Role-Aware Knowledge Representation for Autonomous Robots: Modeling and Reasoning about Situational Roles in Semantic Digital Twins

Abdelrhman Bassiouny[1] , **Michael Beetz**[1]
[1]University of Bremen
abassiou@uni-bremen.de, beetz@cs.uni-bremen.de

Abstract

Autonomous robots operating in dynamic environments must reason about entities' temporary contextual roles alongside their persistent identities (e.g., an object acting as an *Obstacle* during navigation and a *HeldObject* during manipulation). Grounded in OntoClean's metaproperty of anti-rigidity, this paper presents the design and implementation of roles within KRROOD (Knowledge Representation and Reasoning through Object-Oriented Design). We compare two Python implementation strategies: a *delegator-class* approach and an *inheritance with constructor-interception* approach, and provide a decision procedure for choosing between roles, subclasses, and compositions. Role relations are modeled via the transitive predicate `IsRoleFor` and its inverse `HasRole`, using forward-chaining fixpoint inference. Evaluation against OWL2Bench illustrates correct role reclassification and reasoning feasibility [Bassiouny *et al.*, 2026], while a Semantic Digital Twin case study demonstrates qualitative state transitions in dynamic world representations.

1 Introduction

A robot operating in a dynamic environment encounters entities whose physical identity is stable but whose contextual classification changes: the same object is an *Obstacle* during navigation and a *HeldObject* once grasped; a room *plays* the role of *Kitchen* only so long as the kitchen utilities remain there. For a robot to act correctly, its knowledge representation must capture this distinction. This is not a question of inheritance, where being a *Kitchen* is not a specialisation of being a *Room* in the way a *Mammal* is a specialisation of an *Animal* [Guarino and Welty, 2004]. It is a *role*, a temporary, context-dependent property acquired and relinquished while the entity's underlying identity persists.

In classical knowledge representation, roles of this kind are formalised through Description Logics and ontology languages such as OWL [Horrocks *et al.*, 2003], where anti-rigidity (the property that a concept need not apply to all instances in all possible worlds [Welty and Guarino, 2001]) distinguishes roles from natural types. However, deploying these formalisms within modern software systems requires bridging an integration gap: object-oriented programs, which dominate applied software engineering, provide no native construct that expresses the distinction between a persisting entity and a context-dependent role it plays [Steimann, 2000].

For an autonomous robot, these are not abstract queries: before acting, the robot must consult its Semantic Digital Twin [Autili *et al.*, 2022] and ask *does this object currently play the role of an Obstacle?* or *which locations currently serve as storage for cleaning tools?* These are queries about situational, dynamic states, which is the kind of reasoning the qualitative reasoning community has long studied in the context of processes, state transitions, and temporal change [Weld and de Kleer, 1990]. Making such queries expressible natively in a programming language is therefore both a software engineering and a knowledge representation problem.

KRROOD [Bassiouny *et al.*, 2026] (Knowledge Representation and Reasoning through Object-Oriented Design) addresses the broader integration gap by treating knowledge as a first-class programming abstraction in Python. This paper extends that work by enabling role-aware reasoning within a KRROOD-based Semantic Digital Twin. We contribute:

- Two Python implementation strategies for roles with a comparative analysis across semantic correctness, IDE compatibility, and maintainability.
- A decision procedure for modellers to determine whether a concept should be represented as a role, a subclass, or a composition.
- The predicates `HasRole` and `IsRoleFor` with forward-chaining fix-point inference over their inverse and transitive properties.
- Empirical evaluation on OWL2Bench and a Semantic Digital Twin case study for autonomous robotic agents.

2 Background

2.1 Roles and Anti-Rigidity

OntoClean [Guarino and Welty, 2004; Welty and Guarino, 2001] provides a principled metaproperty framework for evaluating ontological soundness. Central to our work is the property of *rigidity*: a concept C is *rigid* if every instance of C is necessarily an instance in all possible worlds (i.e., it cannot cease to be an instance of C while continuing to exist). A concept is *anti-rigid* if no instance is necessarily an instance

in all worlds. Natural kinds such as *Person* or *Gold* are rigid; roles such as *Student*, *Employee*, or *Teacher* are anti-rigid.

