An Ontological Approach for Generating Precedence Matrices of Assemblies Directly from CAD Files*

Amir Jaberi[1,*,†], Mihai Pomarlan[2,†], Oliver Kutz[1,†] and Angelika Peer[1,†]

[1]*Free University of Bozen-Bolzano, Italy*
[2]*University of Bremen, Germany*

Abstract

Assembly sequence planners require information such as eligible assembly sequences and constraints to generate feasible assembly plans. This information can be, for example, encoded in the form of a precedence matrix. This paper presents a knowledge-driven approach that integrates ontological reasoning with geometry and topology-aware properties to systematically generate precedence matrices. We use a bottom-up assembly approach to generate ontological reasoning rules by identifying an initial part and then successively attaching directly connected components until the entire assembly is complete. We implement an ontology-based reasoning framework in which Semantic Web Language rules encode and enforce priority relationships among parts based on their direct connectivity and assembly constraints. Each rule captures a specific precedence condition. We validate the correctness and generality of these rules by applying them to multiple, structurally diverse products. The reasoning mechanism reliably infers sound precedence constraints for each evaluated assembly, producing a correct partial order of operations from which valid assembly or disassembly sequences can be generated.

Keywords

Assembly Sequence Planning, Precedence Matrix, Precedence Constraint, Engineering Ontology, CAD-based Reasoning

1. Introduction

Determining semantic precedence constraints among components is a fundamental issue in assembly and disassembly planning. Precedence constraints define possible sequences in which components/parts can be assembled or disassembled. The precedence constraints can be effectively represented through a precedence matrix, a structured data representation that captures binary relationships between assembly actions related to components/parts; indicating the necessary constraints of the assembly order [1, 2]. A precedence matrix P can record that the assembly action related to part i must be performed after the assembly action related to part j by setting $P(i, j) = 1$ (otherwise empty entries). Such representations ensure that essential predecessor-successor relationships are respected, while violating such a precedence constraint would result in an infeasible assembly plan. The accurate generation of such matrices remains a challenging topic in manufacturing research due to the inherent complexity and variability of product designs [2].

Several systematic approaches have been developed in literature to extract precedence constraints and related precedence matrices from a given assembly. This section reviews the relevant works and summarizes their contributions and shortcomings.

1.1. Assembly Sequence Planning from CAD-based Collision Analysis

In this category, precedence constraints are derived *directly from the CAD geometry* through algorithmic collision, stability, or physics-engine simulation, *without* relying on human-demonstrated trajectories.

Proceedings of the Joint Ontology Workshops (JOWO) - Episode XI: The Sicilian Summer under the Etna, co-located with the 15th International Conference on Formal Ontology in Information Systems (FOIS 2025), September 8-9, 2025, Catania, Italy
*Corresponding author.
†These authors contributed equally.
ajberi@unibz.it (A. Jaberi); mihai.pomarlan@uni-bremen.de (M. Pomarlan); oliver.kutz@unibz.it (O. Kutz); angelika.peer@unibz.it (A. Peer)
0000-0002-1304-581X (M. Pomarlan); 0000-0003-1517-7354 (O. Kutz); 0000-0002-2896-9011 (A. Peer)

Collision or interference tests are the classic starting point. A candidate component is "virtually removed" along its assembly direction while all other components remain fixed; any interference identifies a blocker and therefore a precedence relation. Early sweep-test pipelines such as Zhang et al. (2017) building an interference matrix by collision-detection use bounding-volume and facet-intersection tests to identify colliding part pairs and filter out spurious geometry [3]. Kumar et al. (2022) introduced a purely geometric assembly-sequence-planning method that samples dozens of oblique insertion/removal directions and uses Minkowski-sum collision maps to build a blocking graph, automatically extracting precedence constraints between parts [4]. In [5], authors proposed an automated collision-detection approach to create a matrix of assembly-precedence constraints for axisymmetric components. The collision-detection method, with its bounding-box and 2D section analyses, is specifically tailored for axisymmetric components. Neb and Göke (2021) combined collision analysis and gravity analysis (ensuring stability by checking part dependencies under gravitational forces), translating these into assembly order rules. This integrated approach resolves both spatial and stability constraints to define feasible sequences [6]. Beyond purely geometric tests, physics engines extend CAD-based analysis. Tian *et al.* [7] loaded the CAD model into a signed-distance-field physics engine and casted planning as the reverse of a disassembly-tree search: fast Signed Distance Field (SDF) collision checks, a greedy gravitational-stability filter, and Graph Neural Networks(GNN)-guided part selection together generate robot-executable sequences for assembly of complex products.

Although collision analysis pipelines offer fully automated precedence extraction by simulating part motions and recording physical interferences to find "must-precede" relations, their capabilities are limited by an exclusive reliance on geometric interference detection. Specifically, they overlook semantically rich CAD-defined connection types — such as coincident, concentric, threaded, and gear-to-gear relationships — which are critical for capturing assembly-relevant dependencies.

1.2. Automatic Sequence Planning from Semantic Reasoning

Ontology-based methods represent one of the most promising recent advances in systematically generating precedence matrices. Ontologies provide structured semantic frameworks to formally represent assembly knowledge, capturing both physical relationships such as spatial constraints and functional relationships such as load-bearing constraints. Using ontologies, precedence constraints are inferred by applying semantic rules through reasoners, often encoded using languages such as OWL (Web Ontology Language) and SWRL (Semantic Web Rule Language) [8].

Hu et al. (2024) [9] introduced a human-robot collaborative disassembly (HRCD) pipeline that couples a domain ontology – whose core classes are "FunctionalPart", "AccessoryPart" and a single generic "Fastener" – with SWRL rules to infer precedence constraints, assign human/robot/collaborative execution modes, and output an "optimal" disassembly sequence. But since all fasteners are aggregated into one class and precedence is inferred purely from disassembly – they consider a basic coverage blocking to decide the removal order – the approach cannot differentiate screws, nuts, washers, etc. It also cannot decide their internal order – an essential requirement for forward assembly planning – so it remains effective for disassembly scenarios but inadequate for assembly-oriented fastener sequencing as addressed in our work.

