Planning

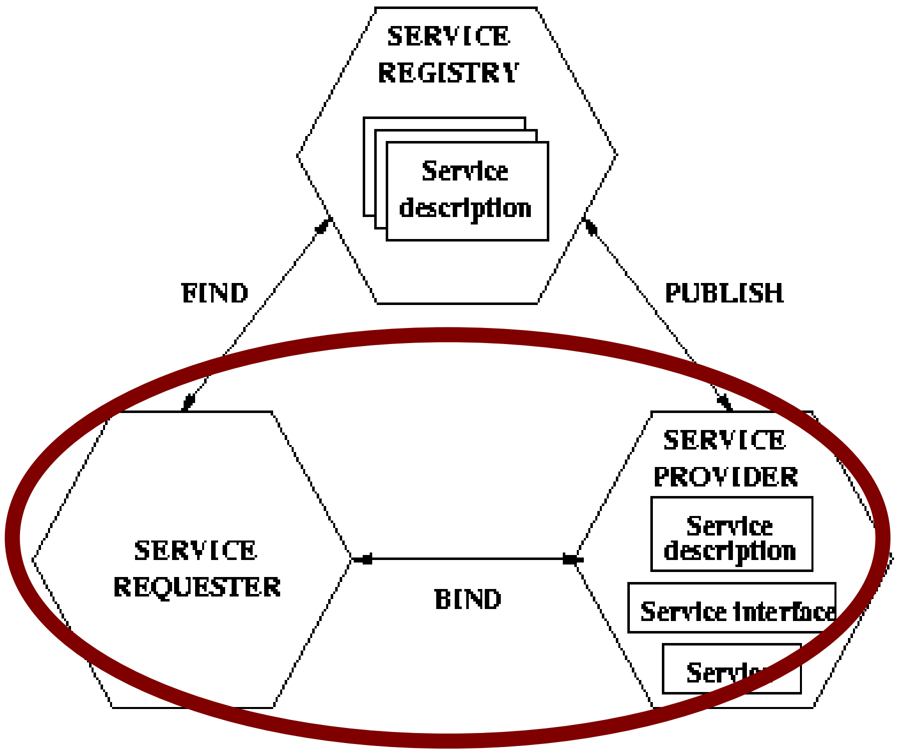
Tuukka Ruotsalo

Contents

- Semantic Web Service Use-Cases
- Information Systems
  - Data, Processes and External events
- Situation Calculus
- Planning, STRIPS

Semantic web services use cases

- Automatic Web service discovery
  - How can we find the service we need from the web (like traditional services from UDDI)
- Automatic Web service invocation
  - How can we execute a service from the web
- **Automatic Web service composition and interoperation**
  - How can we know which service to execute at a certain situation to achieve a goal


SERVICE
REGISTRY
Service
description
FIND
PUBLISH
SERVICE
REQUESTER
BIND
SERVICE
PROVIDER
Service
description
Service interface

Automatic Web Service Composition

- Automatic Web Service composition and interoperation
  - What services are available (and executable according to situation rules) at certain situation
  - Work-flows are planned dynamically based on the state of the system
  - Data-centric systems
    - Processes are not pre-defined, but planned according to state of the data in the system

An example

- Depending on the conditions we might want the “process of ordering skate boots” be different.
- Requires dynamic planning and execution of services

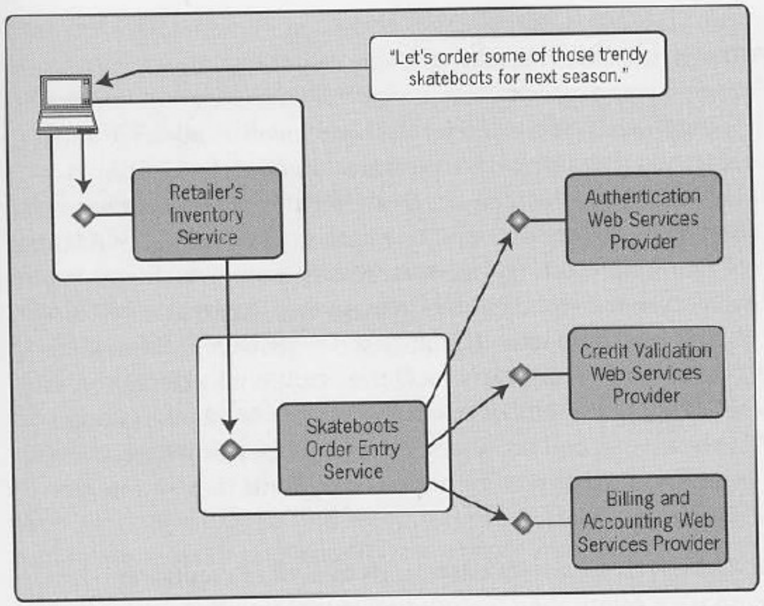


Figure 1-3 The Skateboots order entry service comprises several other Web services.

