OWL-enabled Assembly Planning for Robotic Agents

Robotics Track

Daniel Beßler
University of Bremen
Bremen, Germany
danielb@cs.uni-bremen.de

Mihai Pomarlan
University of Bremen
Bremen, Germany
pomarlan@uni-bremen.de

Michael Beetz
University of Bremen
Bremen, Germany
beetz@cs.uni-bremen.de

ABSTRACT

Assembly cells run by intelligent robotic agents promise highly flexible product customization without the cost implication product individualization has nowadays. One of the main questions an assembly robot has to answer is which sequence of manipulation actions it should perform to create an assembled product from scattered pieces available. We propose a novel approach to assembly planning that employs Description Logics (DL) to describe what an assembled product should look like, and to plan the next action according to faulty and missing assertions in the robot's beliefs about an ongoing assembly task. To this end we extend the KNOWROB knowledge base with representations and inference rules that enable robots to reason about incomplete assemblies. We show that our approach performs well for large batches of assembly pieces available, as well as for varying structural complexity of assembled products.

ACM Reference Format:
Daniel Beßler, Mihai Pomarlan, and Michael Beetz. 2018. OWL-enabled Assembly Planning for Robotic Agents. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 9 pages.

1 INTRODUCTION

Some industrial actors have expressed interest in developing assembly cells using newly available impedance controlled robot arms. Such cells promise to be versatile and reconfigurable, a desired quality when customization is important. However, several challenges remain before such promises can be delivered. We will focus in this paper on issues of knowledge representation and planning.

Every product that an assembly cell can produce requires a different assembly plan, and every time the product changes (for example because new standards require some parts to be replaced) the plan needs to be adapted or recreated from scratch. Further, plans are sensitive to the kinds of resources available (the parts the cell can access) and to the capabilities of the robot. Finally, when creating small batches of customized products it is often the case that the customizations are variations of each other: using one part instead of another, or varying in terms of which accessories are attached to a common skeleton. In principle, assembly re-planning can tackle all of these issues, but is computationally expensive. Instead, we pursue a knowledge-enabled approach where the robotic agent is informed about its own capabilities and available resources, has knowledge about the products it needs to assemble, and the representation is

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden.

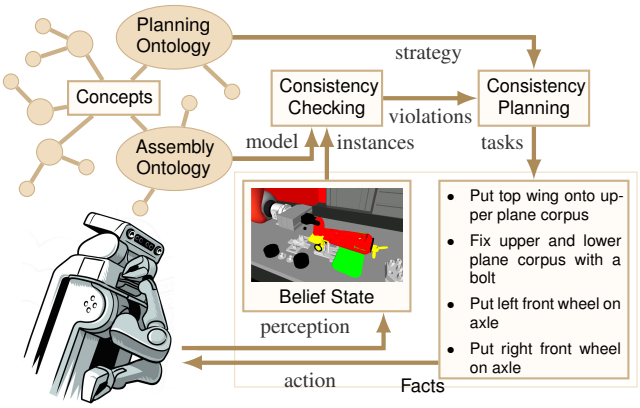

Figure 1: Architecture for OWL-enabled assembly planning that is designed to run within the perception-action loop of a robotic agent.

such that it allows quick adaptation. We use OWL [8], together with rule-based reasoning, to represent and process this knowledge.

Classical planning approaches, such as PDDL [7], make the closed world assumption, and thus may not be able to find a solution if some facts about the world are not known. This is fairly limiting: robots can not begin an assembly until all parts are known, and can not react on external changes of the world state or have to re-plan in such a case. Instead, our planner uses open world semantics such that the robot can begin an assembly activity with incomplete knowledge, identify the missing knowledge pieces, and reason about how the missing information can be obtained. Another difference is that the goal state can be defined in terms of complex class descriptions rather then objects and their roles.

Our approach is to describe, in an ontology, concepts of finished assemblages by their parts, sub-assemblies, how the parts are connected with each other, and how they can be grasped, and to compare these descriptions with what the robot believes about an ongoing assembly task to infer what to do next. Concept definitions include restrictions the finished assemblages must satisfy. The restrictions would initially not be satisfied by an individual, and the robot will act so as to resolve these inconsistencies. In more detail, based on the inconsistencies between an initial state (available scattered parts) and a desired state (a particular assemblage), our system creates an agenda: an ordered list of tasks that, when performed, will transform the assemblage such that the inconsistencies are fixed. This mechanism runs within the perception-action loop of the robot: The robot's beliefs are updated according to objects perceived and actions performed, and assembly actions are selected and parametrized by reasoning about the robot's beliefs and how inconsistencies can be fixed. This architecture is depicted in Figure 1.

The knowledge modeling for assembly processes is also not trivial and, to our knowledge, has not been deeply developed for robots. We extended the robot ontology KNOWROB [20] with general assembly concepts such as mechanical part, atomic part, assemblage, assembly affordance etc. We also added reasoning methods in KNOWROB that allow to reason about assemblages, their parts, connections, and inconsistencies, and how consistency can be established. To this end, we extended KNOWROB's handling of the robot's belief state such that it includes information about (partial) assemblages present, and the connections between their parts.

Summarizing, the contributions of this paper are as follows:

- a novel robot ontology for assembly tasks that describes assemblages, their parts, and how parts should be grasped, held and connected with each other;
- a set of extensions to the KNOWROB reasoner that enables to reason about (partial) assemblages, and to explain inconsistencies of individuals wrt. their terminological definition;
- a novel approach to knowledge-enabled assembly planning that is designed to run within the perception-action loop of a robotic agent, and that uses an agenda of ordered steps that would, when performed, transform an incomplete assemblage into one that is in accordance with its semantic model.

2 ASSEMBLY ONTOLOGY

Assemblages can be described by their parts, sub-assemblies, and connections between them. Robots further need to know how to grasp and hold parts to connect them with each other. We represent this type of information in a novel robot ontology for assembly tasks that we will describe in this section.

