Description du Schéma StateChartSchema.xsd

Antoine Orgerit

Ce document vise à décrire le schéma XSD développé spécifiquement pour pouvoir utiliser des fichiers XML en entrée du comportement StateChartBehaviour de JADE. Tout manquement de conformité avec ce schéma n'entraînera pas le lancement de l'agent ou sa continuité dans le cadre d'une mise à jour du fichier XML d'entrée. Il convient donc de respecter ce dernier en rapport avec les différentes fonctionnalités proposées.

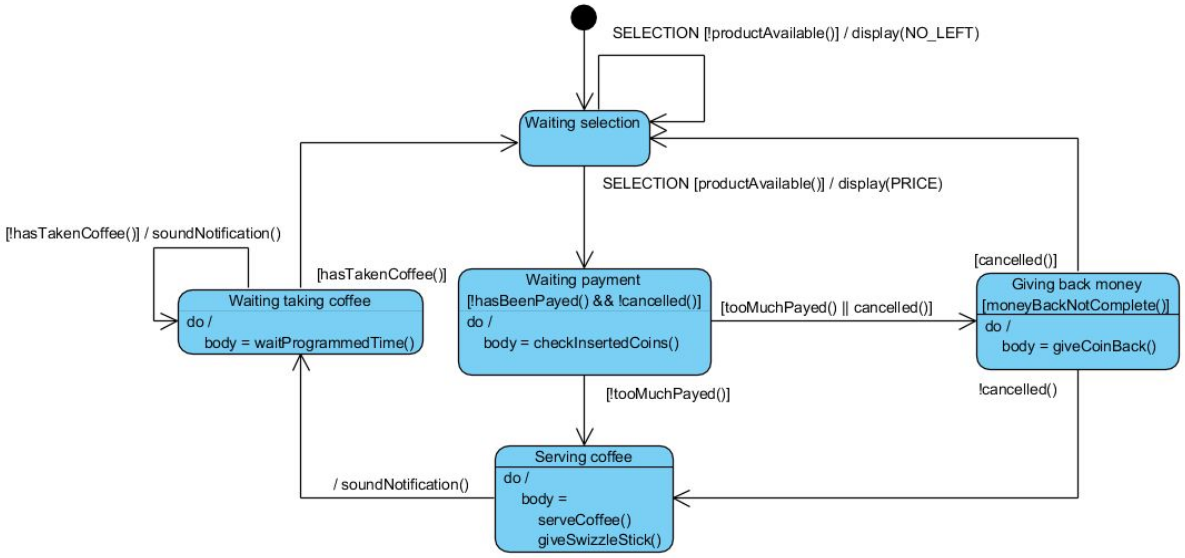
1. Vérification du document XML

Il existe de nombreuses manières de vérifier un document XML par rapport à un schéma XSD. Une grande majorité des éditeurs de documents XML supportent cette fonctionnalité en indiquant par exemple la nature du ou des erreur(s) de vérification. Il est donc recommandé de se munir de ce type d'éditeur afin de valider le document par rapport au schéma avant son utilisation dans le programme.

À titre informatif, l'éditeur XML utilisé durant le développement du schéma a été XML Exchanger Editor.

2. Exemple utilisé dans cette documentation

Pour appuyer les différents éléments présentés dans cette documentation, un exemple global est proposé en fin de documentation. L'exemple proposé est celui d'un distributeur de café, représenté de manière simplifiée de la façon suivante :



3. Description des composants du schéma

L'ensemble de cette description est proposée de manière plus formelle dans le fichier StateChartSchema.xsd.html disponible avec cette documentation.

Le schéma StateChartSchema.xsd repose sur un élément racine *Models*, permettant ainsi de définir plusieurs automates à utiliser par l'application le cas échéant. Chaque *Model* le composant représente ainsi un comportement particulier pouvant être adapté par un agent.

```
<Models>
        <Model ...>
                ...
        </Model>
        ...
</Models>
```

3.1. Élément *Model*

L'élément *Model* du schéma impose à ce dernier de posséder un nom sous l'attribut *name*. Cet attribut est utilisé afin d'aller rechercher le modèle spécifié à un agent lors de son lancement.

