8th Distributed Ledger Technology Workshop (DLT2026)
1-6 June 2026 | Pula, Sardinia, Italy

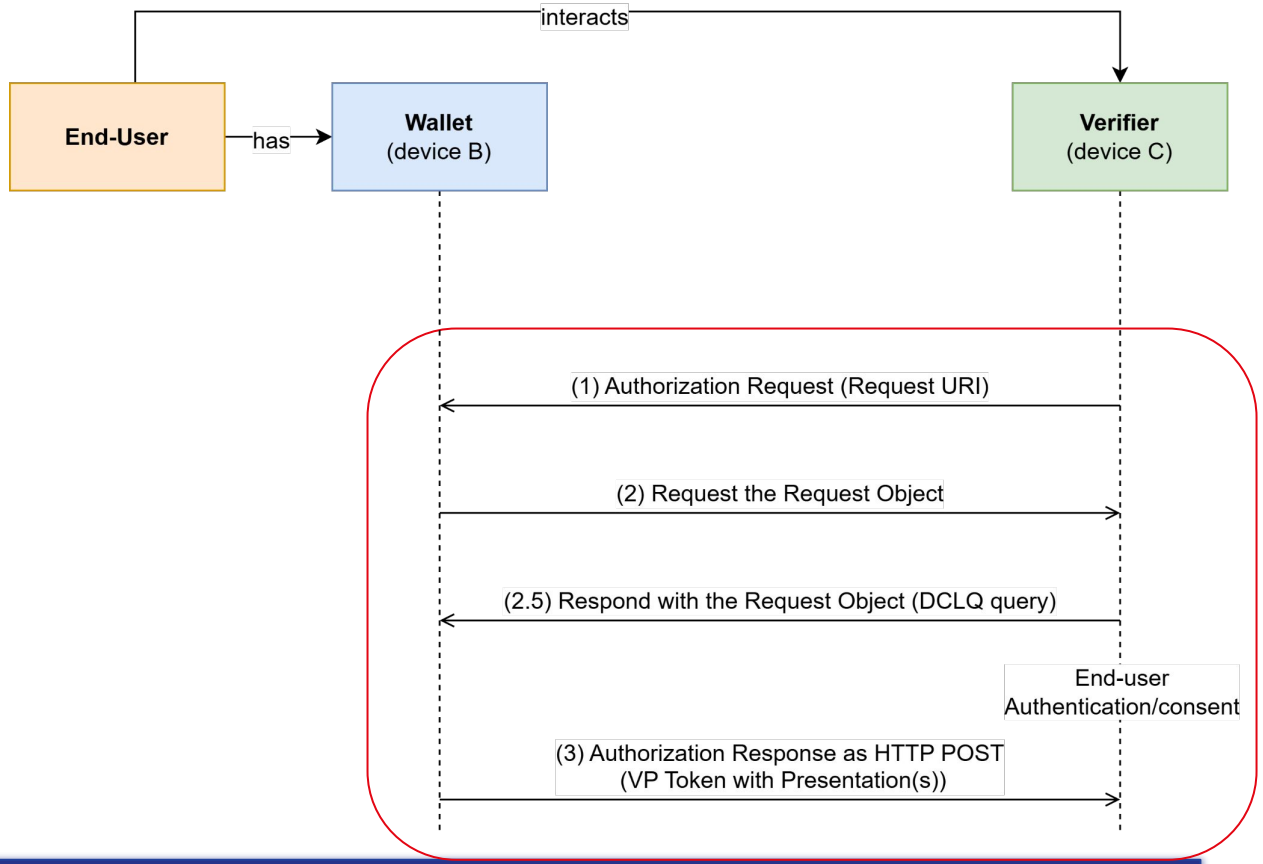
Semantic Reasoning for Verifiable Credential Validation in SSI Systems

Authors
Stefano Bistarelli[2]
Martin Diéguez[3]
Chiara Luchini[1,2]
Eric Monfroy[3]
Francesco Santini[2]

[1]Department of Mathematics and Computer Science "Ulisse Dini", University of Florence
[2]Department of Mathematics and Computer Science, University of Perugia
[3]Laboratoire d'Etude et de Recherche en Informatique d'Angers (LERIA), University of Angers