2.1 Ontology hierarchy

Assembly upper ontology. The most general assembly-related concepts are defined here. These can be divided into concepts subsumed by *PhysicalPartOfObject* (i.e., tangible parts of some spatial thing) and *Connection-Physical* (i.e., tangible parts in physical contact such that they resist spatial separation), which are defined in KNOWROB's upper ontology. In this ontology, we represent general concepts such as *MechanicalPart* $\sqsubseteq$ *PhysicalPartOfObject*, *Assemblage* $\sqsubseteq$ *MechanicalPart*, and *AssemblyConnection* $\sqsubseteq$ *Connection-Physical*, but also some commonly used classes of connections, parts and assembly affordances (e.g., screwing, sliding and snapping connections). The ontology counts 236 logical axioms and 56 classes, has the DL expressivity $\mathcal{ALCROIQ}(D)$, and is provided as an open-source extension of KNOWROB.

Parts ontology. Part classes, affordances they provide, and types of assembly connections they can enter in are defined in this ontology. Each user of our system would define their own part ontology/ies, as these are project-specific. Parts, however, can be reused for different projects, hence they get a special level in our ontology hierarchy.

Assemblages ontology. Concepts for a top-level assemblage, and the sub-assemblages that it contains are defined in this ontology. These concepts are highly project specific. They describe the assembled product in terms of parts and connections between them, and all intermediate steps that determine how it can be constructed (potentially in different variants).

Simulation ontology. The previous ontologies define classes, rather than individuals. In a real-world use case, individuals such as parts may be asserted by perception, and any dependent individuals (such as the affordances the parts are expected to provide) are asserted through reasoning on part definitions. It is convenient however, especially for simulation experiments (see section 7), to have an OWL file describing the initial state of the world: what parts are available, where they are located, whether sub-assemblages already exist etc.

2.2 Defining Connections and Assemblages

Our knowledge modelling of assembly processes is based on the *AssemblyConnection* concept, where a *AssemblyConnection* may need and/or block *AssemblyAffordances* offered by *MechanicalParts*. We look at these concepts in more detail below.

MechanicalPart. The top concept for objects that get manipulated during an assembly operation. It is subdivided into *AtomicPart*, which refers to parts without separable components, and *Assemblage*. *AtomicPart* is further subdivided into *FixedPart*, useful for representing holders, and *MobilePart*. A *MechanicalPart* must offer at least one *AssemblyAffordance*, which allows the part to be used for certain assembly connections. *MobileParts* should also offer at least one *GraspingAffordance*, which allows them to be grasped in certain ways. Note that both *AtomicParts* and *Assemblages* can offer *AssemblyAffordances*; this is because assembly-relevant features may appear only when more parts are put together, for example a tunnel formed by two longitudinal halves. When a *MechanicalPart* is asserted to the belief state (e.g. because it was perceived or a new assemblage was put together) its affordances can be inferred from existential restrictions that constrain the *hasAffordance* relation, and automatically asserted when the part is instantiated.

AssemblyConnection. The central concept in how we describe assemblages. An *AssemblyConnection* needs two *AssemblyAffordances*, and may block several other affordances. Typically an *AssemblyAffordance* that is needed is also blocked, and this is represented by a *consumesAffordance* $\sqsubseteq$ *needsAffordance* object property; but other affordances, e.g. *GraspingAffordances*, may be blocked by a connection: some grasps become impossible once a part slides into a connection. We define connections in terms of affordances they need (rather than the *MechanicalParts* they connect) so as to also capture which connections *prevent* others from being formed, and therefore have information in the ontology to reason about operation sequencing. Parts involved are linked to the connection via the property chain $\mathit{hasAtomicPart} \equiv \mathit{needsAffordance} \circ \mathit{hasAffordance}^{-}$.

Connections are also associated with a transform that describes how parts are placed relative to each other, and use one of the parts as a "reference" (if the *MechanicalPart* in question is an *Assemblage*, then its reference part is used). The current implementation presents a limitation, in that the parts in a connection cannot be of the same type (or else it would be impossible to disambiguate which should be the reference without further data). We would like to solve this rare issue in a future revision of the assembly ontology.

Assemblages. The aggregates of mechanical parts. An *Assemblage* uses exactly one *AssemblyConnection*, and may place further restrictions on the types of involved parts. This is because connections can be fairly general, such as slide-in connections,

whereas an assemblage may require a particular wheel to be slid on a particular axle. Another important property when defining *Assemblage* subclasses is the *linksAssemblage* property chain that we define as: $linksAssemblage \equiv hasAtomicPart \circ hasAtomicPart^{-} \circ usesConnection^{-}$. Using restrictions involving the *linksAssemblage* property we can explicitly encode sequencing information about assembly operations (e.g., place chairs on a chassis only after all the wheels have been placed) as well as infer what are all the parts needed for a particular assembly task.

3 ASSEMBLY BELIEF STATE

KNOWROB has been designed for service robots acting in the kitchen [4]. The belief state, in this case, contains information about (what the robot knows about) the position of objects in the world and which objects are being grasped at the moment. Another way to look at this is that KNOWROB can answer questions such as *"what objects are present?"*, *"what objects are being grasped?"*, *"what is the transform between tool frame and grasped object?"*, etc.

Assembly robots also need information about how objects are connected into (sub)assemblages. We have extended KNOWROB's belief state to include this information. In other words, we make KNOWROB able to answer queries such as *"what are the parts that are contained in an assemblage?"*, *"what parts are (possibly indirectly) connected to a given part?"*, *"what is the transform between two parts that have a connection between them?"*. These are implemented as Prolog queries, and part of an open-source add-on for KNOWROB. Some example queries are shown in section 7.

In terms of implementation, we have adjusted KNOWROB's handling of belief states so that an object pose (for all objects that are not robot links) is reported in a frame in which the object is fixed. For a free-floating object, this is the *world* frame. For an object that is part of an assembly with none of its component parts grasped, either that object is the reference object for the assembly (in which case its pose is given in world coordinates), or the object pose is given relative to the reference object. A grasped object, or an object that is part of an assembly which has one component grasped, has its pose given in the tool frame of the gripper. The reason for this approach is that there could be many objects available to a robot doing assembly, so to reduce the load on the transform message channel and its subscribers that keep track of where everything is, we publish object poses rarely: whenever the reference frame changes (the object has entered/left an assembly, or a grasp/ungrasp happened), or at some large interval (typically a couple of seconds).

