PFIA 2023 Plate-forme Intelligence Artificielle

APIA

7 juillet 2023
Strasbourg

Hybridation opérationnelle des logiques OWL2 et ASP pour améliorer l'expressivité déclarative

Xavier GOBLET (Jeolis Solutions) - Christophe REY (LIMOS – UCA) – Adrien COLLANGE (LIMOS + Jeolis Solutions)

Avec l'aide de France Relance R&D

LIMOS

jeolis solution

Agenda

1 **Introduction**

2 **Travaux antérieurs**

- Oraloos v1
- Combiner des langages logiques
- Les approches opérationnelles

3 **HexLite OWLAPI**

- Principes
- Intégration pour Oraloos v2
- Limites

4 **Exialis: notre proposition**

- Ses objectifs
- Intégration pour Oraloos v3
- Vers une généralisation

5 **Conclusion**

01

INTRODUCTION

Contexte applicatif: e-ETP & ORALOOS

> *déf. OMS : Aider les patients à acquérir ou maintenir les compétences dont ils ont besoin pour gérer au mieux leur vie avec une maladie chronique*

Applications d'Education Thérapeutique du Patient

- Suivi régulier sur un temps long
- Enjeux: engagement et motivation des patients

Proposer des activités ludiques et un parcours personnalisé

- Défis = tâches personnelles de la TCC
- Réaliser des taches de + en + difficiles en fonction du retour patient…

Outil de Recommandations d'Activités Ludiques basé sur une Ontologie Opérationnelle de Suivi [APIA 2020]

Contexte article

> Hybridation opérationnelle des logiques OWL2 et ASP pour améliorer l'expressivité déclarative

Programmation logique déclarative

1. Utilisateur décrit le ''Quoi''
2. ''Comment'' & ''Quand'' délégués au moteur de raisonnement, i.e. Prolog ou Answer Set Programming (ASP)

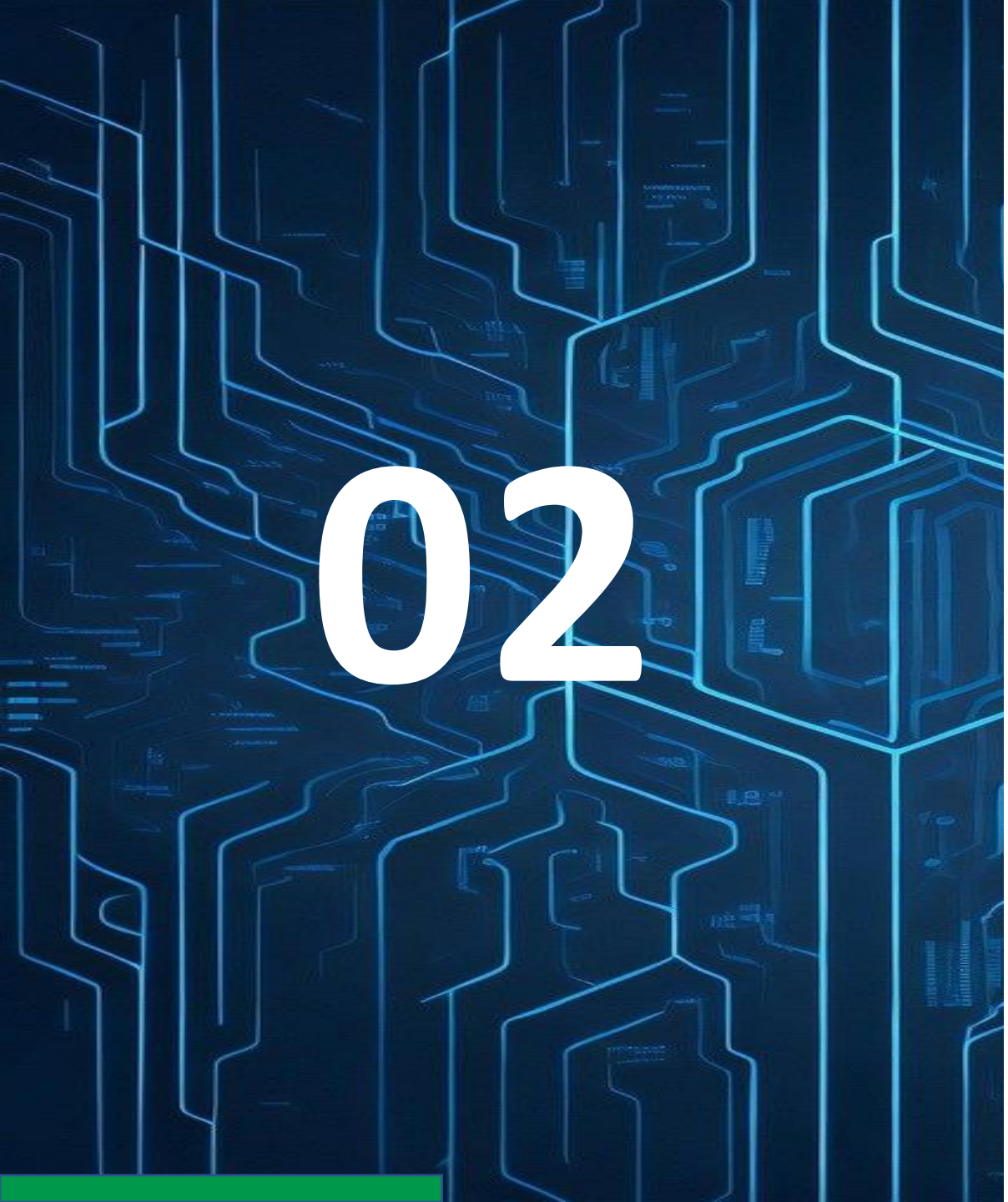
=> Facilite l'ajout de nouvelles règles métier, nouveaux concepts…

Hybridation

- De multiples langages logiques, chacun avec +/-
- Les combiner en gardant le meilleur est une bonne pratique

