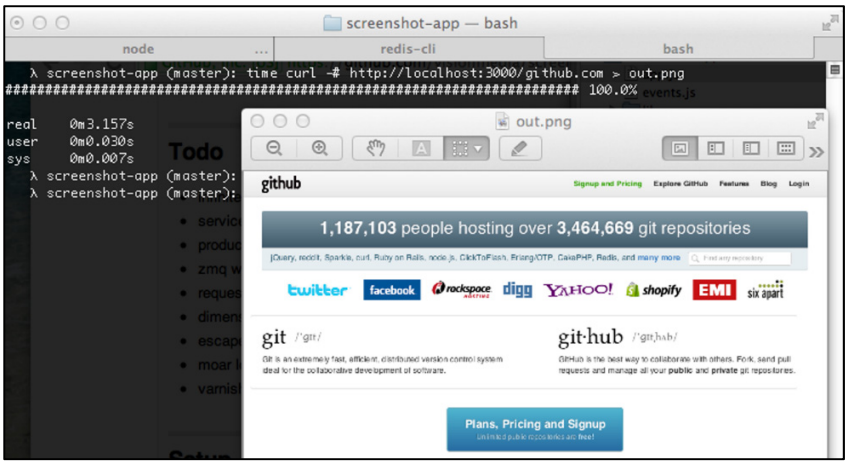

Figure-A XI -4 Système Screenshot. Cet exemple d'utilisation de Screenshot
Tirée de github (https://github.com/tj/screenshot-app)

Shields : Shields est une application qui permet de générer des insignes (badges) pour afficher dans les pages web. Alors que c'est une pratique courante d'utiliser des badges (p. ex.: pour communiquer l'état de la dernière version compilé d'une application ou bien sa version), la plupart sont de mauvaise qualité, Shields tente de générer des badges qui sont faciles à lire et consistants. Elle contient 15 fichiers et 887 déclarations de fonctions.

Figure-A X-5 Système Shields
exemple d'un badge généré sur la page (http://shields.io/)

Babel : Babel est un compilateur JavaScript qui permet de transformer du code ES2015 en ES5. Cela est entre autres utile pour les développeurs qui désirent utiliser dès maintenant ES2015 sans attendre le support de Node.js et des navigateurs web.

KiwiIRC : KiwiIRC est un client web IRC qui permet aux utilisateurs de clavarder directement dans leur navigateur sans passer par un logiciel. C'est une application web qui compte 91 fichiers JavaScript et qui utilise ES5.

Figure-A XI -6 Système KiwiIRC

ANNEXE XII

LES CRITÈRES D'IMPORTANCES

Tableau A XII-1 Importance de base

Groupe	**Importance de base**
js, html, css, js-class,j-Markup, css-markup, js-require, form	100
link, module.exports	90
js-function, js-method, ajax	80
js-object, js-array, js-variable, html-element, form-element	50
Js-method-call	10

Tableau A XII -2 Bonis d'importance selon des conditions*

Condition	**Bonis**
La fonction s'étend sur plus de 10 lignes de code	10
La fonction s'étend sur plus de 20 lignes de code	20
La hiérarchie de l'élément	-2.5 (par niveau)
Si la fonction est privée	-5
Si l'élément ne possède pas de relations	-10
Si la fonction est anonyme	-20

*Ne s'applique que si l'élément n'a pas 100 d'importance

Exemple de calcul pour une fonction anonyme privée s'étalant sur 15 lignes, sur le 3^{e} niveau de hiérarchie ne possédant aucune relation :

```
80  + 10 – (3*2.5) – 5 – 10 - 20 = 47,5 d'importance.
```

ANNEXE XIII

LES PROFILS

Tableau A XIII-1 Les profils

# Profil	#Classes	# Proriétés		Py
1	Maximum		0	100
2	Medium	Medium	50	25
3	Minimum	Maximum	100	0
4		Medium	100	30
5		Minimum	100	50
6		*	100	100

Voici deux exemples de calcul de profils. Le couple (Cx, Py) qui équivaut (50,25) veut dire que tous les éléments ayant une importance plus grande que 50 sont présentés en tant que classes. Tous les éléments en relation avec ces classes ayant une importance plus grande que 25 sont montrés en tant que propriétés. Pour le couple (100,30), seuls les fichiers et classes JavaScript sont présentés comme des classes (élément ayant 100 d'importance) et les éléments ayant une importance plus grande que 30 sont présentés comme des propriétés.