4 REASONING EXTENSIONS

In description logics, ontologies are called consistent if an interpretation exists that satisfies all axioms in the TBox and all assertions in the ABox [1], and it is usually interpreted as an error if no such model exists. The TBox contains intensional knowledge in the form of a terminology, and the ABox contains extensional knowledge. In this paper, we assume that the TBox has no faulty axioms, and exploit that faulty or missing assertions in the ABox can be discovered by checking the assertions against the concepts they describe (e.g., similar to [12]). In this section, we describe reasoning techniques that we employ to *explain* inconsistencies in the ABox.

4.1 Property semantics

Properties can be grouped in a subsumption hierarchy, and axioms in the TBox can restrict their domain and range. Range axioms can be deduced from intersection ($\sqcap$), and union ($\sqcup$) class descriptions. The range of a property P_1 for an intersection class C (denoted $\rho(P_1, C)$) is simply the intersection of, and the range for an union class is the union of inferred range axioms. OWL classes can further constrain property values using universal ($\forall P_1.C_1$), existential ($\exists P_1.C_1$), and has-value ($P_1 : o$) restrictions. Further range axioms $\rho(P_1, C)$ may be inferred for qualified restricitons by exploiting the subsumption hierarchy to find classes K with $C \sqsubseteq K$, and their range for P_1. K can be found by computing the range of the inverse of the restricted property on the class qualified by the restriction. This is, for universal restrictions $\forall P_2.C_2$, the range $K \in \rho(P_2^{-}, C_2)$ which may constrain P_1 for instances of the restriction: All P_1 values must be instance of each range $\rho(P_1, K)$. Finally, cardinality restrictions may be used to specialize range axioms according to constraints on the cardinality of subclasses of the range (i.e., to eliminate subclasses whose cardinality must be zero). The property range of individuals can be deduced from the range of its types, and in the case of a functional or inverse functional property from the specified property value. We expect that the TBox is static during planning and can therefore employ efficient caching for class level range inference.

4.2 Specializability

The assembly ontology restricts assemblages to link certain parts that have certain properties. Initially, some of the restrictions are violated such as that an assemblage must link a particular part type. Given the inferred property range, potential candidates that are instances of the range can be deduced. Individuals that are not instances of the range, on the other hand, could be *specializable* such that they turn into an instance of the range by asserting new facts about them.

DEFINITION 1. *An individual is specializable to a class if one of its types can be specialized, or new (consistent) properties can be asserted such that the individual turns into an instance of the class.*

We write $s \gg C$ if individual s can be specialized to the class description C. Ranges are arbitrary class descriptions, and the specializability test must therefore be defined recursively and for various class constructs that may appear in range descriptions.

ABox specializability. Sufficient conditions for the specializability need to be defined for various possible property range axioms. The most trivial conditions are depicted in Figure 2. An individual s is specializable to a simple class C if one of its types has a subclass in common with C, to a complement class $\neg C$ if it is not an instance of C, and to a union or intersection class if it is specializable to at least one or all of the set members respectively. Similarly, an individual is specializable to a universal restriction $\forall P.C$ if all the existing values of P are specializable to C, and to an existential restriction $\exists P.C$ if at least one of the existing values of P is specializable to C.

Property restrictions may imply minimum cardinality for qualified property values. This is the case for existential, and has-value restrictions, and for qualified cardinality constraints that specify the minimum number of values explicitly. The inferred range of property P may restrict the minimum cardinality of another property P_1 on objects it denotes. Individual s with not enough values for P_1 may

Class axiom	*Specializability condition*
C	$\exists type(s,T) : X \sqsubseteq T \wedge X \sqsubseteq C$
$\neg C$	$\neg C(s)$
$C_1 \sqcap \cdots \sqcap C_n$	$\forall C \in \{C_1,\ldots,C_n\} : s \gg C$
$C_1 \sqcup \cdots \sqcup C_n$	$\exists C \in \{C_1,\ldots,C_n\} : s \gg C$
$\forall P.C$	$\forall P(s,x) : x \gg C$
$\exists P.C$	$\exists P(s,x) : x \gg C$
$P_R : a$	$\exists P(s,a)$

Figure 2: Some sufficient ABox specializability conditions for different class axioms.

still be selected as candidate value for P if it can be deduced that adding the n missing values would not lead to more inconsistencies. We say that s is *decomposable* n-times along the qualified property P_1 if n new qualified values can be asserted:

$$decomposable(s,P_1,C_1,n) \iff$$
$$n = max\_card(s,P_1,C_1) - \#\{x | P_1(s,x) \wedge C_1(x)\}$$

Where $max\_card(s,P_1,C_1)$ is the maximum number of P_1 values that can be instance of C_1. In the next step, it is enforced that C_1 is specializable to every range of P_1: $\forall R \in \rho(P_1,s) : C_1 \gg R$. This ensures that C_1 is a valid type for values of P_1 according to property range constraints on the individual s. Finally, a backwards check ensures there are no range constraints about the inverse of P_1 imposed on C_1 that are conflicting with the description of s: $\forall R^- \in \rho(P_1^-,C_1) : s \gg R^-$.

TBox specializability. Similar to ABox specializability, a simple class is specializable to another class if a common subclass exists, an intersection class is specializable to another class if all its members are specializable, and union classes are specializable to another class if at least one of the members is specializable the other class.

Additionally, qualified restrictions are specializable to a class C if the qualified class C_1 is a valid value for the restricted property P_1 on instances of C (i.e., if the inferred range is specializable to C_1), and if the range of P_1^- on the qualified class is specializable to C (i.e., instances of C_1 must be consistent values for P_1^-):

$$\forall X_1 \in \rho(P_1,C) : X_1 \gg C_1 \wedge$$
$$\forall X_2 \in \rho(P_1^-,C_1) : X_2 \gg C \Rightarrow C_R \gg C$$

Where C_R is either an universal ($\forall P_1.C_1$), existential ($\exists P_1.C_1$), or cardinality ($\leq maxP_1.C_1$ or $\geq minP_1.C_1$) restriction.

