OWLS-FDplan Overview

E. Jaramillo, A. Heggen

March 30, 2021

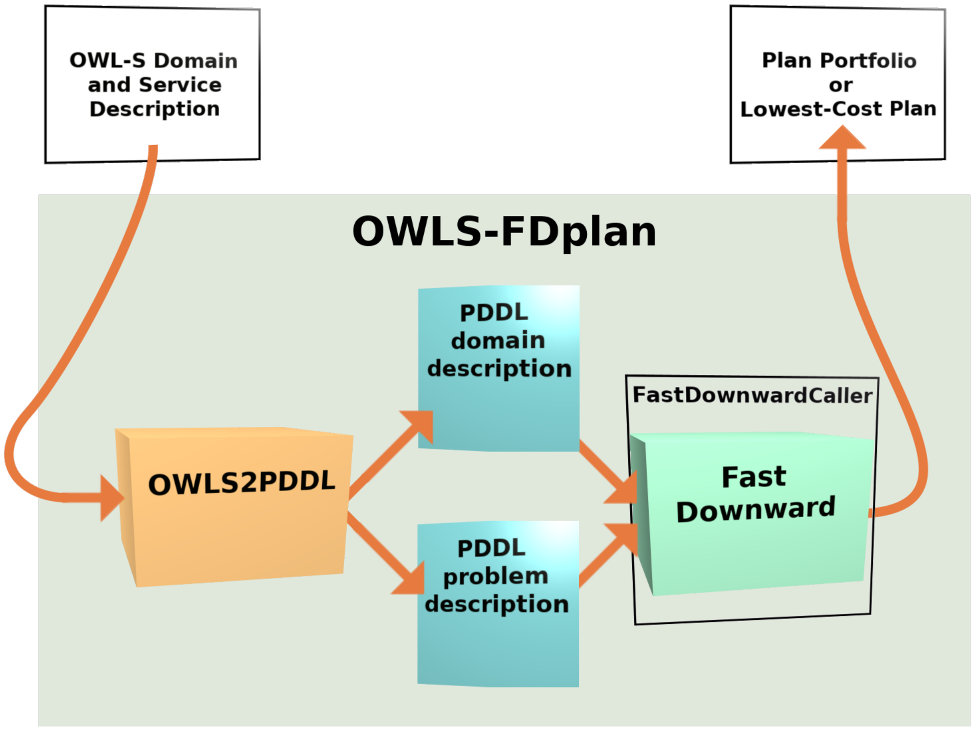

Figure 1: OWLS-FDplan Architecture

Abstract

OWLS-FDplan is a service composition planner: Given a service composition problem expressed in OWL-S 1.1 as input, OWLS-FDplan produces a set of plans solving that problem. To this end, OWLS-FDPlan first uses OWLS2PDDL to map the problem to PDDL 2.1, then invokes FastDownward on this PDDL representation to produce the set of plans. The following subsections will detail the technologies used within OWLS-FDplan.

OWLS2PDDL

OWLS2PDDL is a converter module originally developed for OWLS-Xplan [2]. Although the syntax of its output has been adapted for WELCOME, the purpose of OWLS2PDDL remains the same: to translate service composition problems from OWL-S 1.1 to PDDL 2.1. Indeed, OWLS-Xplan's original implementation of OWLS2PDDL outputs PDDXML, an XML-based dialect of PDDL. OWLS-FDplan's implementation, in contrast, outputs "traditional" PDDL 2.1.

As Figure 6 demonstrates, perhaps the most notable difference between PDDXML and PDDL is that the latter is much more compact and human-readable, lending reason as to why PDDL has been described as the "lingua franca of the planning algorithms used in service composition" [3]. Indeed, PDDL's widespread

use in the planning community led to its adaptation as Fast-Downward's input format, necessitating OWLS2PDDL's update to accommodate this preference.
The OWL-S input of OWLS2PDDL expresses a service composition problem using three subcomponents:

1. An initial ontology describing the initial state of the problem-world
2. A goal ontology describing the goal state of the problem-world
3. One or more services, each of which describe an operation that can be used to advance the initial state towards the goal state.

PDDL, in contrast, describes a service composition problem using only two subcomponents:

1. A problem definition describing both the initial and goal states of the problem-world
2. A domain definition defining the possible operations

One can see, therefore, that OWL-S services resemble PDDL operators. However, the overlap is not perfect: OWL-S services are concerned with *both* world state (via *preconditions* and *effects*) as well as service information (via *inputs* and *outputs*). However, PDDL operators consider only world state. Hence, in order to map service information onto world state, OWLS2PDDL utilizes a special predicate, "`agentHasKnowledgeAbout`". Figure 2 details this relationship.