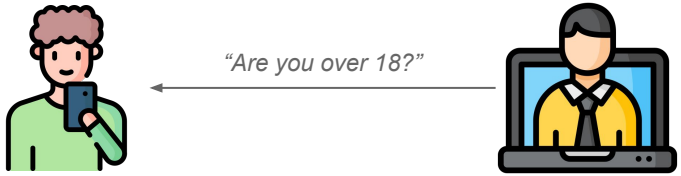
SSI and OID4VP

- The Verifier sends the Wallet an **Authorization Request** containing a `request_uri` pointing to the **Request Object.**
- The Wallet retrieves the **Request Object** by sending an HTTP GET request to the provided `request_uri`.
- The **Request Object** contains the **authorization parameters** and a **DCQL query** specifying the required credentials, formats, and claims. The **Wallet processes the request**, identifies matching credentials, authenticates the user, and collects consent.
- The Wallet generates the requested **presentation(s)** and sends them to the Verifier in an Authorization Response through the `vp_token` parameter.



How is a claim written?

```
…"credentialSubject": {
                "id":
"did:jwk:eyJrdHkiOi...m9aTkFPOEVncmsifQ",
                "given_name": "John",
                "family_name": "Doe",
                "email": "johndoe@example.com",
                "phone_number": "+1-202-555-0101",
                "address": {
                  "street_address": "123 Main St",
                  "locality": "Anytown",
                  "region": "Anystate",
                  "country": "US"
                },
                "birthdate": "1940-01-01",
                "is_over_18": true,
                "is_over_21": true,
                "is_over_65": true
            },
        …
```



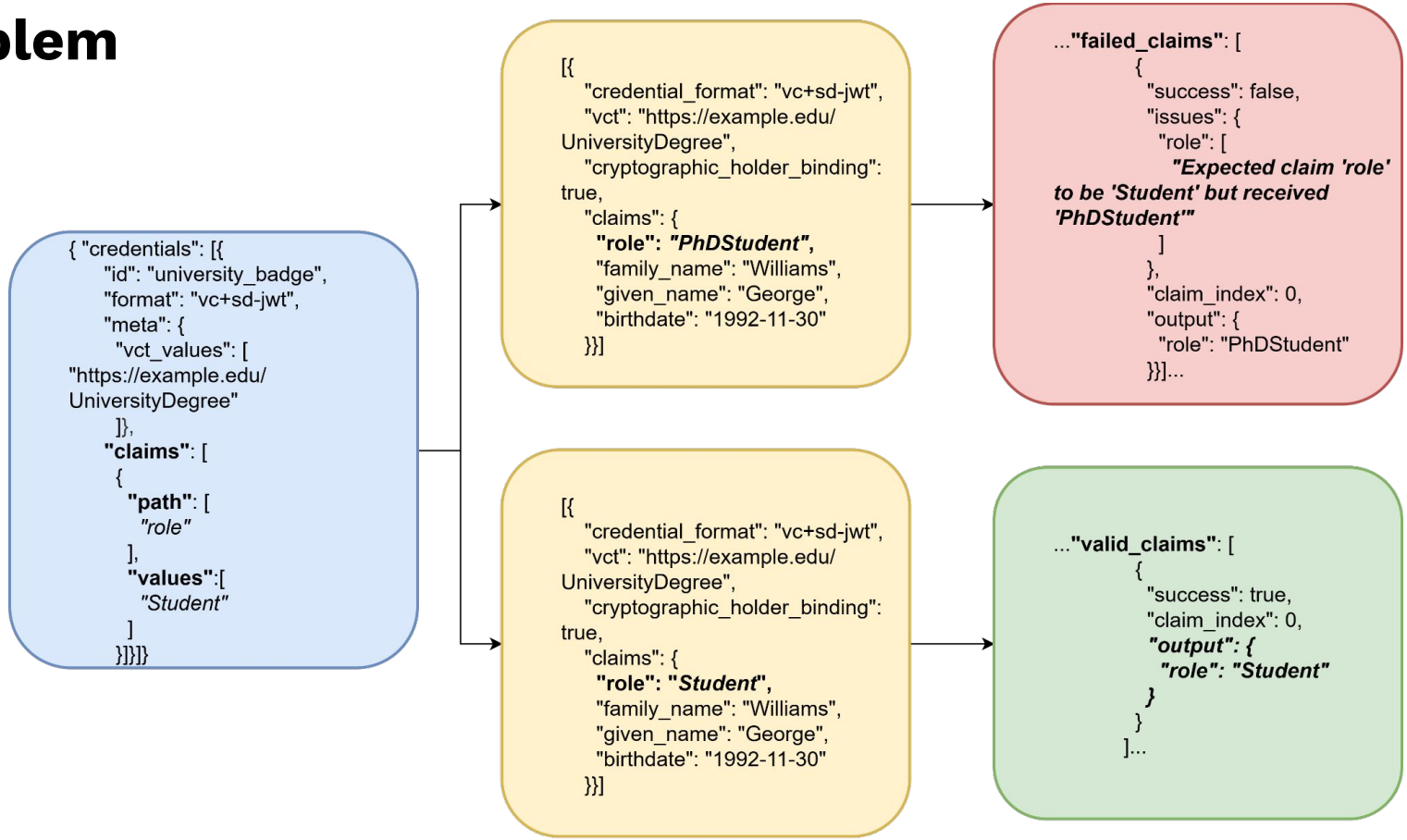
Claims can be of various **types** depending on the type of Credentials.
For instance an **Identity Credential** containing:

- given_name;
- family_name;
- email;
- phone_number;
- birthdate;
- address…

walt.id issuer demo: https://portal.walt.id/credentials?ids=IdentityCredential

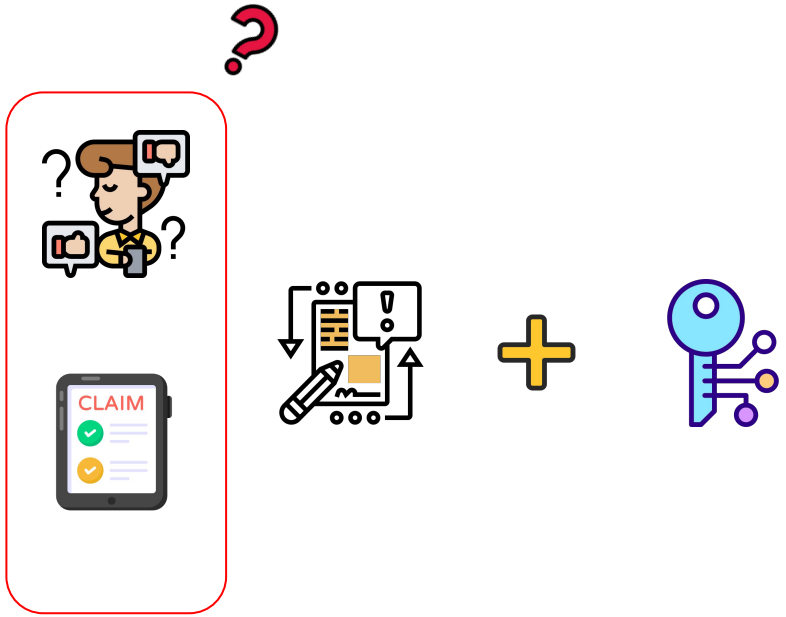
F. Paci, D. Bauer, E. Bertino, D. M. Blough, and A. Squicciarini. 2008. *"Minimal credential disclosure in trust negotiations."* In Proceedings of the 4th ACM workshop on Digital identity management (DIM '08). Association for Computing Machinery, New York, NY, USA, 89–96.

Problem


{ "credentials": [{
"id": "university_badge",
"format": "vc+sd-jwt",
"meta": {
"vct_values": [
"https://example.edu/
UniversityDegree"
]},
"claims": [
{
"path": [
"role"
],
"values":[
"Student"
]
}]}]}
[{
"credential_format": "vc+sd-jwt",
"vct": "https://example.edu/
UniversityDegree",
"cryptographic_holder_binding":
true,
"claims": {
"role": "PhDStudent",
"family_name": "Williams",
"given_name": "George",
"birthdate": "1992-11-30"
}}]
...."failed_claims": [
{
"success": false,
"issues": {
"role": [
"Expected claim 'role'
to be 'Student' but received
'PhDStudent'"
]
},
"claim_index": 0,
"output": {
"role": "PhDStudent"
}}]...
[{
"credential_format": "vc+sd-jwt",
"vct": "https://example.edu/
UniversityDegree",
"cryptographic_holder_binding":
true,
"claims": {
"role": "Student",
"family_name": "Williams",
"given_name": "George",
"birthdate": "1992-11-30"
}}]
..."valid_claims": [
{
"success": true,
"claim_index": 0,
"output": {
"role": "Student"
}
}]...