Qian et al. (2020) [10] proposed a knowledge-based assembly-priority method that combines five numeric indicators with ontology-driven SWRL rules, using OntoSTEP to map STEP geometry to OWL for contact-feasibility checks. Demonstrated only on a small flat-fuze assembly, the approach illustrates how expert knowledge can be systematically encoded, but it offers no evidence of scalability or generalizability to more complex products, provides limited detail on the actual contact-face implementation, and may fail to differentiate parts when indicator values are similar (e.g., symmetrical components); moreover, the indicator thresholds appear tuned to that single case study and are not shown to be transferable to other assemblies. Khan et al. [11] proposed a spatiotemporal ontology to support dynamic assembly motion planning. While their model captures motion semantics using SWRL rules, it does not support precedence constraint generation or reasoning over static passive assemblies extracted from CAD data. Kim et al. [12] formalized joint types with a mereotopological OWL/SWRL model that distinguishes look-alike connections; by contrast, we extract concentric/coincident contacts directly from STEP and use a lightweight ontology to infer precedence. Aameri et al. [13] provided a first-order assembly ontology combining multi-dimensional mereotopology with qualitative boundary/incidence axioms for configuration feasibility; while not addressing CAD parsing or sequencing, their treatment underpins our contact semantics. Our pipeline therefore complements these strands by focusing on automated contact extraction and precedence reasoning.

1.3. Motivation and Objectives

While collision-based pipelines excel at *detecting* geometric clashes, yet they remain essentially *syntactic*: a collision flag explains *that* two parts cannot move simultaneously, but not *why* one part should precede the other in the product's functional logic. Our objective is to unify these two strands – precision geometry and explicit semantics – into a single, fully automated workflow that (i) lifts low-level CAD features to function-aware relations and (ii) exploits those relations to *systematically generate precedence matrices for assembly processes.* In summary, our work makes the following contributions:

1. **Product-agnostic derivation of precedence matrix.** We introduce a generic method that derives a precedence matrix directly from CAD data. The procedure employs OWL/SWRL semantics and class-level rules, avoiding bill of material (BOM)-specific hard-coding and enabling seamless application across a broad spectrum of assemblies.
2. **Automatic extraction of concentric and coincident relationships from native STEP.** We develop a geometric parser that detects directly *concentric (cylindrical)* and the inference of *coincident (planar)* connections from STEP files.
3. **Ontology-based reasoning over StructuralParts and Fasteners.** We design a lightweight domain ontology that *differentiates the principal subclasses of mechanical fasteners* – screws, nuts, bolts, washers, pins, snap-rings—alongside generic `StructuralParts` (any load-bearing or functional component that is not itself a fastening element and whose removal would break the kinematic chain of the assembly). We capture how each fastener constrains the assembly order. A set of SWRL rules encodes generic engineering axioms (e.g. "a nut typically is assembled after its mating screw"), enabling a reasoner to infer precedence relations automatically from an instantiated product model.

Together, these elements yield a pipeline that (i) operates directly on original CAD assemblies, (ii) extracts function-aware precedence constraints, and (iii) scales to structurally diverse products due to the employed semantic extraction and reasoning which are driven by reusable class-level knowledge rather than instance-specific scripts.

2. Methodology: Proposed Ontological Approach

In this research, a domain ontology is developed to represent assembly-related knowledge extracted from CAD models. Figure 1 summarises the implemented pipeline in which the ontology captures key concepts such as components, geometric features, and their spatial relationships, which are essential to determine the precedence matrix. This transformation from the low-level geometric representation (extracted from STEP files) to a high-level ontological model is referred to as the semantic lift. Semantic Web Rule Language (SWRL) rules enable reasoning to infer the assembly priority between each two connected components, which in turn leads to generating a precedence matrix. The emphasis is on practical reasoning and knowledge representation to derive assembly sequences from CAD data. To formalize the semantic model of mechanical assemblies, a layered, top-down ontology is adopted (Figure 2). The upper taxonomy – *artifact, Part, Feature, Connection* – captures what the constituents are. Mereological (e.g., hasPart) and functional/kinematic relations (e.g., hasConcentric-Connection, precedesOver) capture how they interact. Although the Connection and Feature classes are included for conceptual completeness, we do not instantiate them in the current implementation. Detecting contact types (concentric/coincident) requires numeric geometric computation (e.g., axis alignment, interval overlap) that OWL/SWRL cannot perform; we therefore run this in the CAD preprocessor and assert the results as direct part–part properties (`hasConcentricConnectionWith`, `hasCoincidentConnectionWith`). The ontology remains essential as the semantic layer that organizes these assertions, supports SWRL-based precedence rules over `Fastener/StructuralPart` classes, and enables validation and querying at the assembly level, avoiding ontological verbosity while capturing exactly the concepts needed for precedence sequencing (including fastener subclasses such as screw, nut, washer, pin, snap-ring). As we will extend to additional connection types (e.g., threaded engagements, welds, press-fits), we can reify detected links as Connection individuals to attach attributes and check type-specific constraints, without changing the geometric pipeline. Hence, we deliberately designed a lightweight, domain-specific ontology tailored to assembly precedence, rather than reusing a broad upper ontology or existing standards. The primary reason is that general ontologies (e.g., BFO or ISO 15926) are very comprehensive and domain-independent, which would introduce substantial complexity without directly supporting our specific needs in assembly sequence reasoning. Similarly, we didn't use OntoSTEP ontology [14] because pulling in the full STEP schema would add substantial model size/complexity without directly yielding precedence or joint-type assertions. This approach ensured that our semantic rules could be implemented and tested without the overhead of reconciling with a complex upper ontology.

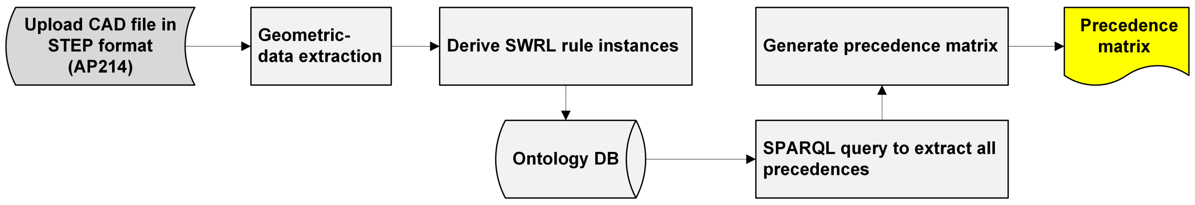

Upload CAD file in STEP format (AP214)
Geometric-data extraction
Derive SWRL rule instances
Ontology DB
SPARQL query to extract all precedences
Generate precedence matrix
Precedence matrix

Figure 1: Overview of the precedence matrix generation process. The process includes a semantic lift from CAD geometry to OWL-based ontology instantiation.

