Administration basée MDE de composants logiciels

Franck Barbier

Université de Pau et des Pays de l'Adour

PauWare Research Group

Administration

- L'administration ou management concerne l'observation (supervision) et le contrôle (monitoring) d'« entités »
- Son développement principal est le monde des réseaux : norme GDMO (*Guidelines for the Definition of Managed Objects*), protocoles spécialisés (CMIP, SNMP…)
- L'approche objet puis l'approche composant/middleware autorise maintenant des approches canoniques illustrées par des technologies comme JMX (*Java Management eXtensions*) et son concept associé de *MBean* ou *Managed Bean*

Administration de logiciel

- L'administration de composants logiciels vise à « instrumenter » ces derniers pour gérer leurs fautes, les (re)-configurer dynamiquement si besoin et plus généralement les visualiser et mieux les comprendre
- L'activité d'administration en développement touche ainsi l'évaluation en général, le test et la prédiction de comportement des composants individuellement, ainsi que ceux des assemblages
- En déploiement, elle offre un véritable cadre de pilotage associée notamment à de technologies comme *BIT* (*Built-In Test*). Son coût peut cependant être dans certains cas prohibitif !

Idées d'application

- Mise au point plus fine de programmes en condition de déploiement : *code tracing/tracking*… (ex. comportement de décodeurs MPEG en mode *streaming*, société Philips Semiconductors)
- Maintenance en ligne et à distance de composants et services distribués avec ou sans fil (cf. enquête Nouvel Observateur, société Nova, « 20% versus 5% »)
- Diagnostic en QoS : détection de goulots d'étranglement, équilibrage de charge (middleware…)

Le MDE dans tout cela ?

Offrir une infrastructure d'administration ne donne pas les techniques effectives qui vont permettre de réaliser cette administration. Idée : rendre les composants logiciels intelligibles en les calquant sur les modèles mais aussi rendre ces modèles persistants à l'exécution (développement et/ou déploiement) -> exécutabilité

Parallèle avec le *model checking* :

- Pas de vérification *a priori* tel que classiquement fait avec les méthodes formelles ou des méthodologies comme Neptune pour UML
- L'interface d'administration (voir démo. ci-après) rend les modèles tangibles, vivants… La vérification reste intuitive et expérimentale mais support pour génération de scénarii de test avec niveau de couverture de la vérification lié à l'utilisateur…

Environnement retenu

Cible :

- *UML State Machine Diagrams* (version augmentée et « sûre »)
- *UML Sequence Diagrams* (version épurée de *alt*, *opt*… notamment)

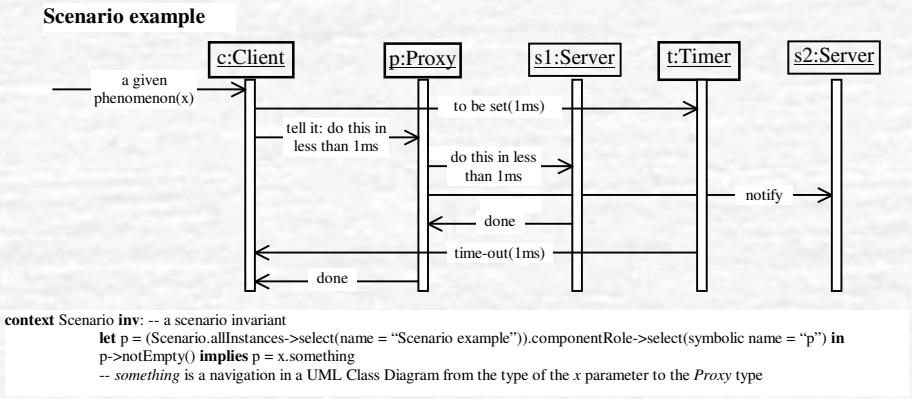
Technique de modélisation/implémentation :

- 1 composant logiciel <-> 1 statechart
- Assemblages décrits par scénarii

Outils :

- Librairie Java *PauWare.Statecharts* (moteur Statecharts) avec mode BIT optionnel (sous-librairie) et support : JMX simple, EJB/RMI et EJB/JMS (*Java Message Service*)
- Extension *PauWare.Velcro* (systèmes mobiles et sans fil) avec support WMA (*Wireless Message API*) mais problème de réflexion Java résiduel

