DEA d'informatique de l'université de Rennes 1

Filière 2 :
Méthodes formelles pour le logiciel

Ce document est disponible et
est mis à jour sous la forme de pages web à l'URL suivante :
`http://www.irisa.fr/lande/dea/mfl/`

23 septembre 2003

Table des matières

1 Présentation 3

2 Organisation des études dans la filière 3

3 Contacts 3

4 Présentation des modules 3
4.1 EFF : Effectivité et efficacité . . . 4
4.1.1 Présentation . . . 4
4.1.2 Plan du module . . . 4
4.1.3 Bibliographie . . . 4
4.2 A2R : analyse, synthèse et supervision des systèmes réactifs et répartis 5
4.2.1 Présentation . . . 5
4.3 SAP : Sémantique et analyse de programmes . . . 7
4.3.1 Présentation . . . 7
4.3.2 Plan du module . . . 7
4.3.3 Bibliographie . . . 7
4.4 COT : Composants et test . . . 8
4.4.1 Présentation . . . 8
4.4.2 Plan du module . . . 8
4.4.3 Bibliographie . . . 9
4.5 PLOG : λProlog . . . 10
4.5.1 Présentation . . . 10
4.5.2 Plan du module . . . 10
4.6 MDV: Méthodes Déductives pour la Vérification . . . 11
4.6.1 Présentation . . . 11
4.6.2 Plan du module . . . 11
4.6.3 Bibliographie . . . 11
4.7 TCPR : Programmation avancée . . . 12
4.7.1 Présentation . . . 12
4.7.2 Plan du module . . . 12
4.8 MLTO : Méthode, langage et technologie objets . . . 13
4.8.1 Présentation . . . 13
4.8.2 Objectifs . . . 13
4.8.3 Bibliographie . . . 13

5 Cours-conférences 14
5.1 Sécurité et méthodes formelles . . . 14
5.2 Modèles de la sécurité informatique . . . 14

6 Equipes de recherche de l'IRISA connectées à la filière "Méthodes formelles pour le logiciel" 15

7 Sujets de stages 16

1 Présentation

Cette filière offre une formation de pointe en génie logiciel et en méthodes formelles, deux domaines de recherche qui sont à la fois en rapide développement et en remarquable complémentarité :

- Le génie logiciel est un ensemble de méthodes visant à fiabiliser et systématiser le développement des logiciels de grande taille. On y trouve notamment des méthodes de conception par objets et des techniques de test du logiciel.
- Les méthodes formelles permettent la preuve de propriétés et la transformation de programmes. Elles ont toutes des fondements rigoureux utilisant des résultats de sémantique, de logique ou de théorie des graphes.

Ces deux domaines sont actuellement en réelle explosion du fait de l'intérêt croissant que porte l'industrie dans l'utilisation combinée de ces techniques. Elles sont complémentaires car les méthodes formelles s'intègrent naturellement dans le cycle de vie du logiciel et viennent de ce fait renforcer les méthodes d'ingénierie du logiciel. A l'inverse, les techniques formelles ne sont utilisables qu'une fois intégrées dans une méthode de génie logiciel.

Les enseignements de la filière se situent à la croisée des deux disciplines et couvrent un large spectre de questions allant des plus fondamentales et théoriques jusqu'aux applications les plus ciblées. Les débouchés au sortir de la filière sont excellents aussi bien dans le monde académique (possibilités de thèses, et ensuite perspectives de recrutement dans l'enseignement supérieur et la recherche) que dans l'industrie (grandes entreprises en télécommunications, avionique, spatial, automobile, médical, etc).

2 Organisation des études dans la filière

Les étudiants doivent suivre sept modules complets : les trois modules de tronc commun et quatre autres modules au choix parmi ceux de la filière. Il est aussi possible, après accord du responsable de filière, de remplacer un des modules de la filière par un module pris parmi ceux des deux autres filières du DEA. En particulier, un des modules suivants, issus de la filière "réseaux et systèmes" peut être suivi dans le cadre de la filière "génie logiciel et méthodes formelles": FSR (Fondements des systèmes répartis) et ALPA (Algorithmique Parallèle). Le demi-module TCPR est optionnel (donne lieu à une bonification).

Les étudiants sont aussi invités à suivre les cours-conférences (d'une durée de trois heures) proposés dans le cadre du DEA ainsi que le séminaire 68NQRT annoncé dans le newsgroup `irisa.seminaires`.

