Integrating Ontologies for Context-based Constraint-based Planning

Uwe Köckemann, Marjan Alirezaie, Lars Karlsson, Amy Loutfi
Center for Applied Autonomous Sensor Systems, Örebro University, Sweden

Abstract

We describe an approach for integrating ontologies with a constraint-based planner to compile configuration planning domains based on the current context. We consider two alternative approaches: The first one integrates SPARQL queries directly with the planner while the second one generates SPARQL queries dynamically from provided triples. The first approach offers the full freedom of the SPARQL query language, while the second offers a more dynamic way for the planner to influence queries based on what is currently relevant for the planner. We evaluate the approach based on how much redundancy is removed by "outsourcing" knowledge into the ontology compared to modeling it directly into the domain of the planner.

1 Introduction

The E-Care@Home project[1] aims to achieve semantic interoperability between information provided by environmental sensors, medical sensors, as well as information from public health records in order to help health care professionals to process and gather information from an Ambient Assisted Living (AAL) environment [Alirezaie *et al.*, 2017]. One major contribution of E-Care@Home is the SmartEnv ontology[2] which as a reusable and publicly accessible knowledge model describes the sensors available in a given environment and provides a way to interpret the data they provide for activity recognition [Alirezaie *et al.*, 2018].

Although the symbolic language of the ontology makes it easy for human users to query the knowledge about the ambient assisted living context, it is still important that users can express their queries in terms of activities and locations without having to refer to specific instances of sensors or actuators. In addition, there may be multiple ways to infer the same information, and available sensors and actuators can vary over time and be different from one home to another. This leads to a need for some form of decision making regarding which sensors are used in what way to achieve information goals. We propose that the problem of automation configuration is seen as a planning problem where configuration planning aims at automatically finding a set of sensor and actuator instances to use for gathering and processing data and to answer information queries.

In this paper, we describe a way to re-use information from the same ontology to dynamically assemble a configuration planning problem. Our main motivation for this is to avoid redundancy in various parts of the E-Care@Home system. It is achieved by compiling the domains for a constraint-based configuration planner for a given context and based on information from the ontology.

Figure 1 illustrates the idea of the approach. We have a model of the environment, a goal in the form of an information request, and an ontology that tells us about available sensors and what information we can obtain from them. From this we want to create and solve a configuration planning problem that allows to configure the smart home environment in order to provide the requested information when it is needed. Using an ontology in this context is convenient because the same set of concepts and relations represented in an ontology can be used for many different purposes (e.g., activity recognition). In most cases, a domain for task planning will only be used for a single purpose and might contain redundancy wrt. other parts of an overall system. We present two approaches that allow a constraint-based planner to include dependencies on ontologies. The first approach is to use SPARQL queries [Prud'hommeaux and Seaborne, 2008] directly. The second approach extends the planner's domain definition language to directly refer to parts of the ontology via triples and prefixes identifying the ontologies that are used. We provide a simple evaluation that shows how we can generate planning operators from an ontology of a realistic smart home environment. Neither approach is linked directly to configuration planning and both can be used for any constraint-based planning problem.

Before going to the details we briefly introduce the SmartEnv ontology in the following:

1.1 SmartEnv Ontology

SmartEnv is an ontology designed to represent different aspects of a smart environment including the objects (e.g., chairs, couches, TVs), their properties (such as pressure, illumination), the feature of interests for a specific observa-

[1] `http://ecareathome.se/`
[2] http://w3id.org/smartenvironment/smartenv.owl

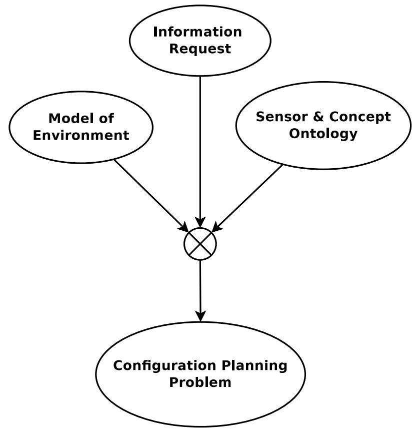

Figure 1: The problem: generate the configuration planning problem from information about the environment, an information request, and ontological information about sensors and concepts.

tion process(e.g., pressure under the couch), the observation (or sensing) process itself including the sensors, rate of sampling and the network configurations. SmartEnv also provides the representational basis required to model the temporal and spatial aspects of the environment, events or activities referring to a change that can interpreted as a specific situation in the environment (e.g., the pressure sensor under the couch is triggered which is interpreted as some one is sitting on the couch). Further details with scripted scenarios can be found in [Alirezaie *et al.*, 2018].