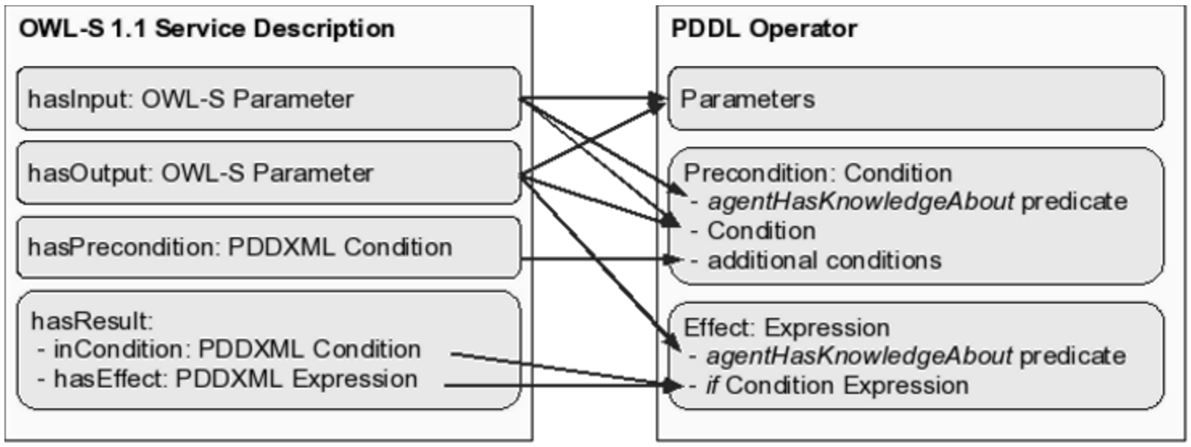
OWL-S 1.1 Service Description
hasInput: OWL-S Parameter
hasOutput: OWL-S Parameter
hasPrecondition: PDDXML Condition
hasResult:
- inCondition: PDDXML Condition
- hasEffect: PDDXML Expression
PDDL Operator
Parameters
Precondition: Condition
- agentHasKnowledgeAbout predicate
- Condition
- additional conditions
Effect: Expression
- agentHasKnowledgeAbout predicate
- if Condition Expression

Figure 2: How OWLS2PDDL maps OWL-S services onto PDDL operators

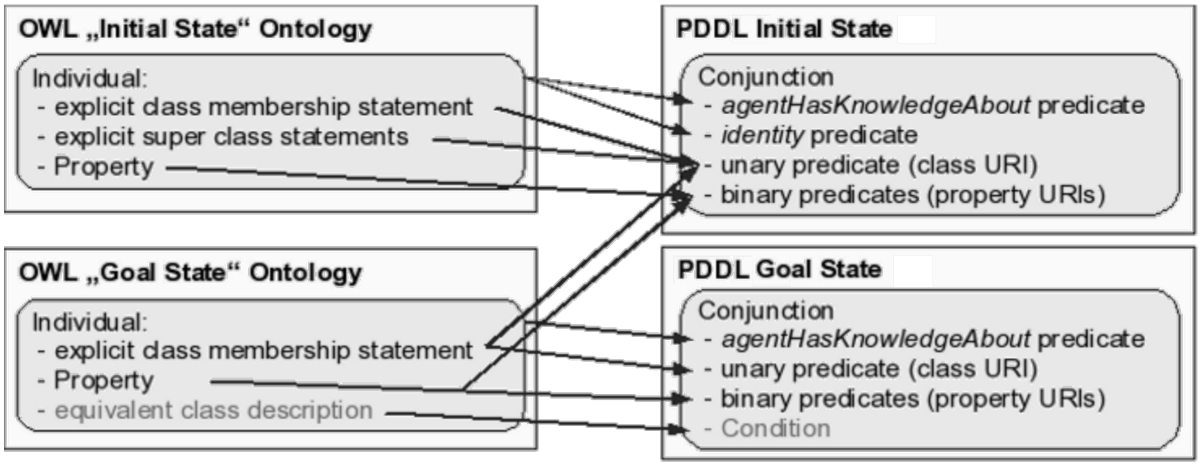
OWL „Initial State" Ontology
Individual:
- explicit class membership statement
- explicit super class statements
- Property
OWL „Goal State" Ontology
Individual:
- explicit class membership statement
- Property
- equivalent class description
PDDL Initial State
Conjunction
- agentHasKnowledgeAbout predicate
- identity predicate
- unary predicate (class URI)
- binary predicates (property URIs)
PDDL Goal State
Conjunction
- agentHasKnowledgeAbout predicate
- unary predicate (class URI)
- binary predicates (property URIs)
- Condition

