

Déclaration des variables

Mode d'utilisation des variables

- Par référence, sauf pour le passage des paramètres de service.
- Portée selon l'endroit de la déclaration : service (premier action state), séquence d'instruction dans un action state.

Déclarer une variable simple

- `<nature> <nomVariable> ["=" <valeur initiale>]`
- La valeur initiale doit être compatible avec la nature
- *infolntervenant intervenant - int compteur = 1*

Déclarer une variable tableau

- `<nature> ["" taille {", " taille} "]" <nomVariable> ["=" <valeur initiale>] {"", valeur initiale}`
- `<taille> (1) ::= '1' .. '9' | '*'`
- Les valeurs initiales doivent être compatibles avec la nature
- *InfoMission[*] IstMissions*

Déclarer un paramètre d'entrée de service

- `in _" <nomVariable> ["" taille {", " taille} "]" <nature>`
- `<taille> (1) ::= '1' .. '9' | '*'`
- *in_flux[*] InfoPlanCoassurance*

(1) Un tableau[*] est équivalent à la déclaration d'une liste

Egalité d'objets

Tester l'égalité entre deux objets (correspondance des attributs)

- `<objet#1> "." equivaut "(" <objet#2> ")"`
- Renvoie un booléen vrai si équivalent (voir aussi opérateur logique '==')
- *infoChantierSinistre.equivaut.infoChantierContrat*

Etat des automates

Manipulation de l'état d'un automate

- La dénomination de l'état se fait sous la forme d'une chaîne de caractères.
- *Si Couverture.etat == "AEvaluer" Alors... Fin Si*

Foncteur verrou

Prise de verrou

- Verrouiller `<identifiant du dossier> "," <type de dossier>`
- Renvoie le signal DossierDejaVerrouille si le dossier est déjà verrouillé
- *verrouiller(sd.infoSinistre.idSinistre, TypeVerrou.Sinistre)*

Libération de verrou

- `deverrouiller "("`
- Déverrouille tous les dossiers verrouillés au préalable pour la session utilisateur.
- *deverrouiller()*

Foncteur journalisation

Foncteur journalisation

- `Journalisation.debuterActeDeGestion "(" <code de l'acte de gestion> ")"`
- `Journalisation.debuterActivite "(" <code de l'activité> "," <identifiant du dossier> "," <type de dossier>")"`
- `Journalisation.terminerActeDeGestion "(")`
- `Journalisation.terminerActivite "(")`
- `Journalisation.abandonnerActeDeGestion "(")`

Foncteur orchestrateur

Foncteur orchestrateur

- `Orchestrateur.Activite "(" <structure de données> "," <code activité> ")"`
- `Orchestrateur.ActeGestionInit "(" <structure de données> "," <code acte de gestion> "," <idDossier> "," <cdTypeDossier> ")"`
- `Orchestrateur.ActeGestionValid "(" <structure de données> "," <code acte de gestion> "," <idDossier> "," <cdTypeDossier> ")"`
- `Orchestrateur.TerminerActeGestion "(")`
- `Orchestrateur.AbandonnerActeGestion "(")`
- `<structure de données> := la structure InfoMLO`

Foncteur règles

Appel au moteur de règles

- `executerRegles "(" <structure de données> ")"`
- `<structure de données> := la structure InfoMLO pour une invocation du moteur de règles au niveau de la strate de l'organisation. La structure InfoMLM en cas d'invocation au niveau de la strate métier.`
- *executerRegles(sd)*

Foncteur transaction

Un service est transactionnel dès l'instant que le modèle l'indique par une annotation : *tagged value* appartenant au profile UML pour Praxeme.

Invocation de service

Invocation de services

- `<nom du composant> "." <nom du service avec les paramètres>`
- Renvoie un attribut du type prévu par le retour du service invoqué
- *sd.IstSouscripteurs = IPersonne.rechercherNaturePersonnes(IPersonne.SOUSCRIPTEUR_RE CHERCHER_SIGNALETIQUE, IstCriteres, NaturePersonne.Souscripteur)*

Liste

Trier une liste

- `<liste> "." trier "(" <attribut de la liste> [", " decroissant] ")"`
- Renvoie une liste triée
- *IstChantiers.trier(cdChantier)*
- *IstChantiers.trier(cdChantier, decroissant)*

Premier et dernier élément d'une liste

- `<liste> "." premier "(")`
- `<liste> "." dernier "(")`
- Renvoie un élément premier ou dernier selon les cas
- *IstChantiers.premier()*

Intersection entre deux listes

- `intersection "(" <liste#1> "," <liste#2> ")"`
- Renvoie une liste avec les éléments en intersection
- *Intersection(IstChantiersSinistre, IstChantiersContrat)*

Union entre deux listes

- `union "(" <liste#1> "," <liste#2> "," "1" | "2" ")"`
- 1 : garde les doublons – 2 : élimine les doublons
- Renvoie une liste qui correspond à l'union
- *union(IstChantiersSinistre, IstChantiersContrat, 1)*

