Cours : Abstraction et raffinement

Article : POSIX file store in Z/Eves : an experiment in the verified software repository

LEO FREITAS, Zheng Fu, and Jim Woocock

Department of Computer Science

University of York YO10 5DD, UK

Présenté par :

BARRY Thierno Boubacar

M2 PLS

Chargée du cours :

Mme Christine CHOPPY

Sommaire

1. Contexte

Les logiciels industriels disposent généralement d'une documentation complète, mais le comportement des logiciels est souvent une surprise totale! La programmation est particulièrement difficile parce que nous devons réduire un problème à un ensemble de règles qui peuvent être suivies aveuglément par un ordinateur. Mais les composants interagissent et interfèrent, puis des propriétés indésirables apparaissent. Il en résulte que les systèmes ne parviennent pas à satisfaire les besoins de leurs utilisateurs.

Les logiciels sont intrinsèquement difficiles à développer car il est difficile de définir les besoins, d'anticiper les interactions et de s'adapter aux nouvelles fonctionnalités. La documentation comprend de grandes quantités de texte, d'images et de diagrammes, mais ceux-ci sont souvent imprécis et ambigus. Les informations importantes sont souvent cachées par des détails non pertinents. Les erreurs de conception sont souvent découvertes trop tard, ce qui les rend coûteuses, voire impossibles à corriger. C'est grâce à la compétence des ingénieurs en logiciel que les systèmes fonctionnent.

2. Résumé

Nous présentons dans ce rapport les résultats du deuxième projet pilote du Grand défi international de vérification : une spécification formellement vérifiée d'un entrepôt de fichiers conforme à la norme POSIX utilisant le prouveur du théorème Z/Eves. L'objectif global du projet est de construire un dépôt de fichiers vérifiés pour les missions de vols spatiaux. Notre spécification du magasin de fichiers est basée sur la spécification du système de classement UNIX de Morgan & Sufrin ; la preuve et sa mécanisation en Z/Eves sont nouvelles. Nous montrons comment notre travail contribue à la construction d'un référentiel logiciel vérifié : un ensemble de théories et d'expériences générales réutilisables dans différents domaines.

Objectifs du Référentiel

1. Accélérer le développement de la technologie de vérification par la mise au point de meilleurs outils, d'une interopérabilité accrue et de repères réalistes.
2. Fournir un point de mire à la communauté de la vérification pour s'assurer que les résultats de la recherche sont pertinents, reproductibles, complémentaires et cumulatifs, et promouvoir une collaboration significative entre les techniques complémentaires.
3. Donner libre accès aux résultats les plus récents et au matériel pédagogique dans les domaines pertinents pour la recherche en vérification.
4. Recueillir un corpus important de code vérifié (spécification, dérivation, preuves, implémentation) qui traite des applications difficiles.
5. Identifier les mesures clés pour évaluer l'échelle, l'efficacité, la profondeur, l'amortissement et la réutilisabilité de la technologie.
6. Énumérez les problèmes de contestation et les domaines à vérifier, de préférence ceux qui nécessitent des techniques multiples.

7. Identifier (et éventuellement normaliser) les formats de représentation et d'échange des spécifications, des programmes, des cas de test, des preuves et des benchmarks, pour favoriser l'interopérabilité et la comparaison des outils.
8. Définir des normes de qualité pour le contenu du référentiel.

3. Introduction

Les logiciels d'aujourd'hui sont accompagnés d'une documentation complète : guides d'utilisation, manuels de référence et documents de conception. Il existe des systèmes d'aide en ligne, des didacticiels interactifs et des présentations conviviales pour les mannequins. Pourtant, le comportement des logiciels est souvent une surprise tant pour les utilisateurs que pour les concepteurs. Les conséquences les plus spectaculaires font la une des journaux : crash d'avion, collision de trains, doses mortelles de radiations, retrait des services téléphoniques d'urgence, etc. Le moins spectaculaire auquel nous sommes confrontés chaque jour : le temps est perdu, le temps est gaspillé, les projets importants sont abandonnés et notre santé est endommagée par la frustration pure et simple. Tout cela, et plus encore, parce que les pannes logicielles sont à la hauteur de nos attentes, ce qui s'explique de nombreuses façons : les exigences imposées à un logiciel sont difficiles à définir, les façons dont un système peut être utilisé sont difficiles à prévoir et il y a toujours une demande de fonctionnalités supplémentaires. En effet, le fait que de nombreux logiciels fonctionnent réellement, et fonctionnent bien, est une indication de la compétence de ceux dont le travail consiste à les développer.

