Rapport : Analyse des Patterns Prolog et Conception des Requêtes Cypher pour la Vérification RGPD

Objectif du Rapport :

Ce document a pour but de synthétiser l'analyse des mécanismes de vérification de conformité RGPD, initialement implémentés en Prolog, et de présenter en détail la conception des requêtes Cypher équivalentes. Ces requêtes sont destinées à être exécutées sur une base de données Neo4j, conformément au schéma de graphe déduit du code de conversion PrologToGraphDB.java. Ce travail constitue la phase de conception préalable à l'implémentation du nouveau solveur de conformité basé sur Cypher.

Méthodologie Générale de Conception :

Pour chacune des quatre exigences RGPD ciblées, la démarche suivante a été adoptée :

- **Analyse Approfondie des Patterns Prolog :** Compréhension de la logique, des prédicats impliqués, des conditions et des objectifs de chaque règle Prolog.
- **Cartographie vers Neo4j :** Identification des labels de nœuds, types de relations et propriétés dans Neo4j correspondant aux faits et structures Prolog, en s'appuyant sur le code `PrologToGraphDB.java`.
- **Définition de l'Objectif de la Requête Cypher :** Chaque requête Cypher est conçue pour identifier et retourner les instances de non-conformité au principe RGPD concernée.
- **Spécification des Paramètres d'Entrée :** Définition des paramètres dynamiques requis par chaque requête (ex: `$currentTime`, limites de durée spécifiques).
- **Conception Itérative de la Requête Cypher :** Traduction progressive de la logique Prolog en clauses Cypher (`MATCH`, `WHERE`, `OPTIONAL MATCH`, `NOT EXISTS {}`, etc.), en discutant et affinant la structure.
- **Définition des Informations de Sortie :** Spécification des champs à retourner par la requête pour permettre la création et l'affichage d'objets `Issue`informatifs en Java.

Conception Détaillée par Exigence RGPD:

1. Limitation de Stockage

Logique Prologue Essentielle :

- Le prédicat storageLimitation(D, TU)identifie une violation si une donnée personnelle DP(source de D), dont la dernière utilisation pertinente (non-suppression) était à TU, n'est plus dans un état "Ok" (\+ storageLimitationOk(DP, TU)).
- storageLimitationOk(DP, TU)est vrai si le délai de conservation depuis TU(basé sur $currentTimeet $storageLimitDuration) n'est pas écoulé, OU si DP a été supprimé (notAvailable) avant l'expiration de ce délai.

Éléments Neo4j Pertinents :

- **Source personnelle donnée :** `(dp_source:Artifact {type: 'personal_data'})`
- **Artefact utilisé :** `(artifact_used:Artifact)`
- **Dérivation :** `(artifact_used)-[:WAS_DERIVED_FROM*0..]->(dp_source)`
- **Dernière utilisation (non-suppression) :** `(process_of_last_usage:Process)-[u_usage:USED {TU: actual_last_usage_TU}]->(artifact_used)` où `process_of_last_usage.action <> 'delete'`.
- **Suppression :** `(p_delete:Process {action: 'delete'})-[u_delete:USED {TU: timestamp_del}]->(dp_source)`

Paramètres Cypher : `$currentTime` (entier), `$storageLimitDuration`(entier, même unité que les timestamps).