Opérationnelle

- Démarche pragmatique dans un contexte industriel



02

TRAVAUX ANTERIEURS

ORALOOS V1 – modélisation déclarative

Ontologie OWL2

(Web Ontology Language – 2009)

- Connaissances statiques
- Logique de description (LD)

1. TBox = définition de concepts, relations entre concepts (Object Property) ou intra concept (Data Property), héritage (spécialisation)…
2. ABox = instances de concepts (individus)

Avantages: description terminologique et hiérarchique, standard W3C, raisonnement par classification…

Inconvénients: une expertise en LD nécessaire, raisonnement en monde ouvert…

Exemples

- **Concepts:** Challenge, Patient, Feedback, RelationCond subclassOf Condition
- **OP:** concernsChallenge(Feedback, Challenge)
- **DP:** hasLevel(Challenge, int)

ORALOOS V1 – modélisation déclarative

Ontologie OWL2

(Web Ontology Language – 2009)

- Connaissances statiques
- Logique de description (LD)

1. TBox = définition de concepts, relations entre concepts (Object Property) ou intra concept (Data Property), héritage (spécialisation)…
2. ABox = instances de concepts (individus)

Avantages: description terminologique et hiérarchique, standard W3C, raisonnement par classification…

Inconvénients: une expertise en LD nécessaire, raisonnement en monde ouvert…

SWRL

(Semantic Web Rule Language - 2004)

- Connaissances dynamiques (comportements)
- Règle: [Prémisses] -> [Conclusion] où prémisses et conclusion sont des expressions logiques combinant (ET logique) des concepts, propriétés et individus d'une ontologie.

Avantages: simple, efficace

Inconvénients: pas de négation, pas de OU, pas standard, raisonnement monotone…

ORALOOS V1 – modélisation déclarative

SWRL
(Semantic Web Rule Language - 2004)

- Connaissances dynamiques (comportements)
- Règle: [Prémisses] -> [Conclusion] où prémisses et conclusion sont des expressions logiques combinant (ET logique) des concepts, propriétés et individus d'une ontologie.

Avantages: simple, efficace
Inconvénients: pas de négation, pas de OU, pas standard, raisonnement monotone…

Exemple:

R1: Feedback(?fb) ^ concernsChallenge(?fb, ?ch) ^ hasLevel(?ch, ?lv) ^ usable(?fb, ?nch) ^ swrlb:greaterThan(?nlv, ?lv) ^ hasLevel(?nch, ?nlv) ^ hasSuccess(?fb, true) -> reinforceLevel(?fb, ?nch)

Si le feedback du patient pour le défi courant est positif, et s'il existe un autre défi de difficulté supérieur et utilisable dans le contexte du feedback, alors ce défi sera proposé au patient comme renforçant sa progression

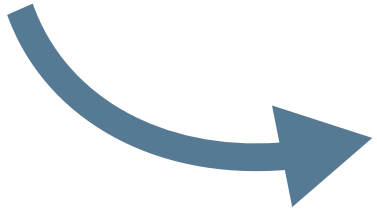
ORALOOS V1 – code impératif

01 Limitations palliées par Owlready2 [Lamy 2017]

- Manipulation CRUD d'ontologies sous la forme d'objets
- Inclut nativement une base de données + les raisonneurs Pellet ou HermiT

02 Les concepts OWL = classes que l'on peut étendre

Extension du Feedback pour calcul usable()



- Les relations usable() d'un Feedback sont calculées dynamiquement en amont du déclenchement SWRL par un filtre

Avec DC le défi courant et UC l'utilisateur courant,

1. usable = tous les défis de même domaine que DC
2. usable = usable – DC
3. usable = usable – (tous les défis déjà réalisés par UC)
4. usable = usable – (tous les défis dont les conditions sont fausses)

- Les 3 dernières étapes sont difficiles à exprimer en OWL2 et SWRL

ORALOOS V1 – code impératif

01 Limitations palliées par Owlready2 [Lamy 2017]

- Manipulation CRUD d'ontologies sous la forme d'objets
- Inclut nativement une base de données + les raisonneurs Pellet ou HermiT

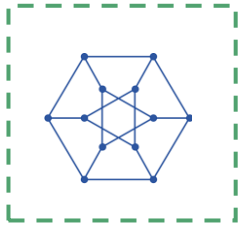
02 Les concepts OWL = classes que l'on peut étendre

Extension du Feedback pour calcul usable()

03 Résultats

- Simple, efficace MAIS impacte le caractère déclaratif portée par OWL2 et SWRL
- 1er objectif : limiter ce code impératif

Combiner les ontologies & programmation logique



Etudiée depuis + 30 ans

- Représentation des connaissances et raisonnements (KRR) avec beaucoup de résultats théoriques surtout concentrés sur l'interrogation ontologique des bases de données
- En se focalisant sur OWL2 et ASP (Answer Set Programming), les approches opérationnelles sont plutôt récentes.

Pourquoi ASP ?

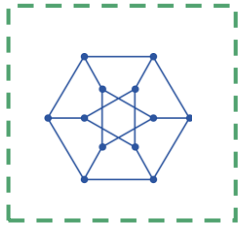
Paradigme de **programmation déclarative** utilisé pour résoudre des problèmes de **raisonnement logique** et de **satisfaction de contraintes**

- Programme ASP = ensemble de règles logiques, de faits et de contraintes
- Objectif = trouver les ensembles d'assignations de valeurs (answer set) aux variables du programme qui satisfont toutes les règles et contraintes
- Résolution ''one-shot'' : Ground → Solve
- Solveurs ''académique & industriel'' : DLV, Clingo

Avantages: langage logique complet (conjonction, disjonction, 2 négations) + contraintes, raisonnement non monotone, complétude…

Inconvénients: écrire un programme ASP, comprendre les modèles stables demandent une forte expertise …

