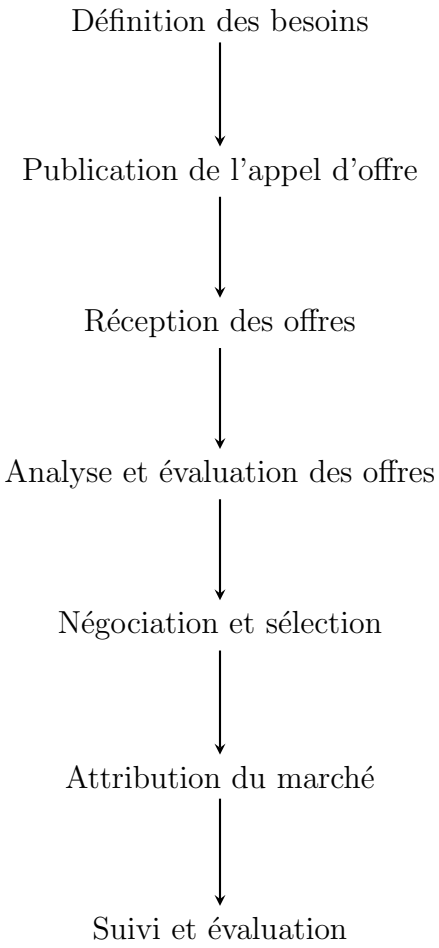
TP: Smart Contract avec le Paradigme ECA et Interrogation d'un Service Oracle: Application des marchés publics

Description des marchés publics

Un marché public est un contrat conclu entre une autorité publique (comme une administration, une collectivité locale ou une entreprise publique) et une entreprise privée ou publique, dans le but de répondre à un besoin spécifique en termes de travaux, de fournitures ou de services. Les marchés publics sont soumis à des règles strictes, souvent dictées par des lois ou des règlements, visant à garantir la transparence, l'égalité de traitement, et la bonne gestion des fonds publics.

Le diagramme ci-dessous illustre ces étapes de manière séquentielle :



Objectifs du TP

L'objectif de ce TP est de comprendre et d'implémenter un smart contract utilisant le paradigme **Event-Condition-Action (ECA)**. Ce contrat interagira avec un service externe via un **oracle**, permettant d'exécuter des actions spécifiques en fonction des événements et des conditions définis.

Étape 1 : Comprendre le paradigme ECA dans les smart contracts

Le paradigme **Event-Condition-Action (ECA)** est une approche essentielle dans le développement des smart contracts. Il permet de gérer des interactions dynamiques en définissant des règles basées sur des *événements déclencheurs*, des *conditions logiques* et des *actions associées*.

1.1. Définition du paradigme ECA

Le paradigme ECA repose sur trois éléments principaux :

- **Événement (Event)** : Une occurrence spécifique qui déclenche un processus. Par exemple, un dépôt d'argent dans un contrat ou une modification d'état sur la blockchain.
- **Condition (Condition)** : Une règle ou un critère qui doit être satisfait pour exécuter une action. Cela peut inclure une vérification de montant, d'adresse ou de toute autre donnée pertinente.
- **Action (Action)** : Une tâche ou une opération effectuée lorsque la condition est validée. Par exemple, transférer des fonds ou modifier une variable d'état.

1.2. Fonctionnement du paradigme

1. **Détection de l'événement** : Lorsqu'un événement défini (comme un appel de fonction ou un transfert de tokens) se produit, le smart contract est alerté.
2. **Évaluation de la condition** : Le contrat vérifie si la condition associée à l'événement est remplie.
3. **Exécution de l'action** : Si la condition est satisfaite, l'action est exécutée conformément à la logique implémentée.

1.3. Exemple concret

Prenons l'exemple d'un smart contract pour une assurance climatique :

- **Événement** : Le service oracle signale qu'une tempête a été détectée.
- **Condition** : Vérifiez si la localisation de l'utilisateur est affectée par la tempête.
- **Action** : Déclenchez automatiquement un remboursement ou un paiement d'indemnisation.

1.4. Avantages du paradigme ECA

- **Automatisation** : Réduit la dépendance à des actions manuelles.
- **Réactivité** : Permet une exécution en temps réel basée sur des données dynamiques.
- **Flexibilité** : Facilite l'ajout de nouvelles règles sans modifier l'ensemble du système.

En maîtrisant le paradigme ECA, il devient possible de concevoir des smart contracts plus efficaces et réactifs pour des cas d'utilisation variés, tels que la finance décentralisée, l'Internet des Objets (IoT) ou les systèmes de gestion d'actifs.

Étape 2 : Prérequis et Préparation de l'environnement de développement

Prérequis

Avant de commencer ce TP, assurez-vous que les éléments suivants sont installés et configurés sur votre machine :

- **Truffle** : Outil de développement pour smart contracts.
- **Ganache** : Blockchain locale pour tester les smart contracts.
- **Python 3.x** : Environnement Python pour interroger des services externes.
- **Node.js et npm** : Outils nécessaires pour Truffle.

Préparation de l'environnement de développement

1. **Installer Node.js et npm** si ce n'est pas déjà fait.
2. **Installer Truffle** : Exécutez la commande suivante pour installer Truffle globalement :

$$npminstall - gtruffle$$

3. **Lancer Ganache** pour créer une blockchain locale.
4. **Créer un projet Truffle** : Exécutez `truffle init` dans le terminal pour initialiser un nouveau projet.
5. **Connecter Truffle à Ganache** en configurant le fichier `truffle-config.js` avec l'adresse de votre Ganache.

