Examen de 2ème session - Juin 2016 - Durée 2h

Cet énoncé comporte 6 pages.

Sans documents. Tout les équipements électroniques (téléphones, montres intelligentes, etc) doivent rester dans le sac et hors de portée. Pas d'échanges (stylo, effaceur, etc) durant l'examen.

Quelques rappels de XText et XTend sont donnés en Annexes 3 et 4.

1. DSL de logs

On désire proposer un DSL permettant de stocker des logs Web comme présenté dans la figure 1. Les règles suivantes doivent être respectées :

- les entrées de log ont lieu à des moments indiqués sous la forme année/mois/jour-heures :minutes :secondes
- les entrées se succèdent dans l'ordre chronologique
- l'utilisateur qui fait une action enregistrée est identifié
- les actions enregistrées sont soit des demandes de pages Web (fichiers .html), soit des appels de code (fichiers .jsp). Dans le second cas il est possible d'avoir des paramètres
- on enregistre pour chaque action le retour du serveur (OK ou KO)

QUESTIONS

- donner le fichier de grammaire de DSL au format XText correspondant au DSL souhaité. Ce DSL doit accepter l'exemple de la figure 1.
- donner le fichier XTend de validation permettant de vérifier les règles qui ne sont pas vérifiées directement par la grammaire XText

2. Génération de diagrammes de séquences

On souhaite générer des pages Web contenant des diagrammes de séquence à partir des fichiers de logs. On doit générer une page par utilisateur référencé dans le fichier de log et une page d'index (qui pointe sur les autres pages). Concernant les fichiers pour les utilisateurs, un fichier pour l'utilisateur $uuid_x$ ne doit contenir que les entrées relatives à l'utilisateur $uuid_x$. Le diagramme de séquence dans une page est généré par utilisation de Plant UML. Cela se fait comme indiqué dans la figure 2 qui donne le code HTML de la page pour uuid1.

QUESTIONS

- donner le fichier XTend permettant de générer les fichiers souhaités à partir d'un fichier dans le DSL ci-dessus.

Les fichiers résultants sont présentés dans la figure 3

```
à 2016/06/27-17:31:39
utilisateur uuid1 demande "index.html"
réponse OK
à 2016/06/27-17:32:10
utilisateur uuid2 demande "index.html"
réponse OK
à 2016/06/27-17:33:15
utilisateur uuid1 demande "register.html"
réponse OK
à 2016/06/27-17:34:10
utilisateur uuid1 appelle "register.jsp"
paramètres id="alice" et nom="Arbre" et prenom="Alice" et motpasse="alice1234"
réponse OK
à 2016/06/27-18:31:39
utilisateur uuid2 demande "login.html"
réponse OK
à 2016/06/27-18:40:00
utilisateur uuid2 appelle "login.jsp"
paramètres id="bob" et motpasse="bob4321"
réponse KO
à 2016/06/27-19:00:00
utilisateur uuid1 demande "login.html"
réponse OK
à 2016/06/27-19:40:01
utilisateur uuid1 appelle "login.jsp"
paramètres id="alice" et motpasse="alice1234"
réponse OK
à 2016/06/27-19:40:02
utilisateur uuid2 appelle "login.jsp"
paramètres id="bob" et motpasse="bob1234"
réponse OK
à 2016/06/27-20:00:00
utilisateur uuid2 demande "logout.html"
réponse OK
à 2016/06/27-20:00:15
utilisateur uuid2 appelle "logout.jsp"
réponse OK
```