La taille et la complexité croissantes des systèmes augmentent la pression pour des logiciels fiables. Dans les domaines où l'intégrité et la sécurité sont essentielles, la fiabilité est une exigence imposée par les organismes de réglementation, mais il est généralement difficile de vérifier l'exactitude et d'assurer la fiabilité au moyen de tests logiciels ou de processus de développement imprécis. Les méthodes formelles sont des techniques de développement de logiciels qui permettent une caractérisation précise du domaine basée sur les mathématiques ; en appliquant des analyses mathématiques standard. Ces méthodes peuvent être utilisées pour prouver l'exactitude des systèmes.

L'idée d'appliquer la modélisation mathématique et l'analyse aux processus de développement de logiciels n'est pas nouvelle. Bien qu'il nécessite des compétences et des processus de développement différents par rapport aux cycles de développement traditionnels, il a été utilisé avec succès dans des applications industrielles, telles que la banque électronique, et il est largement accepté dans les milieux universitaires avec plusieurs décennies d'expériences. Avec des outils appuyés par une théorie mature, les méthodes formelles deviennent plus efficaces et leur utilisation est plus facile à justifier, non pas comme une exigence académique ou légale, mais comme une analyse de rentabilité. C'est-à-dire que, malgré l'effort supplémentaire initial, les gains que les méthodes formelles permettent en termes de fiabilité, de responsabilité et de précision permettent d'économiser de l'argent. En d'autres termes,

malgré l'effort supplémentaire initial, les méthodes formelles permettent d'économiser de l'argent en termes de fiabilité, de responsabilité et de précision.

Ces avancées récentes en matière de théorie et de support d'outils ont inspiré les chercheurs industriels et académiques à se joindre à un grand défi international dans le domaine des logiciels vérifiés, et ont créé un sentiment d'urgence pour faire de ses objectifs une réalité. Les travaux ont commencé par la création d'un dépôt de logiciels vérifiés (VSR) avec deux objectifs principaux : (i) composants logiciels vérifiés ; et (ii) expériences de vérification à l'échelle industrielle ayant une importance théorique considérable et un impact sur les outils.

Travaux connexes

La première expérience pilote VSR a eu lieu en 2006 : la mécanisation réussie d'une version aseptisée de la première application bancaire à carte à puce Mondex ITSEC de niveau 6 de haute intégrité. Dans cette expérience, sept groupes ont utilisé différentes théories et outils pour mécaniser le raffinement de Mondex et sa vérification.

Un autre travail dont nous pourrions bénéficier est l'implémentation en occam d'un petit système d'exploitation "POSIX-aware".

Dans la section suivante, nous décrivons brièvement ce qui constitue un projet pilote et les raisons de notre choix de l'interface POSIX pour les magasins de fichiers. Dans la section 3, nous présentons la portée du travail, ainsi que les décisions importantes en matière de conception et un résumé révisé des spécifications. Ensuite, la section 4 présente une stratégie de raffinement, ainsi que la mise en œuvre concrète pour laquelle le raffinement est prouvé en utilisant Z/Eves. Nous résumons brièvement les conditions préalables calculées pour les opérations abstraites et concrètes dans la section 5. Ensuite, la section 6 présente les questions de mécanisation et les statistiques, qui servent à la fois de théories de base pour d'autres expériences et de repères pour différents récits et outils. Enfin, la section 7 résume nos réalisations et traite des travaux futurs.