Demande Cypher Conçue (pour trouver les violations) :

```
// Étape 1: Trouver la dernière utilisation non-suppression de chaque artefact

    MATCH (p_usage:Process)-[u_usage:USED]->(artifact_used:Artifact)
    WHERE p_usage.action <> 'delete'
    WITH artifact_used, p_usage, u_usage.TU AS lastUsageTime
    ORDER BY artifact_used.name, lastUsageTime DESC
    WITH artifact_used, COLLECT({processNode: p_usage, time: lastUsageTime})[0]
AS lastUsageEventInfo

    // Étape 2: S'assurer que l'événement existe et lier à la donnée personnelle source

    WHERE lastUsageEventInfo IS NOT NULL
    WITH artifact_used, lastUsageEventInfo.processNode AS process_of_last_usage,
lastUsageEventInfo.time AS actual_last_usage_TU
    MATCH (artifact_used)-[:WAS_DERIVED_FROM*0..]->(dp_source:Artifact {type:
'personal_data'})

    // Étape 3: Appliquer la logique de violation

    WHERE ($currentTime - actual_last_usage_TU) >= $storageLimitDuration // Le
délai de conservation est dépassé
    AND NOT EXISTS { // ET la donnée n'a PAS été supprimée à temps
       MATCH (p_delete:Process {action: 'delete'})-[u_delete:USED]->(dp_source)
       WHERE u_delete.TU > actual_last_usage_TU // La suppression doit être après la
dernière utilisation
        AND (u_delete.TU - actual_last_usage_TU) < $storageLimitDuration // ET la
suppression doit avoir eu lieu DANS les délais
    }
```

// Étape 4: Retourner les informations de la violation

```
RETURN DISTINCT
   dp_source.name AS D,
   actual_last_usage_TU AS TU,
   process_of_last_usage.name AS P,
   artifact_used.name AS D_artifact_actually_used
```

=> Champs Retournés pour **Issue**: D (données personnelles source), TU(timestamp de dernière utilisation), P(processus de cette utilisation), D_artifact_actually_used(artefact spécifique utilisé).

2. Droit à l'Effacement

Logique Prologue Essentielle :

- **`eraseCompliant(D,T,P)`** signale une violation si une demande d'effacement (`askErase(D,T,P)`) pour la donnée `D`(par `P`à `T`) n'est pas "Ok" (`\+ dataErasureOk(D,T)`).
- **`dataErasureOk(D,T)`** est vrai si la donnée a été effacée à temps (`eraseComplete(D,T)`) OU si le délai pour l'effacer n'est pas encore expiré (`eraseNotDoneYet(D,T)`).

Éléments Neo4j Pertinents :

- **Demande :** `(p_ask:Process {action: 'askErase'})-[u_ask:USED {TU: TU_askErase}]->(d_target:Artifact)`
- **Suppression :** `(p_delete:Process {action: 'delete'})-[u_delete:USED {TU: TU_delete}]->(d_target)`

Paramètres Cypher : `$currentTime` , `$erasureLimitDuration`.

Demande Cypher Conçue (pour trouver les violations) :

```
// Étape 1: Identifier toutes les demandes d'effacement
MATCH (p_ask:Process)-[u_ask:USED]->(d_target:Artifact)
WHERE p_ask.action = 'askErase'
WITH d_target, p_ask, u_ask.TU AS TU_askErase
```

// Étape 2: Filtrer uniquement celles dont le délai d'action est dépassé

WHERE ($currentTime - TU_askErase) >= $erasureLimitDuration

// Étape 3: Chercher les suppressions éventuelles et trouver la plus rapide

OPTIONAL MATCH (p_delete:Process)-[u_delete:USED]->(d_target)

WHERE p_delete.action = 'delete' AND u_delete.TU > TU_askErase

WITH d_target, p_ask, TU_askErase, COLLECT(u_delete.TU) AS deletionTimes

WITH d_target, p_ask, TU_askErase,

CASE WHEN size(deletionTimes) > 0 THEN min(deletionTimes) ELSE null END AS minDeletionTimestamp

// Étape 4: Déterminer la violation : soit jamais supprimé, soit supprimé trop tard

WHERE minDeletionTimestamp IS NULL OR (minDeletionTimestamp - TU_askErase >= $erasureLimitDuration)

// Étape 5: Retourner les informations de la violation

RETURN DISTINCT

d_target.name AS D,

TU_askErase AS T,

p_ask.name AS P

=> Champs Retournés pour `Issue`**:** `D` (données concernées), `T`(horodatage de la demande), `P`(processus de la demande).