Framework for VC Reasoning

- **Idea:** combining constraint solver with VC claims.
- **Research Gap:** No semantic reasoning on the VC claims.
- **Goal:** Framework for semantic reasoning on VC.



owl2chr and Claim Validation

UniversityDegree DCQL query

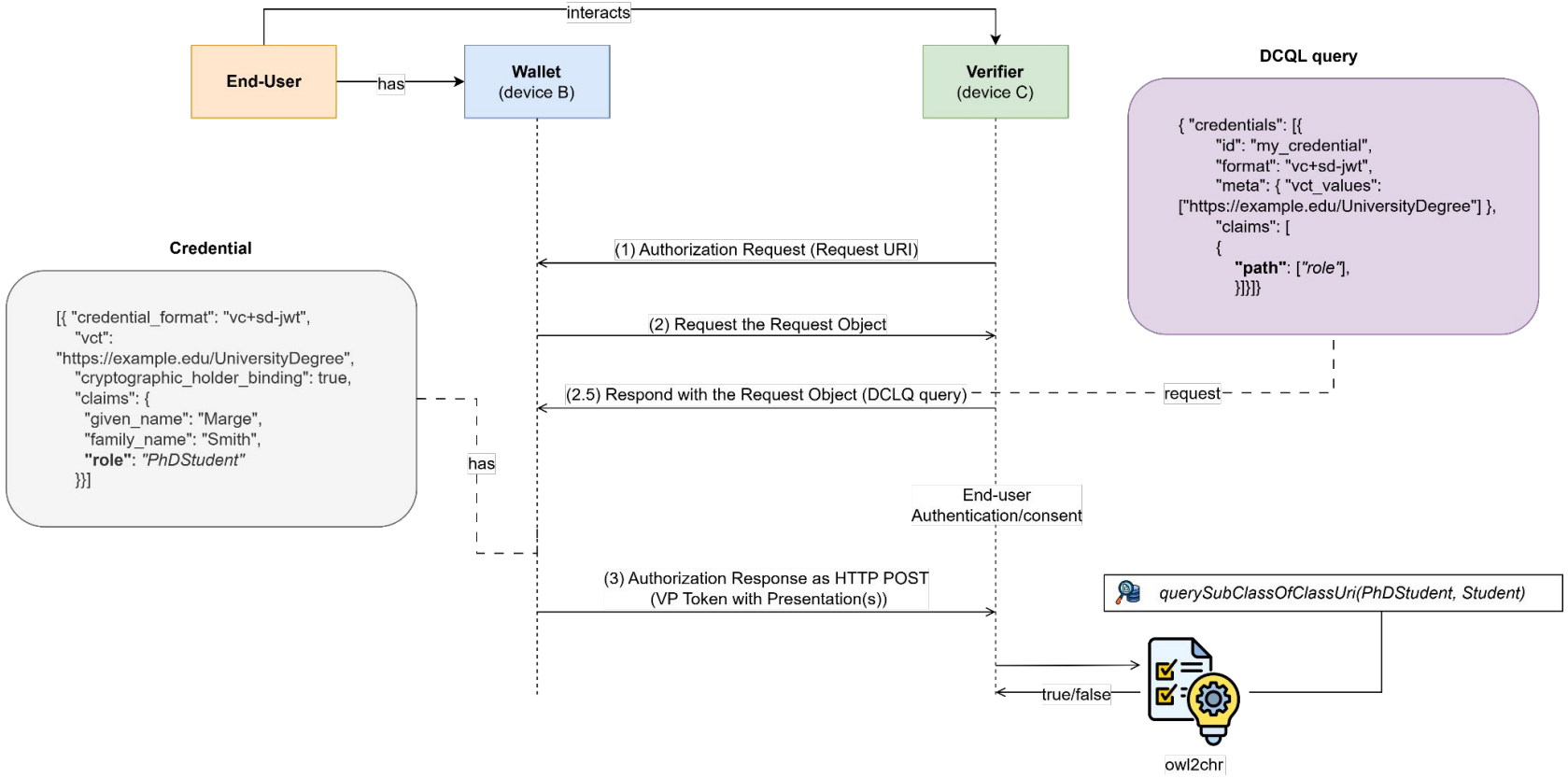
```
{"credentials": [{
        "id": "my_credential",
        "format": "vc+sd-jwt",
        "meta": { "vct_values":
["https://example.edu/UniversityDeg
ree"] },
        "claims": [
        {
            "path": ["role"]
        }
        ]
    }]}
```