4. Qu’est ce qu’un projet pilote?

Dans [14], Joshi & Holzmann proposent un projet pilote pour le grand défi. Ils caractérisent, motivent et justifient un choix intéressant pour la vérification formelle comme un mini-défi. De ce point de vue, le grand défi est divisé en projets plus petits et d'une importance considérable selon les caractéristiques suivantes :

"a) il serait suffisamment complexe que les méthodes traditionnelles, telles que les essais et les revues de codeurs, ne suffisent pas à en établir l'exactitude ;

b) il serait suffisamment simple que les spécifications, la conception et la vérification puissent être effectuées par une équipe spécialisée dans un délai relativement court, par exemple 2 à 3 ans ; et

c) il serait suffisamment important que les résultats obtenus aient un impact au-delà du milieu de la vérification, des universités et des entreprises.

Lors de l'atelier Menlo Park à SRI[24], l'interface de stockage de fichiers POSIX du noyau Linux[13] a été proposée comme projet pilote. En particulier, la suggestion portait sur un petit sous-ensemble de POSIX adapté au matériel à mémoire flash avec des exigences strictes de tolérance aux pannes à utiliser lors des prochaines missions de la NASA. En raison de la nature de l'environnement dans lequel ce petit sous-ensemble devait fonctionner, deux exigences de robustesse importantes pour la tolérance aux pannes ont été convenues par la suite [5] :

(i) pas de corruption en présence d'une perte de puissance inattendue ; et
(ii) récupération à partir de défauts spécifiques au matériel flash (par exemple, blocs défectueux, erreurs de lecture, corruption de bits due au rayonnement, etc). Dans la récupération après une panne de courant en particulier, ils exigent que le système de fichiers soit réinitialisé et fiable dans le sens suivant : si une opération *Op* est en cours au moment de la panne de courant, alors au redémarrage, l'état du système de fichiers sera comme si *Op* s'était achevé avec succès ou n'avait jamais commencé.

Système de fichiers POSIX

Le choix de l'interface système de fichiers POSIX est intéressant pour diverses raisons :

(i) il s'agit d'une interface propre, bien définie et normalisée qui est stable depuis de nombreuses années ;
(ii) les structures de données et les algorithmes sous-jacents ne sont pas non plus bien compris ;
(iii) bien qu'il s'agisse d'une petite partie d'un système opérationnel, il est suffisamment complexe en termes de garanties de fiabilité, comme une panne électrique, un accès concurrent, une corruption des données, et
(iv) la technologie moderne d'information dépend largement de sa disponibilité, sûre et sûre. Toutes ces raisons vont au-delà de l'intérêt de la communauté de vérification, ainsi que de l'utilisation initiale prévue lors des prochaines missions de la NASA, telle que développée par le Jet Propulsion Laboratory (JPL).

Un sous-ensemble initial de POSIX a été choisi pour le projet pilote. Il n'y a pas de soutien pour : (a) les permissions de fichiers ; (b) les liens durs ou symboliques ; et (c) les entités autres que les fichiers et répertoires traditionnels (par exemple, pas de tuyaux, sockets, etc.) L'ajout du support pour (a) n'est pas difficile et peut se faire ultérieurement, alors que le support pour (b) et (c) est plus difficile et peut dépasser la portée du challenge. Dans tous les cas, les systèmes de fichiers à mémoire flash populaires, tels que YAFFS2[29], ne supportent pas non plus ces fonctionnalités car elles ne sont généralement pas nécessaires pour l'environnement embarqué dans lequel les périphériques flash résident habituellement.

Nous avons choisi de nous concentrer sur les fonctionnalités de base pour les fichiers et répertoires, en commençant par le premier, ce qui inclut les API sélectionnées par JPL, comme *creat, open, close, read, write, truncate, ftruncate, stat, fstat, mkdir, rmdir,*

rename, *opendir, readdir, rewinddir, closedir,* etc, où la documentation est trouvée dans[17]. En outre, nous avons également pris en compte les opérations à l'échelle du système, telles que le format, le montage et le démontage, ce qui signifie que nous ne nous occupons pas au départ du chiffrement, de la liste des répertoires, des expressions régulières et d'autres opérations utilitaires, car elles sont généralement construites sur le dessus de cette fonctionnalité de base. En outre, le JPL a opté pour la garantie très conservatrice concernant le comportement simultané que le résultat de l'exécution d'opérations simultanées équivaut à leur exécution dans un ordre de série. Cela se fait bien sûr au prix d'une certaine performance, qui est échangée contre de la simplicité. Cette simplification est peut-être une exigence trop stricte pour un système de fichiers embarqué général, donc cette question reste ouverte dans le contexte du grand défi.