3. Droit d'Accès:

Logique Prologue Essentielle :

- rightAccess(S,TE)signaler une violation si une demande d'accès (action askDataAccess par S à TE, générant A) n'est pas "Ok" (\+ rightAccessOk(P,A,TE)).
- rightAccessOk(P,A,TE)est vrai si le délai pour répondre n'est pas encore écoulé OU si les données ont été envoyées à temps (par P2 avec action:'sendData', et TE2_envoi - TE < $accessLimitDuration).

Éléments Neo4j Pertinents :

- **Demande :** (s:Agent)<-[wcb_ask:WAS_CONTROLLED_BY {ctx:'owner', TE:TE_demande}]-(p_ask:Process {action:'askDataAccess'}) ,(a_req:Artifact)-[:WAS_GENERATED_BY]->(p_ask)
- **Envoi :** (agent_sys)<-[wcb_send:WAS_CONTROLLED_BY {TE:TE_envoi}]-(p_send:Process {action:'sendData'}) ,(p_send)-[:USED]->(a_req)

Paramètres Cypher : $currentTime , $accessLimitDuration.

Demande Cypher Conçue (pour trouver les violations) :

```
// Étape 1: Identifier les demandes d'accès faites par le propriétaire

MATCH (s_demandeur:Agent)<-[wcb_ask:WAS_CONTROLLED_BY]-
(p_demande:Process {action: 'askDataAccess'})

WHERE wcb_ask.ctx = 'owner'

MATCH (a_demande:Artifact)-[:WAS_GENERATED_BY]->(p_demande)

WITH s_demandeur, p_demande, a_demande, wcb_ask.TE AS TE_demande


// Étape 2: Filtrer les demandes dont le délai de réponse est dépassé

WHERE ($currentTime - TE_demande) >= $accessLimitDuration


// Étape 3: Appliquer la logique de violation

AND NOT EXISTS { // ET il n'existe PAS d'envoi de données fait à temps en réponse à cette demande

  MATCH (p_envoi:Process {action: 'sendData'})-[u_send:USED]->(a_demande)
```

MATCH (agent_repondant:Agent)<-[wcb_send:WAS_CONTROLLED_BY]-(p_envoi)

WHERE wcb_send.TE > TE_demande // L'envoi doit être postérieur à la demande

AND (wcb_send.TE - TE_demande) < $accessLimitDuration // ET l'envoi doit être fait DANS les délais

}

// Étape 4: Retourner les informations de la violation

RETURN DISTINCT

s_demandeur.name AS S,

TE_demande AS TE

=> Champs Retournés pour Issue: S (sujet demandeur), TE(horodatage de la demande).

4. Consentement (Legal) :

Logique Prologue Essentielle :

- `legal(P,D,PU,T,DP,C)` signaler une violation si une donnée personnelle `DP`(issue de `D`) utilisée par `P`pour la finalité `PU`à `T`n'a pas de consentement valide (`\+ consentFoundOk(C,DP,PU,T)`).
- `consentFoundOk`est vrai si `PU`est une finalité par défaut OU si un consentement explicite existe, pertinent, donné avant `T`, "dernier" actif et non révoqué avant `T`.

Cartographie Neo4j Clé (basée sur `PrologToGraphDB.java`) :

- **Utilisation :** `(p:Process {action:PU})-[u:USED {TU:T}]->(d:Artifact)`
- **Source personnelle donnée :** `(d)-[:WAS_DERIVED_FROM*0..]->(dp:Artifact {type:'personal_data'})`
- **Consentement :** `(c:Artifact {name: C, consent_type:'purposes_consent'})`
- **Finalités :** Propriété dynamique `c[dp.name + "_purposes"]`(liste de chaînes).
- **Génération du consentement :** `(c)-[wgb_c:WAS_GENERATED_BY {TG: T_consent}]->(process)`
- **Révocation :** `(p_revoke:Process {action:'revokeConsent'})-[u_revoke:USED {TU:T_revoke}]->(c)`

Paramètres Cypher : `$defaultPurposesList` (liste de chaînes).