Cet élément contient également les composants suivants :

- un élément *InitialState* obligatoire, permettant de spécifier l'état initial du modèle à l'aide du nom d'état qu'il contient ;
- un ou plusieurs élément *State* développés ci-après dans le document, représentant les différents états du modèle sans limitation de leur nombre maximum.

```
<Model name="...">
        <InitialState>...</InitialState>
        <State name="...">
                ...
        </State>
        ...
</Model>
```

3.2. Élément *State*

L'élément *State* du schéma représente un état possible du modèle dans lequel il est contenu. Il est caractérisé par les composants suivants :

- un élément *Invariant* facultatif du type complexe *InvariantType*, défini ci-après dans le document et représentant un ensemble de conditions à respecter pour boucler l'exécution du contenu de l'état. En cas d'absence de cet invariant, le contenu sera exécuté seulement une fois ;
- un élément *Content* facultatif représentant le contenu de l'état à exécuter et constitué d'un ensemble de fonctions *Function* définies ci-après dans le document ;
- un élément *Transitions* obligatoire, représentant un ensemble de transitions *Transition* possibles à partir de l'état. Au moins une transition doit y être spécifiée pour éviter une exécution en boucle du comportement par manque de transition.

```
<State name="...">
        <Invariant>
                ...
        </Invariant>
        <Content>
                <Function>
                        ...
                </Function>
                ...
        </Content>
        <Transitions>
                <Transition ...>
                        ...
                </Transition>
                ...
        </Transitions>
</State>
```

3.3. Les types d'invariant

3.3.1. Le type *InvariantType*

Le type *InvariantType* est un type spécifique utilisé pour représenter un conditionnement logique composé d'un élément. Chaque élément utilisant ce type doit contenir obligatoirement un seul des éléments possibles suivants :

- l'élément *AND* représentant un conditionnement logique && du type *BinaryInvariantType* précisé ci-après dans le document ;
- l'élément OR représentant un conditionnement logique || du type *BinaryInvariantType* précisé ci-après dans le document ;
- l'élément NOT représentation la négation d'un élément de conditionnement logique du type *InvariantType* qu'il doit contenir ;
- l'élément *Function*, défini ci-après dans le document, représentant une fonction à appeler auprès de l'agent utilisant le comportement défini et devant dans ce cas renvoyer une valeurs booléenne.

3.3.2. Le type *BinaryInvariantType*

Le type *BinaryInvariantType* est un type spécifique utilisé pour représenter un conditionnement logique composé de deux éléments fils. Chaque élément utilisant ce type doit contenir obligatoirement deux éléments pouvant être du même type que ceux proposés pour les éléments fils possible du type *InvariantType* (voir ci-dessus).

3.3.3. Exemples

```
<Function>...</Function>
```

```
<NOT>
        <Function>...</Function>
</NOT>
```

```
<AND>
        <NOT>
                <Function>...</Function>
        </NOT>
        <Function>...</Function>
</AND>
```

```
<OR>
        <Function>...</Function>
        <AND>
                <Function>...</Function>
                <Function>...</Function>
        </AND>
</OR>
```

3.4. L'élément *Function*

L'élément *Function* du schéma permet de définir une fonction à appeler auprès de l'agent utilisant le comportement et le modèle défini. Une fonction est caractérisée par :

- un élément *FunctionName* obligatoire, représentant le nom de la fonction à appeler en tant que chaîne de caractères ;
- un élément *FunctionParameters* facultatif, représentant les paramètres possibles de la fonction à appeler et composé d'au moins un sous-élément *FunctionParameter* contenant le paramètre en chaîne de caractères.

```
<Function>
        <FunctionName>...</FunctionName>
</Function>
```

```
<Function>
        <FunctionName>...</FunctionName>
        <FunctionParameters>
                <FunctionParameter>...</FunctionParameter>
                ...
        </FunctionParameters>
</Function>
```

3.5. L'élément *Transition*

L'élément *Transition* du schéma permet de définir une transition possible à partir de l'état *State* dans lequel son élément parent *Transitions* est défini. Cet élément est composé de la manière suivante :

- un attribut obligatoire *to*, permettant de définir l'état suivant sur lequel la transition pointe ;
- un élément *Stimulus* facultatif, représentant la stimulation de la transition par réception de message en tant que chaîne de caractères considérée comme le patterne de message à recevoir en tant que stimulation ;
- un élément *Guard* facultatif, représentant une garde à respecter pour pouvoir passer la transition de type *InvariantType* (voir ci-dessus) ;
- un élément *Function* ou *Message* facultatif, représentant une action unique à effectuer soit par appel de fonction, soit par envoi de message. Dans le cas de l'élément *Message*, le contenu doit être une chaîne de caractères représentant le patterne de message à envoyer.

```
<Transition to="..." />
```

```
<Transition to="...">
        <Stimulus>...</Stimulus>
        <Guard>...</Guard>
        <Function>...</Function>
</Transition>
```

```
<Transition to="...">
        <Stimulus>...</Stimulus>
        <Guard>...</Guard>
        <Message>...</Message>
</Transition>
```

3.5.1. Cas d'un modèle à états finis

Dans le cas d'un modèle à états finis, il est possible de terminer l'exécution du comportement à l'aide d'une transition pointant sur le nom d'état *EndState*. Si ce nom d'état est rencontré, l'agent sera programmatiquement stoppé par le comportement. Tout contenu

présent dans la définition de la transition sera néanmoins vérifié/exécuté avant arrêt de l'agent.