OWL2 + ASP: approches opérationnelles récentes



DL-programs

env. 2008

ASP étendu par des sources externes RDF/OWL

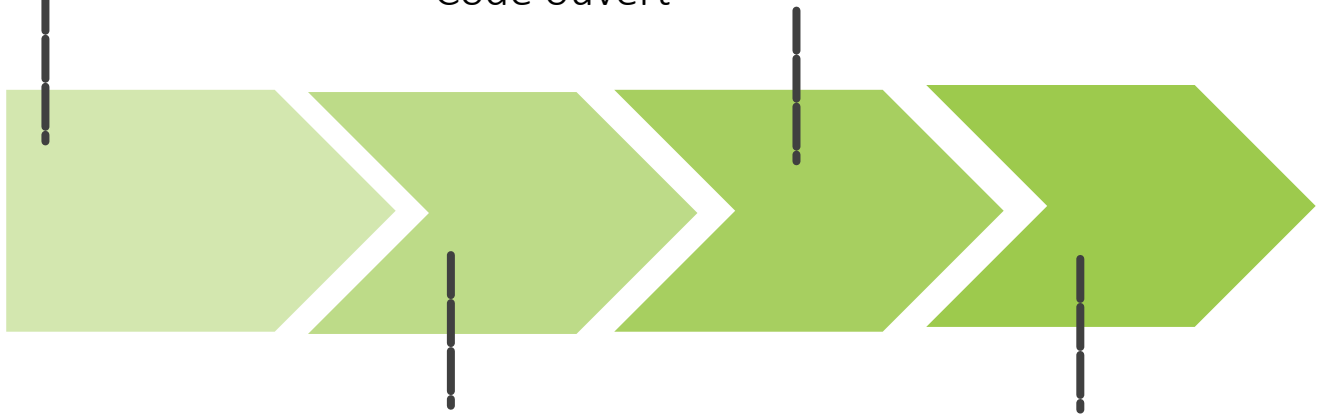
HexLite

Schüller, 2019

Implémentation en Python d'un fragment de HEX

Solveur ASP: Clingo

Code ouvert



HEX-programs

Eiter et al., 2017

High-order logic with External atoms

Solveur ASP: DLV

dlvhex: implémentation de référence en C++

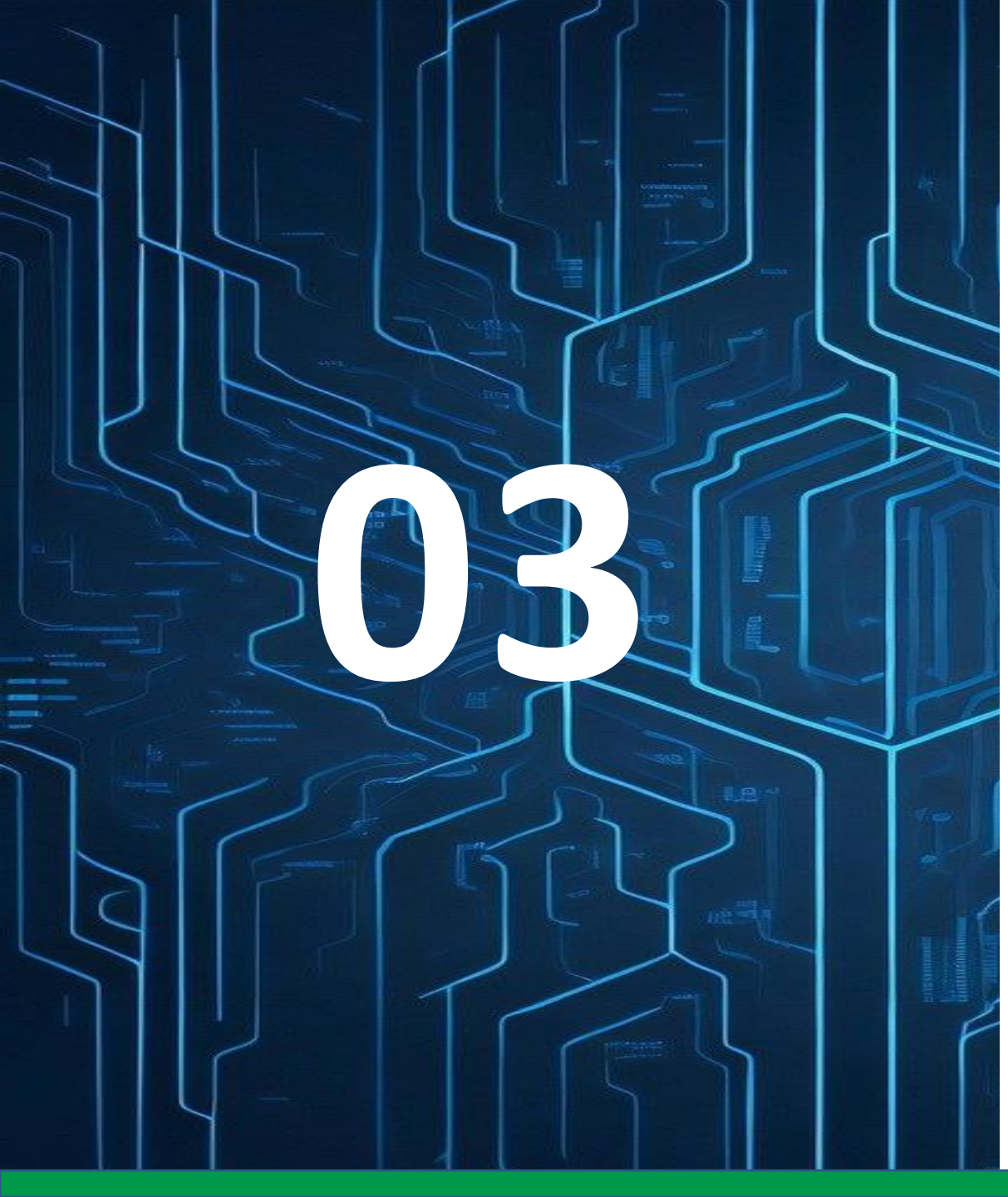
HexLite OWLAPI

Schüller, 2020

Interface bidirectionnelle OWL2 <-> ASP

Via interface Java OWLAPI

Code ouvert



HexLite OWLAPI

HexLite OWLAPI – principes

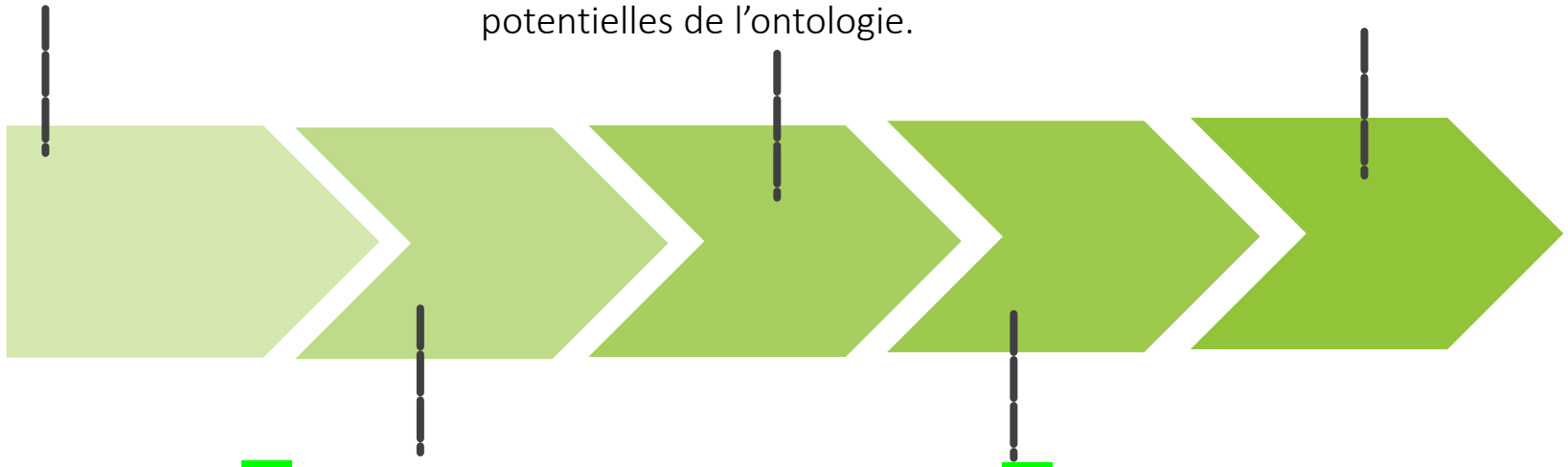
1- OWL2
Description concepts, propriétés…
Individus instanciés.

2- ASP
Lecture d'individus dans l'ontologie alimente en faits.

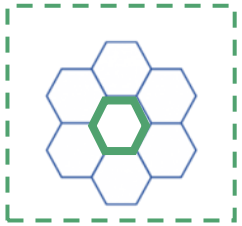
3- ASP
Résolution dynamique génère des nouveaux faits => modifications potentielles de l'ontologie.

4- OWL2
Vérifie la consistance des modifications par raisonnement.

5- ASP
Si consistante alors propagation effective des modifications par écriture.



HexLite OWLAPI – principes



1- OWL2
Description concepts, propriétés…
Individus instanciés.

3- ASP
Résolution dynamique génère des nouveaux faits => modifications potentielles de l'ontologie.

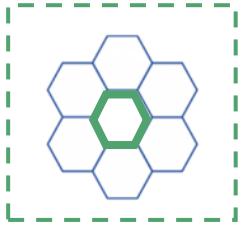
5- ASP
Si consistante alors propagation effective des modifications par écriture.

A RETENIR: les modifications dans l'ontologie portent sur les individus et pas sur la description

2- ASP
Lecture d'individus dans l'ontologie alimente en faits.

4- OWL2
Vérifie la consistance des modifications par raisonnement.

HexLite OWLAPI – interface bidirectionnelle



- **Atomes externes de lecture**

	Retourne
&dlCro[onto, C](I) => &dlCro[onto, ''Challenge''](C)	⟶ Tous les individus Challenge
&dlOPro[onto, OP](I1, I2) => &dlOPro[onto, ''succeeds''](''pat5'', C)	⟶ Tous les défis réussis par le patient 5