Our pipeline distinguishes two types of contact information extracted from the STEP model:

1. **Concentric (cylindrical) connections.** Two components form a concentric connection when they share a coaxial cylindrical interface whose radii, axial directions, and centre-line offsets all lie within the specified geometric tolerances. Practically, this arises when a boss (external cylinder) on one part mates with a hole (internal cylinder) on the other, the cross-sectional profiles of both features being equivalent up to the tolerance band. All threaded joints (e.g., screw-nut engagements) are treated as a subclass of concentric connections, as they inherit the same coaxiality constraint and analogous mating geometry. We first parse the Boundary Representation (B-Rep) data and collect all cylindrical faces. A pair of components $\langle C_i, C_j \rangle$ is marked *concentric* when they meet several conditions (explained in detail in section 2.1).
2. **Coincident connections.** A coincident connection is established when two StructuralPart instances present mating planar faces that are coplanar within the prescribed positional and angular tolerances, creating a flush interface. Because the plane itself offers no intrinsic resistance to separation along its normal, the parts are ultimately retained by external fasteners – e.g., screw/bolt–nut sets or locating pins – inserted through the aligned clearance holes in the coincident faces. A *coincident* relation is not read directly from the CAD file; instead it is inferred from the fastening pattern. Two `StructuralPart` instances P_k and P_ℓ are declared coincident if (i) each has a concentric connection with the *same* fastener F (pin, screw, bolt, …), and (ii) the axes of those concentric connections are colinear. (iii) For cylindrical faces classified as "hole", the relative positioning of their bounding boxes must adhere to a specified tolerance. These rules capture typical face-to-face joints retained by pins or screws and avoid accidental matches.

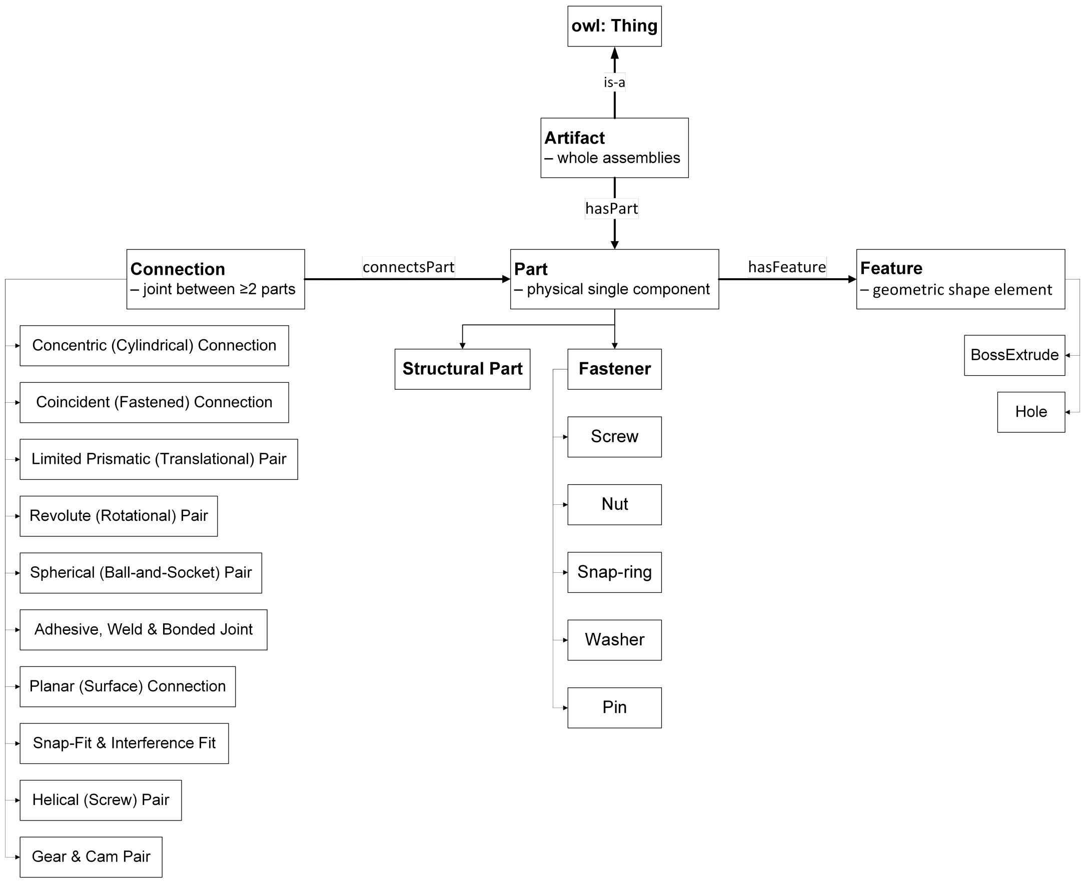

owl: Thing
is-a
Artifact
– whole assemblies
hasPart
Connection
– joint between ≥2 parts
connectsPart
Part
– physical single component
hasFeature
Feature
– geometric shape element
Concentric (Cylindrical) Connection
Coincident (Fastened) Connection
Limited Prismatic (Translational) Pair
Revolute (Rotational) Pair
Spherical (Ball-and-Socket) Pair
Adhesive, Weld & Bonded Joint
Planar (Surface) Connection
Snap-Fit & Interference Fit
Helical (Screw) Pair
Gear & Cam Pair
Structural Part
Fastener
Screw
Nut
Snap-ring
Washer
Pin
BossExtrude
Hole

Figure 2: Top-level class hierarchy and core object properties of the mechanical-assembly ontology.