A role R played by an entity e of type T adds contextual attributes and behaviours to e without altering e's identity or type. Crucially, three properties follow from anti-rigidity:

1. Roles can be acquired and lost without affecting the persisting identity of the base entity.
2. An entity can simultaneously play multiple roles, provided they are not mutually exclusive.
3. Attributes of the base entity remain accessible through the role: a *Teacher*'s name, age, and height are still those of the underlying *Person*.

2.2 Roles in Software Engineering and Knowledge Representation

Role modelling has a long history in software engineering [Steimann, 2000]. While formal frameworks like NextKB and upper ontologies like SUMO represent roles via logical microtheories or axiomatic constraints, they lack native integration with execution environments. Software patterns (e.g., Fowler's analysis patterns [Fowler, 1996] and DCI [Coplien and Bjørnvig, 2009]) address identity, but lack ontological grounding in rigidity. Standard OOP mechanisms (subclassing and composition) conflate the role/type distinction: Inheriting *Student* from *Person* implies that every student is necessarily a new instance of emph Person, and incorrectly inflates population counts when querying for emph Person instances (unless a unique id is used, but that still inflates the memory usage and duplicates data). If modeled as a composition, then access to the original entity attributes is awkward. Now to access a teacher's age, you have to write `"teacher.person.age"`, this gets even worse when the hierarchy is deep, for example, a *VisitingProfessor* is a role for a *Professor*, which is in turn a role for a *Person*, and to access age you have to write `"visiting_professor.professor.person.age"`.

3 Role Modelling in KRROOD

3.1 Design Requirements

Any adequate software representation of roles must satisfy the following requirements derived from anti-rigidity and the knowledge-level accessibility criterion of Newell [Newell, 1982]:

R1 Identity Preservation. The base entity must exist independently of any role it plays.

R2 Attribute Delegation. Base entity attributes must be accessible through the role without duplication of state.

R3 Population Correctness. Querying for entities of a base type must not return role instances.

R4 IDE Accessibility. Role classes must be statically analysable by Python type checkers and IDEs, supporting autocomplete and type inference.

R5 Simultaneous Roles. An entity must be able to take on multiple compatible roles concurrently.

3.2 Method 1: Delegator-Class

The first strategy generates, for each base class T, a proxy class `RoleFor`T that holds a reference to a *role taker* instance of type T and exposes all of T's attributes and methods via explicit delegation. Any class representing a role that T can play inherits from `RoleFor`T.

```python
@dataclass
class RoleForPerson:
    person: Person

    @property
    def age(self) -> int:
        return self.person.age

@dataclass
class Teacher(RoleForPerson):
    courses: List[Course]
```

Listing 1: Delegator-class approach for a Person role.

Advantages. Semantics are explicit and correct: a `Teacher` is not a `Person` in the type hierarchy, so population queries (`R3`) return correct results. The delegator class is reusable across all roles for the same base type, and full IDE support (`R4`) is provided through explicit attribute signatures.
Disadvantages. The approach requires generating a substantial amount of boilerplate delegation code whose only function is forwarding. Every attribute and method of the base class must be duplicated in the proxy class signature. As the base class evolves, the delegator must be kept in sync.

3.3 Method 2: Inheritance with Constructor-Interception and Dynamic Delegation

The second strategy has role classes inherit directly from the base class while holding a `RoleTaker` reference to the specific base instance. The constructor is overridden to suppress re-initialisation of inherited attributes, and `GetAttribute` and `SetAttribute` are overridden to redirect all attribute access dynamically through the role taker at runtime. The code that demonstrates this is available online[1].
Advantages. No extra classes need to be generated. IDE type inference works natively since `Teacher` is a subtype of `Person` in Python's type system (`R4`). Attribute delegation is handled automatically at runtime.
Disadvantages. The inheritance relationship misrepresents the semantics: `isinstance(teacher, Person)` returns `True`, violating `R3`. This must be mitigated by filtering on the persistent identity (role-taker identity) rather than type membership. The non-trivial constructor override makes the implementation harder to debug and maintain at scale.

