Système Orchestrateur Central

Ce module constitue le cœur de coordination de l'application LSF. Il orchestre l'ensemble des composants et sous-systèmes conformément aux diagrammes d'état et à l'arborescence du projet.

# **Architecture**

L'architecture du Système Orchestrateur a été refactorisée pour suivre les principes SOLID avec une séparation claire des responsabilités. Cette nouvelle structure permet :

* Une meilleure modularité et maintenabilité
* Des tests unitaires plus faciles à mettre en œuvre
* Une évolution plus aisée des fonctionnalités
* Une réduction des couplages entre composants

## **Composants Principaux**

### **Orchestrateur Central (`SystemeOrchestrateurCentral.ts`)**

* Point d'entrée principal du système
* Coordonne l'initialisation des composants
* Gère l'état global du système
* Délègue les tâches aux services spécialisés

### **Services Spécialisés**

* CacheService : Gestion intelligente du cache multi-niveaux
* RequestService : Traitement et suivi des requêtes
* OptimizationService : Optimisation des performances et ressources
* AlertManager : Gestion centralisée des alertes système

### **Gestionnaires de Requêtes**

* RequestHandlerFactory : Création des gestionnaires selon le type de requête
* Gestionnaires spécialisés par type de requête (Expression, Traduction, Analyse)

## **Relations avec les Autres Systèmes**

* SystemeControleEthique : Validation éthique des requêtes
* IACore : Traitement cognitif
* SystemeExpressions : Génération d'expressions
* ValidationCollaborative : Validation des expressions par la communauté
* GestionVariantesDiatopiques : Gestion des variantes dialectales
* GestionEspaceSpatial : Gestion de l'espace spatial de signation
* MonitoringUnifie : Supervision et métriques

# **Utilisation**

## **Initialisation**

```typescript  
 import { SystemeOrchestrateurCentral } from '@ai/coordinators';  
   
const orchestrateur = new SystemeOrchestrateurCentral({  
 cacheEnabled: true,  
 monitoringLevel: 'advanced',  
 initialMode: 'balanced',  
 maxPendingRequests: 100,  
 autoRecover: true,  
 preloadModels: true,  
 ethicsLevel: 'standard'  
 });  
   
orchestrateur.on('request:completed', (data) => {  
 console.log(`Request completed in ${data.duration}ms`);  
 });  
 ```

## **Traitement des Requêtes**

```typescript  
 import { RequestType, TaskPriority } from '@ai/coordinators/types';  
   
const request = {  
 type: RequestType.TRANSLATION,  
 modality: 'lsf',  
 data: { text: 'Bonjour, comment ça va?' },  
 userId: 'user123',  
 sessionId: 'session456',  
 timestamp: Date.now(),  
 priority: TaskPriority.NORMAL  
 };  
   
try {  
 const result = await orchestrateur.processRequest(request);  
 console.log('Résultat:', result);  
 } catch (error) {  
 console.error('Erreur:', error);  
 }  
 ```

## **Changement de Mode**

```typescript  
 import { OrchestrationMode } from '[@ai/coordinators/types/orchestrator.types](mailto:@ai/coordinators/types/orchestrator.types)';  
   
orchestrateur.changeOrchestrationMode(OrchestrationMode.HIGH\_PERFORMANCE);  
 ```

## **Arrêt Propre**

```typescript  
 await orchestrateur.shutdown();  
 ```

# **Évolutions Futures**

1. Intégration plus poussée avec le système de pyramide IA
2. Amélioration des capacités d'auto-optimisation
3. Renforcement de la résilience face aux défaillances
4. Intégration de la pyramide IA étendue