ACI Sécurité ALIDECS:

Langages et Atelier Integrés pour le Développement de Composants Embarqués Sûrs

Réunion de démarrage
LIP6, 21 et 22 octobre 2004

Marc Pouzet

Page web

- `http://www-verimag.imag.fr/SYNCHRONE/alidecs/`

Partenaires du projet

- thème SPI, LIP6 (Marc Pouzet)
- Equipe Synchrone, VERIMAG (Florence Maraninchi)
- Projet Pop-Art, INRIA Rhône-Alpes (Pascal Fradet)
- Projet Mimosa, INRIA Sophia (Frédéric Boussinot)
- Equipe CMOS, Université Evry (Jean-Marc Delosme)

Présentation générale

Contexte

- systèmes embarqués critiques
- systèmes *dynamiques* et *concurrents* par nature
- la sûreté du logiciel est importante
- méthodes de description *et* de programmation formels
- avoir des *garanties* sur le comportement à l'exécution

Comment traiter:

- systèmes embarqués de grande taille
- la réutilisation de code devient un pb. crucial

Alidecs

Propoposer un atelier de développement intégré

- fondé sur un modèle sémantique solide (temps, concurrence)
- construction modulaire de systèmes de grande taille
- spécifier, programmer, réutiliser des *composants*
- mécanisme de composition/encapsulation des programmes *et* des spécifications
- outils de validation statique (e.g., typage, causalité)
- outils de simulation des programmes *et* des spécifications
- outils de compilation

Notre approche

Une approche “langage”

- Domain Specific Language (DSL)
- offrir les bonnes constructions de programmation
- adapté au domaine (temps + concurrence)
- limiter le pouvoir expressif (obtenir quelque chose en échange)
- analyse et compilation dédiés

Basée sur la notion de “composant”

- du code + une interface + une opération de composition
- définir formellement

Quoi de neuf sous le soleil?

Conception orientée "composant"

- approche Middleware (CORBA, Fractal, etc.)
- mécanisme d'exécution peu adaptés à l'embarqué critique
- pas de sémantique formelle, pas de garantie *avant* l'exécution

Langages d'usage général

- mécanismes d'encapsulation (modules, objets, etc.)
- formellement définis
- ne permettent pas de décrire des prop/prog. dynamiques et concurrents
- mécanismes de compilation inadaptés aux systèmes embarqués critiques (e.g., GC)

Outils formels spécialisés

- spécifier des prop. temporelles, outils de composition (e.g., "Reactive Modules")
- on ne peut écrire ni compiler des programmes
- on ne peut pas simuler

Outils formels généralistes

- Méthode B, Larch Prover, Coq, etc.
- mélanger programmes et propriétés
- prouver formellement
- mais pas de modèle natif du temps et de la concurrence
- donc pas d'outils d'analyse, de compilation, de simulation dédiés

Outils de l'ingénieur

- UML, (Mathlab) Simulink/StateFlow
- outils de simulation, modélisation *vs* programmation
- pas (ou trop?) de sémantique formelle
- pas de compilation
- peu d'analyses statiques

Programmation Synchrone

- Domain Specific Languages: Lustre, etc.
- sémantique du temps et de la concurrence
- outils de compilation, de test, de simulation
- mécanismes d'abstraction/encapsulation limités
- pas de spécification de composants

Difficultés (verrous technologiques)

Concevoir un environnement intégré, fondé sur un modèle synchrone de la concurrence et du temps pour la construction modulaire de systèmes embarqués sûrs de grande taille

- spécifier un composant
- validation statique et modulaire
- exécution précoce et simulation
- techniques de compilation

Spécifier un composant

- définir précisément ce qu'est un *composant réactif*
- du code + une interface (propriétés) + une opération de composition
- décrire des signatures complexes de manière uniforme
  - propriétés statiques: types, causalité, etc.
  - propriétés dynamiques: horloges, etc.
  - propriétés calculatoires (combinatoires ou temporelles)
- pouvoir décrire *en même temps* le controleur (un programme) et le controlé (un programme non déterministe)
- quelle est la bonne opération de composition?
- notion de *contrat temporel logique*?
- quid des formalismes fondés sur la théorie des types?

Validation statique et modulaire

- donner des *garanties* à la compilation
- exécution en temps et mémoire bornés
- absence de certaines erreurs (e.g., typage, dead-lock)
- programmation par aspect: instrumenter (tisser) le code automatiquement pour assurer une propriété à l'exécution

Exécution précoce et simulation

- exécuter les programmes incomplets, les programmes, les spécifications
- simuler une propriété dès le début (au début, un composant peut n'être défini que par une propriété)
- composition de composants (mécanisme d'*interprétation* à la Boussinot?)

Techniques de compilation

- compilation de programmes de grande taille
- compilation pour le software (*vs* hardware)
- compilation modulaire
- quid de la causalité?

Objectif du projet

- une notion de *composant réactif* pour l'embarqué
- prenant en compte le *temps* et la concurrence
- un composant = un corps et une spécification
- constructions de langage adaptés permettant de les décrire (e.g., objets, modules?)
- définition et implantation d'analyses statiques et modulaires
- définition et implantation de techniques de génération de code

Notre expérience

- Domain-Specific Languages: bon choix des constructions de programme, techniques de compilation, création dynamique...
- techniques de vérification (model-checking, Abstract Int., Automatic Testing)
- simulation et outils de mise au point (debugging)
- programmation par aspects (e.g., garantir des propriétés non fonctionnelles)
- run-time asynchrones (GALS)
- notion de contrats temporels logiques
- une notion de composant de haut niveau
- outils fondés sur la théorie des types