The rest of the paper is formed as follows: Section 3 provides a short description of constraint-based planning. Section 4 describes how the constraint-based planner can be extended to use SPARQL queries. Section 5 describes the alternative triple-based approach. Finally, Section 6 provides an initial evaluation based on the reduction of redundancy offered by the ontology-based approach.

2 Related Work

The constraint-bases planner described in Section 3 was described by [Köckemann, 2016; Köckemann *et al.*, 2014] who used it to provide solutions for several challenges of human-aware planning. Configuration planning combines task planning, which attempts to reach a set of goals by applying operators (or actions), with information dependencies and information goals. This can be viewed as an extension of the task planning problem. Operators may require information to be applicable (*information dependency*). A robot, for instance, may require a camera feed of a ceiling camera in order to move safely through the environment. Indeed, gathering information might even be the goal (i.e., an *information goal*) which requires task planning to be achieved.

[Lundh, 2009] describes an early approach to configuration planning based on an extension of hierarchical task-network planning [Nau *et al.*, 2003]. Operators are extended to functionalities that describe input and output of information. The configuration planning problem is solved as a composition of the HTN planning problem and the problem of finding a configuration that satisfies all information dependencies.

[Di Rocco *et al.*, 2013] go one step further and integrates information dependencies with causal, temporal and resource reasoning. Information is still treated as static in the sense that it does not consider information being available for only a limited temporal interval. Configuration planning problem with multiple, partially-ordered preferences was considered by [Silva-Lopez *et al.*, 2015]. In [Köckemann and Karlsson, 2017], we presented two approaches to integrate information dependencies into the constraint-based planner SpiderPlan[3]. The goal-based approach simply adds information goals and dependencies as goals and preconditions to the planner's operators. This allows to simply use the planner without extension. The second approach defines a sub-problem to satisfy information dependencies as a *Constraint Satisfaction Problem (CSP)* that can be solved on its own but may introduce further task-requirements for the planner. The goal based approach tends to be faster, while the CSP-based approach provides better quality solutions (since it uses optimization on the sub-problem).

Existing work on combining planning with ontologies includes the work by Galindo and Saffiotti [2013] who model norms as part of an ontology to create goals in case robot observations deviate from the norm. McNeill *et al.* [2005] discuss the translation from ontologies to the Planning Domain Defintion Language (PDDL) [Ghallab *et al.*, 1998]. In comparison, our approach is focused on integration, rather than translation. In this way, we can focus on fetching relevant available information from ontologies, while modeling operators and their constraints in the planner's domain definition language. Considering the connection to the ontology as a new constraint type that is solved by a dedicated solver leads to a clear conceptual separation.

3 Constraint-based Planning

The approach to constraint-based planning that we use in this paper models different types of knowledge via different types of constraints. Solvers for constraint-based planning problems can be composed of solvers for individual constraint types. Each of these types poses a sub-problem and solving all sub-problems leads to a solution to the overall problem. A sub-problem for a constraint type can have three different outcomes. It can be *satisfied*, *unsatisfiable*, or require to *resolve a flaw*. In the latter case, there might be a choice of resolvers that need to be explored systematically. For each constraint type we assume a solver that decides if a sub-problem can be resolved and how. This search over resolvers of flaws for all constraint types defines the constraint-based planning problem as a search problem. Our implementation of this approach (SpiderPlan), supports an extendable domain definition language. In the following we will go over the basics of the representation, but we will keep the description of the actual planning approach to a minimum since we are more interested in problem generation.

