Université de Bretagne Occidentale
UFR Sciences
Département d'Informatique

Master1 TIIL
Année 2016-2017

Projet de Compilation

Analyse d'un langage de gestion de scenarii domotiques (DOMUS) et production de code pour le simulateur SiDo

A rendre pour le : vendredi 9 décembre 2016, 18h.

Le but du projet est d'écrire un compilateur de programmes écrits en langage DOMUS.

DOMUS est un langage (inventé pour les besoins de ce projet et décrit en fin de document) permettant de déclarer différents types de matériels, équipements ou automatismes de la maison, de spécifier des scenarii domotiques sur ces équipements, puis d'en programmer l'exécution via des évènements sur des interfaces (interrupteurs, télécommandes, téléphones...) ou de manière automatique (ponctuelle ou périodique).

Le compilateur réalisé doit effectuer les analyses lexicales, syntaxiques et sémantiques du programme DOMUS en entrée, en produire un programme java pour exécution dans le simulateur SiDo (démonstration en séance), et générer un rapport synthétique sur l'ensemble du processus.

Toutes les ressources utiles pour le projet sont disponibles dans : /home/commun_depinfo/enseignants/vnicolas/M1_TIIL_Projet_Compil

- **Documents électroniques à rendre :**
  - Le code de votre réalisation, commenté (fichiers lex et cup, plus d'autres fichiers si nécessaire).
  - Un document textuel (max 4 pages) spécifiant précisément les parties du projet que vous avez réalisées (complètement ou partiellement... préciser), expliquant vos choix de conception ou d'implantation, avec votre grammaire Cup (sans les actions java).
  - Les trois rapports de compilation et programmes java obtenus pour les trois programmes de test donnés lors du projet.
  - Et éventuellement les rapport de compilation et programmes java obtenus pour un programme de test de votre choix permettant de montrer les aspects les plus importants de votre réalisation.

• **Modalités pour rendre les documents :**

Les différents fichiers à rendre sont à mettre dans un répertoire intitulé par vos noms de login, et à zipper dans un fichier vos_login.zip.

Ce fichier zip est ensuite à m'envoyer par email à vnicolas@univ-brest.fr (en demandant un accusé de réception).

Date limite de dépôt (après laquelle sera appliqué un point de pénalité par jour de retard) : vendredi 9 décembre 2016, 18h.

• **Travail à réaliser :**

Il est recommandé de suivre les étapes suivantes pour terminer le projet dans les temps :

1. **Première étape : Analyse lexicale du langage DOMUS.**
2. **Deuxième étape : Analyse syntaxique du langage DOMUS**
   - commencer par l'analyse syntaxique de DOMUS privé des instructions *si* et *pourtout*,
   - ajouter ensuite l'analyse syntaxique de l'instruction *si*,
   - puis ajouter l'analyse syntaxique de l'instruction *pourtout*,
   - ajouter enfin l'analyse syntaxique des ensembles d'appareils et traiter l'inclusion des scenarii via *executer_scenario*.
3. **Troisième étape : Analyse sémantique du langage DOMUS**

   Vérification de propriétés sémantiques générales du programme :
   - *Ident_unique* : tout identificateur est unique dans un programme et différent des mots-clés du langage,
   - *Format_Date* : toute date apparaissant dans une commande de programmation de scenarii respecte les contraintes inhérentes aux dates (numéro de mois entre 1 et 12, année sur 4 chiffres, nombre de jours en fonction du mois, etc.),
   - *Action1* : une action ne s'applique qu'à un appareil préalablement déclaré,
   - *Action2* : une action est applicable à un appareil si et seulement si elle fait partie des actions autorisées sur le type d'appareil concerné,
   - *Comm* : une commande ne s'applique qu'à des interfaces et des scenarii préalablement déclarés dans le programme ;
4. **Quatrième étape : Production du code Java pour le code DOMUS en entrée.**

   Le code est produit si et seulement si les erreurs lexicales, syntaxiques et sémantiques ne sont pas bloquantes, et est sous la forme de deux fichiers *HabitatSpecific.java* et *CMaisonUser.java*.
5. **Dernière étape : Génération du rapport/bilan du processus de compilation.**

   Ce rapport, généré si les erreurs lexicales, syntaxiques et sémantiques ne sont pas bloquantes, contiendra au minimum :
   - un résumé du programme domotique - à savoir le nombre d'appareils déclarés, le nombre d'interfaces déclarées, le nombre et le nom des scenarii déclarés, les associations interfaces/scenarii, et le nom des scenarii programmés,
   - un bilan de l'analyse sémantique - pour chaque propriété, on notera si le programme la vérifie, et si non on indiquera à quels endroits sont les problèmes.