3.6. Application à l'exemple

L'ensemble des éléments définis ci-dessus appliqués à l'exemple proposé permet de constituer le document suivant :

```
<Models>
    <Model name="Coffee">
        <InitialState>Waiting selection</InitialState>
        <State name="Waiting selection">
            <Transitions>
                <Transition to="Waiting selection">
                    <Stimulus>SELECTION</Stimulus>
                    <Guard>
                        <NOT>
                            <Function>

<FunctionName>productAvailable</FunctionName>
                            </Function>
                        </NOT>
                    </Guard>
                    <Function>
                        <FunctionName>display</FunctionName>
                        <FunctionParameters>
                            <FunctionParameter>NO_LEFT</FunctionParameter>
                        </FunctionParameters>
                    </Function>
                </Transition>
                <Transition to="Waiting payment">
                    <Stimulus>SELECTION</Stimulus>
                    <Guard>
                        <Function>
                            <FunctionName>productAvailable</FunctionName>
                        </Function>
                    </Guard>
                    <Function>
                        <FunctionName>display</FunctionName>
                        <FunctionParameters>
                            <FunctionParameter>PRICE</FunctionParameter>
                        </FunctionParameters>
                    </Function>
                </Transition>
            </Transitions>
        </State>
        <State name="Waiting payment">
            <Invariant>
                <AND>
                    <NOT>
                        <Function>
                            <FunctionName>hasBeenPayed</FunctionName>
                        </Function>
                    </NOT>
                    <NOT>
                        <Function>
                            <FunctionName>cancelled</FunctionName>
                        </Function>
```

```
                        </NOT>
                </AND>
        </Invariant>
        <Content>
                <Function>
                        <FunctionName>checkInsertedCoins</FunctionName>
                </Function>
        </Content>
        <Transitions>
                <Transition to="Giving back money">
                        <Guard>
                                <OR>
                                        <Function>
                                                <FunctionName>tooMuchPayed</FunctionName>
                                        </Function>
                                        <Function>
                                                <FunctionName>cancelled</FunctionName>
                                        </Function>
                                </OR>
                        </Guard>
                </Transition>
                <Transition to="Serving coffee">
                        <Guard>
                                <NOT>
                                        <Function>
                                                <FunctionName>tooMuchPayed</FunctionName>
                                        </Function>
                                </NOT>
                        </Guard>
                </Transition>
        </Transitions>
</State>
<State name="Giving back money">
        <Invariant>
                <Function>
                        <FunctionName>moneyBackNotComplete</FunctionName>
                </Function>
        </Invariant>
        <Content>
                <Function>
                        <FunctionName>giveCoinBack</FunctionName>
                </Function>
        </Content>
        <Transitions>
                <Transition to="Waiting selection">
                        <Guard>
                                <Function>
                                        <FunctionName>cancelled</FunctionName>
                                </Function>
                        </Guard>
                </Transition>
                <Transition to="Serving coffee">
                        <Guard>
                                <NOT>
                                        <Function>
                                                <FunctionName>cancelled</FunctionName>
                                        </Function>
                                </NOT>
                        </Guard>
                </Transition>
        </Transitions>
```

```
        </State>
        <State name="Serving coffee">
                <Content>
                        <Function>
                                <FunctionName>serveCoffee</FunctionName>
                        </Function>
                        <Function>
                                <FunctionName>giveSwizzleStick</FunctionName>
                        </Function>
                </Content>
                <Transitions>
                        <Transition to="Waiting taking coffee">
                                <Function>
                                        <FunctionName>soundNotification</FunctionName>
                                </Function>
                        </Transition>
                </Transitions>
        </State>
        <State name="Waiting taking coffee">
                <Content>
                        <Function>
                                <FunctionName>waitProgrammedTime</FunctionName>
                        </Function>
                </Content>
                <Transitions>
                        <Transition to="Waiting taking coffee">
                                <Guard>
                                        <NOT>
                                                <Function>

<FunctionName>hasTakenCoffee</FunctionName>
                                                </Function>
                                        </NOT>
                                </Guard>
                                <Function>
                                        <FunctionName>soundNotification</FunctionName>
                                </Function>
                        </Transition>
                        <Transition to="Waiting selection">
                                <Guard>
                                        <Function>
                                                <FunctionName>hasTakenCoffee</FunctionName>
                                        </Function>
                                </Guard>
                        </Transition>
                </Transitions>
        </State>
    </Model>
</Models>
```