3 Contacts

Pour toute question sur le déroulement des études au sein de la filière, prière de contacter Thomas Genet (Thomas.Genet@irisa.fr, tél. 02 99 84 73 44, bâtiment D, bureau D068) responsable de la filière. Les questions spécifiques à un module sont à adresser aux intervenants du module. Toutes les informations sur la filière sont regroupées dans un ensemble de pages web accessibles à l'URL suivante:

`http://www.irisa.fr/lande/dea/mfl/`

4 Présentation des modules

4.1 EFF : Effectivité et efficacité

- **Intervenant** : Didier Caucal, chargé de recherche au CNRS (Didier.Caucal@irisa.fr)
- **Volume** : 21 heures

4.1.1 Présentation

Ce module fait suite à une demande d'étudiants en DEA et en thèse, et est un complément dans la connaissance générale en informatique.

Dans ce module, nous traitons les deux questions suivantes. Étant donné un problème, peut-on le résoudre de façon effective? Une solution donnée est-elle efficace? L'intérêt est d'éviter de rechercher un algorithme à un problème non effectif, et sinon d'être capable de distinguer les algorithmes efficaces de ceux qui ne le sont pas. Après des rappels nécessaires, nous proposons des études de cas variés permettant de maîtriser ces deux aspects dans des situations concrètes.

Aucun prérequis n'est demandé.

4.1.2 Plan du module

1. **Effectivité** : machines algorithmiques, problèmes sans solution effective, étude de cas.
2. **Efficacité** : combinatoire, récurrences, étude de cas.

4.1.3 Bibliographie

- L. Comtet, Analyse combinatoire, tomes 1 & 2, Presses Universitaires de France, 1970.
- D. Knuth, The art of computer programming, tomes 1 & 3, Addison Wesley, 1973.
- R. Sedgewick, P. Flajolet, Introduction à l'analyse des algorithmes, International Thomson Publishing France, 1996.
- A. Arnold, I. Guessarian, Mathématiques de l'informatique, Masson, 1992.
- C. Froidevaux, M.-C. Gaudel, M. Soria, Types de données et algorithmes, McGraw Hill, 1990.
- T. Cormen, C. Leiserson, R. Rivest, Introduction à l'algorithmique, Dunod, 1992.
- J.M. Autebert "Calculabilité et décidabilité" Masson 1992.
- M. D. Davis, E. J. Weyuker "Computability, complexity and languages, Fundamentals of theoretical science" Academic Press 1983
- D. Harel "The spirit of computing" Addison-Wesley 1987.
- J. Hopcroft, D. Ullman "Introduction to automata theory, languages, and computation" Addison-Wesley 1979.
- A. Salomaa "Introduction à l'informatique théorique" Armand Colin 1989.
- P. Wolper "Introduction à la calculabilité" Interéditions 1991.

4.2 A2R : analyse, synthèse et supervision des systèmes réactifs et répartis

- **Intervenants** : Sophie Pinchinat, maître de conférences à l'Ifsic (Sophie.Pinchinat@irisa.fr)
- **Volume** : 21 heures
- **Equipe de recherche** : S4

4.2.1 Présentation

Ce module est en deux parties enseignées alternativement une année sur deux. Une partie porte sur l'**analyse et la supervision de réseaux de Petri**, et l'autre sur les **méthodes formelles pour la vérification des systèmes réactifs**. Cette année le module qui vous est proposé est:

Méthodes Formelles pour la Vérification des Systèmes Réactifs
proposée par Sophie Pinchinat.

De nos jours, l'omniprésence de l'informatique dans les réseaux de communication et la manière dont les programmes se substituent à l'homme pour des tâches délicates (e.g. contrôle d'appareils d'assistance médicale, régulation thermique de réacteurs nucléaires, etc...) imposent le développement de méthodes rigoureuses et efficaces pour la validation des produits logiciels. Les systèmes qui nous occupent dans ce contexte sortent du champ des programmes dits *transformationnels* : ils n'ont pas forcément d'état final, et notre intérêt porte plutôt sur leur comportement réactif. C'est dans ce sens que Pnueli les appelle *systèmes réactifs*.