Finally, a restriction C_{R1} is specializable to another restriction C_{R2} if the restricted property P_2 is a sub-property of P_1, the qualified class C_1 is specializable to the qualified class C_2 (if any), and (1) C_{R1} and C_{R2} are either both existential or universal restrictions, (2) C_{R1} and C_{R2} are (qualified) cardinality restrictions and C_{R2} subsumes the cardinality of C_{R1} ($min_1 \leq min_2, max_1 \geq max_2$), or (3) C_{R1} and C_{R2} are has-value restrictions with the same value.

4.3 Explaining ABox inconsistencies

In our approach, planning the next step during an assembly activity is driven by identifying and fixing (local) inconsistencies in the ABox of the KB. Individuals are assumed to be consistent wrt. the KB if they are in fact a proper instance of each of their asserted types. Asserted types are arbitrary class descriptions and it can further be deduced up to which statement in the complex class description an individual is an instance of it. We say that a class description is *satisfied up to* the statement at which this test fails. Identifying such statements enables to *explain* why an individual is inconsistent, and is essential for planning actions that establish consistency in the ABox by asserting or retracting facts about an individual.

Consistency checking is restricted to a local context. This is because (1) the KB may contain many facts that are irrelevant for successful assembly, (2) it allows more compact and reusable planning knowledge, and (3) the robot can dynamically switch contexts such that it changes its task without interruption. The context is spanned by the partonomy of the assembly ontology: Assemblages establish connections between affordances of atomic parts that may be linked to some sub-assemblages with some other affordance.

The partonomy of the assembly ontology is reflexive (i.e., the *linksAssemblage* relation). This is why it is not possible to decide the sub-assembly relation solely in the ABox. Instead we need to consider *linksAssemblage* constraints in the TBox that implicitly define the sub-assembly hierarchy. More formally, an assemblage b is a sub-assemblage of another assemblage a if (1) both have a common atomic part: $\exists hasAtomicPart(a,x) : hasAtomicPart(b,x)$, and (2) one of the asserted types of a is a qualified restriction on the *linksAssemblage* property and b is an instance of the qualified class C_1: $\exists type(b,T) : T \sqsubseteq C_1$. An assemblage is considered to be fully specified if it is consistent wrt. its required sub-assemblies (i.e., if the sub-assemblies are fully specified), and if it links all the required direct atomic parts.

Satisfiable Class Descriptions. The consistency of assemblages may break at arbitrary OWL class descriptions such that the inference up to which statement the individual satisfies the description must be recursively defined.

Trivially, an individual a satisfies a simple class description C, with $\neg C(a)$, up to being classified as instance of C. Union or intersection classes are not satisfied if the individual does not satisfy all of or one of the set members respectively. Unions are satisfied up to where a satisfies a union member X for which $a \gg X$ holds (if any), and up to the union class otherwise. Intersections are satisfied up to where a satisfies each of the intersection members.

- *has-value* constraints are violated if the value is not specified, and the restriction is then satisfied up to the specification of this value.
- Universal restrictions are not satisfied by a if not all the values of the restricted property P_1 are instances of the qualified class C_1: $\exists P_1(a,v) : \neg C_1(v)$. The restriction class is satisfied up to where the value satisfies C_1 if the value is specializable to C_1, or up to detaching the value from a otherwise.
- Existential restrictions are violated by a if there is no value of P_1 that is instance of the C_1. The restriction is satisfied up to where one of the existing values satisfies C_1 if any of the existing values is specializable to it, or to the specification of a new value otherwise.

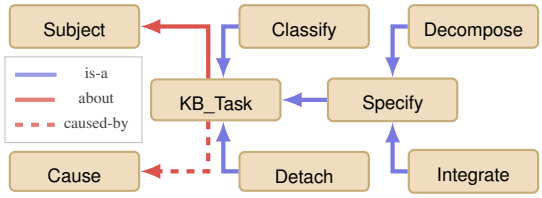

Figure 3: Consistency tasks are caused by some unsatisfied constraint and describe how to fix the inconsistency by asserting or retracting a fact about a subject.

- Qualified cardinality restrictions are not satisfied by a if there are not enough or too many P_1 values that are instance of the class C_1. The restriction is satisfied up to detaching $card - max$ P_1 values that are instance of C_1 if there are too many values (where $card = \#\{v|P_1(a,v) \wedge C_1(v)\}$ is the current, and max the inferred maximum cardinality). In case of not enough values, the restriction is satisfied up to where one of the P_1 values satisfies C_1 if the value is specializable to but not yet an instance of C_1. The restriction is further satisfied up to the specification of $min - card - c$ new values if c existing candidate values were found that can be specialized to C_1, and if $min - card - c > 0$ holds.

5 ESTABLISHING CONSISTENCY

The process of transforming an incomplete and underspecified assemblage to one that is fully specified needs to be *controlled*. We propose a knowledge-enabled approach that allows to switch between different tasks, order actions using domain specific knowledge, explain why actions were planned, and reason about how an action can be performed. In this section, we describe the data structures and knowledge pieces used by this process, and how it can be decomposed into multiple steps.

5.1 Consistency agenda

Actions that were planned to transform an underspecified assemblage into one that is consistent are represented in a data structure that we call *consistency agenda*. Agenda items are generated because (1) an individual needs to be further classified, (2) a required property is not specified and a new value must be instantiated (*decomposition*), (3) a required property is not specified and an existing individual must be used to specify the property (*integration*), or (4) a property value must be retracted (*detaching*). Note that properties can be described as *integratable* if existing values should be used for them. Processing these items requires to select (1) a more specific type, (2) a set of types for newly instantiated individuals, (3) an existing individual, or (4) which existing value should be retracted. The items are caused by some unsatisfied constraint and are concerned with the specification or retraction of a fact about a subject such that the constraint becomes satisfied. Note that the restricted individual with unsatisfied constraints is not necessarily the same as the subject of the item for which a new fact must be asserted (i.e., in case of violations in nested restrictions). Items are represented in an ontology to allow for OWL reasoning about them, and to support the specification of constraints in OWL terms. The consistency agenda ontology is depicted in Figure 3.