UniversityDegree DCQL query + values

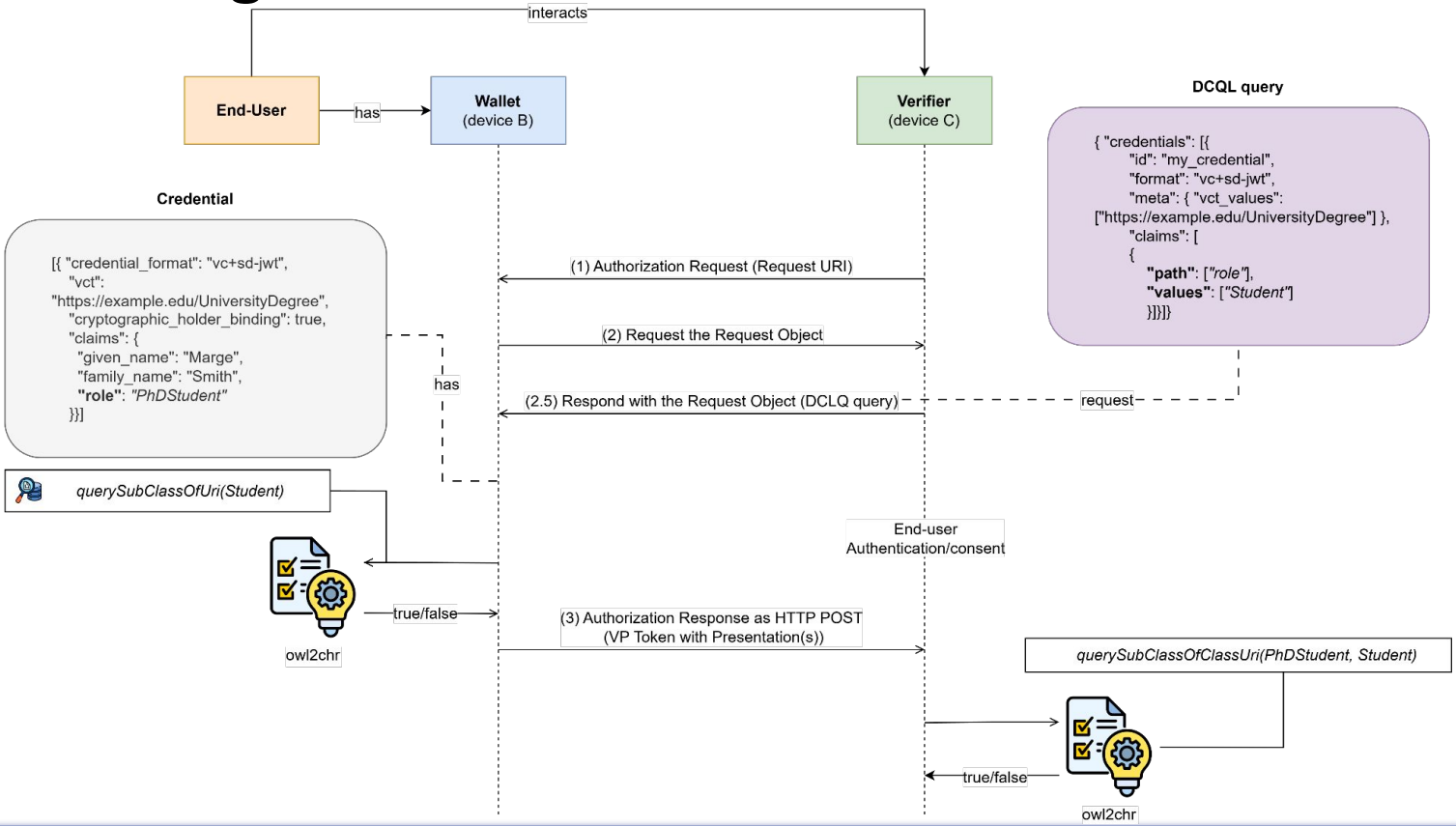
```
{"credentials": [{
        "id": "my_credential",
        "format": "vc+sd-jwt",
        "meta": { "vct_values":
["https://example.edu/UniversityDeg
ree"] },
        "claims": [
        {
            "path": ["role"],
            "values": ["Student"]
        }
        ]
    }]}
```

DCQL Playground: https://dcqlfiddle.com/.