Les méthodes formelles définissent des techniques d'analyse des systèmes réactifs qui se situent en amont de méthodes bien connues de simulation et de test. Elles s'appuient sur des théories mathématiques permettant de construire les systèmes - on parle de *spécification* et de *synthèse* -, d'exprimer leurs propriétés et de raisonner de façon rigoureuse sur leur comportement - on parle encore de *spécification* et de *vérification* -, et, dans l'éventualité de mettre au point ses systèmes par exemple en les contrôlant, - on parle du *contrôle* d'un système -.

Plan du module

- **Cours 1** Introduction à la vérification des systèmes.
- **Cours 2** Représentation des systèmes
- **Cours 3** Logiques temporelles et modales.
- **Cours 4** Model Checking par méthodes itératives.
- **Cours 5** Méthodes formelles par automates d'arbres
- **Cours 6** Procédures de décision basées sur les jeux de parité.
- **Cours 7** Vers les problèmes de synthèse de contrôleurs.

Bibliographie

- Ph. Schnoebelen, B. Bérard, M. Bidoit, F. Laroussinie, and A. Petit. Vérification de logiciels : Techniques et outils du model-checking. Vuibert, 1999, **mais aussi** B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. Systems and Software Verification. Model-Checking Techniques and Tools. Springer, 2001.*
- E. M. Clarke, Orna Grumberg, Doron Peled . Model Checking Mit 314 pages (December 1999)
- E. A. Emerson, Temporal and modal logic. In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, vol B, chapter 16, pages 995–1072. Elsevier Science Publishers, 1990.
- E. A. Emerson, Automated Temporal Reasoning about Reactive Systems. Logics for Concurrency: Structure versus Automata, F. Moller and G. Birtwistle

(eds.), Springer LNCS no. 1043, Pages 111-120.
- Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, March 2000.
- Automata, Logics, and Infinite Games: A Guide to Current Research. Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. Outcome of a Dagstuhl seminar, February 2001, volume 2500 of Lecture Notes in Computer Science. Springer, 2002. `http://www-mgi.informatik.rwth-aachen.de/Misc/LNCS2500/`.

4.3 SAP : Sémantique et analyse de programmes

- **Intervenants** : Bertrand Jeannet, chargé de recherche à l'INRIA (Bertrand.Jeannet@irisa.fr) et Thomas Jensen, chargé de recherche au CNRS (Thomas.Jensen@irisa.fr)
- **Volume** : 21 heures
- **Equipe de recherche** : LANDE et VERTECS

4.3.1 Présentation

Ce module présente les principales idées et méthodes utilisées en sémantique et analyse de programme. Une sémantique définit formellement le sens des programmes. Elle permet d'éviter les ambiguïtés, de prouver la correction de la mise en oeuvre ou de raisonner sur les programmes. L'analyse de programme est un type de raisonnement fondé sur une sémantique. Les analyses permettent de déduire automatiquement et statiquement des propriétés sur les programmes. Elles sont largement utilisés dans les phases de mise au point (découverte d'erreurs) et de compilation (optimisations).

Le domaine couvert fait partie des fondements des langages de programmation. Le module est de ce fait d'intérêt général ; il n'est lié ni à une équipe de recherche, ni à un langage particulier.

4.3.2 Plan du module

1. Sémantiques opérationnelles. Sémantique à grands pas (naturelle). Sémantique à petits pas (SOS). Techniques de preuves.

2. Analyse de flot de données. Exemples. Graphe de flot de contrôle. Treillis et points fixes. Résolution d'équations et de contraintes.

3. Sémantiques dénotationnelles. Ordres partiels complets (CPO) et points fixes. Techniques de preuves.

4. Interprétation abstraite – théorie de l'approximation statique. Treillis de propriétés. Connexion de Galois et approximation de points fixes. Analyse de signes.

5. Systèmes de types.

6. Interprétation abstraite – théorie de l'approximation dynamique. Élargissement et séquences descendantes. Analyse d'intervalles. Autres exemples d'analyse de variables numériques et hiérarchie d'interprétations abstraites. Analyse interprocédurale.

7. Présentation d'articles de recherche récents.

4.3.3 Bibliographie

- H.R. Nielson and F. Nielson, *Semantics with Applications - A Formal Introduction*, Wiley 1992. (chapitre 1) `http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.html`
- D.A. Schmidt, *Denotational Semantics: a Methodology for Language Development*, Allyn and Bacon, 1986. (chapitre 8)
- F. Nielson, H.R. Nielson, C. Hankin, *Principles of program analysis*, Springer, 1999.