A *Constraint Database (CDB)* Φ is a set of constraints of different types. Constraint types considered in this paper are

[3] `spiderplan.org`

statements, arithmetic (math) and cost constraints, temporal constraints, open goals, and interaction constraints. We will go over each type in turn. Later, we also introduce a new type of constraint for configuration planning that will be utilized in our second approach. In our approach to constraint-based planning, CDBs constitute the problem, as well as the solution. In the following section, we show how a complex CDB can be generated from a simpler CDB that depends on information from an ontology. In the remainder of this section we go over some constraint types that we use in the example domains in the next sections.

The most basic type is *statement*. Statements relate state-variable assignments to temporal intervals. They have the form $(\mathcal{I}\ x\ v)$ where $\mathcal{I}$ is a temporal interval, x is the state-variable and v the value. We use statements to model both the context for the task planner (initial state, preconditions, and effects), as well as information that is available during a temporal interval.

Example 1. *The first two statements below represent the location of a robot and an object. The third statement is used to represent which object the robot is facing. The last two statements actually represent information that is available during their intervals.*

```
(:statement (I1 (at robot) location)
  (I2 (at object) location)
  (I3 (facing robot) object)
  (I4 (camera-feed-raw robot object))
  (I5 (camera-feed-processed object)) )
```

A temporal context for statements is established by *temporal constraints* that can impose, e.g., flexible durations, release times or precedence constraints. We express temporal constraints via quantified Allen's interval relations Allen [1984]; Meiri [1996] between statements in CDBs. For convenience, we add disjunctions of conceptually neighboring constraints such as *during-or-equals* Freksa [1992]. We also add the unary temporal constraints *release*, *deadline*, *duration* and *at*.

Example 2. *The following temporal constraints applied to the statements from Example 1 state that the robot is facing the object while they are both at the same location. The raw camera feed is provided during the interval where the robot is facing the object. The two bounds state the difference between start and end times of the two intervals. Finally, the processed camera feed is provided with a temporal delay of 10 time units.*

```
(:temporal (during I3 I1 [1 inf] [1 inf])
  (during I3 I2 [1 inf] [1 inf])
  (during I4 I3 [1 inf] [1 inf])
  (overlaps I5 I4 [10 inf] [1 inf]) )
```

Goals are statements that the task planner is supposed to achieve. Temporal constraints on the intervals of goal statements allow to define during what interval the goal has to be reached. Goal constraints require operators that define which statements can be reached from a CDB. An operator o is a tuple $(name, \mathcal{P}, \mathcal{E}, \mathcal{C})$ that consists of a name, a set of precondition statements $\mathcal{P}$ that must be in a CDB for the operator to be applicable, a set of effect statements $\mathcal{E}$ that are added when the operator is applied, and a set of constraints $\mathcal{C}$ that must be satisfied when the operator is applied. Usually, $\mathcal{C}$ contains temporal constraints that relate the temporal intervals of preconditions and effects to the operator itself. If an operator's effect adds a statement that can be matched to a goal that goal can be satisfied by connecting them with a temporal equals constraint (forcing effect and goal intervals to have the same solution space). In the next section we will see how operators can be used to establish information links.

Example 3. *The following operator requires a raw camera feed and produces a processed one. Information is represented by precondition and effect statements. The temporal overlaps constraint assures that the required information is available before processing (interval ?P has to start at least 10 time units before interval ?E). The operators interval (represented by ?THIS) is equal to the effect interval.*

```
(:operator (process-camera-feed)
  (:preconditions (?P (camera-feed-raw ?R ?O)) )
  (:effects (?E (camera-feed-processed ?O)) )
  (:constraints (:temporal (equals ?THIS ?E)
     (overlaps ?P ?E [10 inf] [1 inf]) ) ) )
```

The following goal can be achieved by the example operator if the raw camera feed is available during a temporal interval that can satisfy temporal constraints of both goal and operator. The temporal equals constraint is used to link effects with goals that they achieve (as shown below).

```
(:goal (G (camera-feed-processed object)) )
(:temporal (at G [0 50] [80 inf])
  (equals G I5) ) ;; G is satisfied by I5
```