Figure 3: How OWLS2PDDL converts OWL-S initial and goal ontologies to PDDL

Fast Downward

Fast Downward is a classical planning system that relies on heuristic search [1]. Fast Downward is notable for its speed and efficiency, and has only been refined over the years: it first won the "classical" track of the 4th International Planning Competition in 2004, and has regularly placed as a finalist since then. Generally speaking, Fast Downward takes three steps to solve a service composition problem (Figure 4):
[1]

1. Translation. Fast Downward re-expresses the PDDL problem as a multi-valued planning task to allow hierarchical planning, as hierarchical planning tends to be more efficient than planning in the original PDDL space. If a re-expressed variable is not dependent on other variables to change its state, then it ranks high in the hierarchy. However, most problems are not perfectly hierarchical: relaxation in the form of ignoring certain casual dependencies is used in these cases.

2. Knowledge Compilation. Fast Downward takes the multi-valued planning task and, from it, prepares four data structures needed for the search:

   - Domain transition graph, which encodes the circumstances under which a variable can change
   - Casual graph, which encodes hierarchical dependencies between different state variables
   - Two data structures used by forward-searching algorithms to expedite their search: successor generator and axiom evaluator. The former computes the operators available in a given world state, while the latter determines the value of axioms at a given state

3. Search. Fast Downward calls a basic search algorithm to complete the actual search. In Helmert's original paper on Fast Downward, the algorithms to choose from were greedy best-first search, multi-heuristic best-first search, and focused iterative-broadening search.

FastDownwardCaller

FastDownwardCaller is a Java wrapper for Fast-Downward: It calls Fast Downward, then returns either a `String`, `Plan`, or `PlanProfile` conveying Fast Downward's output (Figure 5). The `String` and `Plan` detail the shortest plan Fast-Downward can find from a given domain and problem pddl, while the `PlanProfile` includes all plans found by Fast Downward in the given time. Three components are necessary are necessary to use `FastDownwardCaller`:

1. `FastDownwardCaller.jar`
2. A compiled version of Fast-Downward
3. A problem and domain PDDL

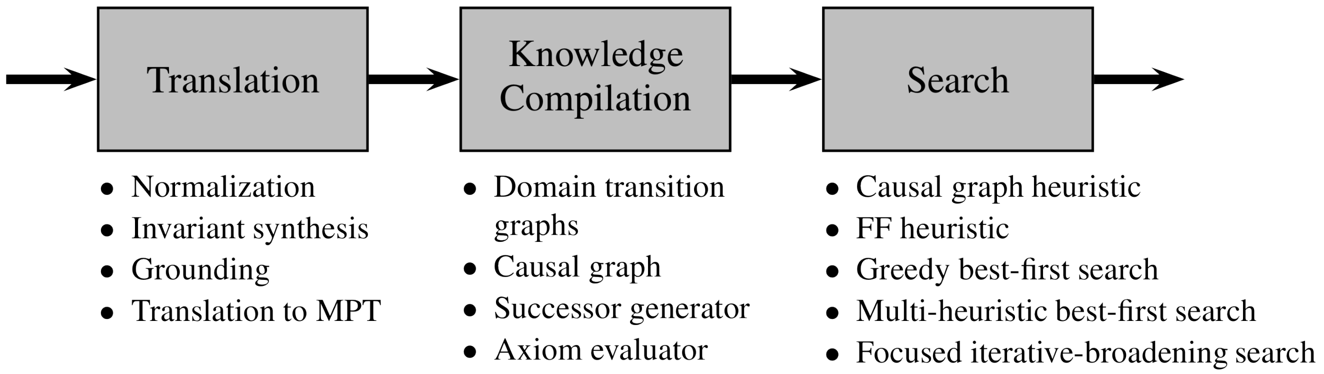

Figure 4: The three steps utilized by Fast-Downward. MPT stands for "multi-valued planning task", FF stands for "Fast-Forward".