5. Conclusions

Dans ce travail, nous avons réussi à mécaniser et à abstraire la spécification des fiichiers de type POSIX en utilisant le prouveur du théorème Z/Eves. Nous avons également fourni une implémentation concrète basée sur Java HashMaps qui est supprimée des annotations JML.

A partir des annotations JML. Ces annotations sont des conditions préalables et postérieures, ainsi que des invariants de classe et de boucle. A partir de ces deux types de données, nous avons prouvé qu'une simulation prospective montrant l'implémentation de HashMap affine le stockage des fichiers abstraits.

Dans le but de spécifier formellement un magasin de fichiers POSIX, nous avons divisé le travail suggéré en suivant une architecture orthogonale qui permet la séparation et la combinaison ultérieure de préoccupations, telles que les exigences fonctionnelles, les impératifs de tolérance aux pannes et les divers dispositifs matériels. C'est d'une importance cruciale pour permettre à des scientifiques ayant des intérêts et des antécédents différents de collaborer en vue de relever le défi.

Dans cet article, nous abordons une partie des exigences fonctionnelles d'un sous-ensemble d'API pertinentes pour le défi, et nous prouvons un raffinement vers une structure de données concrète qui se prête au prototypage. Dans ce processus, nous avons trouvé une opportunité de généraliser les propriétés prouvées pour les entités injectives afin que le travail dans d'autres formalismes et les outils correspondants puissent en bénéficier. En fait, nous avons réutilisé certaines de ces conclusions générales tirées de travaux antérieurs dans un autre projet pilote sur les cartes à puce. Ceci est en accord avec le dépôt de logiciels vérifiés qui fournit un port pour les conceptions vérifiées et les expériences de vérification importantes.

Nos efforts visent un " utilisateur " particulier (le JPL de la NASA), c'est pourquoi nous nous concentrons sur un petit sous-ensemble initial de fonctionnalités qui les intéresse. Mais le travail est construit de manière à ce que ce choix ne compromette pas le grand défi des idéaux scientifiques, de généralité et de précision au-delà des besoins actuels. Ceci est possible parce que nous suivons l'architecture d'Intel mentionnée ci-dessus, donc nous avons une stratégie de développement de projet modulaire. Les résultats de ces travaux sont rassemblés et disponibles en ligne dans le dépôt VSR de SourceForge.

6. Travaux futurs

Dans la spécification UNIX de[16], il y a encore deux autres parties à développer concernant les liens et répertoires. Dans [3], des progrès ont été réalisés dans l'intégration matérielle de la mémoire flash vers l'avant. Et en suivant l'architecture de[12], il y a beaucoup de questions ouvertes à traiter, comme les différents périphériques matériels, les aspects de tolérance aux pannes, et d'autres aspects de fonctionnalité comme le multithreading et le temps réel, la mise en réseau, ou les permissions de fichiers et le chiffrement. On pourrait également affiner l'implémentation concrète de HashMap jusqu'à un programme plus séquentiel, qui peut être utilisé pour comparer avec les spécifications JML. Comme le chevalier Perceval de Camelot, nous devons persévérer dans la quête du compilateur vérificateur.

7. Références

[1] L. Burdy et al. An Overview of JML Tools and Applica-tions. In8thInternational Workshop on Formal Methods forIndustrial Critical Systems (FMICS), ENTCS, pages 73–89.University of Nijmegen, Elsevier, Mar. 2003.

[2] L. Burdy et al.Java Applet Correctness: a Developer-Oriented Approach. In Proceedings of Formal Methods Eu-rope, Pisa, number 2805 in LNCS, pages 422–439. Formal Methods Europe, Springer, 2003.

[3] A. Butterfield and J. Woodcock. Formalising flash memory: first steps. In12th ICECCS, Auckland New Zealand, Jul. 2007. IEEE.

[4] Fred Barnes and others. RMoX: A Raw-Metal occam Ex-periment. In Communicating Process Architectures, pages2 69–288. IOS Press, 2003.