Cost constraints are used to add up costs. *Math constraints* can be used to perform calculations. In this paper we use these constraints to calculate and sum up the costs of using sensors based on information from the SmartEnv ontology. *Interaction Constraints (ICs)* essentially allow to directly model flaws and resolvers in terms of any constraint type used by the planner. Flaws and resolvers if ICs are user-provided CDBs. This can be used, for instance, to model social acceptability of generated plans. Using a camera during specific human activities may be considered unacceptable. This could be modelled by two statements (representing camera usage and human activity) and a temporal constraint (these statements can overlap in time). Possible resolvers could impose temporal constraints to make sure that the camera is used before or after this human activity. *ICs* are a very powerful tool to express complex situations and how they should be resolved. In this paper we use them to dynamically assemble value-domains for variables based on the information found in an ontology.

Since we focus on context-based problem construction, we omit the details of the algorithms and the description of individual solvers. Details can be found, e.g., in Köckemann [2016].

4 Integrating SPARQL Queries into Constraint-based Planning

SPARQL is a semantic query language which is applied to query the data represented in RDF (Resource Description Framework) [Carroll *et al.*, 2004]. In this section we describe how we integrate SPARQL (under the Jena framework) into constraint-based planning. The basic idea is to include a SPARQL query from a separate file and define how the variables in this query are linked to the variables used by the planner.

As mentioned above, SmartEnv is the ontology that we use in this work. On the side of the constraint-based planner our interest is to execute queries that provide us all ways to produce information, together with the circumstances (i.e., task requirements) under which the information is valid.

One of the main advantages of the constraint-based planning approach described in Sections 3 is that it can be extended easily with new types of constraints. To achieve our goal of integrating ontologies and configuration planning we introduced a constraint type that utilizes SPARQL queries and allows to substitute their results into our configuration planning domain. For this to be possible, we need several components:

1. Ontology description (TBox), referring to the general knowledge common to all sensors and smart homes.
2. Ontology data (ABox), referring to individuals defining specific smart homes and sensors.
3. SPARQL query (posing questions to the ontology)
4. SPARQL expressions (connecting SPARQL query to constraint-based planner)

Given the TBox and the ABox of the ontology, this solution requires the user to define queries in SPARQL and relate the variables retrieved as answers to these queries to expressions used in the planner's domain definition. The query is then executed and its results are substituted into the planner's domain. This effectively allows to define operators in a context determined by the content of the ontology. As a result we can remove all domain specific knowledge (except the relations used in the queries) on the planner's side. A comprehensive example is provided below.

Example 1. *Here we import the ontology ABox together with the scripts of seven different queries. This will allow us to refer to the corresponding queries in Example 4. The file names could be substituted by URLs:*

```
(:include
  (ecare-ontology-data "./ecare-data.rdf")
  (query-sensor-none "./queries/
      query-sensor-none.sparql")
  (query-sensor-config "./queries/
      query-sensor-config.sparql")
  (query-sensor-location "./queries/
      query-sensor-location.sparql")
  (query-sensor-targeting "./queries/
      query-sensor-targeting.sparql")
  (query-link-2 "./queries/query-link-2.
      sparql")
  (query-link-3 "./queries/query-link-3.
      sparql")
  (query-link-4 "./queries/query-link-4.
      sparql") )
```

Example 2. *A simple SPARQL query that requests all the information that can be provided by sensors without any special requirements. The prefixes assigned to the ontologies are used in the body of the queries as the replacement of the ontology URIs to make the queries shorter.*

```
PREFIX ecare: <http://aass.oru.se/~mae/files/
    ontologies/SmartHome_TEMPLATE.owl#>
PREFIX sensing: <http://aass.oru.se/~mae/
    files/ontologies/patterns/
    SmartHome_Sensing.owl#>
PREFIX foi: <http://aass.oru.se/~mae/files/
    ontologies/patterns/
    SmartHome_FeatureOfInterest.owl#>
PREFIX dul: <http://www.
    ontologydesignpatterns.org/ont/dul/DUL.
    owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf
    -syntax-ns#>

SELECT ?Sensor ?Info ?Cost

WHERE {
    ?Sensor rdf:type sensing:SmartHomeSensor.
    ?sensingProcess dul:implementedBy ?Sensor.
    ?sensingProcess rdf:type sensing:
        SensingProcess.
    ?sensingProcess dul:hasConstituent ?Info.
    ?Info rdf:type foi:
        SmartHomeFeatureOfInterest.
    ?sensingProcess sensing:hasCost ?Cost.
    ?sensingProcess sensing:
        requiresConfiguration none.
      ?sensingProcess sensing:
          requiresLocation none.
    ?sensingProcess sensing:requiresTargeting
        none. }
```