Information System meta-model

- Information Systems are
  - Models of the universe of discourse
  - In Computer World models of
    - Data
      - What entities the system contains and in which states they are at a certain time
    - Processes
      - What are the actions that enable transformations of a state of entities to another state of entities
    - + Interaction to cause external events
      - User interfaces (how to get the system to do something)

What do we need to compose a configurable “semantic” IS?

- Descriptions of data
  - Ontology (of data) (i.e. Entities of the given discourse) = data model
- Descriptions of processes
  - Ontology (of possible actions) = Situation Calculus processes available

Situation

- Definition
  - A situation s is the complete state of the universe at an instant of time.
  - Sit:= {s | s is situation}
- Since the universe is too large, we‘ll never describe the universe completely, but give only facts
  - These facts will be used to deduce further facts about that situation, future situations, situations that agents can bring about
  - Requires also to consider hypothetical situations
  - These situations can‘t be completely described, but with sufficient details for some purposes

Fluents

- Idea: Partial description of a situations
- Definition
  - A fluent f is a function with domain Sit
  - Range of f {true,false}: **propositional fluent**
  - Sit: **situational fluent**
- Fluents are often the values of functions
  - T(President(Finland) = JKPaasikivi,1950)
  - Note! It is not logically equivalent that President(Finland) = JKPaasikivi,1950, but the function returns the correct value for a time slice 1950
  - It is logically identical that the situation in 1950 was that JKPaasikivi was the President of Finland

Causality

- Make assumptions about causality by means of a fluent F(π) (where π is a fluent itself!)
- F(π,s) asserts that the situations will be followed(after some time) by a situation that satisfies the fluent π
- Example:
  - All x, All a [raining(x) and at(a,x) and outside(x) -> F(wet(a))]
- Other useful operators
  - F(π,s): For some situation s‘ in the future of s holds π(s‘)
  - G(π,s): For all situation s‘ in the future of s holds π(s‘)
  - P(π,s): For some situation s‘ in the past of s holds π(s‘)
  - H(π,s): For all situation s‘ in the past of s holds π(s‘)

Actions

- Fundamental role in our study of actions:
  - Situational fluent: result(a,σ,s) where…
  - a:Agent, σ: Action/Strategy, s:Situation
- The value of result(a,σ,s) is the situation that results when a carries out σ, starting in situation s
- When σ is non-terminating in that context, this value is undefined

Planning as problem solving

- Problem solving based on the search strategies consists of:
  - Actions
    - Available actions that may change the state of the entities
  - State description
    - Initial state, descriptions of the states before and after the actions
  - Goal description
    - Description of the state when the problem is solved
  - Plan
    - Ordered list of actions that solve the problem

Problems in situation calculus

- Search to solve the problem takes exponential time with respect to the length of the path
- First Order Logic is only semi-decidable
- We can proof that a solution exists, but it is not known if it is an optimal solution
  - Solution [do(x)] is as good as [do(nothing)| do(x)]
- More effective languages have been developed
  - Less possible solutions and more effective algorithms

STRIPS

- STRIPS is a classic planning language
- Describes states and operators with limited language
- Describes situations with literals that are not functions
  - Predicates that contain constants as values, negation is allowed
  - For example: At(Home) /\ ^(Have(Cake))
- Goals are represented as conjunction of literals
  - For Example: At(Home) /\ Have(Milk)
- Variables are allowed: At(x) /\ Sell(x, Milk)