ANNEXE XIV

GRAPHIQUE D'APPEL DE BROWSENPM

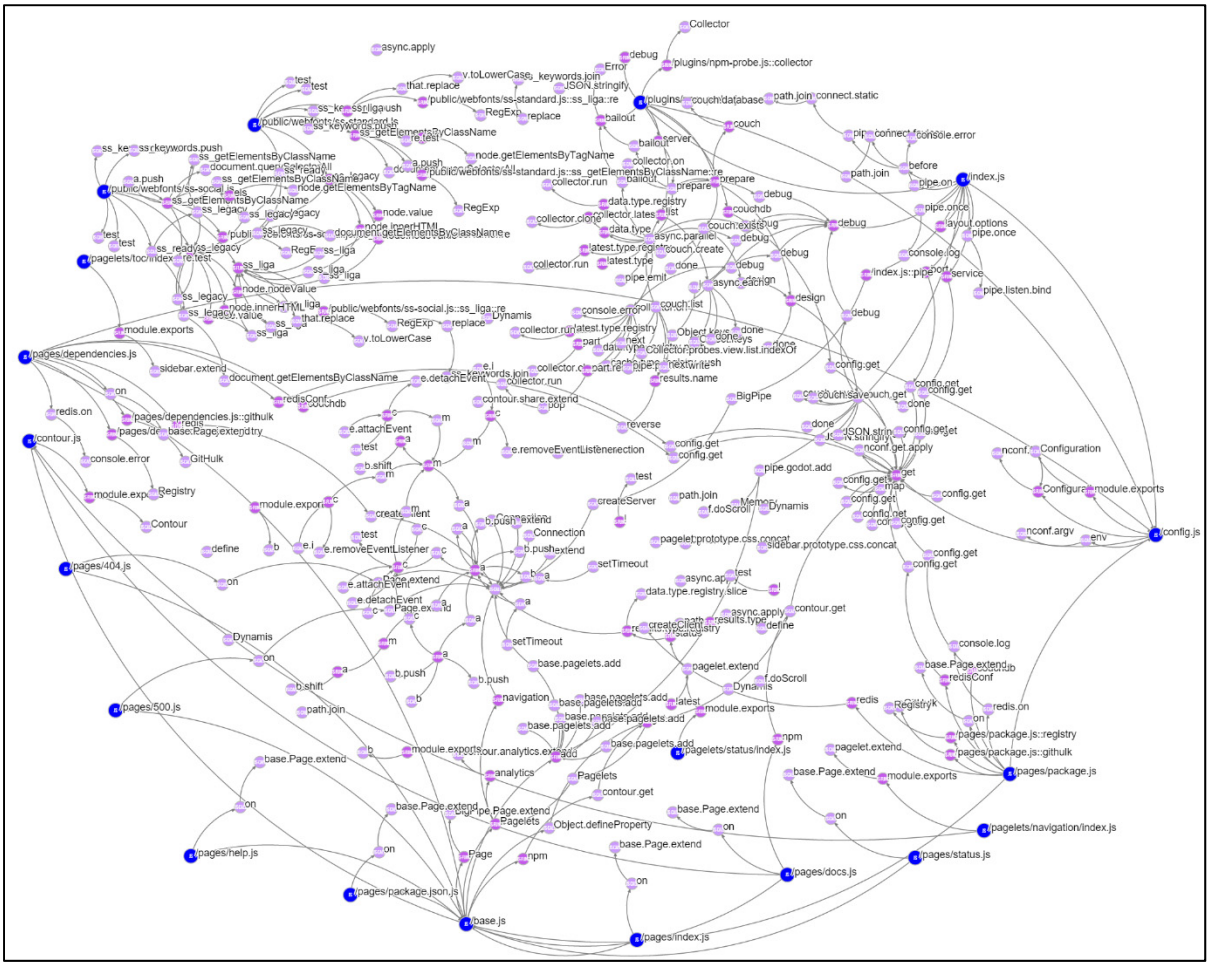

Figure-A XIV-6 Graphique d'appel de BrowseNPM

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Amalfitano, D., A. R. Fasolino, A. Polcaro, and P. Tramontana. 2010. "DynaRIA: A Tool for Ajax Web Application Comprehension." In *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*, 46-47.