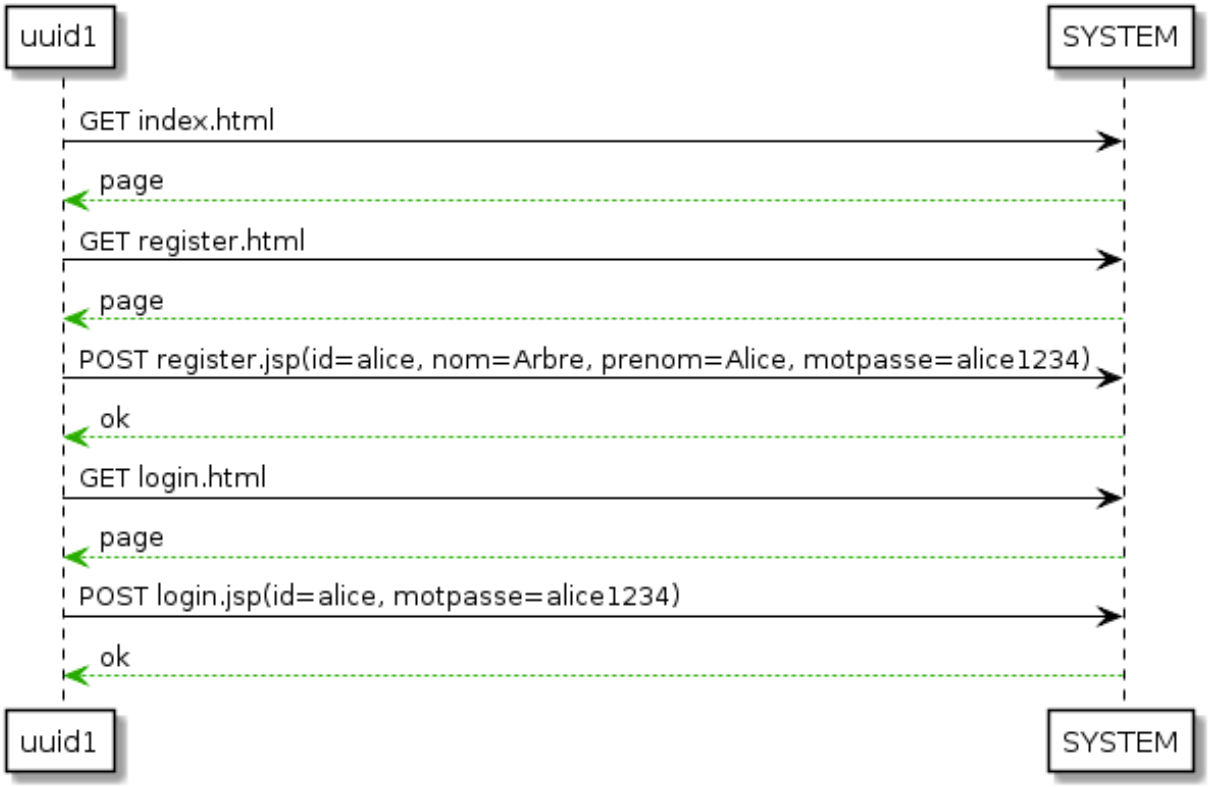
FIGURE1 – Fichier de log écrit dans le DSL à concevoir.

```
<HTML>
<HEAD>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript" src="jquery_plantuml.js"></script>
<TITLE>uuid2</TITLE>
</HEAD>
<BODY>
<H1>uuid2</H1>
<IMG UML="@startuml
skinparam sequence {
ArrowColor Black
ActorBorderColor Black
LifeLineBorderColor Black
LifeLineBackgroundColor Black
ParticipantBorderColor Black
ParticipantBackgroundColor White
ParticipantFontColor Black
}

uuid2 -> SYSTEM : GET index.html
SYSTEM -[#00AA00]-> uuid2 : page
uuid2 -> SYSTEM : GET login.html
SYSTEM -[#00AA00]-> uuid2 : page
uuid2 -> SYSTEM : POST login.jsp(id=bob, motpasse=bob4321)
SYSTEM -[#AA0000]-> uuid2 : error
uuid2 -> SYSTEM : POST login.jsp(id=bob, motpasse=bob1234)
SYSTEM -[#00AA00]-> uuid2 : ok
uuid2 -> SYSTEM : GET logout.html
SYSTEM -[#00AA00]-> uuid2 : page
uuid2 -> SYSTEM : POST logout.jsp()
SYSTEM -[#00AA00]-> uuid2 : ok
@enduml
">
</BODY>
</HTML>
```

FIGURE2 – Code de la page HTML pour l'utilisateur uuid1 (uuid1.html).

