langage PRISM) basé sur une version simplifiée de Reactive Modules.

Ce langage a été adopté comme un standard *de facto*. Il a néanmoins plusieurs faiblesses : les seuls types disponibles sont les entiers et les booléens, il n'existe pas de structures de contrôle (`if`, `for`, *etc.*). Ce dernier point est probablement le plus gênant car lorsqu'il s'agit de décrire des systèmes conséquents, la spécification devient vite pénible. Certaines personnes mettent au point des solutions « ad hoc » pour pallier ce problème : l'utilisation de scripts shell, du préprocesseur du C (`cpp`) ou du langage `M4` sont des exemples courants.

Le but de cet exposé est d'étendre le langage PRISM pour résoudre certains de ces problèmes. Néanmoins, comme il faut rester compatible avec le parseur de PRISM, il faut traduire les systèmes écrits dans la version étendue du langage vers du code PRISM pur. Cette traduction est assurée par le package XRM qui fournis également de nombreux outils (parseurs, *pretty-printers*).

OLENA

OLENA est une bibliothèque de traitement d'images générique et performante. Un algorithme est écrit une fois et peut s'appliquer sur tout type d'images (binaire, en couleur, 2D, 3D, etc). OLENA sert également de cadre aux recherches sur le traitement d'images.

Conception d'un langage statique, un aperçu de SCOOL

par *Q. Hocquet, T. Moulard* — *14 juin, 14h30, P004*

Lors du développement d'OLENA, deux bibliothèques ont été créées afin de contourner les différentes limitations du C++ : Metalic et Static.

Static est une bibliothèque permettant d'utiliser des classes abstraites et des méthodes virtuelles sans payer le prix du typage dynamique : une fonction travaillant sur des abstractions connaît à la compilation le type exact de ses arguments, évitant le surcoût à l'exécution.

Metalic est une bibliothèque offrant beaucoup de constructions de métaprogrammation (`if`, `switch` statiques notamment), des `typedefs` virtuels, des vérifications sur les types à la compilation (vérification que deux types sont égaux ou possèdent une relation d'héritage).

Cependant, l'utilisation de ces dernières est loin d'être simple pour un développeur qui ne connaît pas les mécanismes de fonctionnement interne de ces outils. L'utilisation abondante de *templates* et des subtilités associées à ces derniers rend, de plus, la compréhension de ces bibliothèques difficile.

Afin de permettre une écriture simple, la création d'un langage suffisamment expressif pour incorporer dans sa syntaxe les fonctionnalités de Metalic et Static a été décidé. Ce langage doit également être très performant afin de satisfaire les contraintes de rapidité d'exécution liés à OLENA. C'est de ce constat qu'est né SCOOL, un langage statique orienté objet. Ses principales caractéristiques sont : un typage statique ; des `typedefs` virtuels ; des multiméthodes (à résolution statique) ; des contraintes sur les arguments des procédures ; la programmation par contrat ; *etc.*

Ces fonctionnalités apparaissent directement dans la syntaxe du langage donc toutes les constructions complexes que nous voulions écrire en C++ trouvent ici une représentation simple et courte.

SCOOL est un langage transformé vers C++ : le « compilateur » génère des fichiers C++ qui sont ensuite compilés classiquement. Dans cette optique, il est possible d'incorporer du code C++ directement dans SCOOL. Cette transformation n'est malheureusement pas aisée, à cause de la grande complexité de Metalic, Static ainsi que de la grammaire du C++.

Ce séminaire se déroulera en deux parties. Tout d'abord un aperçu des fonctionnalités de SCOOL et de sa syntaxe seront expliqués, puis nous passerons à la façon de transformer cette syntaxe en C++.

Dynamisation de bibliothèques C++ statiques

par *Tristan Croiset* — *14 juin, 14h00, amphi P004*

L'utilisation de bibliothèques statiques en C++ est fréquente notamment pour proposer des ensembles d'outils de traitements scientifiques. Cette approche qui permet un développement plus efficace et une importante généricité, cause plusieurs désavantages à l'utilisation et particulièrement des temps de compilation rédhibitoires.