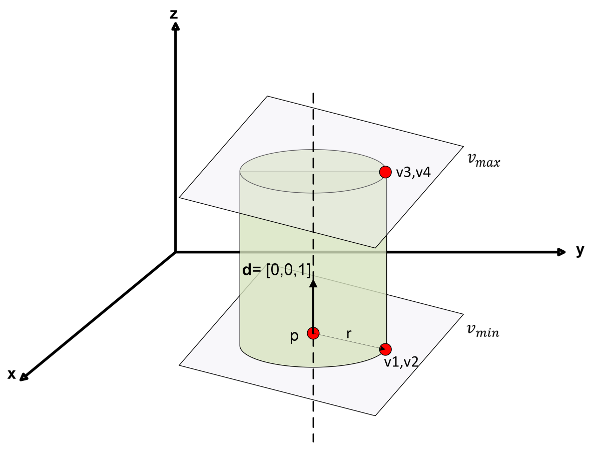


Figure 3: Parametric definition of a trimmed cylindrical face; the axis direction d shown is illustrative only.

2.1. Geometry Analysis

The geometric-topological pre-processing is performed entirely in Python scripts by using pythonOCC – a high-level wrapper around the *Open Cascade CAD Technology* (OCCT) kernel [15]. By parsing CAD files, we extract a list of part names, and the full set of cylindrical faces. Each face is classified *in situ* as a **hole** or **boss** by inspecting its orientation with respect to the solid's outer shell. Moreover, for each cylindrical face, its radius r, unit axis direction $\mathbf{d}$, a reference point on the axis $\mathbf{p}$, and four corner vertices $\{\mathbf{v}_1, \ldots, \mathbf{v}_4\}$ – two per trimming plane – are extracted. Indeed, a lateral cylindrical face is represented as the parametric patch $\{(U, V) \mid U \in [0, 2\pi],\ V \in [v_{\min}, v_{\max}]\}$ on the canonical surface

$P(U,V) = \mathbf{Loc} + V\,\mathbf{Z}_{\text{dir}} + r\big(\mathbf{X}_{\text{dir}}\cos U + \mathbf{Y}_{\text{dir}}\sin U\big)$. Consequently, its boundary comprises two *seam* edges at $U = 0$ and $U = 2\pi$ and two *trim* edges at $V = v_{\min}, v_{\max}$; the intersections of these seams and trims yield the four topological "corner" vertices $\{v_1, \ldots, v_4\}$ returned by the kernel. Figure 3 visualizes the minimal geometric parametric description that our pre-processor harvests from each cylindrical face and immediately collapses to a signed 1-D interval along $\mathbf{d}$. Algorithm 1 then processes this description: for every hole-boss pair it evaluates, in turn, the radius tolerance, coaxiality (the hole and boss feature axes must be coaxial (parallel and radially coincident), and their origins must match within ε_p in the two coordinates orthogonal to that axis; a mismatch is tolerated only along the axis direction), and interval-overlap predicates, admitting the pair as a *concentric (cylindrical) connection* only when all three conditions succeed. In the context of our work, a cylindrical face is a face whose supporting surface is right-circular cylindrical; it is this entity from which we extract the radius r, axis direction $\mathbf{d}$, origin $\mathbf{p}$, and end-face vertices $\{\mathbf{v}_1, \ldots, \mathbf{v}_4\}$.

Algorithm 1: Interval-based concentric-connection detection

```
Input: Assembly model $\mathcal{A}$; tolerances $\varepsilon_r$ (radius), $\varepsilon_p$ (origin), $\varepsilon_l$ (interval)
Output: Set $\mathcal{C}$ of concentric connections $(p, q)$
foreach part $k \in \mathcal{A}$ do
    $F_k \leftarrow$ OCCT GetCylindricalFaces$(k)$;
    foreach face $f \in F_k$ do
        store $(r_f, \mathbf{d}_f, \mathbf{p}_f, \text{type}_f)$ *[r]type $\in \{hole, boss\}$
    end
end
foreach unordered face pair $(f_h, f_b)$ with type$_{f_h}$ = hole and type$_{f_b}$ = boss do
    if $|r_{f_h} - r_{f_b}| < \varepsilon_r$ then
        if $\sum_{i\in\{x,y,z\}} \mathbf{1}\big(|o_h^i - o_b^i| < \varepsilon_p\big) \geq 2$ then
            Project vertices of $f_h$ and $f_b$ onto $\mathbf{d}_{f_h}$;
            $[s^h_{\min}, s^h_{\max}],\ [s^b_{\min}, s^b_{\max}] \leftarrow$ axial intervals;
            $\Delta \leftarrow \min\big(s^h_{\max}, s^b_{\max}\big) - \max\big(s^h_{\min}, s^b_{\min}\big)$;
            if $\Delta \geq \varepsilon_l$ then
                $\mathcal{C} \leftarrow \mathcal{C} \cup \{\big(\text{part}(f_h), \text{part}(f_b)\big)\}$;
            end
        end
    end
end
return $\mathcal{C}$;
```

2.2. Property Schema

The ontology distinguishes *object properties*, which connect individuals, from *data-type properties* that encode geometric or literal metadata. Domains, ranges and cardinalities are declared in OWL to enable logical validation and SWRL rule execution.

Data properties. Table 1 summarises the core data-type properties that underpin our feature-level ontology. Geometric attributes apply to every `Feature`: the axis-aligned bounding box is delimited by its extremal vertices (`minCornerX/Y/Z`, `maxCornerX/Y/Z`); the frame is defined by the reference point `originX/Y/Z` and the unit axis vector `directionX/Y/Z`. For hole and boss features, a one-dimensional bounding interval, `intervalMin/Max`, measured along the feature's principal axis. The scalar `radius` specifies the cross-sectional size. Assembly semantics are declared only for `StructuralParts`: an integer `hasLayer` encodes for each `StructuralPart` to belong to which build layer, and the Boolean `isBasePart` identifies the component that must be placed first. Unlike previous approaches that compute a single axis-aligned bounding box (AABB) for an entire solid, we generate feature-level AABBs exclusively for hole and boss primitives. Restricting the enclosure to these functional features prevents spurious spatial relations that arise when part-level envelopes of disjoint components overlap or merely touch in highly intricate assemblies. Consequently, by building one-to-one correspondence between genuine mating features, eliminates false positives without sacrificing computational efficiency.