Étape 3 : Définir et écrire le smart contract

- **Définir un contrat de base avec un événement** : Vous allez définir un événement dans le smart contract qui sera déclenché lorsqu'une condition est remplie (par exemple, un dépôt dépassant un certain montant).
- **Définir la condition** : Cette condition sera associée à la logique de votre événement, par exemple, si un dépôt dépasse un montant spécifique, un événement sera déclenché.
- **Exécuter l'action** : Lorsqu'une condition est vérifiée, une action sera exécutée, comme appeler une fonction pour envoyer des fonds ou émettre un autre événement.

Étape 4 : Interroger un service externe via un Oracle

Un **oracle** est un service externe qui permet de récupérer des informations du monde réel (comme des données météorologiques, des prix, etc.) pour être utilisées dans la blockchain.

1. Créez un service Oracle en Python pour interroger une API externe, comme `OpenWeatherMap`, pour obtenir des données telles que la température actuelle.
2. Utilisez l'API pour récupérer des données spécifiques, par exemple, la température.
3. Définissez des règles sur la blockchain pour que le smart contract agisse en fonction des données récupérées par l'Oracle.

Conclusion

Dans ce TP, vous avez appris à utiliser le paradigme **Event-Condition-Action (ECA)** pour gérer les événements dans un smart contract. Vous avez également vu comment interroger un service externe via un **oracle** pour récupérer des données du monde réel et les utiliser pour déclencher des actions spécifiques sur la blockchain. Cela permet de rendre les smart contracts plus dynamiques et réactifs aux informations externes.

Exercice

Ces exercices visent à explorer l'application des contrats intelligents (*smart contracts*) et des services oracles dans la gestion des marchés publics en utilisant le paradigme **ECA** (*Événement-Condition-Action*). Chaque exercice s'appuie sur des scénarios concrets pour illustrer les concepts fondamentaux.

Exercice 1 : Publication automatique d'un appel d'offres

Objectif : Créer un smart contract qui publie automatiquement un appel d'offres lorsque :

- Le budget est validé par le ministère des Finances.
- Les documents de l'appel d'offres sont prêts et validés.

Tâche :

1. Modéliser le processus avec le paradigme ECA :
   - **Événement :** Le ministère des Finances valide le budget.
   - **Condition :** Les documents sont complets et validés.
   - **Action :** Publier l'appel d'offres sur une plateforme décentralisée.
2. Simuler l'utilisation d'un service oracle pour récupérer les informations :
   - Validation du budget depuis une base de données externe.
   - Vérification de la validation des documents.

Questions :

1. Comment intégrer un oracle pour valider les documents et le budget ?
2. Quels sont les avantages d'automatiser ce processus avec un smart contract ?

Exercice 2 : Suivi des paiements intermédiaires

Objectif : Mettre en place un smart contract qui gère les paiements intermédiaires lorsque :

- Les travaux sont achevés à une étape donnée.
- Un inspecteur valide cette étape via un service oracle.

Tâche :

1. Définir les règles ECA pour chaque étape :
    - **Événement :** L'inspecteur envoie une confirmation via l'oracle.
    - **Condition :** Les travaux sont achevés selon les spécifications.
    - **Action :** Libérer le paiement pour cette étape.
2. Utiliser un service oracle pour obtenir :
    - Le rapport de validation des travaux.
    - L'état des fonds disponibles.

Questions :

1. Quels mécanismes garantissent que l'inspecteur agit de manière impartiale ?
2. Comment gérer les conflits en cas de non-validation des travaux ?

Exercice 3 : Résiliation automatique d'un contrat

Objectif : Modéliser un smart contract qui résilie automatiquement un marché public si :

- Les délais contractuels sont dépassés sans justification.
- Les travaux réalisés sont non conformes selon les rapports des inspecteurs.

Tâche :

1. Établir les règles ECA :
    - **Événement :** Détection d'un dépassement de délai ou rapport de non-conformité.
    - **Condition :** Aucune justification ou mesure corrective soumise dans les 7 jours.
    - **Action :** Résilier automatiquement le contrat et notifier les parties concernées.
2. Simuler un oracle pour vérifier :
    - Le statut des délais contractuels.
    - Les rapports de conformité des travaux.

Questions :

1. Quels mécanismes doivent être intégrés pour éviter les résiliations abusives ?
2. Comment assurer la transparence de l'oracle utilisé ?

Exercice 4 : Sélection automatisée de l'offre gagnante

Objectif : Créer un smart contract qui sélectionne automatiquement le prestataire gagnant en fonction des critères suivants :

- L'offre financière respecte le budget alloué.
- Tous les documents requis sont soumis et validés par un oracle.
- Le score total (qualité/prix/délais) est le plus élevé.

Tâche :

1. Définir les règles ECA :
   - **Événement :** Clôture des soumissions.
   - **Condition :**
     - Budget respecté.
     - Documents validés.
     - Score supérieur à celui des autres candidats.
   - **Action :** Attribuer le marché au prestataire gagnant.
2. Connecter un oracle pour :
   - Valider les documents soumis par chaque candidat.
   - Calculer les scores en fonction des données externes (prix, qualité, délais).

Questions :

1. Comment garantir l'impartialité dans l'attribution des scores ?
2. Quels mécanismes permettent de contester la décision automatisée ?

Ces exercices montrent comment les *smart contracts* et les services oracles, associés au paradigme ECA, peuvent rendre les processus des marchés publics plus transparents, automatisés et efficaces.