Example 3. *An extension of the previous example that returns all information provided by sensors that have a specific configuration ?Config. The difference to the previous example is the extra variable (prefixes are omitted). The following corresponds to* $query - sensor - config$*:*

```
SELECT ?Sensor ?Info ?Config ?Cost

WHERE {
    ?Sensor rdf:type sensing:SmartHomeSensor.
    ?sensingProcess dul:implementedBy ?Sensor.
    ?sensingProcess rdf:type sensing:
        SensingProcess.
    ?sensingProcess dul:hasConstituent ?Info.
    ?Info rdf:type foi:
        SmartHomeFeatureOfInterest.
    ?sensingProcess sensing:hasCost ?Cost.
    ?sensingProcess sensing:
        requiresConfiguration ?Config.
      ?sensingProcess sensing:
          requiresLocation none.
?sensingProcess sensing:requiresTargeting
    none. }
```

Once imported, queries like the ones shown above can be used by operators to decide when they are applicable.

Example 4. *The operator below uses the query shown in Example 2 to determine all sensors that can provide information without special requirements. Notice that the variables used by this operator appear in various other places in the operators definition. If we assume that the data in the ontology does not change, we can replace the operator by a set of operators using every combination of the variables* $?Sensor$, $?Info$, *and* $?Cost$ *that are returned by the query* $query - sensor - none$.

```
(:operator
  (infer-sensor ?Sensor - sensor ?Info -
     concept ?Cost - cost)
  (:preconditions  )
  (:effects
    (?E1 (inferring ?Info))
    (?E2 (sensor-state ?Sensor) on) )
  (:constraints
    (:temporal
      (equals ?THIS ?E1)
      (during ?THIS ?E2 [1 inf] [1 inf]) )
    (:sparql
      (query-sensor-none ecare-ontology-data ?
         Sensor ?Info ?Cost) )
    (:math (eval-int (intervalCost ?E1) (mult (
       sub (EET ?E1) (LST ?E1)) ?cost)))
    (:cost (add link-cost (intervalCost ?E1)))
       ) )
```

Since we can use any existing constraint type as part of an interaction constraint, the following examples shows how we can use *ICs* to create domains for the temporal configuration planning approach presented by Köckemann and Karlsson [2017].

Example 5. *The IC below allows us to add information links based on SPARQL queries. ICs can be preprocessed in the same way as operators. In the case below the IC would be replaced by one for every allowed combination for sensor, information, and cost. The resulting IC's condition is always satisfied since it's only condition was already taken care of. Thus all links will be added as resolvers.*

```
(:ic
  (add-basic-sensor-link ?Sensor - sensor ?
     Info - concept ?Cost - cost)
  (:condition
    (:sparql
      (query-sensor-none ecare-ontology-data ?
         Sensor ?Info ?Cost) ) )
  (:resolver (:configuration-planning (link ?
     Info {?Sensor} ?Cost)) ) )
```

5 Dynamically Generate SPARQL using Triples

As shown above, the planner can make use of SPARQL to "natively" query an ontology. There are, however, good reasons to avoid including SPARQL queries in the planner's domain, not least the fact that adding another language fragment to the domain definition language can affect its readability. We therefore investigate how queries to an ontology can be expressed directly in the language of the planner, in the form of *ontology expressions*. These expressions are converted to a SPARQL query, whose answers are then substituted to the relevant variables used by the planner. The main difference to the approach presented in the previous section is in the inclusion of triples in the domain definition language of the planner. As a result, we can substitute into these triples based on the outcome of other reasoning processes. This could be used, for instance, to determine which relation to use in a triple.