&dlDPro[onto, DP](I, D) => &dlDPro[onto, ''hasDomain''](C, ''Nutrition'')	⟶ Tous les défis du domaine Nutrition

- **Atomes externes d'écriture**

Prérequis: des atomes ASP delta portant les modifications

delta(T, <op>) avec op: addc(C,I), delc(C,I), addop(OP,I1,I2), delop(OP,I1,I2), etc

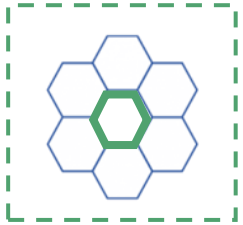
Écriture: update puis query

&dlC[onto, delta, T, C](I) , &dlOP[onto, delta, T, OP](I1,I2), &dlDP[onto, delta, T, DP](I,D)

- **Avant d'appliquer le delta, vérifier consistance de l'ontologie**

&dlConsistent[onto, delta, T]() ⟶ True ou False

ORALOOS V2 – Feedback avec HexLite OWLAPI



code Python + SWRL

```python
class Feedback(Thing):
    def preFilter(self, core, app):
        dom = self.concernsChallenge.hasDomain
        forbids = self.concernsPerson.forbids
        usables = app.search(type=core.Challenge, hasDomain=dom)
        # le defi courant doit etre enleve
        current = self.concernsChallenge
        usables.remove(current)
        # on enlève tous les défis interdits (déjà joués)
        if len(forbids) != 0:
            cp_usables = usables.copy()
            for df in cp_usables:
                if df in forbids: usables.remove(df)
        # on traite maintenant les conditions des défis usables
        for df in usables:
            conds = df.hasCondition
            if conds is None or len(conds) == 0 or self.check(conds, self):
                self.usable.append(df)
        return True

    def check(self, conditions, context):
        for cond in conditions:
            cond.hasContext = context
            if not cond.execute(): return False
        return True

class ClassCond(Thing):
    def execute(self):
        classe = self.hasObject[0].__class__
        subject = self.hasContext.concernsPerson
        if subject == None: return False
        subclasses = subject.is_a
        if classe in subclasses: return True
        return False
```

