

# Outils systèmes pour le stockage des données du génie logiciel

Yoann Padioleau

Olivier Ridoux

Benjamin Sigonneau

IFSIC/IRISA

Campus Universitaire de Beaulieu

35 042 Rennes Cedex, France

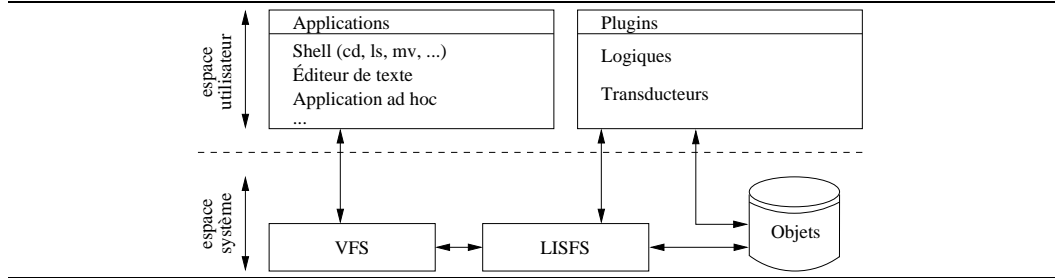
## Résumé

Les données du génie logiciel sont variées et entretiennent des relations complexes. De plus, quel que soit le niveau d'action choisi, les acteurs d'un projet et les rôles d'un acteur sont multiples, tant dans l'espace que dans le temps, menant à autant de points de vue sur le projet.

Cette observation est classique et a conduit à la réalisation d'outils dédiés, comme PCTE. Cependant, on assiste encore à la création et au développement de tels outils et environnements. On peut ainsi citer Eclipse. En effet, l'espace des objets et des relations n'est pas défini ; les programmeurs en inventent toujours de nouveaux. Il faut donc plutôt se tourner vers une solution générique.

Pour cela, nous utilisons le cadre des *systèmes d'information logiques*, et plus particulièrement sa mise en œuvre sous forme d'un *système de fichiers logique*. Nous présentons ensuite des *exemples d'application* et nous dressons quelques conclusions et pistes de recherche.

**FIG. 1** Architecture de LISFS



## 1 Système d'information logique

Le cadre des systèmes d'information logiques, ou SIL, offre les moyens de traiter et gérer des données de façon générique et flexible. Ce cadre constitue une alternative aux systèmes hiérarchiques (p. ex. systèmes de fichiers à la UNIX), aux systèmes booléens (à la Google) et aux systèmes relationnels (p. ex. bases de données). Il se base sur une variante de l'analyse de concepts formels [5, 4].

La structure de stockage des informations est un *contexte formel* qui associe à des objets des propriétés logiques. Un SIL peut indifféremment être interrogé ou vu comme un support de navigation. L'interrogation répond à une requête, exprimée sous la forme d'une propriété, en calculant l'ensemble des objets dont la propriété entraîne logiquement la requête. La navigation introduit la notion d'*endroit*, lui aussi exprimé par une propriété, et calcule les endroits accessibles depuis celui-ci et les objets contenus dans cet endroit. Cela revient à calculer la différence entre une propriété (l'endroit courant) et les propriétés de tous les objets satisfaisant cette requête. Si ces différences sont calculées avec soin, elles forment les bases d'un outils d'exploration progressive.

Dans un SIL, les objets ont deux types de propriétés. Les propriétés *intrinsèques* sont calculées d'après le contenu des objets tandis que les propriétés *extrinsèques* sont données par l'utilisateur selon des critères qui ne sont pas dans l'objet ou ne peuvent être calculées. Par exemple, le nom des variables globales qu'elle utilise est une propriété intrinsèque d'une procédure, mais le fait qu'elle soit correcte est une propriété extrinsèque.

## 2 Système de fichiers logique

LISFS (Logical Information System File System) met en œuvre les principes d'un SIL au niveau système, sous l'interface d'un système de fichiers. Dans ce cadre, les objets sont des fichiers ou des parties de fichiers, et les endroits sont présentés comme des répertoires. L'approche système de cette mise en œuvre permet de banaliser l'accès logique aux fichiers, quel que soit leur type et les applications utilisées [3, 8].

Pour les systèmes d'exploitation modernes, un système de fichiers est un ensemble de méthodes permettant de gérer les fichiers et les répertoires. Ainsi, sous Linux, VFS (Virtual File System) agit comme une interface dont chaque méthode est virtuelle. Un système de fichiers est simplement une implémentation de ces méthodes ; c'est le cas de LISFS,

qui fonctionne sous Linux 2.4. Comme le montre la figure 1, LISFS possède un système de *plug-ins*. Il existe deux types de plug-ins : les *logiques*, qui déterminent quelles sont les logiques utilisées pour interroger/classer l'ensemble d'objets, et les *transducteurs* qui sont des programmes calculant les propriétés intrinsèques des objets. Soulignons enfin que LISFS agit à deux niveaux. Au premier niveau, dit *inter-fichiers*, les objets du SIL sont les fichiers. Au second niveau, dit *intra-fichiers*, les objets sont des parties de fichiers.