(1) Prefer items that were processed less often (Inhibition)
(2) Prefer to stick to the same individual (Continuity)
(3) Prefer *classify* tasks (Pattern)
(4) Prefer *usesConnection* relations of *Assemblage* instances (Pattern)

Figure 4: Sequences of prioritized selection criteria are used to sort the tasks deduced from local inconsistencies.

The agenda can be filtered according to patterns of agenda items. These partial descriptions are represented as OWL classes, and to infer if an agenda item matches a pattern it can be checked whether it is an instance of it. Patterns may include constraints about the items type, subject, and its cause. This allows, for example, to match items that are concerned about subjects with a specific type, or subjects with specific property values.

The fact to be asserted is restricted by the violated axiom that caused the existence of the item. Such constraints are represented using universal property restrictions on the *about* property. The classification of a subject as instance of type T is represented as: $\forall$ *about*.T, and the specification of an instance of class description C as value for property P as: $\forall$ *about*.($\exists$ P.C).

Whenever an item is selected its validity is ensured. Items are caused by a restriction R that is satisfied up to an axiom addressed by the item. The item represents the restriction as: $\forall$ *causedBy*.R, and its validity is checked by testing if there is still an unsatisfied axiom up to which R is satisfied, and which is a specialization of (or the same as) the axiom that caused the item.

As an illustration, some example agenda items are listed and explained below:

- *"decompose TopWingInBody1 (uses$^-$ TopWingLooseOnBody)"* The existing connection *TopWingInBody1* is missing an assemblage that uses the connection. The assemblage must be an instance of *TopWingLooseOnBody*.
- *"integrate TopWingInBody1 (consumes PlaneTopWingSlideInM)"* The existing connection *TopWingInBody1* is missing one of the parts it must link. The part must have an affordance that is an instance of *PlaneTopWingSlideInM*.
- *"detach BoltInHolder1 (consumes BoltAffordance1)"* The bolt with affordance *BoltAffordance1* must be removed from the connection *BoltInHolder1* that attaches the bolt to a holder.

5.2 Consistency strategy

Consistency may break at multiple axioms in the ABox such that a decision has to be made about which axiom to establish next. This is, for example, the case for assemblages with multiple connections that link some parts that are not yet specified. The selection procedure also implies the search strategy (e.g., following the partonomy implies depth first search). Users need to have fine control over the selection procedure to allow for flexible adaption to new planning tasks (i.e., new or adapted assembly tasks). This is one of the reasons we also employ ontologies for the representation of such *selection knowledge*. Figure 4 shows a simple example of a selection strategy.

In section 4.3, we have listed reasons why enforcing consistency in the ABox is restricted to local context. Meaningful context is

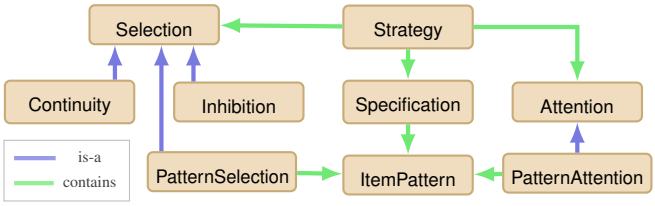

Figure 5: Consistency strategies are composed of strategies for focusing on relevant aspects, the selection of the next task, and procedures that specify consistent property values (or further restrictions).

highly domain dependent. Assembly focuses on atomic parts and their connections with each other while a different planning task could be concerned with cooking a meal from a set of ingredients according to a recipe that is formalized using DL, or even with instantiating a motion controller based on what the robot knows about the task ahead. To allow for such versatile planning tasks we also employ knowledge pieces that describe what is relevant for a given planning task (we say *attention knowledge*) as ontology.

Attention knowledge allows to decompose assembly tasks into multiple phases that are concerned about establishing consistency in different local contexts. For example, the robot may need to put a part onto a holder such that one of its affordances is accessible. This implies a connection between part and holder that is not part of the final product, and thus can not be planned using the description of it. Instead, the process can be decomposed into first putting the parts on holders, and next putting parts together.

The agenda item selection procedure needs domain dependent heuristics to make good decisions. The robot could, for example, decide based on distance or reachability which part would be a good next selection. Further, agenda items may need to be deferred, for example, in case none of the candidate parts is reachable for manipulation. To allow for user defined methods for the specification, restriction, and deference of unsatisfied class statements we also employ knowledge pieces that describe methods for the specification of new facts (we say *specification knowledge*) as ontology.

All *control knowledge* pieces are composed into one data structure that we call *consistency strategy* (depicted in Figure 5). Strategies can be dynamically switched such that the robotic agent can flexibly change between different assembly tasks.

5.3 Procedural view

The ultimate goal is to transform a scene of cluttered parts into an assembled product according to the semantic description provided. This transformation process can be decomposed into following steps:

(1) *Strategy selection* Strategies are collections of knowledge pieces to control the establishment of local consistency using control knowledge. The strategy is selected externally by the robot's plan executive.

(2) *Initialization* Initially, the target assemblage class describing the final product is instantiated. It is also possible to start with an existing partly specified assemblage. The assemblage is traversed such that items are generated for all of its initially linked parts. Items are generated for all the statements up to which the traversed individuals satisfy their types.

(3) *Task selection* Agenda items are sorted according to prioritized criteria that are described in the selection knowledge pieces of the active strategy. The item with highest priority is selected and further processed such that consistency is established wrt. the unsatisfied class statement that caused the existence of the item. Other items that would result in more specific values for the same property and on the same subject are also taken into consideration for additional restrictions on candidate values to avoid unnecessary computation.

(4) *Specification* Consistency is enforced for candidate values wrt. constraints implied by their role in the assemblage. The specification process (e.g., finding a more specific type in case of classify agenda items) can be controlled by methods described in the specification knowledge pieces of the active strategy. It is also allowed that these methods further restrict candidate values instead of fully specifying the unsatisfied axiom. The item is inhibited and re-added to the agenda if no consistent value was found.