[5] L. Freitas. Workshop on the VSR Grand Challenge: POSIXfile stores, Dec. 2006.

[6] L. Freitas et al. Verified Software Repository @ Source-Forge. http://vsr.sourceforge.net, 2006.

[7] Z. Fu.A refinement of the UNIX Filing System using Z/Eves. Master's thesis, University of York, Oct. 2006.

[8] T. Hoare. The Verifying Compiler Software Grand Challenge. Journal of ACM, 50(1):63–69, 2003.

[9] M. Huisma Reasoning about Java Programs in Higher-Order Logic using PVS and Isabelle. PhD thesis, Universiteit Nijmegen, 2001.

[10] M. Huisman. Verification of Java's AbstractCollection Class: a Case Study. In Proceedings of 6th conference on Mathematics of Program Construction, number 2386 in LNCS, pages 175–194. Springer, 2002.

[11] IEEE POSIX Working Group. Interface Requirements for Realtime Distributed Systems Communication. Technical Report IEEE P1003.21, IEEE, Jul. 1995.

[12] Intel® Flash File System Core Reference Guide, version 1.Technical Report 304436001, Intel Coorporation, Oct. 2004.

[13] A. Josey, editor.The Single UNIX Specification Version 3.Open Group, 2004. ISBN: 193162447X.

[14] R. Joshi and G. J. Holzmann. A Mini-Challenge: Build A Verifiable Filesystem. InVerified Software: Theories, Tools, Experiments (VSTTE), Zurich, Switzerlan, 2005. IFIP Work-ing Conference.

[15] I. Lakatos.Proofs and Refutations: The Logic of Math-ematical Discovery.Cambrdige University Press, 2005.ISBN: 0521290384.

[16] C. Morgan and B. Sufrin. Specification of the UNIX FilingSystem. In Transactions on Software Engineering, volume SE-10, pages 128–142. IEEE, 1984.

[17] Open Group Technical Standard. Protocols for Inter-working: XNFS, Version 3W.Technical Report C702,The Open Group, Feb. 1998. ISBN: 1859121845.

[18] P. Place. POSIX 1003.21—Real Time Distributed Systems Communication. Technical report, Software Engineering In-stitute @ Carnegie Mellon University, Aug. 1995.

[19] G. Polya. How to Solve It: a New Aspect of Mathematical Method. Princeton University Press, 2004.

[20] B. Russell. Recent Work in Philosophy of Mathematics. International Monthly, 1901. Reprinted in Mysticism and Logic and Other Essays(p.59–74) Barnes & Noble, 1976.

[21] M. Saaltink. Z/Eves 2.0 User's Guide. ORA Canada, 1999.TR-99-5493-06a.

[22] S. Schneider.The B-Method—an Introduction. Palgrave,2002.

[23] J. M. Spivey.The Z Notation: A Reference Manual.Prentice-Hall, 1998.

[24] SRI.Workshop on the Verification Grand Challenge.www.csl.sri.com/users/shankar/VGC05, Feb. 2005.

[25] S. Stepney et al. An Electronic Purse: Specification, Refine-ment, and Proof. PRG 126, Oxford University, Jul. 2000.

[26] J. Woodcock and J. Davies.Using Z: Specification, Refine-ment, and Proof. Prentice-Hall, 1996.

[27] J. Woodcock and L. Freitas.Z/Evesand the Mondex Elec-tronic Purse. In3rdICTAC, volume 4281 ofLNCS, pages15–34. Springer, 2006.

[28] J. C. P. Woodcock and A. L. C. Cavalcanti. A ConcurrentLanguage for Refinement. In A. Butterfield and C. Pahl, editors, IWFM'01: 5th Irish Workshop in Formal Methods,BCS Electronic Workshops in Computing, Dublin, Ireland,July 2001.

[29] YAFFS Direct Interface(YDI) User's Guide, Jul. 2006.

https://www.researchgate.net/publication/224483525_Specification_of_the_UNIX_Filing_System

https://www.computer.org/csdl/proceedings-article/iceccs/2007/28950003/12OmNBtUdMq

http://www.cs.cmu.edu/~15819/zedbook.pdf