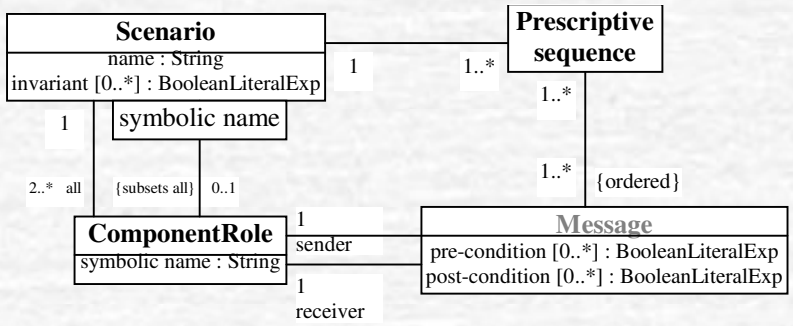
Eléments techniques, *UML Sequence Diagrams* (1/2)

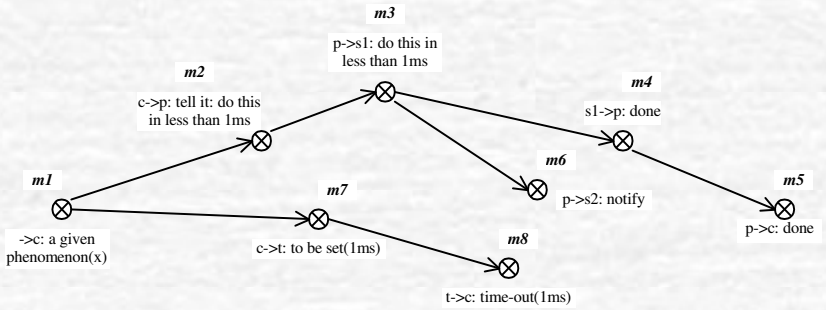


Réf. bibliographiques (parmi d'autres) :

- T. Ziadi, L. Hélouët and J.-M. Jézéquel, *Revisiting Statecharts Synthesis with an Algebraic Approach*, ICSE'2004
- F. Barbier and O. Constant, *Executable Component-Based Scenarios*, ICSE'2005/"Scenarios and State Machines: Models, Algorithmes and Tools" workshop

Eléments techniques, *UML Sequence Diagrams* (2/2)