R1: Feedback(?fb) ^ concernsChallenge(?fb, ?ch) ^ hasLevel(?ch, ?lv) ^ usable(?fb, ?nch) ^ swrlb:greaterThan(?nlv, ?lv) ^ hasLevel(?nch, ?nlv) ^ hasSuccess(?fb, true) -> reinforceLevel(?fb, ?nch)

R2: Feedback(?fb) ^ concernsChallenge(?fb, ?ch) ^ hasLevel(?ch, ?lv) ^ usable(?fb, ?nch) ^ swrlb:lessThan(?nlv, ?lv) ^ hasLevel(?nch, ?nlv) ^ hasSuccess(?fb, false) -> remedyLevel(?fb, ?nch)

R3: Feedback(?fb) ^ concernsChallenge(?fb, ?ch) ^ hasLevel(?ch, ?lv) ^ usable(?fb, ?nch) ^ hasLevel(?nch, ?lv) -> sameLevel(?fb, ?nch)

Etapes 1, 2 et 3 du filtre:

- usable = tous les défis du même domaine
- usable = usable - défi courant
- usable = usable - (les défis déjà joués par le patient)

Etape 4

- usable = usable - (les défis dont les conditions sont fausses)

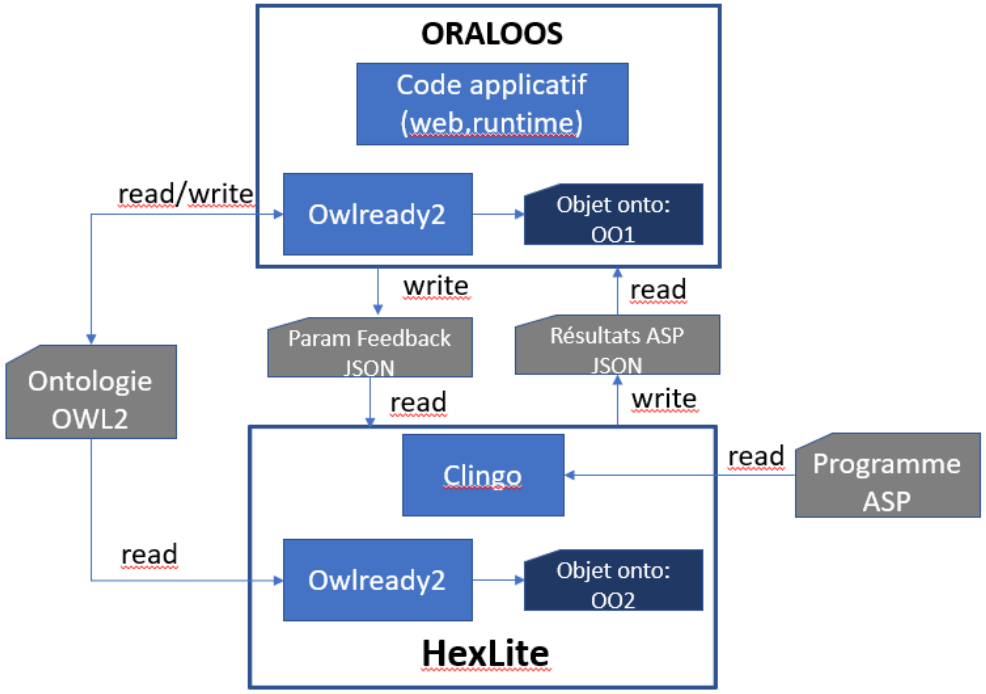
En sortie du filtre, équivalent SWRL

code HexLite

```
#const onto="in/oraloos-meta.json".
% === INIT from ontology ===
challenge(Challenge) :- &dlCro[onto, "Challenge"](Challenge).
domain(Challenge, Domain) :- &dlDPro[onto, "hasDomain"](Challenge, Domain).
forbids(Person, Challenge) :- &dlOPro[onto, "forbids"](Person, Challenge).

has_condition(Challenge, Object) :- &dlOPro[onto, "hasCondition"](Challenge, Condition),
    &dlDPro[onto, "hasVerb"](Condition, Verb), &dlOPro[onto, "hasObject"](Condition, Object),
    &dlOPro[onto, "hasSubject"](Condition, Subject), Verb = is_a.
has_condition(Challenge, Verb, Object) :- &dlOPro[onto, "hasCondition"](Challenge, Condition),
    &dlDPro[onto, "hasVerb"](Condition, Verb), &dlOPro[onto, "hasObject"](Condition, Object),
    &dlOPro[onto, "hasSubject"](Condition, Subject), Verb != is_a.
% === USABLE ===
usable_before(Challenge) :- challenge(Challenge), domain(Challenge, Domain), current_domain(Domain),
    not current_challenge(Challenge), current_user(User), not forbids(User, Challenge).
% Challenges without condition
usable(Challenge) :- usable_before(Challenge), not has_condition(Challenge, _), not has_condition(Challenge, _, _).

is_checked(Challenge, ClassCond) :- usable_before(Challenge), has_condition(Challenge, ClassCond), current_user(User),
    &dlGetType[onto, User](ClassCond).
is_checked(Challenge, Verb, RelationCond) :- usable_before(Challenge), has_condition(Challenge, Verb, RelationCond).
% Challenges with condition(s)
usable(Challenge) :- usable_before(Challenge), N={has_condition(Challenge, Condition)},
    M={is_checked(Challenge, Condition)}, N=M,  I={has_condition(Challenge, Verb, Condition)},
    J={is_checked(Challenge, Verb, Condition)}, I=J.
% === LEVEL MANAGEMENT ====
current_level(CurrentLevel) :- current_challenge(Challenge), &dlDPro[onto, "hasLevel"](Challenge, CurrentLevel).
usable_level(Challenge, UsableLevel) :- usable(Challenge), &dlDPro[onto, "hasLevel"](Challenge, UsableLevel).

reinforce_level(Challenge) :- usable_level(Challenge, UsableLevel), current_level(CurrentLevel),
    UsableLevel > CurrentLevel.
remedy_level(Challenge) :- usable_level(Challenge, UsableLevel), current_level(CurrentLevel),
    UsableLevel < CurrentLevel.
same_level(Challenge) :- usable_level(Challenge, UsableLevel), current_level(CurrentLevel),
    UsableLevel = CurrentLevel.
```