(5) *Projection* The specification process does not assert new beliefs, this is done in a separate projection step. New individuals are instantiated according to inferred type constraints, and for each asserted property value, it is checked whether a more specific property can be deduced from constraints on the subject for which the property needs to be specified.

6 ASSEMBLY ACTIONS

We consider two general actions for this paper: Grasping of parts and assemblages, and connecting two parts to form an assemblage. These actions may be performed in many different ways, and with different part and connection types (e.g., screwing and slide-in connections need to be operated with different motions).

General action descriptions, called *action designators*, can be derived from knowledge about the structure of an assemblage, and how parts can be put together. It is up to the plan executive how action designators are instantiated such that they are executable on a particular robot. Additional action relevant information, such as the location of the object, where to grasp it, or its physical properties, can be queried by the plan executive during action designator instantiation to enhance the reusability of the plan schemata.

Connecting Something. Action designators are generated in a bottom-up fashion starting from fully specified assemblages without sub-assemblages, and following the inverse of the sub-assemblage relation until an underspecified assemblage was reached. They include information about the involved parts and their intended connection with each other, and thus can only be generated after this information was specified. An example action designator is depicted in Figure 6. Mapping of partially specified assemblages to action designators executable by the robot is done in an additional step: Each time when new facts about an assemblage were asserted it is checked whether an action can be derived from its description.

The robot has to reason about which of the parts needs to be moved into the other part, and which part should remain fixed during action execution. The fixed part must be held such that the assembly affordance that is intended to connect it to the other part is exposed. Single armed robots potentially need additional tools, such as holders or clamps, to fix one of the parts. Dual arm robots may choose to

```
[an, action,
  [type: 'AssembleTwoParts'],
  [part: [an, object,
    [type: 'PlaneBottomWing'],
    [name: 'PlaneBottomWing_1']]],
  [with-part: [an, object,
    [type: 'PlaneChassis'],
    [name: 'PlaneChassis_1']]],
  [with-connection: [a, thing,
    [type: 'BottomWingSlideInChassis'],
    [name: 'BottomWingSlideInChassis_1']]]
]
```

Figure 6: An action designator generated during assembly planning to be executed by the robot.

hold the fixed part with one gripper in such a way that the other gripper can move the other part into it. The assembly ontology does not represent this type of information (it only models how parts are connected in the final product), and thus it can not be directly inferred which of the objects should remain fixed.

We employ a heuristic for the determination of mobile and fixed part in an assembly action that uses prioritized criteria to decide which of the parts is better suited to be moved into the other one. The prioritized list is as follows: (1) prefer parts with an unblocked grasping affordance, or which are connected to a part with an unblocked grasping affordance (i.e., try not to destroy existing assemblages to be able to grasp some part); (2) prefer parts whose assembly affordance is blocked by a fixture (they must be moved anyway to expose the affordance); (3) prefer parts that are not attached to a fixture; and (4) prefer parts connected to a small number of other parts (move small assemblages into big ones).

Grasping Something. The robot needs to reason about if and how parts can be grasped. Each part may have multiple grasping affordances that imply some way to grasp the part. Assemblages may block the grasping affordance in case it is occluded by some other part. Parts are only graspable if they have at least one grasping affordance that is not blocked by some assemblage. The robot may indirectly grasp a part, however, if it is connected to some other part with an unblocked grasping affordance.

We employ a simple grasping subsumption hierarchy with different types of power and precision grasps. Assembly parts can further describe the contact point, pre-grasp pose, grasping force, and how wide the robot should open its gripper when approaching the object. A connection is established between the end effector of a robot and the part it holds. Grasping connections may block assembly affordances. This allows robots to reason about whether they hold the part in the right way to perform an assembly action.

7 EXPERIMENTS

We characterize the performance of the proposed assembly planning along following three dimensions: Steps required to formalize a new assembly task, what types of queries can be answered about assemblages, and how fast the next action can be deduced.

We selected a toy plane assembly targeted at 4 year old children for evaluation. The toy plane is made of 21 plastic parts that are mainly put together using loose slide in connections, and fixed with bolts afterwards. The parts are comparably huge such that grasping them is easier. We use a single armed robot with a KUKA LWR arm, and test the assembly planning in a Gazebo [10] simulation environment. The plane is part of the YCB Object and Model Set [5].

Modeling. First we create a parts ontology for the toy plane, and populate it with concepts corresponding to the parts (in this case, there are 12 part types). Each part concept defines the assembly and grasp affordances that it offers. We have added the affordances to the definitions by hand, but we will look at ways to automate this, either based on semantic annotations provided with the CAD files as in [15], or possibly some geometric reasoning on part features.

The next step is to create an assemblages ontology and populate it with sub-assemblages, up to the complete toy plane (in this case, 22 sub-assemblages). Each sub-assemblage also defines what sub-assemblages it should contain, which places some ordering constraints on how they are created. In this way we allow the user to describe not just sequencing knowledge but also knowledge about variations: sub-assemblages can be specified in terms of more general classes from the parts ontology (e.g. Wing) as long as any subclass is appropriate (e.g., StraightWing, BacksweptWing).

For the toy plane, we did the parts and assemblages modeling in protégé [13], which took 4 hours. We expect that a dedicated tool (e.g., with a form to specify a list of affordances, rather than the many Add Object Property operations required in protégé) would considerably speed up the process. We had CAD models of the parts available and an instruction sheet on how to assemble the plane.

Querying. In the assembly domain, querying is mostly concerned with the structure and the current state of an assemblage, as well as how the robot should grasp, hold and put together parts.