STRIPS representation of actions

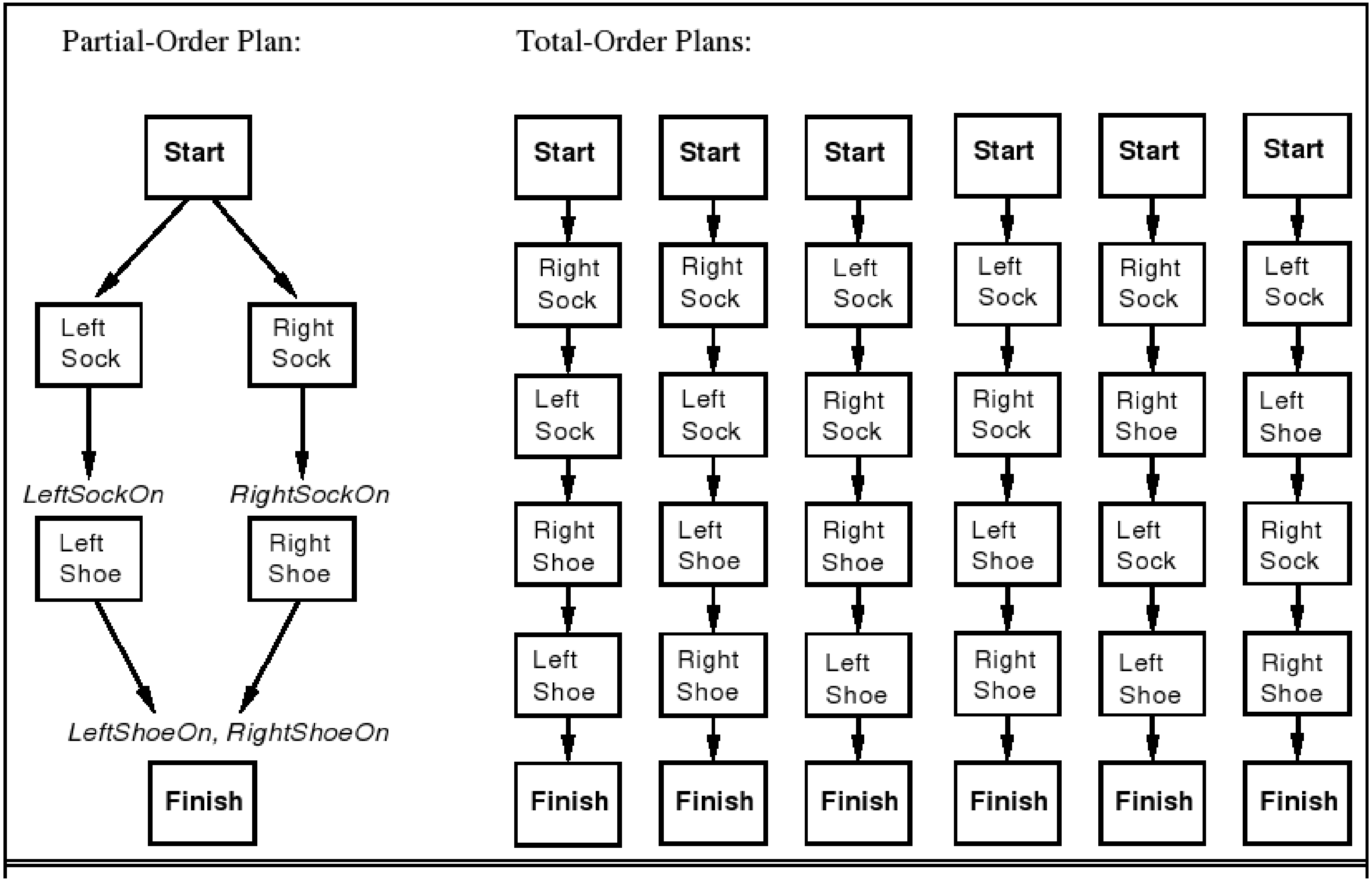
- Actions are represented with IOPEs
  - Inputs
  - Outputs
  - Preconditions
    - In which state the action may be performed
    - Conjunction of facts (only positive literals)
  - Effects
    - What are the changes in the universe according to the outputs of the action
- For example:
  - Action Bake(Cake)
    - Precondition Have(Milk)
    - Effect Have(Cake)

An Example

- System has a set of data and rules what to do if a certain situation is satisfied
- We do not give exact definition what to do and in which order, just a set of data and set of rules to apply to the situation
- Example
- Init(Have(Cake))
- Goal (Have(Cake) & Eaten(Cake))
- Action (Eat(Cake))
  - Precondition(Have(Cake))
  - Effect not(Have(Cake)) & Eaten(Cake)
- Action Bake(Cake)
  - Effect Have(Cake)