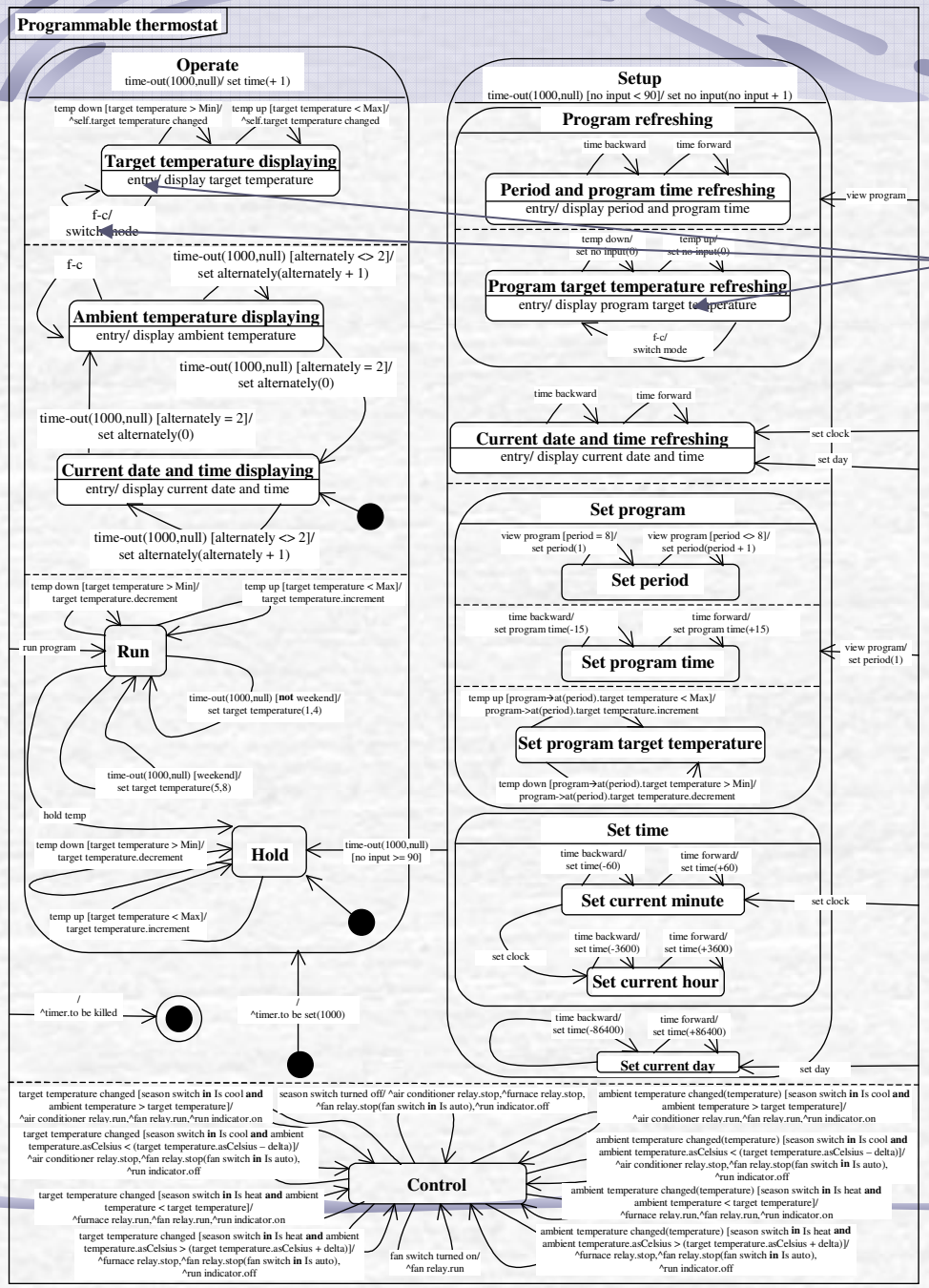




Eléments techniques, *UML State Machine Diagrams* (1/2)

- Mode d'exécution des modèles *run-to-completion* :
  - Gestion de la réentrance (ex. : un composant s'envoie un message à lui-même, états concurrents) : cycle ininterruptible avec nécessité de prise en compte du message
  - Compatibilité avec modèles de composants technologiques (ex. gestion des transactions en EJB), compatibilité avec Fractal ?
- Formalisation directe et intuitive de la configuration : *public void reset() {to_state("Operate");}*
- *Broadcast* et *multicast* supportés
- Partie systèmes mobiles et sans fil : approche par cache (réplication) du composant logiciel (*i.e.*, automate) sur le tiers « management » : communication intensive entre composant embarqué et composant image, simulation possible de services locaux (*e.g.*, *timer event services*)

Eléments techniques, *UML State Machine Diagrams* (2/2)



Event

State-based programming support

```
public void f_c() throws Statechart_exception {

_Programmable_thermostat.fires(_Ambient_temperature_displaying,_Ambient_temperature_displaying);

_Programmable_thermostat.fires(_Target_temperature_displaying,_Target_temperature_displaying,true,this,"switch_mode");

_Programmable_thermostat.fires(_Program_target_temperature_refreshing,_Program_target_temperature_refreshing,true,this,"switch_mode");

    _Programmable_thermostat.run_to_completion();

}
```

UML-2 *run-to-completion* model execution mode

Perspectives

- Transformateur XMI 2 -> *PauWare.Statecharts*
- Transformateur *PauWare.Statecharts* -> XMI 2 (prototypage rapide, documentation…)
- *UML 2 Sequence Diagrams* (terrain hautement instable, voir release de fin 2004)
- Composabilité basée état
- Philosophie *Open Source* : pourparlers avec www.fujada.de

Démo.

Questions en suivant ?