### 3 Exemples d'application

**Recherche de module** Les méthodes de packages Java ont été indexées par leur types, leur visibilité, les exceptions qu'elles lèvent ainsi que des mots-clés extraits de leur noms (p. ex. *set* et *value* pour *setValue*) et de leurs commentaires. Ces propriétés, associées à un ordre sur les types et à des isomorphismes de types [1], permettent de définir différents critères (formel, semi-formel, informel) pour la recherche de méthodes.

Nous avons appliqué ce procédé au package `java.awt`, qui est un *toolkit* graphique de 5 500 méthodes. La recherche de méthodes dans cet ensemble est efficace. La possibilité d'utiliser plusieurs critères dans un ordre quelconque la rend plus simple qu'en utilisant des outils conventionnels comme la documentation Javadoc de référence. Nous avons fait des expériences similaires avec les pages *man* d'une installation UNIX standard.

**Exploration de traces** Des fichiers de traces d'exécution de programme Prolog (et C) ont été constitués dans le but de les explorer. Chaque événement de la trace est décrit par une ligne de texte contenant les valeurs d'un ensemble d'attributs (numéro d'événement, port, prédicat et but appelés, profondeur d'appel). Il est alors possible, en utilisant la navigation intra-fichier, d'explorer la trace en en demandant des vues partielles. Par exemple, `cd depth:>5` fournit une vue partielle de la trace relative aux buts dont la profondeur d'appel est supérieure à cinq.

On pourrait aussi naviguer dans plusieurs traces d'exécution d'un même programme pour différents jeux de tests, ou de plusieurs variantes d'un programme pour un même jeu de tests. LISFS serait alors utilisé pour faire du recoupement de traces dans le but de localiser des erreurs [6].

**Navigation multi-vues dans des sources/documentations** La possibilité offerte par LISFS d'avoir une navigation intra-fichier permet de visualiser un code source sous différents points de vue. On peut demander à ne voir qu'une seule fonction, à enlever les commentaires ou les aspects de débogage, etc. Le premier prototype de LISFS, qui consistait en un seul fichier source de 2 800 lignes de Perl, a été ainsi placé sous ce système. Par exemple, la commande `cd !assert` permet de cacher les assertions de débogage. Il est alors possible de travailler sur cette vue réduite. Lors d'une sauvegarde, les modifications sont propagées dans le fichier initial ; il n'y a pas de problème de mise à jour de vue pour LISFS. Nous avons aussi appliqué cette méthode au code source du noyau Linux.

Bien que prenant le problème dans l'autre sens, LISFS permet ainsi de résoudre des problèmes qui ont conduit à définir la programmation par aspects [7]. L'écriture de com-

mentaires normalisés permet de définir aisément des propriétés dans le but de travailler sur ces vues partielles. Dans le futur, l'utilisation de propriétés issues de l'analyse statique (graphe d'appel, flot de contrôle, analyse de type, ...) devrait fournir une aide plus forte encore au développeur.

## 4 Conclusion

Nous avons présenté les SIL qui permettent d'organiser une collection d'objets suivant un schéma non fixé *a priori* et non hiérarchique. Il est alors possible de gérer cette collection en combinant interrogation et navigation dans un cadre à forte expressivité. La mise en œuvre sous forme d'un système de fichiers permet à toutes les applications d'en bénéficier. Nos expériences utilisant la navigation inter- et intra-objets montrent que LISFS peut être utilisé pour fournir une aide au développement logiciel.

Le modèle des SIL offre des opérations de *data-mining* (calcul de règles d'associations, ...) et d'apprentissage automatique de propriétés. Ces opérations ne sont pas encore intégrées à LISFS. Lorsque cela sera fait, de nouvelles applications pour l'aide au développement logiciel devraient apparaître.

L'analyse statique des programmes a pour objectif d'extraire des propriétés des programmes, sans les exécuter. Il serait donc intéressant d'intégrer de telles analyses dans les transducteurs de LISFS.

## Références

- [1] R. Di Cosmo. Deciding type isomorphisms in a type-assignment framework. *Journal of Logic Programming*, 3(4) :485–525, 1993.
- [2] Eclipse Project. <http://www.eclipse.org>
- [3] S. Ferré and O. Ridoux. A file system based on concept analysis. In *Int. Conf. on Rules and Objects in Databases*, LNCS 1861. Springer, 2000.
- [4] S. Ferré and O. Ridoux. A logical generalization of formal concept analysis. In *Int. Conf. on Conceptual Structures*, LNCS 1867. Springer, 2000.
- [5] B. Ganter and R. Wille. *Formal concept analysis : mathematic foundations*. Springer, 1999.
- [6] J. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *24th Int. Conf. on Software Engineering*, 2002.
- [7] G. Kiczales *et al.* Aspect-oriented programming. In *ECOOP '97*, LNCS 1241, pages 220–242. Springer, 1997.
- [8] Y. Padioleau and O. Ridoux. A logic file system. In *USENIX Annual Technical Conference*, 2003.
- [9] I. Thomas. PCTE interfaces : Supporting tools in software-engineering environments. *IEEE Software*, 6(6) :15–23, November 1989.