Table 1
Principal datatype properties. Units: millimetres.

Data Property	Domain → Range	Description
`minCornerX/Y/Z`	Feature → `xsd:decimal`	AABB lower corner
`maxCornerX/Y/Z`	Feature → `xsd:decimal`	AABB upper corner
`originX/Y/Z`	Feature → `xsd:decimal`	Feature reference point
`directionX/Y/Z`	Feature → `xsd:decimal`	Unit axis vector
`intervalMin/Max`	Feature → `xsd:decimal`	Extent along axis
`radius`	Feature → `xsd:decimal`	Cylinder radius
`hasLayer`	StructuralPart → `xsd:integer`	Specifies the assigned layer of each StructuralPart.
`isBasePart`	StructuralPart → `xsd:Boolean`	Specifies the selection of the BasePart, which is assembled before all other components.

Object properties. Table 2 lists the ontology's core object properties that characterize how two components (either StructuralPart or Fastener) relate in an assembly. The `precedesOver` imposes an explicit assembly order, while `isAdjacentWith` flags axial proximity. Collectively, these properties provide a compact yet expressive vocabulary for inferring connection typology and precedence between any pair of components.

Table 2
Core object properties.

Object Property	Domain → Range	Characteristics
`precedesOver`	Part → Part	Expresses an explicit precedence or assembly order between two parts.
`isAdjacentWith`	Part → Part	Indicates that the one-dimensional intervals of two distinct components fall within a prescribed clearance tolerance. The relation is primarily invoked to verify adjacency either (i) between two `StructuralParts` or (ii) between a `StructuralPart` and a `SnapRing/Washer` instance. Example: *SnapRingR1 isAdjacentWith Rod1*.
`hasConcentricConnectionWith`	Part → Part	Denotes that there is a concentric connection between two parts either Fastener or StructuralPart classes.
`hasCoincidentConnectionWith`	StructuralPart → StructuralPart	Shows that there is a coincident connection between two parts that both belong to StructuralPart class.

2.3. Proposed SWRL Rules

2.3.1. Layer-based Ontology Approach

Our method initiates by selecting a single component from the `StructuralPart` class to serve as the *BasePart*, which is assigned to *Layer0*. Subsequently, all components from the `StructuralPart` class that are geometrically *connected* to the *BasePart* – either via cylindrical or coincident connections—are identified and assigned to *Layer1*. This process is applied recursively: for each subsequent layer (*Layer2*, *Layer3*, etc.), we identify components connected to any component in the previous layer, continuing until all `StructuralPart` components have been assigned to a specific layer.

2.3.2. Precedence Resolution between Fasteners and Structural Parts

Table 3 encodes assembly knowledge in the form of SWRL rules to determine precedence relationships between two connected components – either both belonging to the Fastener class, or one from the Fastener class and the other from the StructuralPart class. When two parts are clamped using fasteners (e.g., screw and nut), the fasteners are typically assembled after the structural parts. Proposed SWRL

Table 3
SWRL rules for determining assembly precedence among Fastener and StructuralParts classes.