4.4 COT : Composants et test

- **Intervenants** :
  Thierry Jéron (Resp. du module), chargé de recherche INRIA (Thierry.Jeron@irisa.fr),
  Arnaud Gotlieb, chargé de recherche INRIA (Arnaud.Gotlieb@irisa.fr)
  Yves Le Traon, maître de conférences à l'IFSIC (yletraon@irisa.fr)
- **Volume** : 21 heures
- **Equipes de recherche** : Vertecs Lande Triskell

4.4.1 Présentation

Le test est aujourd'hui la technique de validation la plus utilisée pour s'assurer de la correction du logiciel. Cependant, sa pratique reste encore bien souvent artisanale. L'objectif du cours est de montrer que le test de logiciel peut être une activité rigoureuse, fondée sur des modèles, des théories et des algorithmes permettant d'automatiser les différentes phases du test pour des types de logiciel variés.

Après une introduction à la problématique du test et son automatisation, le cours est divisé en 3 parties complémentaires. La première partie s'intéresse au test de programmes impératifs, pour lesquels les seuls modèles disponibles sont ceux issus de l'analyse de leur code source. On abordera les techniques de test structurel qui ont pour but de couvrir le modèle suivant des critères liés au contrôle ou aux données. La deuxième partie du cours est consacrée au test de conformité des systèmes réactifs, fondé sur des spécifications. Après une introduction à la problématique de la génération automatique de tests de conformité, on présentera deux techniques principales, l'une fondée sur le modèle des automates, l'autre sur le modèle des systèmes de transitions. La troisième partie s'intéresse au test de systèmes à base de composants objets. On abordera le problème de la qualification des composants par le test, puis les techniques de test d'intégration lors de la phase d'assemblage et enfin les aspects méthodologiques et le test système.

4.4.2 Plan du module

Cours 0 : Introduction : Historique, définitions, problématique du test (A. Gotlieb)

1ère Partie : Test structurel pour les programmes impératifs (A. Gotlieb)

- **Cours 1 :** Test et test structurel: principes
- **Cours 2 :** Approches probabilistes pour le test structurel
- **Cours 3 :** Approches déterministes pour le test structurel

2ème Partie : Génération de tests de conformité pour les systèmes réactifs (T. Jéron)

- **Cours 4 :** Introduction au test de conformité et
  Modèles et techniques fondées sur les automates
- **Cours 5 :** Modèles et techniques fondées sur les systèmes de transition
- **Cours 6 :** Techniques avancées: test réparti, techniques symboliques

3ème Partie : Test de composants orientés objet (Y. Le Traon)

- **Cours 7 :** Test et qualification de composants (mutation et contrats)
- **Cours 8 :** Test d'intégration à partir d'un diagramme de classes
- **Cours 9 :** Test système et patrons de test, testabilité d'un système OO

4.4.3 Bibliographie

1. C. Karapoulios, P. Regnier, S. Xanthakis - Le test des logiciels, Hermès.
2. R. A. Demillo and A. J. Offut, Constraint-Based Automatic Test Data Generation, IEEE Transactions on Software Engineering, SE-17 (9), pp 900-910, September,1991.
3. H. Zhu, P. A. V. Hall and J. H. R. May, Software unit test coverage and adequacy, ACM Computing Surveys (CSUR), 29 (4), pp 366–427, 1997, ACM Press, http://doi.acm.org/10.1145/267580.267590.
4. D. Lee, M. Yannakakis, Principles and methods of testing finite state machines. A survey. Proceedings of the IEEE, Vol 84, NU 8, August 1996.
5. T. Jéron, TGV: théorie, principes et algorithmes, Techniques et Sciences Informatiques, numéro spécial Test de Logiciels, (21), 2002.
6. H. Vu Le, K. Akif, Y. Le Traon, J.-M. Jézéquel, Selecting an Efficient OO Integration Testing Strategy: An Experimental Comparison of Actual Strategies, published in ECOOP (European Conference for Object-Oriented Programming) proceedings, Lecture Notes in Computer Science, Springer Verlag, June 2001, pp. 381-401.