Assemblages are made of connected assembly affordances. The robot can reason about this structure by asking questions such as *"what are the assembly affordances used in an assemblage?"*:

```
?- assemblage_connection(Assemblage, Connection),
   rdf_has(Connection, 'consumesAffordance', Affordance),
   rdf_has(Part, 'hasAffordance', Affordance).
```

The robot needs to identify incomplete assemblages to reason about what to do next. Local context is enforced by only following the *subassemblage* relation:

```
?- subassemblage(Assemblage, SubAssembly),
   assemblage_underspecified(SubAssembly).
```

The robot can explain why an assemblage is incomplete by identifying unsatisfied axioms in the terminological model:

```
?- owl_instance_from_class(kb:'ToyPlaneBody', Assemblage),
   owl_type_of(Assemblage, Type),
   owl_satisfies_up_to(Assemblage, Type, UpTo).
Type = kb:'Restriction_1',
UpTo = specify(Assemblage, kb:usesConnection, (exactly 1
            (UpperBodySlideInFrame and
            (linksAssemblage some ...))), 1).
```

The robot can also reason about if and how a part can be grasped to assemble it. For example, the grasp is not possible if the assembly affordance would be blocked by the grasp. The robot may put the query as follows:

```
?- assemblage_connection(Assemblage, Connection),
   assemblage_possible_grasp(Assemblage,
         Connection, (Part,Affordance,Grasp)).
```

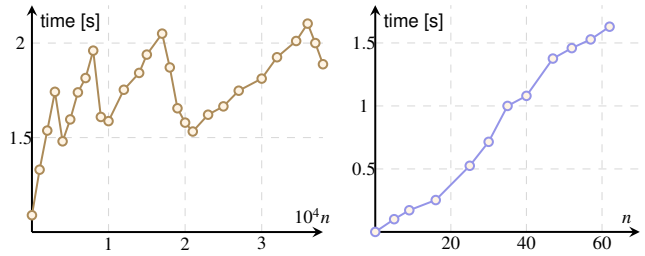

Figure 7: Run time performance for toy plane assembly planning with varying number of available assembly parts (left), and with varying number of constraints the assemblage must satisfy (right).

Run time performance. The run time performance of our approach to assembly planning depends on factors such as the size of the KB, and the structural complexity of the assembled product. Varying the complexity of the assembly ontology itself is difficult. Instead, we choose to vary the structural complexity by testing our approach with different assemblages that the plane ontology describes. Further, we measure the planning performance for growing size of statements in the ABox by varying the number of parts available for assembly.

In the first test, we assert n random toy plane parts in addition to the 21 parts that are required. The parts are asserted together with their affordances, which yields, depending on the part type, from 8 up to 29 triples for each of the asserted parts. We vary n from 0 to 3.8×10^4. We repeat this test 30 times to average out the random selection of part types. With only 21 parts, the planner is able to find a solution in roughly $1.1s$ within 32 planning steps. On average, each step takes $34ms$. In our experiments, the runtime performance dropped maximally to $2.103s$ for the test with 3.6×10^4 additional parts. The performance does, however, not drop continuously. This is caused by hash table behavior of the triple storage employed in SWI Prolog [22]. Anyway, the test shows that the current implementation can deal with huge batches of parts: even with 5×10^5 additional parts a solution was found in $1.75s$

In the second test, we vary the complexity of the assemblage task in terms of constraints the final assemblage must satisfy. The toy plane ontology includes 22 assemblage descriptions with varying complexity from which we choose 11 for this test. The complexity is measured by counting the number of assertions and retractions that were required to make an assemblage consistent. The most simple assemblage requires 5 KB transactions and can be planned in $0.101s$ (averaged over 20 trials), while the most complex assemblage with 62 transactions requires $1.629s$ of planning time. However, the average time between transactions remains nearly constant at $0.025s$ such that online planning is not affected much by the structural complexity of the assembled product.

In more detail, the run time test results are shown in Figure 7.

8 RELATED WORK

There are several efforts to provide ontologies for robotics. By far the largest is that of the IEEE-RAS working group ORA (Ontologies for Robotics and Automation)[18], which aims at standardizing knowledge representation for robotics. The ORA core ontology[17] has been augmented with others for specific industrial tasks[6]. These extensions cover tasks such as kitting, where a robot places a set of parts on a tray or similar receptacle to be carried towards an assembly cell. Other robotic ontologies are the Affordance Ontology[21] and the open-source KNOWROB ontology[20], the latter of which we have chosen to use and expand. None of the mentioned ontologies model assembly concepts as of yet.

As mentioned, the knowledge-enabled approach has been successfully employed for some industrial processes such as kitting[2, 3, 16]. Knowledge-enabled assembly has been investigated by the EU ROSETTA project[9, 14, 19]; their knowledge representation includes concepts for tasks and sequences of tasks (including concepts such as partial order constraints and graphs of tasks) and basic robot skills. Other approaches to knowledge-enabled assembly convert OWL descriptions into PDDL specifications[3, 11]. Knowledge-enabled programming has also been employed as a means to ease teaching a robot cell to assemble new products[15]; similar to our knowledge modelling approach, they insist on annotating geometric data about mechanical parts with information about semantically meaningful features which can then be used to construct constraints and sub-goals for an assembly task description.

In the cited works, the represented knowledge is to be used and exchanged between various system components such as planning, perception, and executives (so as to allow reasoning and replanning) or training interfaces (in which case they provide the "vocabulary" to describe a sequence of tasks in). Generation of action sequences based on the semantic descriptions themselves is not done, unlike our approach. Also, to the extent that knowledge modeling intended for assembly appears, it is either very generally about sequences of tasks (as in [9]) or focuses on geometric features of atomic parts (as in [15]) as opposed to affordances and intermediary sub-assemblages.

9 DISCUSSION

Some relevant issues remain unconsidered in the scope of this paper. This is, for example, the ability to take back decisions in the face of changes that conflict with what was planned. Further, strategy selection could be done rule-based to make plan schemata more reusable. We would also like to apply our approach to other domains such as perception or motion control. Further, we have only worked with a simulated robot. In the future, we want to employ our system on a real robot, and in multi agent scenarios. Finally, some of the manual steps could be automated. For example, we believe that some parts of the terminological model can be auto-generated from existing documents such as technical drawings or construction manuals.