In comparison, SPARQL queries in the previous approach are static. As we will see below, the advantage in using SPARQL directly is that we can rely on its full set of features. Dynamically generated queries are limited to triples we require to be present and triples we do not allow. When the planner solves a set of ontology constraints, it will generate a SPARQL query, run i, and provide all possible answers. The last part uses the same mechanism as the previous approach. In the following we will look at a series of examples to illustrate the approach.

Example 6. *First we include an ontology from a file. Then, we define a set of prefixes (essentially a way to shorthand queries). Finally, we provide an operator that uses a set of ontology expressions to decide to which situations it can be applied. We require the sensor object to be of type* smh_sensing:SmartHomeSensor. *The sensing process* $?Sensing$ *has to be of type* smh_sensing:SensingProcess. *The sensor has to implement the sensing process.*

```
(:initial−context
  (:include (sensor-ont "./SmartHome_ORU_Home1.
     owl"))
  (:ontology sensor-ont
     (prefix rdf "<http://www.w3.org
        /1999/02/22-rdf-syntax-ns#>")
     (prefix owl "<http://www.w3.org/2002/07/
        owl#>")
     ...
     (prefix ssn "<http://purl.oclc.org/NET/
        ssnx/ssn#>")
     (prefix dul "<http://www.
        ontologydesignpatterns.org/ont/dul/
        DUL.owl#>") ))

(:operator (sense-basic ?Sensor ?Sensing)
  (:signature sensor sensing)
  (:preconditions
    (?P (available ?Sensor)) )
  (:effects
    (?E (sensing ?Sensing)) )
  (:constraints
    (:temporal
      (during ?E ?P [1 inf] [1 inf]) )
    (:ontology sensor-ont
       (triple ?Sensor "rdf:type" "
          smh_sensing:SmartHomeSensor")
       (triple ?Sensing "rdf:type" "
          smh_sensing:SensingProcess")
      (triple ?Sensing "ssn:implementedBy" ?
          Sensor) )))
```

The advantage of this approach is in its integration into the domain definition language. This makes it possible to dynamically decide, for instance, which relations are used in a query by using variables to represent them. Sometimes we need negative relations to express that a certain property does not hold in the ontology. Consider the following examples (omitting initial context).

Example 7. *Again we have an operator for using a sensor to provide a sensing process. This time we also consider that the sensor needs a specific configuration (e.g., settings its sampling rate). The main difference to Example 6 is the extra effect that assigns the sensor configuration to a temporal interval and the two extra triples that relate the configuration to sensor and sensing process. This allows the planner to assure (via scheduling) that each sensor only uses a single configuration at any given time. The presence of this operator, however, creates a problem with Example 6. The operator in Example 6 will allow every operator* $sense - config$ *allows because its triples are a subset.*

```
(:operator (sense-config ?Sensor ?Sensing)
  (:signature sensor sensing)
  (:preconditions
    (?P (available ?Sensor)))
  (:effects
    (?E1 (sensing ?Sensing))
    (?E2 (config ?Sensor) ?Config))
  (:constraints
    (:temporal
      (during ?E1 ?P [1 inf] [1 inf])
      (equals ?E1 ?E2) )
    (:ontology sensor-ont
      (triple ?Sensor "rdf:type" "
          smh_sensing:SmartHomeSensor")
      (triple ?Sensing "rdf:type" "
          smh_sensing:SensingProcess")
      (triple ?Sensing "ssn:implementedBy" ?
          Sensor)
      (triple ?Sensor "dul:hasSetting" ?Config
          )
      (triple ?Config "rdf:type" "
          smh_sensing:SensorConfiguration"))))
```

What we really want in the $sense - basic$ operator is to get all the cases sensors and sensing processes that do not require a specific configuration to be used. This is where negative triple come in. The following example is a version of $sense - basic$ that works as intended.