HEXLITE OWLAPI

ORALOOS V2 – Feedback avec HexLite OWLAPI



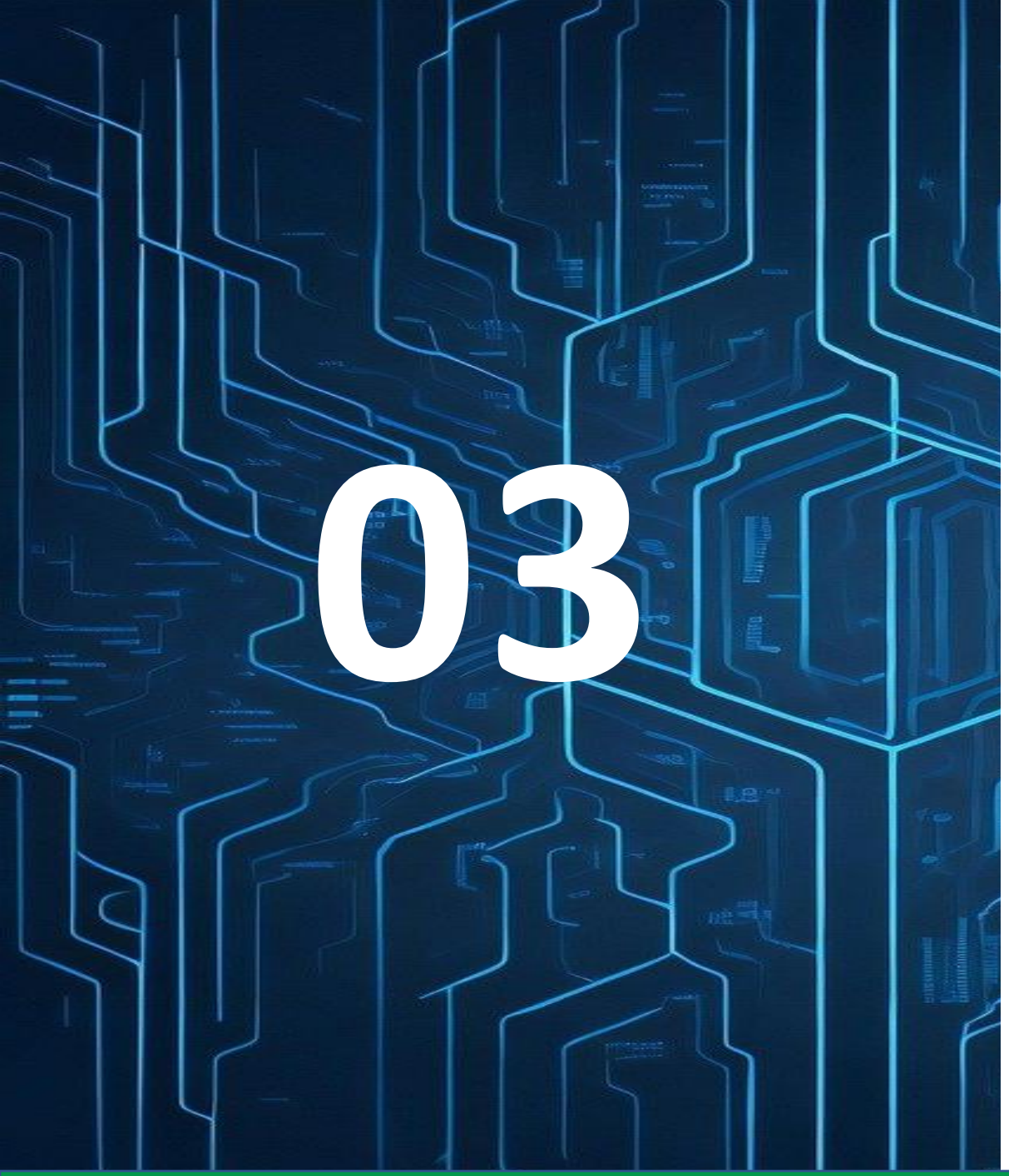
- **Intégration avec Appli Oraloos**
  - HexLite: processus/script Python autonome, indépendant
  - Communication inter processus avec échange de fichiers d'entrée et de sortie

✓ Mise en pratique difficile (déploiement, peu documenté, dépôt sources non maintenu)

✓ Extension possible par plugin: réécriture avec Owlready2, uniquement atomes externes de lecture

✓ L'ontologie est chargée 2 fois => 2 objets ontologie distincts

- **HexLite OWLAPI:** bien fonctionnellement mais difficile à utiliser avec une application tierce et architecture complexe
  - Du pre-processing nécessaire pour partitionner les atomes externes dans un cycle « one-shot » ASP



03

EXIALIS

Notre proposition

Objectifs

01 Fonctionnellement comme HexLite OWLAPI: interface bidirectionnelle OWL2 (via Owlready2) et ASP (via Clingo)

- Traduire les atomes externes en fonctions externes Clingo
- Même principe du delta pour la propagation des modifications

02 Plus simplement que Hexlite en utilisant des fonctionnalités natives Clingo [Clingo 2021]

- Multi-Shot & Incremental Solving

03 Intégrable et opérationnelle avec une application tierce

EXIALIS

Multi-Shot Solving Clingo (MSS)