Nous avons donc cherché à proposer un pont qui permette de faciliter l'utilisation de ces bibliothèques dans un contexte purement dynamique. Plusieurs travaux ont déjà été réalisés au sein du laboratoire et ont permis d'une part d'en préciser les besoins et d'autre part de démontrer sa faisabilité. Les solutions retenues conservent l'utilisation du C++ et font appel aux bibliothèques statiques via une compilation « juste à temps ».

Ce séminaire sera axé sur la proposition d'une solution simple qui doit allier un fonctionnement parfaitement décrit et délimité avec une souplesse suffisante. Nous verrons ensuite comment ce socle stable pourra ensuite servir de base à des extensions d'usage (« sucres ») pour différents cas d'utilisations.

L'air de rien N° 5

Séminaires CSI de juin

L'aléastriel du **L**aboratoire de **R**echerche et de **D**éveloppement de l'**E**PITA[1] Numéro 5, Juin 2006

Edito

par *Akim Demaille*

Les séminaires CSI sont à l'intersection des trois missions du LRDE : enseignement, recherche et développement. Les étudiants de l'option CSI, cursus alternatif de formation par la recherche, travaillent avec les enseignants-chercheurs du laboratoire sur des aspects théoriques ou sur la conception de solutions logicielles. Selon les cas, ces travaux se matérialisent par des publications présentées dans des conférences internationales (voire dans de prestigieuses revues scientifiques), ou bien par la diffusion de composants logiciels qui contribuent à établir des collaborations avec d'autres écoles d'ingénieurs ou universités, ou bien encore avec le développement de prototypes vendus à l'industrie.

Ces activités[2] contribuent à l'attractivité de l'EPITA, aussi bien pour les jeunes candidats, que pour de futurs enseignants, des thésards cherchant un lieu d'accueil pour leur doctorat, ou bien des industriels à la recherche d'innovation. Gageons que les séminaires du 7 juin[3] et 14 juin[4] 2006 vous donneront une idée plus précise des thèmes de travail… ou peut-être même l'envie de venir nous voir !

Venez nombreux, et bonne lecture !

En bref

Les publications (disponibles sur publis.lrde.epita.fr)
– *Beating C in Scientific Computing Applications* par Didier Verna accepté au *European Lisp Workshop* à *European Conference on Object-Oriented Programming (ECOOP)*.

Programmation génétique & *Corewar*

par *Jérémy Marc* — *7 juin, 15h00, amphi P01*

La programmation génétique est un des nombreux dérivés de l'algorithmique évolutive communément assimilée à de « l'intelligence artificielle » bas niveau.

Ce paradigme de programmation s'inspire de phénomènes omniprésents en biologie tels que la sélection naturelle, la mutation, l'héritage génétique, le croisement (*crossing-over*), *etc.*

Son fonctionnement est donc calqué sur la nature. Après avoir défini les gènes caractéristiques d'un individu, on génère (plus ou moins aléatoirement) une population suffisamment diversifiée afin de pouvoir la faire évoluer et couvrir au maximum l'ensemble des possibles. Ensuite durant un certain nombre de générations, les recombinaisons successives vont mettre à jour de nouveaux individus qui seront évalués et conservés si toutefois ils sont plus performants que leurs géniteurs.

Lorsque l'individu est un programme, la tâche devient nettement plus ardue. Il est en effet délicat d'exhiber les paramètres de tels programmes. Dans le cadre du séminaire, nous aborderons les difficultés liées à la création d'un programme robuste pour le célèbre *Corewar*.

[1] L'air de rien, `http://publis.lrde.epita.fr/LrdeBulletin`.
[2] Les actualités du LRDE, `http://www.lrde.epita.fr/cgi-bin/twiki/view/Lrde/News`.
[3] Séminaire CSI du 7 juin 2006, `http://publis.lrde.epita.fr/Seminar-2006-06-07`.
[4] Séminaire CSI du 14 juin 2006, `http://publis.lrde.epita.fr/Seminar-2006-06-14`.

MARKOV

Les modèles probabilistes sont très en vogue dans le milieu de la recherche et de l'industrie. Beaucoup d'algorithmes s'appuient sur les hypothèses de Markov.

Vérification du locuteur

par *Charles Deledalle* — *7 juin, 15h45, amphi P01*

La vérification du locuteur consiste à détecter si un segment de parole est réellement prononcé par un individu présumé, contrairement à l'identification dont le but est de reconnaître l'auteur du message. Une des contraintes est de fournir un système fiable quelles que soient les conditions d'acquisition. Notre travail consiste donc à extraire les caractéristiques acoustiques du locuteur tout en s'affranchissant des bruits d'acquisition et du canal de transmission.