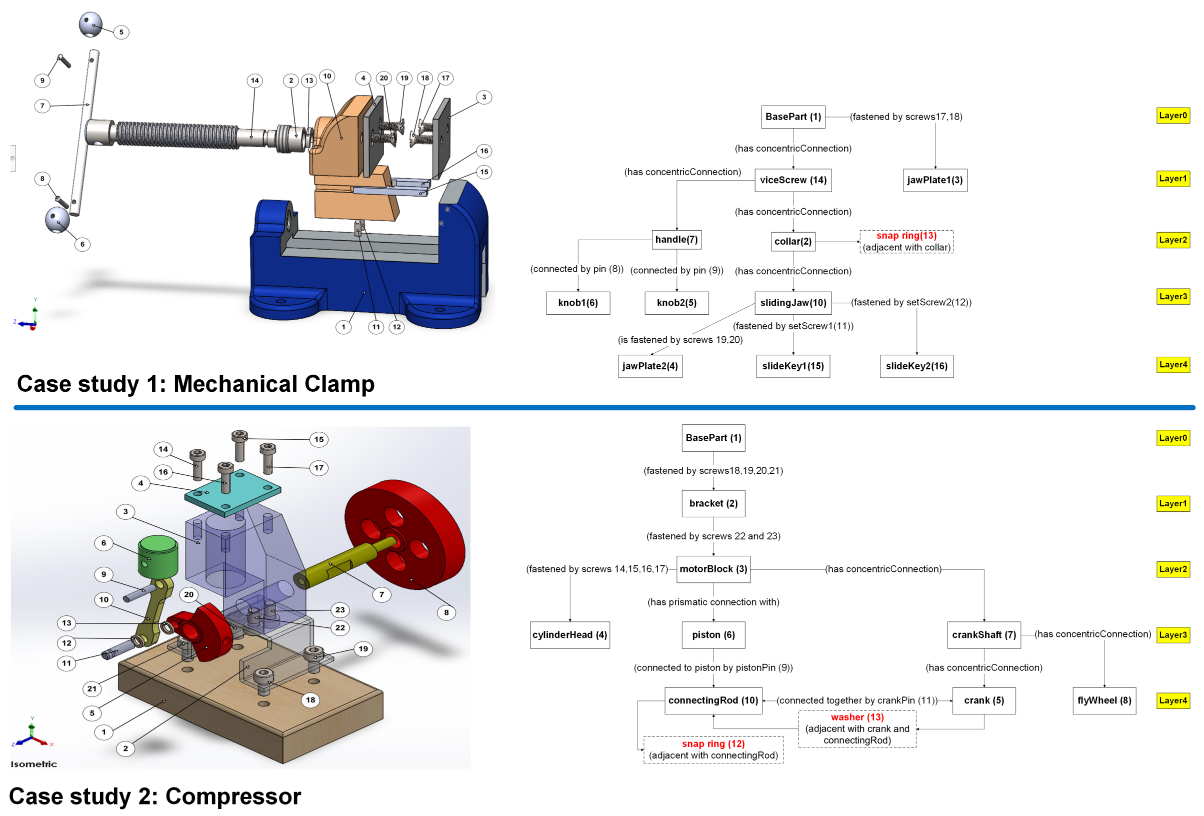
#	SWRL Rules	Explanation
1	`Fastener(?p) ^ StructuralPart(?q) ^ hasConcentricConnectionWith(?p, ?q) → precedesOver(?q, ?p)`	Ensures that when a concentric connection exists between a component of the Fastener class and a component of the StructuralPart class, the StructuralPart is assigned precedence over the Fastener in the assembly sequence.
2	`SnapRing(?p) ^ StructuralPart(?q) ^ isAjacentWith(?p,?q) → precedesOver(?q, ?p)`	When a snap ring prevents the axial movement of a Structural Part to which it is adjacent, the Structural Part must precede the snap ring in the assembly sequence to allow proper insertion and engagement.
3	`Screw(?p) ^ Nut(?q) ^ hasConcentricConnectionWith(?p, ?q) → precedesOver(?p, ?q)`	When a screw and nut are both required to fasten two Structural Parts, the screw is assigned precedence over the nut, as it must be installed first to accommodate the subsequent threading and tightening of the nut.
4	`Washer(?p) ^ Nut(?q) ^ StructuralPart(?r) ^ isAjacentWith(?p,?q) ^ isAjacentWith(?p,?r) ^ differentFrom(?q, ?r) → precedesOver(?r, ?p),precedesOver(?p, ?q)`	When a screw, washer, and nut are all required to fasten two Structural Parts, and the washer has been positioned before the nut during assembly, the washer is assigned precedence over the nut, as it must be installed first to ensure proper load distribution and to accommodate the subsequent threading and tightening of the nut.
5	`Washer(?p) ^ StructuralPart(?q1) ^ StructuralPart(?q2) ^ isAdjacentWith(?p, ?q1) ^ isAdjacentWith(?p, ?q2) ^ differentFrom(?q1, ?q2) ^ hasLayer(?q1, ?L1) ^ hasLayer(?q2, ?L2) ^ swrlb:lessThan(?L1, ?L2) → precedesOver(?q1, ?w), precedesOver(?w, ?q2)`	A washer placed between two StructualParts (`?q1` and `?q2`) must follow whichever part has a `lower` layer value and precede the part with a `higher` layer. This ensures a correct "stacking" order between washer and two structural parts. (Note: Since SWRL rules do not support the 'OR' operator, we need to rewrite the rule separately for each option.)
6-7	`StructuralPart(?p) ^ StructuralPart(?q) ^ hasConcentricConnectionWith(?p,?q) ^ hasLayer(?p, ?L1) ^ hasLayer(?q, ?L2) ^ swrlb:lessThan(?L1, ?L2) → precedesOver(?p, ?q)`	When a concentric (cylindrical) connection exists between two StructuralPart instances—p and q, the part positioned on the lower layer is assembled first. Due to the absence of a logical OR operator in SWRL, this condition is represented using two separate sub-rules (**(L1)** and **(L2)**), each capturing one possible ordering scenario.
8-9	`StructuralPart(?p) ^ StructuralPart(?q) ^ hasCoincidentConnectionWith(?p, ?q) ^ hasLayer(?p, ?L1) ^ hasLayer(?q, ?L2) ^ swrlb:lessThan(?L1, ?L2) → precedesOver(?p, ?q)`	When a coincident connection—fastened by an instance of the Fastener class—exists between two StructuralPart instances, p and q, the part located on the lower layer is assigned assembly precedence.

rules are relatively concise, however, their development requires careful attention to domain semantics. Fewer than 10 rules were sufficient to cover common fasteners and structural parts in the current case studies. The method scales well with increasing part count, since reasoning operates at the component level and avoids low-level feature modeling in OWL. However, certain limitations remain: Our current framework assumes a flat assembly structure without nested sub-assemblies and without significant symmetric layouts.

3. Results

Following the workflow of Section 2, the STEP file of each assembly is processed as follows: **1. Geometry & connection export (JSON):** the parser emits a JSON scene graph that records unique part identifiers and canonical geometry, every detected mating feature, typed as *cylindrical* (hole–boss) or *planar coincident*, and auxiliary relations (e.g., `adjacentTo` for washers and snap-rings). **2. Semantic lift (OWL/-Turtle):** The JSON graph is automatically translated to OWL and loaded into Protégé, instantiating object properties such as `hasConcentricConnectionWith`, `hasCoincidentConnectionWith`, `precedesOver`, class assertions such as `StructuralPart`, `Fastener::Screw`, and the data properties such as `hasLayer`. **3. Rule-based inference (SWRL):** a library of SWRL rules derives precedence predicates between every connected pair, thereby populating the precedence matrix.
We ran the entire pipeline on two open CAD datasets from GrabCAD[1] – a 20-part bench clamp and a 23-part air-compressor (see Figure 4, including their exploded views, connection graphs, and layer-based

[1]https://grabcad.com/library



Case study 1: Mechanical Clamp
BasePart (1)
(fastened by screws17,18)
(has concentricConnection)
viceScrew (14)
jawPlate1(3)
handle(7)
collar(2)
snap ring(13)
(adjacent with collar)
(connected by pin (8))
(connected by pin (9))
knob1(6)
knob2(5)
slidingJaw(10)
(fastened by setScrew2(12))
(fastened by setScrew1(11))
(is fastened by screws 19,20)
jawPlate2(4)
slideKey1(15)
slideKey2(16)
Layer0
Layer1
Layer2
Layer3
Layer4
Case study 2: Compressor
Isometric
BasePart (1)
(fastened by screws18,19,20,21)
bracket (2)
(fastened by screws 22 and 23)
(fastened by screws 14,15,16,17)
motorBlock (3)
(has concentricConnection)
(has prismatic connection with)
cylinderHead (4)
piston (6)
crankShaft (7)
flyWheel (8)
(connected to piston by pistonPin (9))
connectingRod (10)
(connected together by crankPin (11))
crank (5)
washer (13)
(adjacent with crank and connectingRod)
snap ring (12)
(adjacent with connectingRod)

Figure 4: Exploded views and layer-based assembly hierarchy of two case studies; (1) Mechanical Clamp (2) Compressor.

assembly hierarchies.). The ontology-inferred precedence for the Clamp is visualized as a layer-based