Partial order planning

- In partial order planning only the decisions that have to be made at certain situation are made
  - We get partial orders (linearization -> total order)
  - For example having cake has to be true before eating it, how to have the cake is irrelevant for ordering the eat and have situations
- Plan is a data-structure that has four parts
  - Set of steps (actions that must be executed)
  - Set of ordering constraints (Have(Cake) < Eat(Cake)
  - Set of variable bindings (Value = Variable)
  - Set of causal links (Bake(Cake)-Have(Cake)->Eat(Cake))
    - Express a precondition to be set for another situation


Partial-Order Plan:
Total-Order Plans:
Start
Left Sock
Right Sock
LeftSockOn
RightSockOn
Left Shoe
Right Shoe
LeftShoeOn, RightShoeOn
Finish
Start
Right Sock
Left Sock
Right Shoe
Left Shoe
Finish
Start
Right Sock
Left Sock
Left Shoe
Right Shoe
Finish
Start
Left Sock
Right Sock
Right Shoe
Left Shoe
Finish
Start
Left Sock
Right Sock
Left Shoe
Right Shoe
Finish
Start
Right Sock
Right Shoe
Left Sock
Left Shoe
Finish
Start
Left Sock
Left Shoe
Right Sock
Right Shoe
Finish

Partial order planning

- Solution to partial order planning problem is a plan that certainly leads from initial state to goal state.
  - Partial order plan is allowed as long as it is consistent and complete
- Plan is complete if
  - Preconditions for each action are true because of an effect of another action
  - In between there is no action that falsifies the conditions
- Plan is consistent if
  - Ordering and binding constraints are not conflicting
    - (s1<s2 /\ s2 < s1) or (v=A /\ v=B /\ A not (B))

Example

- Init(Have(Cake))
- Goal (Have(Cake) & Eaten(Cake))
- Action Eat(Cake)
  - Precondition(Have(Cake))
  - Effect not(Have(Cake)) & Eaten(Cake)
- Action Bake(Cake)
  - Effect Have(Cake)
- Step1: Eat(Cake)-not(Have(Cake)) /\ Eaten(Cake)
- Step2: Bake(Cake)-Have(Cake)
- Step3: Goal Satisfied
- POP: Eat<Goal
- No variable bindings in this case (only constants used)
- Only one linearisation in this case
  - There is just one plan that satisfies the goal

Partial order planning

- Frame problem does not have to be considered: all other than given effects stay as they are (STRIPS)
- Regressive planning
  - From goal state to given initial state
  - May be more economical
- Progressive
  - From initial state to goal state

Finding a plan

1. In the beginning set start and finish states and start < finish and open preconditions for finish
2. Select open precondition p for action B and find such A that effects that p
3. Add causal link A-p->B and A<B
   1. if A not in plan add A and Start < A < Finish
4. Solve conflicts
   1. For example if C conflicts with A-p->B set B<C or C<A
5. If preconditions can not be satisfied or conflict solved rollback
6. If open preconditions start from 2.

STRIPS **Language**	ADL **Language**
Only positive literals in states: $Poor \land Unknown$	Positive and negative literals in states: $\neg Rich \land \neg Famous$
Closed World Assumption: Unmentioned literals are false.	Open World Assumption: Unmentioned literals are unknown.
Effect $P \land \neg Q$ means add P and delete Q.	Effect $P \land \neg Q$ means add P and $\neg Q$ and delete $\neg P$ and Q.
Only ground literals in goals: $Rich \land Famous$	Quantified variables in goals: $\exists x\, At(P_1, x) \land At(P_2, x)$ is the goal of having P_1 and P_2 in the same place.
Goals are conjunctions: $Rich \land Famous$	Goals allow conjunction and disjunction: $\neg Poor \land (Famous \lor Smart)$
Effects are conjunctions.	Conditional effects allowed: **when** P: E means E is an effect only if P is satisfied.
No support for equality.	Equality predicate $(x = y)$ is built in.
No support for types.	Variables can have types, as in $(p : Plane)$.

Figure 11.1 Comparison of STRIPS and ADL languages for representing planning problems. In both cases, goals behave as the preconditions of an action with no parameters.