Amalfitano, Domenico. 2011. 'Reverse engineering and testing of rich internet applications'. PhD thesis, Universita degli Studi di Napoli Federico II

Amalfitano, Domenico, Anna Rita Fasolino, Armando Polcaro, and Porfirio Tramontana. 2014. 'The DynaRIA tool for the comprehension of Ajax web applications by dynamic analysis', *Innovations in Systems and Software Engineering*, 10: 41-57.

Antoniol, Giuliano, Gerardo Canfora, G Casazza, and A De Lucia. 2000. "Web site reengineering using RMM." In Proc. of the International Workshop on Web Site Evolution, 9-16.

Antoniol, G., M. Di Penta, and M. Zazzara. 2004. "Understanding Web applications through dynamic analysis." In *Program Comprehension, 2004. Proceedings. 12th IEEE International Workshop on*, 120-29.

Bellettini, Carlo, Alessandro Marchetto, and Andrea Trentini. 2004. "WebUml: reverse engineering of web applications." In *Proceedings of the 2004 ACM symposium on Applied computing*, 1662-69. ACM.

Canfora, G., and M. Di Penta. 2007. "New Frontiers of Reverse Engineering." In *Future of Software Engineering, 2007. FOSE '07*, 326-41.

Conallen, Jim. 1999. 'Modeling Web application architectures with UML', *Commun. ACM*, 42: 63-70.

Conallen, Jim. 2003. Building Web applications with UML (Addison-Wesley Longman Publishing Co., Inc.).

D3. 2014. 'Data-Driven Documents', Accessed 27-04-2014. http://d3js.org/.

Di Lucca, G. A., and M. Di Penta. 2005. "Integrating static and dynamic analysis to improve the comprehension of existing Web applications." In *Web Site Evolution, 2005. (WSE 2005). Seventh IEEE International Symposium on*, 87-94.

Di Lucca, G. A., M. Di Penta, G. Antoniol, and G. Casazza. 2001. "An approach for reverse engineering of web-based applications." In *Reverse Engineering, 2001. Proceedings. Eighth Working Conference on*, 231-40.

Di Lucca, G. A., A. R. Fasolino, F. Pace, P. Tramontana, and U. De Carlini. 2002. "WARE: a tool for the reverse engineering of Web applications." In *Software Maintenance and Reengineering, 2002. Proceedings. Sixth European Conference on*, 241-50.

Di Lucca, Giuseppe Antonio, Anna Rita Fasolino, and Porfirio Tramontana. 2004. 'Reverse engineering Web applications: the WARE approach', *Journal of Software maintenance and evolution: Research and practice*, 16: 71-101.

Estievenart, F., A. Francois, J. Henrard, and J. L. Hainaut. 2003. "A tool-supported method to extract data and schema from Web sites." In *Web Site Evolution, 2003. Theme: Architecture. Proceedings. Fifth IEEE International Workshop on*, 3-11.

Feldthaus, Asger, Markus Schafer, Manu Sridharan, Julian Dolby, and Frank Tip. 2013. "Efficient construction of approximate call graphs for JavaScript IDE services." In *Software Engineering (ICSE), 2013 35th International Conference on*, 752-61. IEEE.

Fielding, Roy, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. 1999. "Hypertext transfer protocol–HTTP/1.1." In.: RFC 2616, June.

Fitzgerald, G., S. Counsell, and J. Peters. 2013. "A Study of Web Maintenance in an Industrial Setting." In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, 391-94.

Gama, W., M. H. Alalfi, J. R. Cordy, and T. R. Dean. 2012. "Normalizing object-oriented class styles in JavaScript." In *Web Systems Evolution (WSE), 2012 14th IEEE International Symposium on*, 79-83.

Ginige, A., and S. Murugesan. 2001. 'The essence of web engineering - managing the diversity and complexity of web application development', *MultiMedia, IEEE*, 8: 22-25.