Une phase d'apprentissage est tout d'abord effectuée. Le but est de modéliser au mieux chaque locuteur. Pour cela il existe différentes méthodes basées sur l'étude des vecteurs acoustiques. Ensuite, lors de la phase de test, l'énoncé est comparé au modèle du locuteur présumé.

Au cours de ce séminaire nous examinerons une première approche, probabiliste (état de l'art), utilisant les modèles de mixture gaussienne. Puis nous proposerons deux solutions basées sur les méthodes SVM (voir classification des galaxies dans *L'air de rien* n°1), dont la différence se situe au niveau des noyaux et des caractéristiques utilisés. Notre travail s'inscrit dans le prolongement de l'évaluation NIST[5], et dans le cadre de l'évaluation ISCSLP[6] en cours.

Model-checking qualitatif approché

par *J-Ph Garcia Ballester* — *7 juin, 16h15, amphi P01*

Le model-checking consiste à vérifier certaines propriétés sur un modèle donné. Les méthodes pour vérifier une propriété ou calculer sa probabilité de manière sûre, c'est-à-dire sans erreur possible, sont très coûteuses : les plus efficaces peuvent nécessiter le calcul d'intégrales ou d'exponentielles de matrices.

Une méthode approchée a donc été trouvée, permettant de réduire le temps de calcul. L'algorithme échantillonne les chemins possibles, et vérifie la propriété sur l'échantillon. Si elle est vraie sur t chemins et fausse sur f chemins, alors on peut calculer la probabilité approchée : $t/(t+f)$, avec une précision directement liée à la taille de l'échantillon.

Calculer une probabilité est quantitatif : ce qui nous intéresse est la valeur de la probabilité. Mais il y a des cas où le qualitatif est suffisant : ce qui nous intéresse est de savoir si cette probabilité est supérieure ou inférieure à un certain seuil. Dans ce cas, la précision nécessaire pour pouvoir décider est plus ou moins grande suivant la valeur réelle de cette probabilité et le seuil demandé par l'utilisateur.

Ce séminaire présente les bases du model-checking, et une optimisation de l'algorithme de Sylvain Peyronnet pour la vérification qualitative de propriétés.

Chaînes de Markov continues en présences d'incertitudes

par *Nicolas Neri* — *7 juin, 16h45, amphi P01*

Le 26 Mars 2006 lors de la 12e conférence internationale TACAS[7], Khoushik SEN[8] a présenté un article sur la vérification de programmes en utilisant les chaînes de Markov finies à temps discret pour lesquelles la probabilité exacte des transitions n'est pas connue : les IDTMCs (*Interval-valued Discrete-time Markov Chains*).

Ce séminaire présentera les travaux en cours sur l'extension de l'algorithme de Koushik SEN pour les chaînes de Markov à temps continu présentant des incertitudes. Koushik SEN considère deux sens d'interprétation pour ses IDTMCs. La présentation portera sur la sémantique UCTMC (*Uncertain Continuous-times Markov Chain*) des ICTMCs (*Interval-valued Continuous-times Markov Chains*). Le but sera de trouver un ensemble de variables satisfaisant certaines contraintes et de vérifier la faisabilité de celles-ci grâce à la résolution des BMIs (*Bilinear Matrix Inequalities*).

DIAGRAMMES DE DÉCISION

Les diagrammes de décision sont des structures utilisées dans de nombreux domaines où l'utilisation mémoire est critique, par exemple la vérification de programmes. Les modèles étant de plus en plus complexes, la mémoire d'une seule machine ne suffit souvent plus.

[5] NIST, http://www.nist.gov/speech/.
[6] ISCSLP, http://www.iscslp2006.org/.
[7] TACAS, Tools and Algorithms for the Construction and Analysis of Systems, http://depend.cs.uni-sb.de/index.php?id=329.
[8] Khoushik SEN, Department of Computer Science, University of Illinois at Urbana-Champaign, http://osl.cs.uiuc.edu/~ksen/.