4.5 PLOG : λProlog

- **Intervenant** : Yves Bekkers, professeur à l'IFSIC (Yves.Bekkers@irisa.fr)
- **Volume** : 21 heures

4.5.1 Présentation

L'objectif de ce cours est de présenter de nouveaux outils de conception de logiciel, basés sur un langage déclaratif λProlog. Un mélange uniforme et homogène de deux théories, le λ-calcul et la logique des prédicats permet de spécifier des solutions à des problèmes complexes dans les quels la notion de portée est fondamentale. On rencontre ainsi les problèmes de typages, de compilation ou de réécriture de programme ou encore les problèmes de représentation de connaissance en intelligence artificielle. Nous concentrerons notre étude sur deux aspects qui se complètent mutuellement : les nouvelles possibilités offertes par les λ-termes en tant méthode de représentation de structures de données et les nouveaux outils de controles qui permettent de spécifier des parcours inductifs sur la structure de ces nouveaux objets. Nous insisterons particulièrement sur les questions difficiles posées par la nature de ce mélange de deux théories. Le cours sera illustré par de nombreux exemples dans les quels on pourra constater que, grâce à cette méthode, une *spécification* peut devenir *exécutoire*.

Les deux dernières séances sont consacrées à la pratique

4.5.2 Plan du module

Cours 1 : Introduction aux concepts : le mariage de la logique et du λ-calcul

Cours 2 : Les λ-termes simplement typés, bien représenter les portées

Cours 3 : Deux notions de substitutions complémentaires, l'unification, la β-réduction

Cours 4 : L'ordre supérieur

Cours 5 : Gérer les attributs sémantiques en analyse sytaxique, les DCG d'ordre supérieures

Cours 6 : Pratique sur un exemple concrêt

Cours 7 : Pratique sur un second exemple concrêt

4.6 MDV: Méthodes Déductives pour la Vérification

- **Intervenants** : Thomas Genet, Vlad Rusu
- **Volume** : 21 heures
- **Equipe de recherche** : LANDE et VERTECS

4.6.1 Présentation

La vérification formelle de systèmes informatiques répond à un besoin croissant de sécurité de la part des utilisateurs et constitue un axe majeur dans la recherche informatique actuelle. Le but de ce cours est de présenter certains concepts communs aux méthodes formelles, en les illustrant par la vérification de programmes à l'aide de l'assistant de preuves PVS. Chacune des séances sera partagée en une partie théorique et une partie d'application sur machine. L'évaluation se fera par un examen écrit et par un projet à réaliser avec PVS.

4.6.2 Plan du module

1. Introduction aux méthodes formelles: approches déductives et algorithmiques. Logique propositionnelle et des prédicats. Calcul des séquents.

2. Indécidabilité de l'arithmétique non-linéaire. Théorème de Gödel. Arithmétique de Presburger, procédures de décision. Combinaison de procédures de décision pour les théories décidables.

3. Termes, réécriture, preuve de propriétés non arithmétiques. Le problème de la décision dans les théories équationnelles.

4. Fonctions récursives, mesures et preuves par induction. Ordres et principe d'induction bien fondée.

5. Le système de types de PVS: sous-typage, typage dépendant, types abstraits. Leur utilisation pour détecter des erreurs tôt dans le processus de spécification.

6. Spécification et preuves de programmes avec données. La méthode de renforcement systématique d'invariants. Abstractions et model-checking en PVS.

7. Séance de récapitulation. Présentation, choix, et aide initiale pour l'élaboration des projets.

4.6.3 Bibliographie

- Un livre de logique, par exemple: "Logique et fondements de l'informatique" de Richard Lassaigne et Michel de Rougemont. Hermes 1993.
- A Tutorial Introduction to PVS, http://www.csl.sri.com/papers/wift-tutorial/
- J.-F. Monin. Comprendre les méthodes formelles. Masson 1996.

4.7 TCPR : Programmation avancée

- **Intervenant** : Jean-Pierre Banâtre, professeur à l'IFSIC (Jean-Pierre.Banatre@irisa.fr)
- **Volume** : 18 heures (demi-module optionnel)

4.7.1 Présentation

Ce module a pour objet de faire le point sur certains aspects essentiels concernant la Programmation. Il sera constitué de trois interventions traitant des sujets suivants:

4.7.2 Plan du module

Cours 1 : aspects de la programmation fonctionnelle.

Cours 2 : construction de programmes séquentiels.

Cours 3 : aspects de la programmation parallèle.