- **Objectif du MSS** vs. cycle « standard » ASP
  - ✓ Résoudre des problèmes logiques dont le domaine d'évolution est dynamique, sans repartir de zéro
  - ✓ Entrelacer plusieurs cycles (Ground → Solve)*
- **Programme ASP** partitionné en sous-programmes via directive
  - #program name(p*1*,…,p*k*) avec p*i* des paramètres d'entrée
    - Par défaut, #program base sans paramètres
- API Clingo spécifie quand **ground()+solve() les #program**
  - ✓ Ground s'applique aux #program donnés en argument
  - ✓ Solve déclenche un raisonnement par rapport à toutes les règles « ground » accumulées

EXIALIS

ORALOOS V3 – Feedback avec MSS

Programme base pour initialisation

Programme step paramétré pour un individu

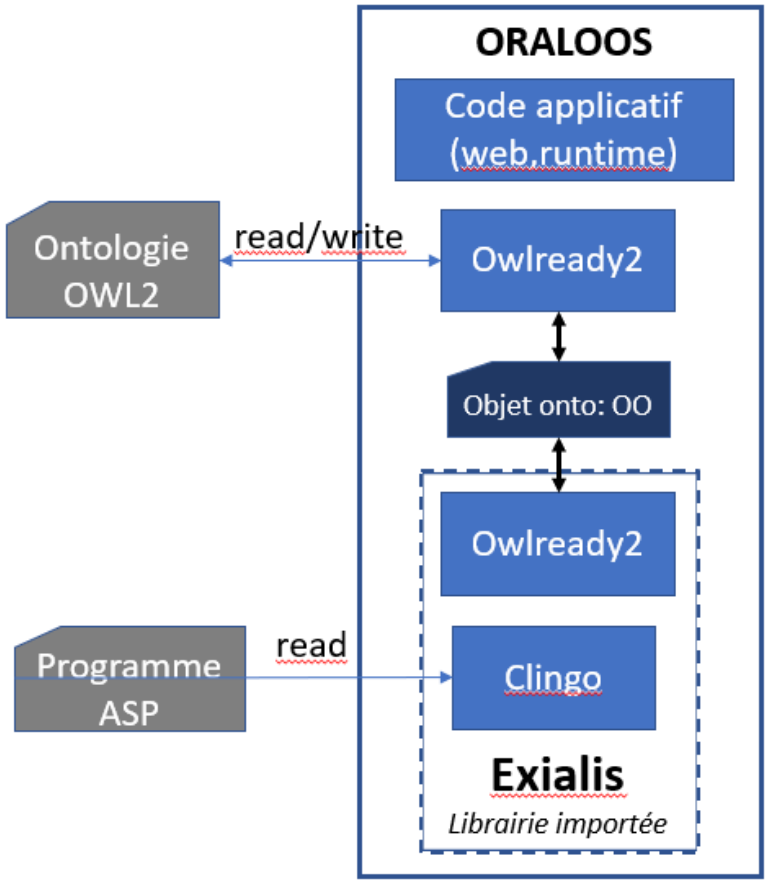
```
#program base.
#const onto="in/oraloos-meta.json".
% Initialisation de données non dépendantes du feedback
% par ex. tous les défis
challenge(@dlCro(onto, "Challenge",()).
...

#program step(feedbackID).
% Récupération de données dépendantes du feedback
% Par ex. l'utilisateur courant et le défi par lecture de propriétes
current_user(CU) :- @dlOPro(onto, "concernsPerson", ())=(feedbackID, CU).
current_challenge(CC) :- @dlOPro(onto, "concernsChallenge", ())=(feedbackID, CC).
current_level(CL) :- @dlDPro(onto, "hasLevel", ())=(Challenge, CL),
    current_challenge(Challenge).
...
% Régles ASP de calcul des nouveaux défis pertinents: usable(Challenge)
...
% Régles ASP de partitionnement en fonction du niveau du défi courant
reinforceLevel(Challenge) :- usable_level(Challenge, UsableLevel),
    current_level(Level), Usable_level > Level.
remedyLevel(Challenge) :- usable_level(Challenge, UsableLevel),
    current_level(Level), Usable_level < Level.
sameLevel(Challenge):- usable_level(Challenge, UsableLevel),
    current_level(Level), Usable_level = Level.

% delta de modifications par ajout de propriétés Objet au feedback
% le paramètre T est indifférent: variable anonyme ASP '_'
delta(_, addOP("reinforceLevel", feedbackID, Challenge)) :- reinforceLevel(Challenge).
% idem pour les propriétes remedyLevel et sameLevel

% Les modifications doivent laisser l'ontologie consistante => contrainte ASP
:- not @dlConsistent(onto, delta, _).

% Modifications effectives dans l'ontologie
reinforceBy(Challenge) :- @dlOP(onto, delta, _, "remedyLevel")=(feedbackID, Challenge).
% idem pour remedy et same
```

```
from clingo.symbol import String
from clingo.control import Control
ctl = Control()
ctl.load("feedback.lp")
ctl.ground([("base", [])])
ctl.solve(on_model=print)
ctl.ground([("step", [String(feedbackID)])])
ctl.solve(on_model=print)
```

Usage de l'API Control clingo

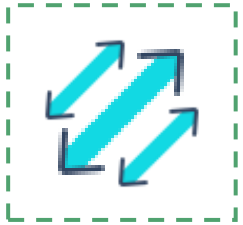
EXIALIS

ORALOOS V3 – Intégration Exialis



- **Intégration avec Appli Oraloos**
  - Exialis: librairie Python
  - Communication intra-processus avec échange par objet partagé
- Ontologie lue 1 seule fois =>objet ontologie permet le paramétrage et la propagation des résultats nativement
- **Exialis:** comme HexLite OWLAPI mais intégration facilitée et architecture plus simple