uuid1



uuid2

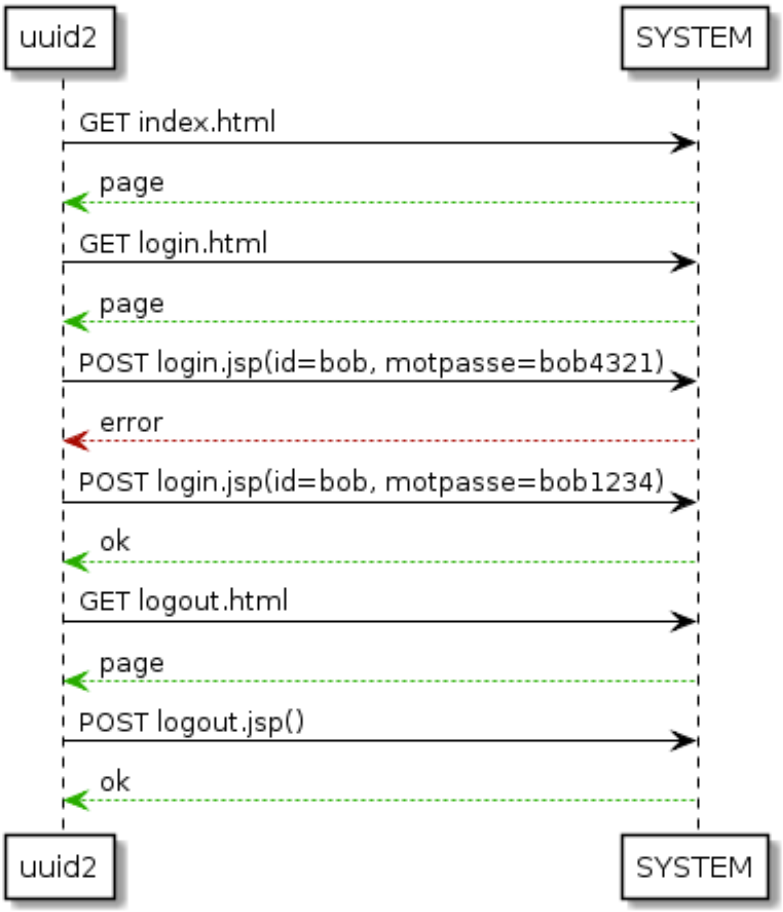


FIGURE3 – Résultats pour l'utilisateur uuid1 (uuid1.html) et pour l'utilisateur uuid2 (uuid2.html).

3. RAPPELS – Énumérations et options avec XText et XTend

Un **choix énuméré entre plusieurs modes** peut s'écrire en XText comme suit.

```
Definition :
        mode=Mode (estOption='option')? name=ID
;

enum Mode :
        MODE1='m1' | MODE2='m2' | MODE3='m3'
;
```

L'effet est de créér, dans le code `Java` généré à partir de la grammaire, une énumération `Mode`. Il sera ensuite possible en XTend, étant donné une variable `v` de type `Definition` de vérifier par exemple si `v.mode == Mode.MODE1`.

On note aussi ici la possibilité de définir un **élément optionnel** (ex : `m1 option Foo` *vs* `m2 Foo`). Pour savoir en XTend si l'on est dans un cas ou dans l'autre, on peut tester si `v.estOption==null` ou pas.

4. RAPPELS – Éléments d'XTend

Le **transtypage** (casting) en XTend ne se fait pas comme en `Java`. Soit la grammaire XText suivante, et en XTend une variable `m` de type `M`.

```
M : M1 | M2 ;
M1 : attr1=ID attr2=ID ;
M2 : attr1=ID attr3=ID ;
```

Il est possible de faire `m.attr1` car `attr1` est défini dans `M1` et `M2`, et donc "présent" dans `M`. Par contre `m.attr2` (indéfini) et `((M1)m).attr2` (erreur de syntaxe) sont incorrects. En XTend on écrira `(m as M1).attr2`.

XTend propose aussi des **constructions très riches pour les collections**. Soit `promo` une variable contenant une collection `etudiants` de `Etudiant`, avec `name` et `age` des attributs de `Etudiant` (voir ci-dessous la grammaire). Alors on a :

```
promo.etudiants                                  // étudiants
promo.etudiants.map[name]                        // noms des étudiants
promo.etudiants.map[name].join(", ")             // String des noms séparés par des virgules
promo.etudiants.filter[age>=18]                  // étudiants majeurs
promo.etudiants.filter[age>=18].size             // nombre d'étudiants majeurs
promo.etudiants.filter[it.addresse.ville="Paris"].size  // nombre d'étudiants vivant à Paris
if(promo.etudiants.exists[age<18]) { ... }       // s'il existe au moins un étudiant mineur ...
```

XTend dispose de plusieurs méthodes pour construire des collections à partir d'une itération :

```
promo.etudiants.map[name]          // noms des étudiants (itérateur)
promo.etudiants.map[name].toList   // la liste qui correspond
promo.etudiants.map[name].toSet    // l'ensemble qui correspond
promo.etudiants.toMap[name]        // le tableau associatif Map<String,Etudiant>, it.name -> it
```

Enfin, un dernier exemple. Soit la grammaire :

```
Promotion : name=ID '=' etudiants+=Etudiant (',' etudiants+=Etudiant)*;
Etudiant : name=ID ':' age:INT
```

Soit le modèle suivant :

```
MIAGE = Alice:18, Bob:16, Charles:20, Daniel:21
```

Pour générer ceci (les majeurs) :

```
MIAGE {
        - Alice a 18 ans;
        - Charles a 20 ans;
        - Daniel a 21 ans
}
```

on peut écrire (remplacer << par « et >> par ») :

```
def doGenerate(Promo p) '''
        <<p.name>> {
        <<p.etudiants.filter[age>=18].map[doGenerateEtudiant].join(";\n")>>
        }
'''
def doGenerateEtudiant(Etudiant e) '''
        - <<e.name>> a <<e.age>> ans
'''
```

Bien sur, on peut aussi le faire (modulo le ; sur la dernière personne) avec les instructions de contrôle de XTend (IF ... ENDIF et FOR ... ENDFOR) et la notion de receveur implicite (a.doGenerateA == doGenerateA(a)) :

```
def doGenerate(Promo p) '''
        <<p.name>> {
        <<FOR etudiant : p.etudiants>>
        <<IF etudiant.age>=18>>
                - <<etudiant.doGenerateEtudiant>> ;
        <<ENDIF>>
        <<ENDFOR>>
        }
'''
```

ou bien encore d'autres façons.