4.8 MLTO : Méthode, langage et technologie objets

- **Intervenants** : Antoine Beugnard, enseignant-chercheur à l'ENST Bretagne (Antoine.Beugnard@enst-bretagne.fr), P. Geffroy, J. Le Drezen, (G. Nomitch), R. Ogor et R. Rannou.
- Module dispensé à Brest.
- **Volume** : 19h30

4.8.1 Présentation

Les objets sont omniprésents dans l'informatique d'aujourd'hui. Le concept n'est pas nouveau ; il est apparu dans les années 60. Ce module est l'occasion de faire le point sur leur utilisation en analyse, conception et réalisation. Une partie du module sera consacrée à la modélisation objet en s'appuyant sur la notation UML et des méthodes comme *fusion*, *syntropy* ou *catalysis*. Nous présenterons des moules de solutions appelés "Design Patterns". La troisième partie sera dédiée aux différents langages pour illustrer les concepts et les mécanismes objets.

4.8.2 Objectifs

Faire une synthèse sur les technologies objet:
- l'analyse et la conception de systèmes (modèles objets, démarche de modélisation, utilisation de l'héritage, des contraintes),
- la programmation et l'implantation de systèmes (limites, liens avec les types, patterns, separation of concern, meta-information),
- la mise en œuvre des langages objets (mise en œuvre de l'héritage, du typage, etc),
- aspects théoriques des langages objets.

4.8.3 Bibliographie

1. Abady, A Theory of objects, Springer, 1996

2. G. Castagna, Object-Oriented Programming: A Unified Foundation, Birkhäuser, 1997.

3. Desmond Francis D'Souza, Alan Cameron Wills, Objects, Components, and Frameworks With Uml : The Catalysis Approach, Addison-Wesley Object Technology Series, 1998.

3. Cook S. et Daniels J., Designing object systems, object-oriented modelling with Syntropy, Prentice Hall, 1994.

4. Coleman, Derek, Fusion : la méthode orientée-objet de 2e génération, 1996.

5. Ducourneau et all, Langages et Modèles à Objets, état des recherches et perspective, Collection Didactique, INRIA, 1999.

6. Gamma et all., Design Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

5 Cours-conférences

Les cours-conférence traitent pendant 3 heures d'un sujet important qui n'a pas déjà été couvert par un module. Ils s'adressent à tous les étudiants de DEA.

Voici quelques exemples de cours-conférences reliés directement à la filière 2. (N.B.: Le programme risque d'évoluer).

5.1 Sécurité et méthodes formelles

- **Intervenant** : Thomas Genet, maître de conférences IFSIC (Thomas.Genet@irisa.fr)
- **Durée** : 2 heures
- **Equipe de recherche** : Lande

Au travers d'exemples de systèmes sécurisés de la vie courante, (essentiellement la carte bancaire mais également des protocoles d'authentification, un protocole de paiement bancaire, le navigateur netscape, ...), le but de ce cours est de montrer

- qu'assurer la sécurité globale de tels systèmes est difficile
- qu'il existe des failles logicielles célèbres et astucieuses
- que dans le futur la complexité de ces systèmes sera très supérieure
- que les méthodes formelles sont un des rares moyens de garantir la sécurité des systèmes présents et futurs

5.2 Modèles de la sécurité informatique

- **Intervenant** : Thomas Jensen, chargé de recherche au CNRS (Thomas.Jensen@irisa.fr)
- **Durée** : 3 heures
- **Equipe de recherche** : Lande

Dans ce cours-conférence, nous présenterons les différents modèles de sécurité et leurs application dans le contrôle d'accès, le contrôle de flux d'informations etc.

6 Equipes de recherche de l'IRISA connectées à la filière "Méthodes formelles pour le logiciel"

ESPRESSO : Environnement de Spécification de Programmes RÉactifS SynchrOnes (/http://www.irisa.fr/espresso/espresso-francais.html)

Groupe GALION : Informatique théorique. Graphes et logique (http://www.irisa.fr/galion/)

LANDE : Logiciel : analyse et développement (http://www.irisa.fr/lande/)

S4 : Synthèse et supervision de systèmes, scénarios (http://www.irisa.fr/s4/fiche-projet-f.html)

TRISKELL : Construction fiable et efficace d'applications par assemblage de composants (http://www.irisa.fr/triskell/)

VERTECS : Modèles et techniques de VÉRification appliquées au TEst et au Contrôle de Systèmes réactifs (http://www.irisa.fr/vertecs/index.htm)

7 Sujets de stages

Les sujets de stage n'étant pas encore tous connus au moment de la réalisation de ce document, nous vous conseillons de vous référer à la page des stages du DEA: (http://www.irisa.fr/dea/sujetsStages/SUJETS.html)