4 Role vs. Subclass vs. Composition

A persistent modelling challenge is deciding whether a concept should be represented as a role, a subclass, or a composition. We provide the following decision procedure, grounded in OntoClean metaproperties:

[1]Implementation available at `https://anonymous.4open.science/r/cognitive_robot_abstract_machine-FDDF/krrood/src/krrood/patterns/role.py`

1. **Apply the rigidity test.** Can an entity cease to be an instance of this concept while continuing to exist? If yes, the concept is anti-rigid and should be modelled as a *role*. Example: a *Person* can cease to be a *Student*; use a role. An *Orange* cannot cease to be a *Fruit*; do not use a role.
2. **Apply the identity test.** Does the concept carry its own identity criterion, independent of the base entity? If yes, model as a *subclass*. Example: *Orange* has its own biological identity independent of being a generic *Fruit*.
3. **Apply the part-whole test.** Does the relationship describe structural containment or assembly rather than classification? If yes, use *composition*. Example: a *Car* is composed of an *Engine*, not a subtype of it.

5 Role Relations and Fix-Point Inference

KRROOD captures role relations through two predicates: `HasRole(entity, role)`, asserting that an entity currently plays a role, and `IsRoleFor(role, entity)`, its inverse. `IsRoleFor` is transitive, so if a *VisitingProfessor* is a role for a *Professor*, and a *Professor* is a role for a *Person*, then a *VisitingProfessor* is also a role for a *Person* without needing to state that explicitly.

When a role assertion is made, two forward-chaining rules fire eagerly until no new facts can be derived:

$$\texttt{IsRoleFor}(r, e) \Rightarrow \texttt{HasRole}(e, r) \qquad \text{(Inv)}$$

$$\texttt{IsRoleFor}(r_2, r_1) \wedge \texttt{IsRoleFor}(r_1, e) \Rightarrow \texttt{IsRoleFor}(r_2, e) \qquad \text{(Trans)}$$

This reaches a fix-point over a global directed acyclic graph (DAG) where nodes are entity instances and edges are role relations. Because role assertions in KRROOD rely on class definition hierarchy (`Role[T]`), cyclic dependencies (`IsRoleFor`) are prevented by construction at runtime via Python's module import resolution, ensuring DAG acyclicity and guaranteed fix-point termination. Role takers do not store explicit dynamic references to their roles; instead, `HasRole(e, r)` is retrieved via KRROOD's central role registry, maintaining base entity decoupling.

6 Evaluation and Applications

6.1 OWL2Bench

OWL2Bench [Singh *et al.*, 2020] is a benchmark for evaluating OWL 2 reasoners using synthetically generated ontologies in the university domain. The correctness of KRROOD's reasoning over role relations has already been established in [Bassiouny *et al.*, 2026]; what we focus on here is how the decision procedure introduced in this paper applies to a real ontology.

OWL2Bench models everything as subclasses. There is no explicit notion of roles anywhere in the ontology. But looking at it through the lens of anti-rigidity, it becomes clear that several of its classes should not be subclasses at all. A *Chair*, *Dean*, or *Director* is not a permanent kind of thing; these are positions a *Professor* occupies temporarily and can relinquish. Similarly, *Employee* is better understood as a role for a *Person*, while *Professor* itself, being a genuine classification (a kind), remains a subclass of *Employee*. Following the decision procedure, we reclassified these concepts, introduced `HasRole` and `IsRoleFor` as properties in the OWL file, and attached them to the relevant classes.

It is worth noting that this identification could be done implicitly: if two subclasses are explicitly stated as mutually exclusive in the ontology, they are likely genuine subclasses; if no such exclusion exists, they are good candidates for roles. This heuristic does not replace the rigidity test but can serve as a practical starting point when working with large existing ontologies, as reported in [Bassiouny *et al.*, 2026].