Vers une généralisation



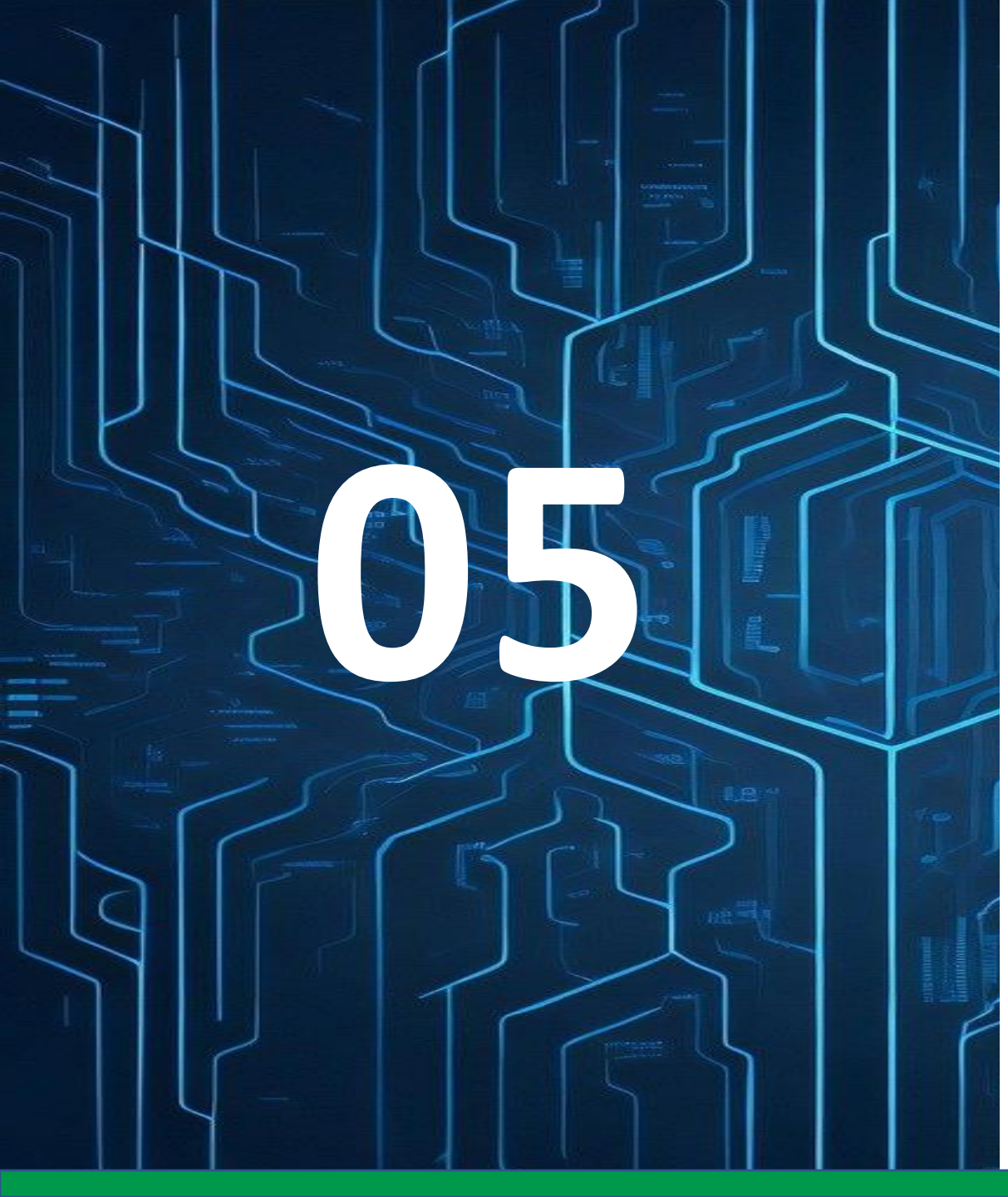
- Natif clingo: Incremental Solving
  - Extension du MSS en intégrant une temporalité de raisonnement : t = 0, 1, …, n

- Pattern encodage ASP:

  #program base → *Données et calcul ASP non dépendant de t…*
  #program step(t, …) → *Données et calcul ASP dépendant de t…*
  #program check(t, …) → *Contrainte(s) atteinte objectif(s) dépendant de t…*

A quoi ça sert ?

- Problèmes de planification, comme le « Box painting » dans Schüller (2020)
- Mais aussi la configuration de produits industriels, comme chez Siemens AG



05

CONCLUSION

Bilan et perspectives

- HexLite OWLAPI & Exialis améliore l'expressivité déclarative pour ORALOOS
- Capacités natives Clingo simplifie Exialis vs. HexLite
- Intégration facilitée pour Exialis

- Comparer les 3 versions d'ORALOOS
- Finaliser la généralisation par Incremental Solving
- Alternative(s) à IS ?

- Ambition finale: librairie Exialis en code ouvert, documentée et maintenue

Bibliographie

1

Clingo 2021

Roland Kaminski, Javier Romero, Torsten Schaub, and Philipp Wanko. How to build your own ASP-based system ? ! Theory and Practice of Logic Programming, page 1–63, 2021.

2

Schüller 2019 & 2020

The Hexlite Solver - lightweight and efficient evaluation of hex programs. In European Conference on Logics in Artificial Intelligence, 2019.
A new OWLAPI interface for HEX-programs applied to explaining contingencies in production planning. In Workshop at ECAI, 2020.

3

Eiter et al. 2017

Thomas Eiter, Tobias Kaminski, Christoph Redl, Peter Schüller, and Antonius Weinzierl. Answer set programming with external source access. In Reasoning Web, 2017.

4

Lamy 2017

Jean-Baptiste Lamy. Owlready : Ontology-oriented programming in python with automatic classification and high-level constructs for biomedical ontologies. Artificial Intelligence in Medicine, 80 :11–28, 2017.

5

APIA 2020

Xavier Goblet and Christophe Rey. Suivi thérapeutique intelligent par recommandation à base d'ontologie et de règles. Conférence Nationale sur les Applications Pratiques de l'Intelligence Artificielle, Afia (Ed), pages 50–57, 2020.

Merci Pour Votre Attention !