[1] This section has been extracetd from Helmert (2006) [1]

```
The lowest-cost plan has 3 steps. Its full description is:

1.http://127.0.0.1:8000/health-scallops/services/createflightaccount.owl\#createflightaccountservice
http://127.0.0.1:8000/health-scallops/simple\_initialontology.owl\#emacreditcard
http://127.0.0.1:8000/health-scallops/simple\_initialontology.owl\#airportnearfarohospital
http://127.0.0.1:8000/health-scallops/simple\_initialontology.owl\#emaattendant

2.http://127.0.0.1:8000/health-scallops/services/proposeflight.owl\#proposeflightservice
http://127.0.0.1:8000/health-scallops/simple\_initialontology.owl\#flightparameters
http://127.0.0.1:8000/health-scallops/simple\_goalontology.owl\#flight

3.http://127.0.0.1:8000/health-scallops/services/bookflight.owl\#bookflightservice
http://127.0.0.1:8000/health-scallops/simple\_initialontology.owl\#airportnearfarohospital
http://127.0.0.1:8000/health-scallops/simple\_initialontology.owl\#emaattendant
http://127.0.0.1:8000/health-scallops/simple\_goalontology.owl\#flight

and, without http format:

1. createflightaccountservice emacreditcard airportnearfarohospital emaattendant

2. proposeflightservice flightparameters flight

3. bookflightservice airportnearfarohospital emaattendant flight
```

Figure 5: An example output by FastDownwardCaller.

```
<action name="CreateFlightAccountService">
  <parameters>
    <param type="object">?CreditcardInformation</param>
    <param type="object">?DesiredAccountData</param>
    <param type="object">?UserData</param>
  </parameters>
  <precondition>
    <and>
      <pred name="agentHasKnowledgeAbout">
        <param>?CreditcardInformation</param>
      </pred>
      <or>
        <pred name="Creditcard">
          <param>?CreditcardInformation</param>
        </pred>
      </or>
      <pred name="agentHasKnowledgeAbout">
        <param>?DesiredAccountData</param>
      </pred>
      <or>
        <pred name="Account">
          <param>?DesiredAccountData</param>
        </pred>
      </or>
      <pred name="agentHasKnowledgeAbout">
        <param>?UserData</param>
      </pred>
      <or>
        <pred name="Person">
          <param>?UserData</param>
        </pred>
      </or>
      <exists>
        <variables>
          <variable type="object">?UserAddress</variable>
        </variables>
        <pred name="hasAddress">
          <param>?UserData</param>
          <param>?UserAddress</param>
        </pred>
      </exists>
      <pred name="isOwnedBy">
        <param>?CreditcardInformation</param>
        <param>?UserData</param>
      </pred>
    </and>
  </precondition>
  <effect>
    <and>
      <pred name="isValid">
        <param>?DesiredAccountData</param>
      </pred>
    </and>
  </effect>
</action>
```

`CreateFlightAccountService` in PDDXML

```
(:action CreateFlightAccountService

    :parameters (?CreditcardInformation - object
    ?DesiredAccountData - object
    ?UserData - object)

    :precondition(and
    (or (agentHasKnowledgeAbout ?CreditcardInformation)
        (Creditcard ?CreditcardInformation))
    (or (agentHasKnowledgeAbout ?DesiredAccountData)
        (Account ?DesiredAccountData))
    (or (agentHasKnowledgeAbout ?UserData)
        (Person ?UserData))
    (isOwnedBy ?CreditcardInformation ?UserData))

    :effect(and
    (isValid ?DesiredAccountData))

)
```

`CreateFlightAccountService` in PDDL

Figure 6: Excerpts of equivalent outputs: From OWLS-Xplan's implementation of OWLS2PDDL (left), and OWLS-FDplan's implementation of OWLS2PDDL (right).

0.1 OWLS2PDDL

0.2 Fast Downward

0.3 OWLS-FDplan

1 Bibliography

References

[1] Helmert, M. (2006). *The Fast Downward Planning System Journal of Artificial Intelligence Research 26*, 191-246.
`https://dl.acm.org/doi/10.5555/1622559.1622565`

[2] Klusch, M., Gerber, G., & Schmidt M. (2005). Semantic Web Service Composition Planning with OWLS-Xplan. *Proceedings of the AAAI Fall Symposium on Semantic Web and Agents*
`https://www.aaai.org/Papers/Symposia/Fall/2005/FS-05-01/FS05-01-008.pdf`

[3] Schumacher, M., Helin, H., Schuldt, H., Calisti, M., Walliser, M., Brantschen, S., & Herbstritt M. (2008). CASCOM: Intelligent Service Coordination in the Semantic Web. *Whitestein Series in Software Agent Technologies and Autonomic Computing*
`https://link.springer.com/book/10.1007%2F978-3-7643-8575-0`