Récupérer un élément dans une liste

- `<liste> "." recuperer "(" <attribut de recherche> ")"`
- Renvoie un élément de la liste
- *IstCouvertures.recuperer(infoReevaluation.idCouverture)*

Déterminer si un élément est présent dans une liste

- `<liste> "." contient "(" <element> ")"`
- Renvoie un élément de la liste
- *IstChantiers.contient(idChantier)*

Retourner une liste réduite aux valeurs d'un attribut

- `<liste> "." isoler "(" <attribut> ")"`
- Renvoie une liste avec une seule colonne `<attribut>`
- *IstChantiers.contient(idChantier)*

Remplacer un élément par un autre dans une liste

- `<liste> "." remplace "(" <element#1> "," <element#2> ")"`
- `<element#1>` est remplacé par `<element#2>`
- *IstChantiers.remplace(idSinistreAvant, idSinistreApres)*

Filtrer une liste à partir d'un prédicat

- `<liste> "." selectionner "(" <expression booléenne> ")"`
- `<expression booléenne>` porte sur les attributs de la liste
- *IstChantiers.selectionner(cout>seuilMaximum)*

Taille d'une liste

- `<liste> "." taille "(")`
- Renvoie la taille de la liste (nature int)
- *IstChantiers.taille()*

Ajouter un élément dans une liste

- `<liste> "." ajouter "(" <element> ")"`
- *IstChantiers.ajouter(chantierSinistre)*

Supprimer un élément dans une liste

- `<liste> "." supprimer "(" <element> ")"`
- *IstChantiers.supprimer(chantierSinistre)*

Trouver la position d'un élément dans une liste

- `<liste> "." trouverPosition "(" <élément> ")"`
- Renvoie la position (nature int)
- *IstChantiers.trouverPosition(idChantier)*

Manipulation des données

- On retient la notation pointée.
- *adresse.cdTypeUtilisation=TypeUtilAdresse.Principale*

Opérateurs logiques

<operateur logique> ::= 'et' | 'ou' | 'ou bien' | '=' | '==' | 'dans' | '<' | '>' | '{' | '}' | 'U'

- *Si sd.pourcentage.estVide() ou sd.cdRoleCoassureur.estVide() ou sd.cdRoleCoassureur.estVide() et sd.refExtCieAssurance.estVide()...*
- Exemple de test d'égalité
- *Si gar.idGarantiePK == in_oid Alors garantie = gar*
- *Fin Si*

Retour du service

Retour du service

- retourner
- Le mot-clef retourner est utilisé en fin de service pour préciser la valeur ou la variable renvoyée par le service. La plupart du temps les services d'une machine logique retourne la structure de données (sd).
- La valeur retournée doit être compatible avec la signature du service
- *retourner sd*

Service de lecture, recherche et création

lire

- lire "(" <oid> "," <flux> | null [" , " <nature>] ")"
- <flux> ::= liste d'InfoMLM (si oid n'est pas dans la liste alors accès base)
- <nature> ::= nature d'une classe fille en cas d'héritage de MLM (string)
- Renvoie un InfoMLM
- *sd.sinistre = ISinistre.lireSinistres(sd.sinistre.idSinistre, null)*

recherche

- rechercher "(" <requete> "," <liste criteres> [" , " <nature>] ")"
- <requete> ::= nom de la requête de recherche (constate sur l'interface de l'atelier qui expose le service de recherche)
- <liste criteres> ::= listes des valeurs pour les critères de la recherche
- <nature> ::= nature d'une classe fille en cas d'héritage de MLM (string)
- Renvoie une liste d'InfoMLM
- *sd.IstDossiersArchive = IArchivage.rechercherDossierArchives(IArchivage.DOSSIERARCHIVE_R ECHERCHER_IDSIN, IstCriteres)*

créer

- creer "(" infoMLM ")"
- Renvoie une variable de nature InfoMLM initialisé avec les données du paramètre d'entrée
- *infoChantierNouveau = creer(infoChantierPrecedent)*

Signalisation

Utilisation d'un signal

- signal "(" <nom du signal> ")"
- *Si signal (ParametreObligatoireNonRenseigne) Alors...*

Emission d'un signal

- Emettre "(" signal "(" <nom du signal> ")" [" , " <causes>] ")"
- <causes> ::= liste de signaux. C'est la liste des signaux à l'origine de l'émission du signal courant.
- *Si parmAbsent ou in_nature.estVide() Alors*
Emettre(signal(ParametreObligatoireNonRenseigne))
Fin Si

Structure alternative

Structure alternative

- Si <predicat> Alors <instruction> [Sinon <instruction>] Fin Si
- *Si sd.infoSinistre.idSinistre.estVide()*
Alors #preparation des variables du message utilisateur
émettre (signal(ParametreObligatoireNonRenseigne))
Fin Si

Structure énumérative

Structure énumérative

- Pour chaque <attribut> dans <liste d'élément du même type que attribut> <instruction> Fin Pour chaque
- *Pour chaque mouvement dans IstMouvements*
*mtOperation+=abs(mouvement.mt)*mouvement.nature.cdSens*
Fin Pour chaque