6.2 Semantic Digital Twin for Autonomous Robots

The problem motivating this work is precisely here: a Semantic Digital Twin (SDT) [Autili *et al.*, 2022] mirrors a physical environment for use by autonomous robotic agents, and role-aware representation is essential to its correctness. Representing certain SDT concepts as roles rather than subclasses leads to more semantically accurate models. A *Room*'s purpose; whether it functions as a *Kitchen*, *Bedroom*, or *Living Room*, is dependent on what's inside the room and how it is used, all of which are temporary, and is better captured as a role the room plays rather than a permanent type. Similarly, a *Pose* only becomes a *GraspPose* in the context of a manipulation task, gaining a relevant arm and grasp description for the duration. In both cases, the underlying entity retains its identity while its roles come and go.

This representation allows the agent to query the SDT in role-aware terms using KRROOD's Entity Query Language, asking, for instance, what roles a given room currently plays, or which poses in the environment are currently *GraspPoses*. The same idea extends naturally to object-level semantics: a physical object in the scene might play the role of a *HeldObject* during manipulation, an *Obstacle* during navigation, or a *PerceivedObject* when first detected by the robot's sensors, with each role coming and going as the task context evolves [Forbus, 1984].

7 Conclusion

Autonomous robots need knowledge representations that capture the situational roles entities play in a dynamic world; we have addressed this within KRROOD, grounded in the OntoClean metaproperty of anti-rigidity. Two Python implementation strategies are introduced and compared across semantic correctness, IDE accessibility, and maintainability. A formal decision procedure guides modellers in choosing between roles, subclasses, and compositions. Fix-point inference over transitive and inverse role predicates enables expressive qualitative reasoning over dynamic entity states. Empirical evaluation on OWL2Bench validates the correctness of inference, and a Semantic Digital Twin application demonstrates the framework's utility for autonomous agents reasoning about a changing world.

References

[Autili *et al.*, 2022] Marco Autili, Paola Inverardi, and Patrizio Pelliccione. Semantic digital twins for autonomous

systems. In *Proceedings of the IEEE/ACM 44th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 61–65, 2022.

[Bassiouny *et al.*, 2026] Abdelrhman Bassiouny, Tom Schierenbeck, Sorin Arion, Benjamin Alt, Naren Vasantakumaar, Giang Nguyen, and Michael Beetz. Implementing knowledge representation and reasoning with object oriented design, 2026.

[Coplien and Bjørnvig, 2009] James O. Coplien and Gertrud Bjørnvig. *Lean Architecture: for Agile Software Development*. Wiley, 2009.

[Forbus, 1984] Kenneth D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24(1–3):85–168, 1984.

[Fowler, 1996] Martin Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, 1996.

[Guarino and Welty, 2004] Nicola Guarino and Christopher A. Welty. An overview of OntoClean. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, pages 151–172. Springer, Berlin, Heidelberg, 2004.

[Horrocks *et al.*, 2003] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. volume 1, pages 7–26, 2003.

[Newell, 1982] Allen Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.

[Singh *et al.*, 2020] Gunjan Singh, Sumit Bhatia, and Raghava Mutharaju. Owl2bench: A benchmark for owl 2 reasoners. In *The Semantic Web – ISWC 2020*, pages 81–96, Cham, 2020. Springer International Publishing.

[Steimann, 2000] Friedrich Steimann. On the representation of roles in object-oriented and conceptual modelling. *Data & Knowledge Engineering*, 35(1):83–106, 2000.

[Weld and de Kleer, 1990] Daniel S. Weld and Johan de Kleer. Readings in qualitative reasoning about physical systems. 1990.

[Welty and Guarino, 2001] Christopher Welty and Nicola Guarino. Supporting ontological analysis of taxonomic relationships. *Data & Knowledge Engineering*, 39(1):51–74, 2001.