Example 8. *The operator below is identical with Example 6 except for including three* "not" *triples that impose that there is no sensor configuration specified for a given sensor and sensing process combination.*

```
(:operator (sense-basic ?Sensor ?Sensing)
  (:signature sensor sensing)
  (:preconditions
    (?P (available ?Sensor)))
  (:effects
    (?E (sensing ?Sensing)))
  (:constraints
    (:temporal
      (during ?E ?P [1 inf] [1 inf]))
    (:ontology sensor-ont
      (triple ?Sensor "rdf:type" "
          smh_sensing:SmartHomeSensor")
      (triple ?Sensing "ssn:implementedBy" ?
          Sensor)
      (triple ?Sensing "rdf:type" "
          smh_sensing:SensingProcess")
      (not ?Config "rdf:type" "
          smh_sensing:SensorConfiguration")
      (not ?Config "dul:isSettingFor" ?Sensor
          )
      (not ?Config "dul:isSettingFor" ?
          Sensing) )))
```

One issue with the approach presented in this section is that it lacks the modeling freedom offered by writing queries in SPARQL directly. We might, for instance, want to count the number of information inputs needed to compute some new information. The following query is an example that counts how many events are required as preconditions for a complex event.

```
SELECT ?event (COUNT(?event) AS ?
    numberOfPreconditions)
WHERE {
      ?event rdf:type event:ComplexEvent.
      ?event dul:hasPrecondition ?ec.
      ?ec rdf:type event:EventCondition.
} Group by ?event
```

In such cases we can revert to the approach presented in the previous section. In fact, the constraint-based planner can easily use both approaches in a single domain.

6 Evaluation

In this section we provide a preliminary evaluation of our second approach. We use the SmartEnv ontology with an instantiation of a real home. For this we provide a domain definition for the planner that uses ontology expressions presented in the previous section to compile a concrete set of operators and to populate type domains. We query the ontology to create operators (as shown in the previous section) and to populate the domains of the types used by the planner. We then use the number of domain and operator expressions in the domain before and after this step as an indicator for the amount of information that was taken from the ontology.

Example 9. *The following example uses an Interaction Constraint* (IC) *to add every sensor known to the ontology to the domain named* sensor.

```
(:ic (add-sensors-to-domain ?Sensor)
  (:condition (:ontology sensor-ont (triple ?Sensor
       "rdf:type" "
      smh_sensing:SmartHomeSensor")))
  (:resolver (:domain (enum sensor { ?Sensor }) )
      ))
```

The pattern shown in Example 9 is used for all types in the domain (sensors, sensing processes, features of interest, and

	Before	After
Operators	4	62
Domain constraints	4	80

Table 1: Number of operators and domain constraints before and after information was fetched from the ontology.

situation, see Section 1.1). This technique is very convenient since it maintains a strong connection to the ontology by importing its terms.

To measure how much redundancy is avoided we compare the size of the planner's domain before and after the ontology is queried. Specifically, we will measure how many operators we have before and after and how many domain constraints. The four initial domain constraints simply define the types with empty domains. The four planning operators include the ones from Examples 7 and 8 and two operators that extract a feature of interest form a sensing process and a situation from a feature of interest, in turn. The resulting domain definition is completely independent of the concrete smart home environment at hand.

Table 1 shows the resulting numbers of operators and domain constraints. While we could avoid writing down all 62 operators by other means (e.g., Prolog rules or preconditions), we would have to repeat this effort for every instance of a smart home environment instead of relying on existing knowledge. Domain constraints would be completely redundant wrt. the ontology as the usual approach is to list all elements of a domain in the planner's domain definition directly. All it takes to change to another environment is to replace the file containing the ontology instances (see *include* in Example 6).

7 Conclusion

We proposed a solution to re-use information from an ontology to dynamically assemble a configuration planning problem in order to avoid redundancy in various parts of the E-Care@Home system. For this, we presented two alternative solutions for integrating ontologies into a constraint-based planner. In the first approach we include SPARQL queries into the planner's domain definition language. In the second one we rely on triples and prefixes. The first approach allows more flexibility when it comes to queries since we can write SPARQL directly we have access to all it's features. The second approach automatically generates SPARQL. It is less flexible in that way but allows us to dynamically substitute the results of other solvers used by the planner into a query. In this way we could, for instance, dynamically decide which relation is of interest for a specific query depending on the context.