Ginige, A., and San Murugesan. 2001. 'Web engineering: an introduction', *MultiMedia, IEEE*, 8: 14-18.

Hamou-Lhadj, A., A. En-Nouaary, and K. Sultan. 2007. "Reverse Engineering of Web Based Systems." In *Innovations in Information Technology, 2007. IIT '07. 4th International Conference on*, 193-97.

Hassan, A. E., and R. C. Holt. 2002. "Architecture recovery of Web applications." In *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, 349-59.

Humberto Silva, Leonardo, Miguel Ramos, Marco Tulio Valente, Alexandre Bergel, and Nicolas Anquetil. 2015. "Does JavaScript software embrace classes?" In *SANER 2015*

: International Conference on Software Analysis, Evolution, and Reengineering, 73 - 82. Montreal, Canada.

Jang, Dongseok, and Kwang-Moo Choe. 2009. "Points-to analysis for JavaScript." In *Proceedings of the 2009 ACM symposium on Applied Computing*, 1930-37. Honolulu, Hawaii: ACM.

Jeary, Sheridan, Keith Phalp, and Jonathan Vincent. 2009. 'An evaluation of the utility of web development methods', *Software Quality Journal*, 17: 125-50.

Jensen, Simon Holm, Anders M, #248, ller, and Peter Thiemann. 2009. "Type Analysis for JavaScript." In *Proceedings of the 16th International Symposium on Static Analysis*, 238-55. Los Angeles, CA: Springer-Verlag.

Jensen, Simon Holm, Anders M, #248, ller, and Peter Thiemann. 2010. "Interprocedural analysis with lazy propagation." In *Proceedings of the 17th international conference on Static analysis*, 320-39. Perpignan, France: Springer-Verlag.

Jensen, Simon Holm, Magnus Madsen, Anders M, #248, and ller. 2011. "Modeling the HTML DOM and browser API in static analysis of JavaScript web applications." In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 59-69. Szeged, Hungary: ACM.

Junhua, Wu, and Xu Baowen. 2009. "A Method to Support Web Evolution by Modeling Static Structure and Dynamic Behavior." In *Computer Engineering and Technology, 2009. ICCET '09. International Conference on*, 458-62.

Kashyap, Vineeth, John Sarracino, John Wagner, Ben Wiedermann, and Ben Hardekopf. 2013. "Type refinement for static analysis of JavaScript." In *Proceedings of the 9th symposium on Dynamic languages*, 17-26. Indianapolis, Indiana, USA: ACM.

Kienle, Holger M., and Damiano Distante. 2014. 'Evolution of Web Systems.' in Tom Mens, Alexander Serebrenik and Anthony Cleve (eds.), Evolving Software Systems (Springer Berlin Heidelberg: Berlin, Heidelberg).

Liauw, DA. 2012. 'ArchWiki: Using Web 2.0 for Architecture Knowledge Management', TU Delft, Delft University of Technology.

Madsen, Magnus. 2015. 'Static Analysis of Dynamic Languages', Aarhus Universitet Aarhus University, Science and Technology Science and Technology, Institut for Datalogi Department of Computer Science.

Matthijssen, Nick, and Andy Zaidman. 2011. "FireDetective: understanding ajax client/server interactions." In *Proceedings of the 33rd International Conference on Software Engineering*, 998-1000. Waikiki, Honolulu, HI, USA: ACM.

Matthijssen, Nick, Andy Zaidman, Margaret-Anne Storey, Ian Bull, and Arie van Deursen.

2010. "Connecting Traces: Understanding Client-Server Interactions in Ajax Applications." In *Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension*, 216-25. IEEE Computer Society.

Mendes, Emilia. 2014. 'Web Development Versus Software Development.' in, Practitioner's Knowledge Representation: A Pathway to Improve Software Effort Estimation (Springer Berlin Heidelberg: Berlin, Heidelberg).

Murray, Scott. 2013. *Interactive data visualization for the Web* (" O'Reilly Media, Inc.").

Park, Changhee, Hongki Lee, and Sukyoung Ryu. 2013. "All about the with statement in JavaScript: removing with statements in JavaScript applications." In *Proceedings of the 9th symposium on Dynamic languages*, 73-84. Indianapolis, Indiana, USA: ACM.