Table 4
Produced precedence matrix for the assembly of the mechanical clamp.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1: basePart	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1
2: collar	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
3: jawPlate1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
4: jawPlate2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
5: knob1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6: knob2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
7: handle	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
8: pin2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9: pin1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10: slidingJaw	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0
11: setScrew1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12: setScrew2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13: snapRing	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
14: viceScrew	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
15: slideKey1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
16: slideKey2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
$17: screw_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$18: screw_2$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$19: screw_3$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$20: screw_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

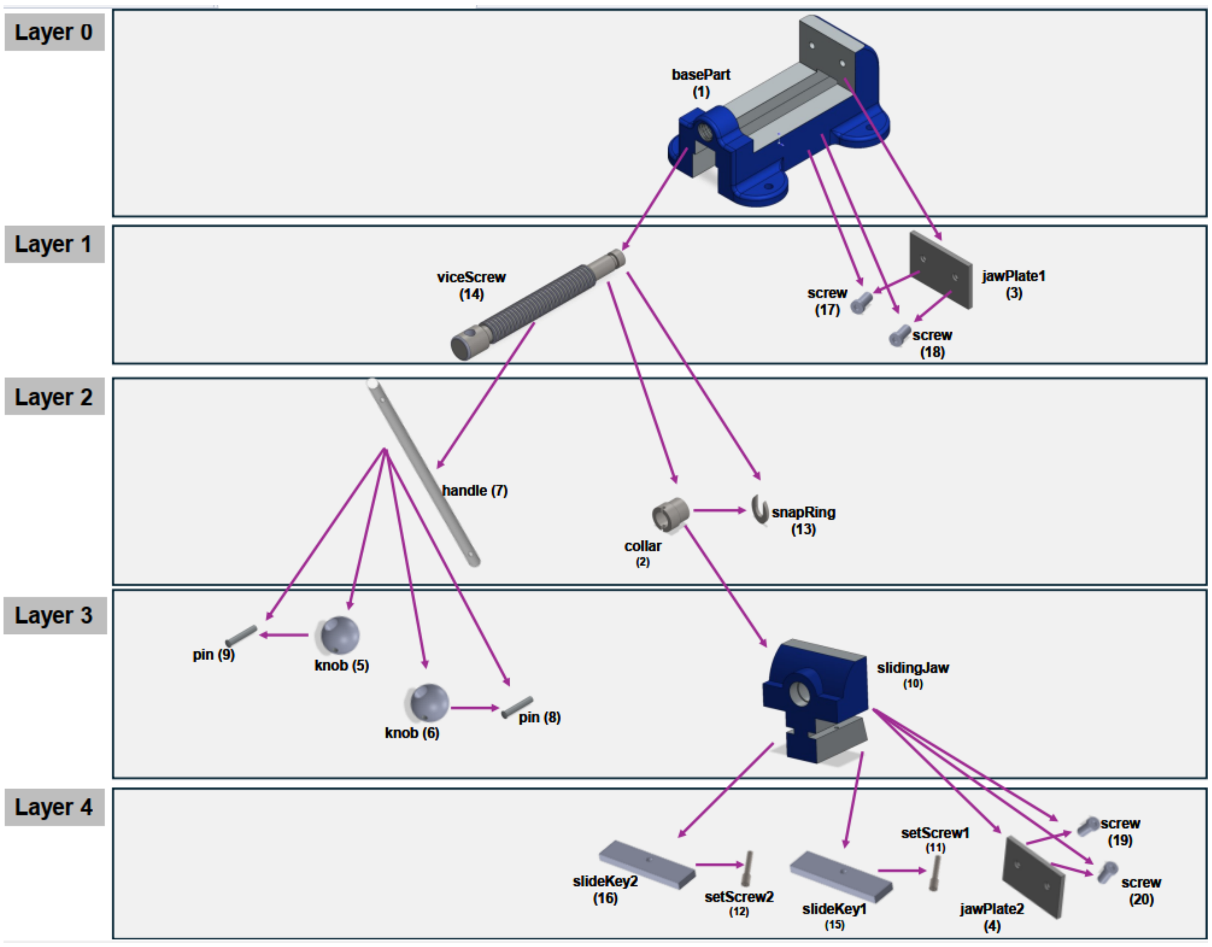


Figure 5: Layer-based decomposition of the Mechanical Clamp extracted from the CAD model. Each component is assigned to a layer based on the assembly process. A directed arrow from component p to q (i.e., $p \rightarrow q$) indicates that p must precede q during assembly, as inferred by ontological reasoning.

decomposition in Figure 5, where components are stratified into defined layers. Tables 4,5 convey the same precedence information in formal, machine-readable form via the binary precedence matrix, enabling direct algorithmic validation and quantitative comparison of assembly orders returned by the SPARQL query after OWL + SWRL reasoning. A "1" in row i, column j means "part i must be assembled before part j", for instance, `basePart` dominates all components in both case studies. Such a precedence matrix compactly encodes the partial-order constraints among assembly tasks; any human or robotic planner can then derive one or more admissible sequences that satisfy those constraints.

4. Conclusion

This work has established a fully automated, ontology-driven pipeline that transforms native STEP geometry into a function-aware precedence matrix for assembly planning. We extracted automatically concentric and coincident connections, instantiated a lightweight OWL domain ontology, and enriched it with SWRL rules that encode generic engineering axioms for functional parts and fasteners and then derived the precedence matrix. Parts are hierarchically stratified into connection-based layers—starting with the BasePart in Layer 0—such that each subsequent layer contains only those components physically attached to parts in all preceding layers. Validation on a 20-component mechanical clamp and a 23-component air-compressor confirms that the inferred matrices reproduce expert-verified ordering constraints.

In any assembly-planning approach, the search space grows with the number of inter-part constraints.

This work is no exception; the phenomenon is inherent to the task rather than to a particular algorithm. A notable advantage of our method, however, is that its layer-based ontological reasoning imposes no upper bound on the number of build layers extracted from the CAD model: precedence inference operates over an arbitrary layered hierarchy without degrading in correctness or completeness. The current implementation relies on manually annotated part names to identify fasteners, which limits full automation when names are missing, non-standard, or ambiguous. In future work we plan to incorporate feature-based recognition. Furthermore, the current two-layer detector handles only *concentric* and *coincident* relations. In future work we will extend the ontology and detection logic to additional constraint types that commonly appear in CAD assemblies – most importantly *threaded engagements* and *gear-gear meshes*, and planar contacts. We also plan to explore aligning our ontology with established standards (for parthood, connection, shape, and function) in our future work.