Our evaluation gives a first impression on the way in which integrating ontologies can avoid redundancies in the domain definition of the planner. Our current domain does not yet cover aspects such as sensors that require targeting or mobile sensors (or robots). Since the evaluation presented here is based on a real smart home setup, we simply do not have any mobile sensors or sensors which require targeting in our domain. They can be added easily, however, in the same way in which we added configurations.

For now the presented approach provides operators only from existing structures. That is the operators themselves exist, but information from the ontology instantiates them. An interesting direction for future work is to dynamically generate structures of operators based on the same reasoning. As an example, consider configuration planning operators that require task planning for different reasons: A sensor may need to be pointed at an object and/or in the same location as the object and/or be in a specific configuration to provide the required information. These three things could be modeled as preconditions. In the current approach, however, we would need to supply 8 template operators for all different combinations of preconditions. Adding another type of task requirement would further multiply the number of templates needed. This problem could be solved by dynamically deciding when to add a precondition to an operator. A general mechanism for this type of problem would be very interesting for constraint-based planning in general.

Acknowledgments:
The work is supported by the project E-care@home funded by the Swedish Knowledge Foundation 2015-2019.

References

Marjan Alirezaie, Jennifer Renoux, Uwe Köckemann, Annica Kristoffersson, Lars Karlsson, Eva Blomqvist, Nicolas Tsiftes, Thiemo Voigt, and Amy Loutfi. An Ontology-based Context-aware System for Smart Homes: E-care@home. *Sensors*, 17(7), 2017.

Marjan Alirezaie, Karl Hammar, and Eva Blomqvist. Smartenv as a network of ontology patterns. *Semantic Web Journal (SWJ)*, 2018.

J.F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: Implementing the semantic web recommendations. In *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers &Amp; Posters*, WWW Alt. '04, pages 74–83, New York, NY, USA, 2004. ACM.

Maurizio Di Rocco, Federico Pecora, and Alessandro Saffiotti. When robots are late: Configuration planning for multiple robots with dynamic goals. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 9515–5922. IEEE, 2013.

C. Freksa. Temporal Reasoning Based on Semi-Intervals. *Artificial Intelligence*, 54:199–227, 1992.

Cipriano Galindo and Alessandro Saffiotti. Inferring robot goals from violations of semantic knowledge. *Robot. Auton. Syst.*, 61(10):1131–1143, October 2013.

M Ghallab, A Howe, C Knoblock, D McDermott, A Ram, M Veloso, D Weld, and D Wilkins. PDDL - The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

Uwe Köckemann and Lars Karlsson. Configuration planning with temporal constraints. In *Proceedings of the 31st Conference on Artificial Intelligence (AAAI)*, 2017.

Uwe Köckemann, Federico Pecora, and Lars Karlsson. Grandpa Hates Robots — Interaction Constraints for Planning in Inhabited Environments. In *Proceedings of the 28th Conference on Artificial Intelligence (AAAI)*, 2014.

Uwe Köckemann. *Constraint-based Methods for Human-aware Planning*. PhD thesis, Örebro university, 2016.

Robert Lundh. *Robots that Help Each Other: Self-Configuration of Distributed Robot Systems*. PhD thesis, Örebro University, School of Science and Technology, 2009.

Fiona McNeill, Alan Bundy, and Chris Walton. Planning from rich ontologies through translation between representations. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, 2005.

I. Meiri. Combining Qualitative and Quantitative Constraints in Temporal Reasoning. In *Artificial Intelligence*, pages 260–267, 1996.

Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, January 2008. `http://www.w3.org/TR/rdf-sparql-query/`.

Lia Susana d. C. Silva-Lopez, Mathias Broxvall, Amy Loutfi, and Lars Karlsson. Towards configuration planning with partially ordered preferences: Representation and results. *KI*, 29(2):173–183, 2015.