10 CONCLUSION

In this paper, we have proposed a novel approach to assembly planning that employs formal descriptions of what an assembled product should look like to plan the next action according to faulty and missing assertions in the belief state of a robotic agent. We have shown that our approach scales well with growing number of parts available, and also with varying structural complexity of the assembled product such that online planning is feasible. We believe that our approach is a step towards versatile and reconfigurable assembly cells, and that these flexible assembly cells will, in the near future, allow customers to individualize products without the cost implication customization has nowadays.

REFERENCES

[1] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (Eds.). 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA.

[2] Stephen Balakirsky. 2015. Ontology based action planning and verification for agile manufacturing. *Robotics and Computer-Integrated Manufacturing* 33, Supplement C (2015), 21 – 28. Special Issue on Knowledge Driven Robotics and Manufacturing.

[3] Stephen Balakirsky, Zeid Kootbally, Thomas Kramer, Anthony Pietromartire, Craig Schlenoff, and Satyandra Gupta. 2013. Knowledge Driven Robotics for Kitting Applications. *Robot. Auton. Syst.* 61, 11 (Nov. 2013), 1205–1214.

[4] Michael Beetz, Ulrich Klank, Ingo Kresse, Alexis Maldonado, Lorenz Mösenlechner, Dejan Pangercic, Thomas Rühr, and Moritz Tenorth. 2011. Robotic Roommates Making Pancakes. In *11th IEEE-RAS International Conference on Humanoid Robots*. Bled, Slovenia.

[5] Berk Çalli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. 2015. Benchmarking in Manipulation Research: The YCB Object and Model Set and Benchmarking Protocols. *CoRR* abs/1502.03143 (2015).

[6] Sandro Rama Fiorini, Joel Luis Carbonera, Paulo Gonçalves, Vitor A.M. Jorge, Vítor Fortes Rey, Tamás Haidegger, Mara Abel, Signe A. Redfield, Stephen Balakirsky, Veera Ragavan, Howard Li, Craig Schlenoff, and Edson Prestes. 2015. Extensions to the Core Ontology for Robotics and Automation. *Robot. Comput.-Integr. Manuf.* 33, C (June 2015), 3–11.

[7] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. 1998. PDDL–the planning domain definition language. *AIPS-98 planning committee* (1998).

[8] Ian Horrocks, Peter F. Patel-Schneider, and Frank Van Harmelen. 2003. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics* 1 (2003), 2003.

[9] Malec J., Nilsson K., and Bruyninckx H. 2013. Describing assembly tasks in declarative way. In *IEEE/ICRA Workshop on Semantics*.

[10] Nathan Koenig and Andrew Howard. 2004. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In *In IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2149–2154.

[11] Z. Kootbally, C. Schlenoff, C. Lawler, T. Kramer, and S.K. Gupta. 2015. Towards Robust Assembly with Knowledge Representation for the Planning Domain Definition Language (PDDL). *Robot. Comput.-Integr. Manuf.* 33, C (June 2015), 42–55.

[12] Christian Meilicke, Daniel Ruffinelli, Andreas Nolle, Heiko Paulheim, and Heiner Stuckenschmidt. 2017. Fast ABox Consistency Checking Using Incomplete Reasoning and Caching. In *Rules and Reasoning - International Joint Conference, RuleML+RR 2017, London, UK, July 12-15, 2017, Proceedings*. 168–183. https://doi.org/10.1007/978-3-319-61252-2_12

[13] Mark A. Musen. 2015. The ProtÉGÉ Project: A Look Back and a Look Forward. *AI Matters* 1, 4 (June 2015), 4–12. https://doi.org/10.1145/2757001.2757003

[14] Rajendra Patel, Mikael Hedelind, and Pablo Lozan-Villegas. 2012. Enabling robots in small-part assembly lines: The "ROSETTA approach" - an industrial perspective. In *ROBOTIK*. VDE-Verlag.

[15] Alexander Perzylo, Nikhil Somani, Stefan Profanter, Ingmar Kessler, Markus Rickert, and Alois Knoll. 2016. Intuitive instruction of industrial robots: Semantic process descriptions for small lot production. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2293–2300.

[16] Athanasios S. Polydoros, Bjarne Großmann, Francesco Rovida, Lazaros Nalpantidis, and Volker Krüger. 2016. Accurate and Versatile Automation of Industrial Kitting Operations with SkiROS. In *Towards Autonomous Robotic Systems - 17th Annual Conference (TAROS)*. 255–268.

[17] Edson Prestes, Joel Luis Carbonera, Sandro Rama Fiorini, Vitor A. M. Jorge, Mara Abel, Raj Madhavan, Angela Locoro, Paulo Goncalves, Marcos E. Barreto, Maki Habib, Abdelghani Chibani, Sébastien Gérard, Yacine Amirat, and Craig Schlenoff. 2013. Towards a Core Ontology for Robotics and Automation. *Robot. Auton. Syst.* 61, 11 (Nov. 2013), 1193–1204.

[18] Craig Schlenoff, Edson Prestes, Raj Madhavan, Paulo Goncalves, Howard Li, Stephen Balakirsky, Thomas Kramer, and Emilio Miguelanez. 2012. An IEEE standard ontology for robotics and automation. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 1337–1342.

[19] Maj Stenmark, Jacek Malec, Klas Nilsson, and Anders Robertsson. 2015. On Distributed Knowledge Bases for Robotized Small-Batch Assembly. 12, 2 (2015), 519–528.

[20] Moritz Tenorth and Michael Beetz. 2013. KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots. *Int. Journal of Robotics Research* 32, 5 (April 2013), 566 – 590.

[21] Karthik Mahesh Varadarajan and Markus Vincze. 2012. Afrob: The affordance network ontology for robots. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 1343–1350.

[22] Jan Wielemaker, Guus Schreiber, and Bob Wielinga. 2003. Prolog-based infrastructure for RDF: performance and scalability. In *The Semantic Web - Proceedings ISWC'03, Sanibel Island, Florida*, D. Fensel, K. Sycara, and J. Mylopoulos (Eds.). Springer Verlag, Berlin, Germany, 644–658. LNCS 2870.