Structure itérative

Itération avec décision en entrée

- Tant que <predicat> <instruction> Fin Tant que
- *Tant que mtAstreinte < borneMaximum*
mtAstreinte += sd.valeurSinistre – borneLegale
Fin Tant que

Itération avec décision en sortie

- Faire <instruction> Jusqu'à <predicat>
- *Faire*
mtAstreinte += sd.valeurSinistre – borneLegale
jusqu'à mtAstreinte < borneMaximum

Structure optative

Structure optative

- Selon <variable> { <valeur> ":" <instruction> } Fin Selon
- <variable> ::= variable atomique ou énuméré toutes natures possibles
- Quand la variable prend une valeur qui n'est pas représentée aucune instruction n'est exécutée dans la structure
- Chaque bloc d'instructions contient un 'break' implicite. Si à la fin d'un bloc d'instructions, l'exécution doit se poursuivre sur le suivant, on utilise le mot-clef 'continuer'.
- Il est possible de nommer plusieurs variables devant un même bloc d'instructions. En revanche, une même valeur ne peut être indiquée qu'une seule fois.
- Pour désigner toutes les valeurs que peut prendre la variable autres que celles mentionnées explicitement dans la structure, on utilise le mot-clef 'autreValeur'
- *Selon IstPersonnes.taille()*
0 : emettre(signal(PersonneVide))
1 : personne = IstPersonnes[0]
autreValeur : emettre(signal(RefcePersonneMultiple))
Fin Selon

Valeur absolue

Valeur absolue d'un nombre

- abs "(" <nombre> ")"
- Renvoie un nombre en valeur absolue
- *mtOperation+=abs(mouvement.mt)*mvt.natureMouvement.cdSens*

Ce document existe grâce à la contribution de la société SMABTP utilisatrice de la méthode Praxeme – Creative Commons Paternité



Auteur et contact : pierre.bonnet@orchestranetworks.com - <http://soa.orchestranetworks.com>

Orchestra Networks – 75 Boulevard Haussmann – 75008 Paris
Tel: 01 42 68 50 80 - info@orchestranetworks.com - www.orchestranetworks.com

Le pseudo-code Praxeme appliqué à SOA

Praxeme est une méthode *open source* pour la construction des systèmes d'information, avec une composante SOA pour l'aspect logique (http://www.orchestranetworks.com/fr/soa/triptyque_praxeme.cfm).

La spécification détaillée des services s'appuie sur un pseudo-code formel pris en main par les concepteurs de l'aspect logique. Les équipes de développement traduisent le pseudo-code dans le langage d'implémentation retenu pour le logiciel. Progressivement, l'équipe de développement reprend la main sur le pseudo-code afin d'assurer la maintenance du système.

Définition commune de la grammaire

Style de notation

- Utilisation de Backus-Naur ⁽¹⁾ pour l'expression de la grammaire.
- CAMEL : mise en majuscule de la première lettre des mots liés.
- Les majuscules et les caractères accentués ne sont pas obligatoires.
- On n'utilise pas de caractère de fin de ligne.
- La manipulation des données se fait par notation pointée, c'est-à-dire de manière formelle sous la forme classe "." attribut

<caractere> ::= n'importe quel caractère ASCII

<nature> ::= 'boolean' | 'char' | 'date' | 'int' | 'long' | 'object' | 'real' | 'string'

<predicat> ::= expression logique qui renvoie un booléen

<instruction> ::= ligne ou séquence d'instructions. Chaque séquence commence sur une nouvelle ligne avec indentation identique

(1) <http://cui.unige.ch/db-research/Enseignement/analyseinfo/BNFweb.html>

Commentaires

<commentaire sur une ligne> ::= "#" <texte commentaire> ["#"]

<commentaire multi-lignes> ::= "<<" <texte commentaire> ">>"

<texte commentaire> ::= {caractere} sans '/' | '*' | '{' | '}' | '|' | '|'

Date

Tester si date#1 est avant date#2

- <date#1> "." avant "(" <date#2> ")"
- Renvoie un booléen
- *dateDuJour.avant(dateSinistre)*

Tester si date#1 est après date#2

- <date#1> "." apres "(" <date#2> ")"
- Renvoie un booléen
- *dateDuJour.apres(dateSinistre)*

Tester l'égalité entre deux dates

- <date#1> "." egal "(" <date#2> ")"
- Renvoie un booléen
- *dateDuJour.egal(dateSinistre)*

Ajouter ou enlever un jour à une date

- <date concernee> "." ajouterJour "(" <incrementJour> ")"
- <incrementJour> ::= "+" | "-" "1"
- Renvoie une date
- *dateDuJour.ajouterJour (+1)*

Trier une liste de dates

- <liste de date> "." trier "(" <attribut date de la liste> [" , " decroissant] ")"
- Renvoie une liste triée
- *IstDate.trier(dtDebut) - IstDate.trier(dtDebut, decroissant)*