owl2chr integration



owl2chr integration



OWL and CHR

Ontologies and OWL

- **Ontology:** a formal specification of a knowledge domain
- It is built on **four elements**:
  - *classes*: concepts (i.e. Student,Person);
  - *properties*: relations (i.e. hasStudent);
  - *individuals*: instances (i.e. David);
  - *axioms*: semantic definition (i.e. subClassOf).
- *Axioms* impose constraints (**subsumption, cardinality**) that structure relations between entities

- **Web Ontology Language** (**OWL**) 2 is the W3C standard for Semantic Web ontologies based on **description logics**
- It enables complex class definitions and precise property semantics
- **Class restrictions** (SubClassOf, etc.) define **subsumption** and **constraints** on property values
- **Property characteristics (transitive, symmetric, inverse)** specify relation semantics

owl2chr

owl2chr is a declarative reasoning engine for OWL 2 RL based on CHR++ (Constraint Handling Rules) and the COWL library for ontology parsing.

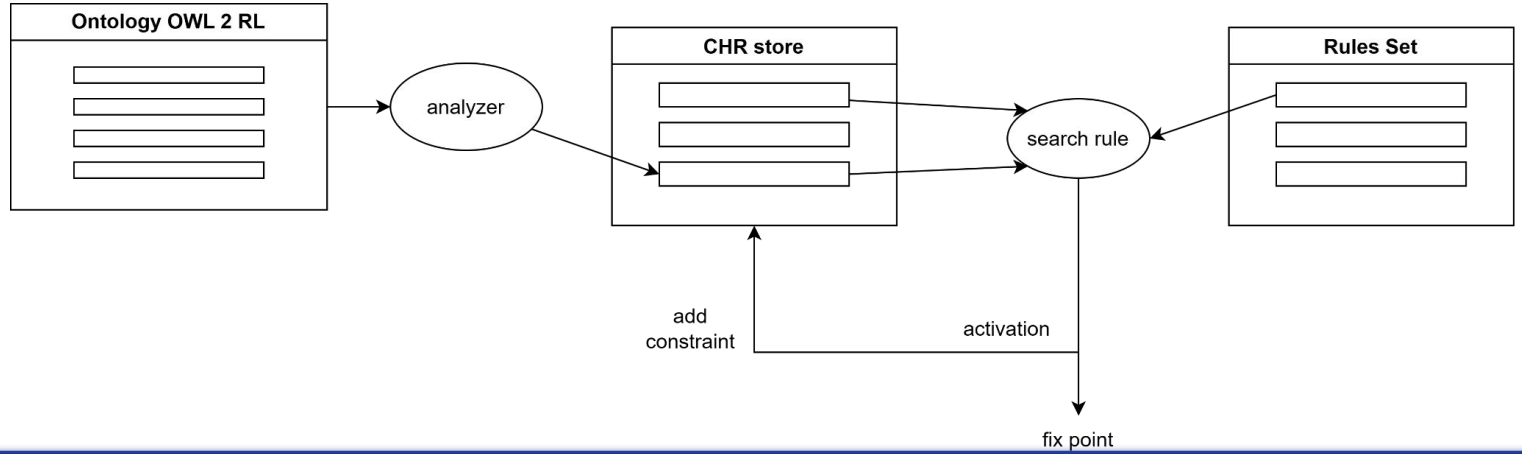
It implements an inference and reasoning engine for **OWL 2 (Web Ontology Language)** ontologies. It combines:

- CHR++: Constraint rule system for logical inference
- COWL: C library for parsing OWL 2 ontologies
- Queries: Query system for querying ontologies

A. Abed, V. Barichard, D. Genest, Éric Monfroy, owl2chr — owl 2 rl declarative reasoner based on chr++, https://github.com/arigraphitech/owlChrpp, 2025.

How owlChrpp works

- **OWL 2 RL axioms are mapped to CHR constraints** (e.g., SubClassOf ->owlSubClassOf).
- CHR rules mimic OWL reasoner semantics.
- A **parser** converts **ontology axioms** into **constraints** and inserts them into the **CHR store**.
- **Adding a constraint** triggers **rule execution**, potentially generating new inferred constraints (**forward chaining**).
- This continues until no more rules apply or inconsistency is detected.