Ricca, F., M. Di Penta, Marco Torchiano, P. Tonella, and M. Ceccato. 2010. 'How Developers' Experience and Ability Influence Web Application Comprehension Tasks Supported by UML Stereotypes: A Series of Four Experiments', *Software Engineering, IEEE Transactions on*, 36: 96-118.

Ricca, Filippo, Massimiliano Di Penta, Marco Torchiano, Paolo Tonella, and Mariano Ceccato. 2006. "An empirical study on the usefulness of Conallen's stereotypes inWeb application comprehension." In *Proceedings of the Eighth IEEE International Symposium on Web Site Evolution*, 58-68. IEEE Computer Society.

Ricca, F., and P. Tonella. 2000. "Web site analysis: structure and evolution." In *Software Maintenance, 2000. Proceedings. International Conference on*, 76-86.

Ricca, F., and P. Tonella. 2001. 'Understanding and restructuring Web sites with ReWeb', *MultiMedia, IEEE*, 8: 40-51.

Richards, Gregor, Christian Hammer, Brian Burg, and Jan Vitek. 2011. "The eval that men do: A large-scale study of the use of eval in javascript applications." In *Proceedings of the 25th European conference on Object-oriented programming*, 52-78. Lancaster, UK: Springer-Verlag.

Richards, Gregor, Sylvain Lebresne, Brian Burg, and Jan Vitek. 2010. "An analysis of the dynamic behavior of JavaScript programs." In *Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 1-12. Toronto, Ontario, Canada: ACM.

Schwinger, Wieland, and Nora Koch. 2006. '3 Modeling Web Applications'.

Schwinger, Wieland, Werner Retschitzegger, Andrea Schauerhuber, Gerti Kappel, Manuel Wimmer, Birgit Pröll, Cristina Cachero Castro, Sven Casteleyn, Olga De Troyer, Piero Fraternali, Irene Garrigos, Franca Garzotto, Athula Ginige, Geert-Jan Houben,

Nora Koch, Nathalie Moreno, Oscar Pastor, Paolo Paolini, Vicente Pelechano Ferragud, Gustavo Rossi, Daniel Schwabe, Massimo Tisi, Antonio Vallecillo, Kees van der Sluijs, and Gefei Zhang. 2008. 'A survey on web modeling approaches for ubiquitous web applications', *International Journal of Web Information Systems*, 4: 234-305.

Sridharan, Manu, Julian Dolby, Satish Chandra, Max Sch, #228, fer, and Frank Tip. 2012. "Correlation tracking for points-to analysis of javascript." In *Proceedings of the 26th European conference on Object-Oriented Programming*, 435-58. Beijing, China: Springer-Verlag.

Toma, Tajkia Rahman, and Md Shariful Islam. 2014. "An efficient mechanism of generating call graph for JavaScript using dynamic analysis in web application." In *Informatics, Electronics & Vision (ICIEV), 2014 International Conference on*, 1-6. IEEE.

Tonella, F. Ricca and P. 2000. 'Visualization of Web Site History', *Proc. of WSE'2000, International Workshop on Web Site Evolution*: 30-33.

Tramontana, P. 2005. "Reverse engineering Web applications." In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, 705-08.

Tramontana, P., D. Amalfitano, and A. R. Fasolino. 2013. "Reverse engineering techniques: From web applications to rich Internet applications." In *Web Systems Evolution (WSE), 2013 15th IEEE International Symposium on*, 83-86.

Wei, Shiyi, and BarbaraG Ryder. 2014. 'State-Sensitive Points-to Analysis for the Dynamic Behavior of JavaScript Objects.' in Richard Jones (ed.), *ECOOP 2014 – Object-Oriented Programming* (Springer Berlin Heidelberg).

Wei, Shiyi, and Barbara G. Ryder. 2013. "Practical blended taint analysis for JavaScript." In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, 336-46. Lugano, Switzerland: ACM.

Welland, Andrew Mcdonald and Ray. 2001. 'Web Engineering in Practice'.

Zaidman, Andy, Nick Matthijssen, Margaret-Anne Storey, and Arie van Deursen. 2013. 'Understanding Ajax applications by connecting client and server-side execution traces', *Empirical Software Engineering*, 18: 181-218.