Demande Cypher Conçue (pour trouver les violations) :

```
// Étape 1: Identifier toutes les utilisations de données personnelles
MATCH (p_using_data:Process)-[u:USED]->(d_artifact_used:Artifact)
MATCH (d_artifact_used)-[:WAS_DERIVED_FROM*0..]->(dp_source:Artifact {type: 'personal_data'})
WITH p_using_data, dp_source, p_using_data.action AS PU, u.TU AS T_utilisation

// Étape 2: Exclure les finalités qui n'exigent pas de consentement explicite
WHERE NOT PU IN $defaultPurposesList

// Étape 3: Vérifier qu'il N'EXISTE PAS de consentement valide pour cette utilisation précise
AND NOT EXISTS {
  // A. Chercher un consentement pertinent (donné avant, pour la bonne donnée/finalité)
  MATCH (c:Artifact {consent_type: 'purposes_consent'})-[wgb_c:WAS_GENERATED_BY]->(:Process)
  WHERE wgb_c.TG < T_utilisation
   AND c[dp_source.name + "_purposes"] IS NOT NULL AND PU IN c[dp_source.name + "_purposes"]

  // B. S'assurer qu'il n'a pas été révoqué avant l'utilisation
  AND NOT EXISTS {
    MATCH (c)<-[u_revoke:USED]-(p_revoke:Process {action: 'revokeConsent'})
    WHERE u_revoke.TU < T_utilisation AND u_revoke.TU >= wgb_c.TG
  }

  // C. S'assurer qu'il s'agit du "dernier" consentement pertinent (pas de consentement plus récent et valide)
  AND NOT EXISTS {
    MATCH (c_next:Artifact {consent_type: 'purposes_consent'})-[wgb_c_next:WAS_GENERATED_BY]->(:Process)
    WHERE c_next <> c
     AND wgb_c_next.TG > wgb_c.TG AND wgb_c_next.TG < T_utilisation
```

```
        AND c_next[dp_source.name + "_purposes"] IS NOT NULL AND PU IN
c_next[dp_source.name + "_purposes"]
        // Et ce consentement plus récent ne doit pas lui-même être révoqué
        AND NOT EXISTS {
          MATCH (c_next)<-[u_revoke_next:USED]-(:Process {action: 'revokeConsent'})
          WHERE u_revoke_next.TU < T_utilisation AND u_revoke_next.TU >=
wgb_c_next.TG
        }
    }
}

// Étape 4: Retourner les informations de la violation
RETURN
    p_using_data.name AS P,
    dp_source.name AS DP,
    PU,
    T_utilisation AS T
```

=> Champs Retournés pour Issue: P (processus), DP(données personnelles source, à aliaser D), PU(finalité), T(temps d'utilisation).

Conclusion de la Phase de Conception

Ce rapport a détaillé la traduction de la logique des patterns Prolog RGPD en conceptions de requêtes Cypher pour les quatre principes étudiés. Pour chaque principe, nous avons :

- Rappelé la logique Prolog.
- Identifié la structure de données Neo4j correspondante basée sur `PrologToGraphDB.java`.
- Définissez l'objectif de la requête Cypher et ses paramètres.
- Proposez une requête Cypher robuste pour identifier les non-conformités.
- Spécifié les informations à retourner pour les objets `Issue`.

Les prochaines étapes critiques incluent la finalisation syntaxique fine de ces requêtes, la confirmation des champs de retour pour l'intégration Java, et surtout, des tests rigoureux de chaque requête sur des jeux de données Neo4j représentatifs (générés via `PrologToGraphDB`) pour valider leur exactitude dans divers scénarios de conformité et de non-conformité.