BDD distribués et Java

par *Guillaume Guirado* — *7 juin, 14h00, amphi P01*

Les BDD (Diagrammes de Décision Binaires) ont été très largement étudiés depuis leur introduction par Randy E. Bryant à la fin des années 80. De très nombreuses implémentations ont été proposées, la plupart reposant sur le langage C.

Cependant, rares sont les bibliothèques permettant de distribuer notre diagramme sur un cluster. Encore plus rares sont celles qui permettent d'utiliser un ensemble de machines de bureau. Partant de ce constat, l'implémentation, dans un langage moderne, d'une bibliothèque de gestion de BDD permettant la répartition sur un grand nombre de machines est nécessaire. Cela permettra d'augmenter considérablement la mémoire disponible, et donc de résoudre des problèmes de plus complexité accrue.

Dans cette optique, nous avons choisi le langage Java, connu pour sa portabilité, sa gestion automatique de la mémoire ainsi que la simplicité avec laquelle les processus peuvent communiquer (sérialisation). Durant la présentation, nous verrons les principales propriétés des BDD et leurs opérations à travers différents exemples simples. Ensuite, nous analyserons les résultats d'une implémentation (en Java) non répartie, avant de finalement exposer les choix, forces et faiblesses, de notre bibliothèque répartie.

Répartition des DDD en Erlang

par *Samuel Charron* — *7 juin, 14h30, amphi P01*

Les DDD (*Data Decision Diagrams*) généralisent le concept de BDD, en permettant de ne pas se limiter aux fonctions booléennes. Cependant, ils n'apportent pas de solution supplémentaire au problème de mémoire.

La répartition de BDD a déjà été longuement étudiée sans donner de résultat réellement satisfaisant. Une étude sur les propriétés des DDD nous a permis de réaliser la répartition.

Nous utilisons pour cela Erlang, un langage fonctionnel dédié aux télécommunications créé dans les années 90. Les primitives de communication et de parallélisation font partie intégrante de ce langage, permettant d'envoyer de façon portable n'importe quelle donnée en une seule ligne de code.

Lors de ce séminaire, nous vous présenterons les DDD. Ensuite, une introduction à Erlang et a l'implémentation réalisée sera effectuée. Enfin, nous comparerons plusieurs algorithmes de répartition existants sur les BDD en y ajoutant d'autres algorithmes nouveaux.

TRANSFORMERS

TRANSFORMERS est une plate-forme de transformation de programmes dédiée au C et au C++. Le projet repose sur Stratego/XT et sur de nombreux outils développés en interne, tels qu'un moteur de grammaire attribuée utilisé lors de la phase de désambiguïsation.

Aperçu des Grammaires Attribuées

par *Nicolas Pierron* — *14 juin, 15h45, amphi P004*

La notion de grammaire attribuée fut introduite par Knuth en 1968. Cette notion permet l'implémentation de certaines passes des compilateurs actuels.

Les attributs sont des symboles attachés aux règles de la grammaire, ils portent une information telle que le résultat d'une évaluation ou bien une table de symboles. L'information peut être propagée entre les attributs en suivant les règles de la grammaire.

Plusieurs types de propagations d'attributs existent, des propagations de haut en bas, de bas en haut, et de gauche à droite. Ces catégories se répartissent sous les noms de L-attributs, S-attributs, et LR-attributs.

Les grammaires attribuées permettent de faire de la désambiguïsation, ou de la vérification de types, ...

De même, il existe quelques systèmes qui gèrent certains types de propagations, en vérifiant les problèmes de cycles à la compilation de l'évaluateur tel que UU-AG, et le module *sdf-attribute* du projet TRANSFORMERS.

Cet exposé vous donnera un aperçu des grammaires attribuées, ainsi que des propagations possibles et de la sémantique associée.

eXtended Reactive Modules (XRM)

par *Benoît Sigoure* — *14 juin, 16h15, amphi P004*

Reactive Modules[9] est un formalisme utilisé pour modéliser des systèmes concurrents. De nombreuses applications de *model-checking* se basent dessus. PRISM[10], un logiciel dedié au *model-checking* probabiliste, utilise un langage du même nom (le

[9] Reactive Modules, http://mtc.epfl.ch/~tah/Publications/reactive_modules.html.
[10] PRISM, http://www.cs.bham.ac.uk/~dxp/prism/.