Le langage DOMUS

** Structure d'un programme en langage DOMUS :*

```
<PROGRAMME_DOMUS>
// Commentaire : déclaration des appareils
<DECLARATION_APPAREILS>
    eclairage e1, e2, e3.
    volet v1, v2, v3, v4.
    chauffage rad1.
    alarme a1.
    fenetre fen.
    autre_appareil(cafetiere) cafe.
    autre_appareil(hifi) hf.

    // déclaration d'un ensemble d'appareils (type énumeré utilisateur)
    definir mon_eclairage_salon = {e2,e3}.
</DECLARATION_APPAREILS>

// déclaration des interfaces
<DECLARATION_INTERFACES>
    interrupteur b1, b2.
    mobile t1.
    telecommande c1.
</DECLARATION_INTERFACES>

// déclaration des scenarii
<DECLARATION_SCENARII>
   // déclaration du scenario bonjour
   <SCENARIO bonjour>
      pourtout v:volet faire v.ouvrir; fait;
        a1.eteindre;
        cafe.allumer;
        si (rad1.etat == eteint) alors rad1.allumer; fsi;
        hf.allumer;
   </SCENARIO bonjour>

   <SCENARIO soiree>
        pourtout v:volet faire v.fermer; fait;
        pourtout e:mon_eclairage_salon faire e.allumer; fait;
   </SCENARIO soiree>
```

```
   <SCENARIO soiree_musique>
        executer_scenario soiree;
        pourtout e:autre_appareil(hifi) faire e.allumer; fait;
   </SCENARIO soiree_musique>

   <SCENARIO depart>
        message("Scenario Départ");
        pourtout v:volet faire v.fermer; fait;
        pourtout e:eclairage faire e.eteindre; fait;
        pourtout x:autre_appareil faire x.eteindre; fait;
        si (fen.etat == ferme) alors a1.allumer;
        sinon message("Attention : la fenêtre ",fen," est ouverte !"); fsi;
   </SCENARIO depart>

   <SCENARIO noel1>
        e1.allumer;
   </SCENARIO noel1>

   <SCENARIO noel2>
        e2.allumer;
   </SCENARIO noel2>

   <SCENARIO noel3>
        e3.allumer;
   </SCENARIO noel3>
</DECLARATION_SCENARII>

// déclaration des commandes
<DECLARATION_COMMANDES>
   associer b1 = bonjour.
   associer b2 = depart.
   associer t1 = depart.
   associer c1 = {noel1,noel2,noel3}.

// le scenario soirée est lancé tous les soirs du 1er jour de chaque mois 2012 à 18h.
   programmer soiree = (2012,_,1,18,0).
   programmer soiree_musique = {(2012,11,20,19,30),(2012,11,21,19,30)}.
</DECLARATION_COMMANDES>
</PROGRAMME_DOMUS>
```

** Description des différents types d'appareils, et actions associées, manipulés dans le langage DOMUS :*

- éclairage : état (allumé, éteint, demi), allumer, éteindre, tamiser
- alarme : état (allumé, éteint, demi), allumer, allumer_partiel, éteindre
- chauffage : état (allumé, éteint, éco), allumer, allumer_éco, éteindre
- fenêtre : état (ouvert, fermé, demi), ouvrir, ouvrir_partiel, fermer, fermer_partiel
- volet : état (ouvert, fermé, demi), ouvrir, ouvrir_partiel, fermer, fermer_partiel
- autre_appareil : état (allumé, éteint), allumer, éteindre (tv, hifi, cafetière, etc.)

On peut également déclarer des interfaces de commande à distance des scenarii de différents types (interrupteur, mobile, téléphone, télécommande, tablette, etc.) et leur associer des scenarii dans les commandes.

Les mots-clés du langage sont : PROGRAMME_DOMUS, DECLARATION_APPAREILS, DECLARATION_INTERFACES, DECLARATION_SCENARII, DECLARATION_COMMANDES, SCENARIO, definir, executer_scenario, associer, programmer, message, pourtout, faire, fait, si, alors, sinon, fsi, ouvrir, fermer, eteindre, allumer, tamiser, etat, allumer_partiel, allumer_eco, ouvrir_partiel, fermer_partiel, allume, eteint, demi, eco, ouvert, ferme, eclairage, volet, chauffage, alarme, fenetre, autre_appareil, interrupteur, mobile, telephone, telecommande, tablette, tv, hifi, cafetiere, video_proj, lave_vaisselle, lave_linge, seche_linge, ordinateur, portail.

Le langage permet d'insérer des commentaires après le symbole //.

Utilisation du simulateur SiDo

Pour tester vos programmes générés (CMaisonUser.java et HabitatSpecific.java), vous pouvez les visualiser et les manipuler avec l'interface SiDo.

Pour cela, recopier dans votre espace le répertoire /home/commun_depinfo/enseignants/vnicolas/M1_TIIL_Projet_Compil/test

Dans ce répertoire se trouvent :
- le répertoire SiDoInit (version d'initialisation du simulateur, à ne pas modifier)
- le script initSiDo.sh (à lancer pour exécuteur le simulateur)
- le répertoire code_exemple_projet qui contient les fichiers CMaisonUser.java et HabitatSpecific.java générés pour l'exemple du sujet
- le répertoire code_ex1, vide, destiné à recevoir les fichiers CMaisonUser.java et HabitatSpecific.java que votre programme génère pour l'exemple ex1.

Vous pouvez créer dans le répertoire test autant de répertoires que vous avez d'exemples à traiter, à l'image du répertoire code_ex1.

L'appel du script initSiDo.sh se fait avec trois arguments :
- en 1er argument : le chemin du dossier où se trouve SiDoInit
- en 2ème argument : le nom du dossier (au même niveau que SiDoInit) où se trouvent les versions de CMaisonUser.java et HabitatSpecific.java à utiliser
- en 3ème argument : le nom du dossier résultat (mis dans le même dossier que SiDoInit), contenant une version du simulateur configurée avec les fichiers CMaisonUser.java et HabitatSpecific.java du 2ème argument.

Exemple, si on se trouve dans le répertoire test :

```
./initSiDo.sh . code_exemple_projet SiDoExemple
```