Acknowledgments

This work was supported by the European Union Next-Generation EU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 3.3 – Decreto del Ministero dell'Università e della Ricerca n.117 del 02/03/2023) within the Advanced-Systems Engineering Ph.D. Program of the Free University of Bolzano and Durst Group AG. We also acknowledge the financial support through the 'Abstractron' project funded by the Autonome Provinz Bozen - Südtirol (Autonomous Province of Bolzano-Bozen) through the Research Südtirol/Alto Adige 2022 Call.

Declaration on Generative AI

During the preparation of this work, the authors used GPT-4 in order to improve grammar and spelling. After using this service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

Table 5
Produced precedence matrix for the assembly of the compressor.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1: basePart	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
2: bracket	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
3: motorBlock	0	0	0	1	0	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1
4: cylinderHead	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
5: crank	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
6: piston	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7: crankShaft	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8: flyWheel	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9: pistonPin	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
10: rod	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
11: crankPin	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
12: snapRing	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13: washer	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$14 : screwM3_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$15 : screwM3_2$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$16 : screwM3_3$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$17 : screwM3_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$18 : screwM4_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$19 : screwM4_2$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$20 : screwM4_3$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$21 : screwM4_4$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$22 : screwM4_5$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$23 : screwM4_6$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

References

[1] M. Rashid, W. Hutabarat, A. Tiwari, A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches, International Journal of Advanced Manufacturing Technology - INT J ADV MANUF TECHNOL 59 (2012). doi:10.1007/s00170-011-3499-8.

[2] X. Guo, M. Zhou, A. Abusorrah, F. Alsokhiry, K. Sedraoui, Disassembly sequence planning: A survey, IEEE/CAA Journal of Automatica Sinica 8 (2021) 1308–1324. doi:10.1109/JAS.2020.1003515.

[3] W. Zhang, M. Ma, H. Li, J. Yu, Generating interference matrices for automatic assembly sequence planning, The International Journal of Advanced Manufacturing Technology 90 (2017). doi:10.1007/s00170-016-9410-x.

[4] G. A. Kumar, M. Bahubalendruni, V. Vara Prasad, D. Ashok, K. Sankaranarayanasamy, A novel geometric feasibility method to perform assembly sequence planning through oblique orientations, Engineering Science and Technology, an International Journal 26 (2022) 100994. URL: https://www.sciencedirect.com/science/article/pii/S2215098621001038. doi:https://doi.org/10.1016/j.jestch.2021.04.013.

[5] B. Bonino, F. Giannini, M. Monti, R. Raffaeli, Automatic assembly precedence detection in axisymmetric products, Computer-Aided Design and Applications (2023) 1175–1189. doi:10.14733/cadaps.2023.1175-1189.

[6] A. Neb, J. Göke, Generation of assembly restrictions and evaluation of assembly criteria from 3d assembly models by collision analysis, Procedia CIRP 97 (2021) 33–38. URL: https://www.sciencedirect.com/science/article/pii/S2212827120314189. doi:https://doi.org/10.1016/j.procir.2020.05.201, 8th CIRP Conference of Assembly Technology and Systems.

[7] Y. Tian, K. D. D. Willis, B. A. Omari, J. Luo, P. Ma, Y. Li, F. Javid, E. Gu, J. Jacob, S. Sueda, H. Li, S. Chitta, W. Matusik, Asap: Automated sequence planning for complex robotic assembly with physical feasibility, 2024. URL: https://arxiv.org/abs/2309.16909. arXiv:2309.16909.

[8] L. Qiao, Y. Qie, Z. Zhu, Y. Zhu, U. K. U. Zaman, N. Anwer, An ontology-based modelling and reasoning framework for assembly sequence planning, The International Journal of Advanced Manufacturing Technology 94 (2018). doi:10.1007/s00170-017-1077-4.

[9] Y. Hu, C. Liu, M. Zhang, Y. Lu, Y. Jia, Y. Xu, An ontology and rule-based method for human–robot collaborative disassembly planning in smart remanufacturing, Robotics and Computer-Integrated Manufacturing 89 (2024) 102766. URL: https://www.sciencedirect.com/science/article/pii/S0736584524000528. doi:https://doi.org/10.1016/j.rcim.2024.102766.

[10] J. Qian, Z. Zhang, C. Shao, H. Gong, D. Liu, Assembly sequence planning method based on knowledge and ontostep, Procedia CIRP 97 (2021) 502–507. URL: https://www.sciencedirect.com/science/article/pii/S2212827120314980. doi:https://doi.org/10.1016/j.procir.2020.05.266, 8th CIRP Conference of Assembly Technology and Systems.

[11] M. T. H. Khan, F. Demoly, K.-Y. Kim, Constructing assembly design model capable of capturing and sharing semantic dynamic motion information in heterogeneous cad systems, International Journal of Advanced Manufacturing Technology 111 (2020). doi:10.1007/s00170-020-06046-7.

[12] K.-Y. Kim, H. Yang, D.-W. Kim, Mereotopological assembly joint information representation for collaborative product design, Robotics and Computer-Integrated Manufacturing 24 (2008) 744–754. URL: https://www.sciencedirect.com/science/article/pii/S0736584508000367. doi:https://doi.org/10.1016/j.rcim.2008.03.010, fAIM 2007.

[13] B. Aameri, H. Cheong, J. Beck, Towards an ontology for generative design of mechanical assemblies, Applied Ontology 14 (2019) 1–27. doi:10.3233/AO-190207.

[14] S. Krima, R. Barbau, X. Fiorentini, R. Sudarsan, R. D. Sriram, Ontostep ::owl-dl ontology for step, 2009. URL: https://doi.org/10.6028/NIST.IR.7561. doi:10.6028/NIST.IR.7561.

[15] Open Cascade SAS, Open CASCADE Technology (occt), https://dev.opencascade.org/, 2025. Accessed: 20 Jun. 2025.