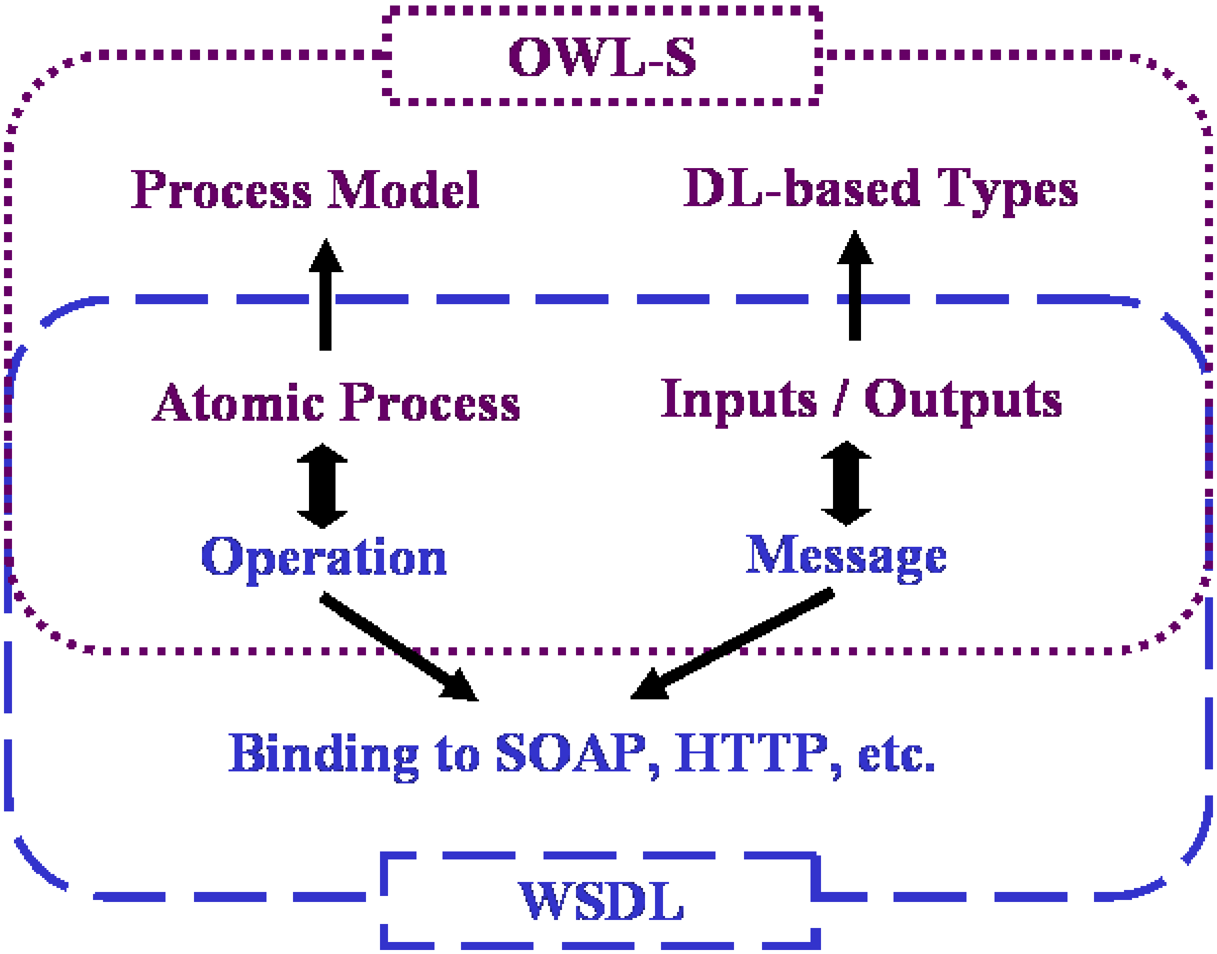
SeCo
semantic computing

An example of a semantic web service language: OWL-S

OWL-S

- Actions are grounded to web services
  - Actions (Functions with inputs and outputs) are RPCs
- Processes are described as situations
  - Depending of the state of the system, according to the preconditions of the actions some of the services are possible candidates to be executed
  - Depending of the response we'll get (output) effects are applied to the ontology = States of the objects are changed
- => We can use POP or more sophisticated planning techniques to derive plans

OWL-S and WSDL



OWL-S Processes