SSI and owl2chr

querySubClassOfClassUri

```
//querySubClassOfClass: return true if X is subclass of Y

r_querySubClassOfClassURI @ logicalName(URIX, X) #passive,
logicalName(URIY, Y) #passive \ querySubClassOfClassUri(URIX, URIY) <=>
querySubClassOfClass(X, Y);;

owlSubclassOf(X, Y)\ querySubClassOfClass(X, Y) <=> print("true") ;;

querySubClassOfClass(X, Y) <=> print("false") ;;
```

Luchinzzz - owlChrpp: https://github.com/Luchinzzz/owlChrpp/tree/develop.

Example workflow

1. querySubClassOfClassUri(URIPhDStudent, URIStudent): the initial query expressed using URIs.
2. querySubClassOfClass(PhDStudent, Student): the query is rewritten using logical names instead of URIs.
3. owlSubclassOf(PhDStudent, Student): the system finds that PhDStudent is a subclass of Student.
4. print("true"): the query succeeds, and the result true is returned.

Recursion Example

1. querySubClassOfClassUri(URIPhDStudent, URIUniversity): the initial query expressed using URIs.
2. querySubClassOfClass(PhDStudent, University): the query is rewritten using logical names instead of URIs.
3. owlSubclassOf(PhDStudent, University): this constraint is not initially present, since the relationship is not direct; therefore, the recursive reasoning process starts.
4. owlSubclassOf(PhDStudent, Student): the system finds that PhDStudent is a subclass of Student.
5. owlSubclassOf(Student, University): it then finds that Student is a subclass of University.
6. owlSubclassOf(PhDStudent, University): by transitivity, the system infers that PhDStudent is a subclass of University.
7. print("true"): the query succeeds, and the result true is returned.

OWL Axioms in CHR Rules

$$\textit{sub-herit} \;@\; \mathrm{owlSubClassOf}(A,B),\ \mathrm{owlClassAssertion}(X,A,\mathrm{true}) \Longrightarrow \mathrm{owlClassAssertion}(X,B,\mathrm{true})$$

$$\textit{sub-trans} \;@\; \mathrm{owlSubClassOf}(A,B),\ \mathrm{owlSubClassOf}(B,C) \Longrightarrow \mathrm{owlSubClassOf}(A,C)$$

$$\textit{sub-idem} \;@\; \mathrm{owlSubclassOf}(A,B) \backslash \mathrm{owlSubclassOf}(A,B) \Longleftrightarrow \top$$

$$\textit{disj-verif} \;@\; \mathrm{owlDisjointClasses}(A,B), \mathrm{owlClassAssertion}(X,A,\mathrm{true}), \mathrm{owlClassAssertion}(X,B,\mathrm{true}) \Longrightarrow \bot$$

$$\textit{comp-init} \;@\; \mathrm{owlComplementOf}(A,NOTA), \mathrm{owlNamedIndividual}(X) \Longrightarrow \mathrm{owlClassAssertion}(X,NOTA,\mathrm{false})$$

$$\textit{comp-valid} \;@\; \mathrm{owlClassAssertion}(X,NOTA,\mathrm{true}) \backslash \mathrm{owlClassAssertion}(X,NOTA,\mathrm{false}) \Longleftrightarrow \top$$

$$\textit{comp-elim} \;@\; \mathrm{owlComplementOf}(A,NOTA), \mathrm{owlClassAssertion}(X,A,\mathrm{true}) \backslash \mathrm{owlClassAssertion}(X,NOTA,\mathrm{false}) \Longleftrightarrow \top$$

$$\textit{comp-verif} \;@\; \mathrm{owlComplementOf}(A,NOTA), \mathrm{owlClassAssertion}(X,A,\mathrm{true}), \mathrm{owlClassAssertion}(X,NOTA,\mathrm{true}) \Longleftrightarrow \bot$$

Conclusion and Future Works

- ✔ Overcome limitations of syntactic claim verification in SSI systems.
- ✔ A CHR-based ontology reasoner in C++ is proposed to enable semantic verification of claims.
- ✔ Semantic reasoning improves claim validation by considering the meaning and context of information

- 💡 Focus on minimal disclosure through informativeness-based approaches as an alternative to computationally expensive ZKPs.
- 💡 Extend to support more expressive constraints and logical reasoning beyond simple subclass relationships.

Thank you for your attention!

chiara.luchini@collaboratori.unipg.it

"Semantic Reasoning for Verifiable Credential Validation in SSI Systems"
Stefano Bistarelli, Martin Diéguez, **Chiara Luchini**, Eric Monfroy and Francesco Santini