Service composition and Schema matching

Tuukka Ruotsalo

Contents

- Information Systems as
  - Data, Processes and External events
- Schema matching
  - Matching
  - Alignment
- Service Composition
  - Orchestration
  - Choreography
  - Planning

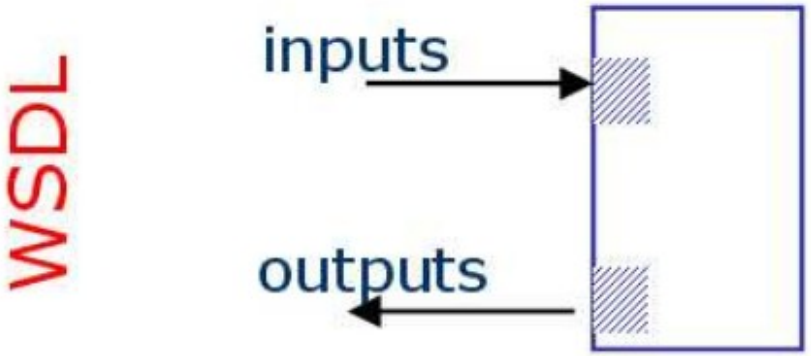
Information System meta-model

- 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)

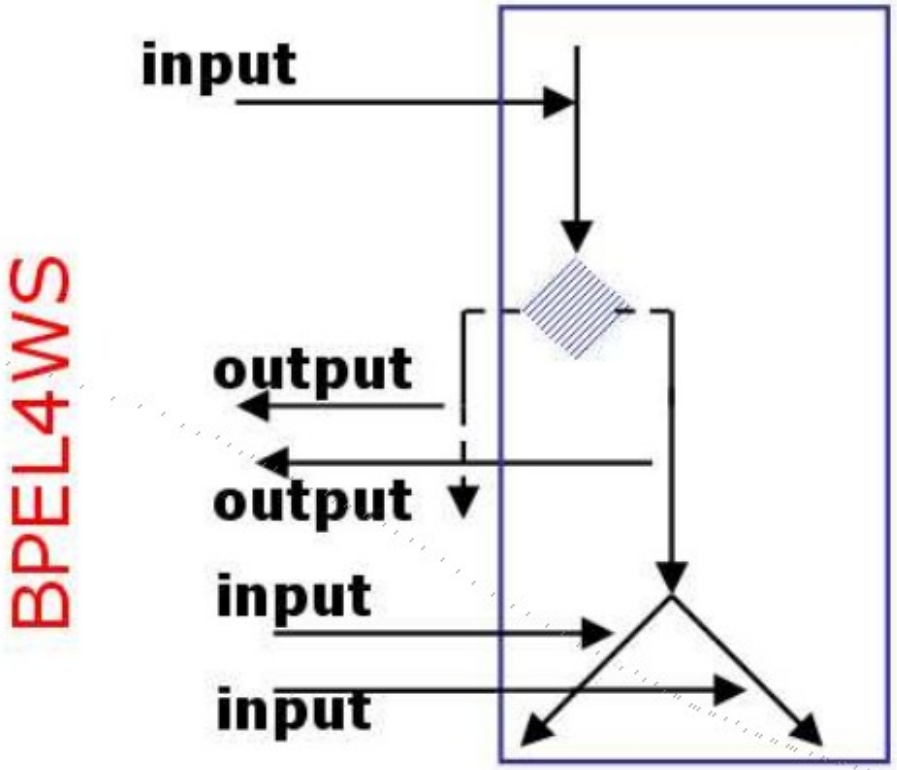
Semantic Problems

- Matching the datamodel (Schema mapping)
- Matching the process model (Service composition)

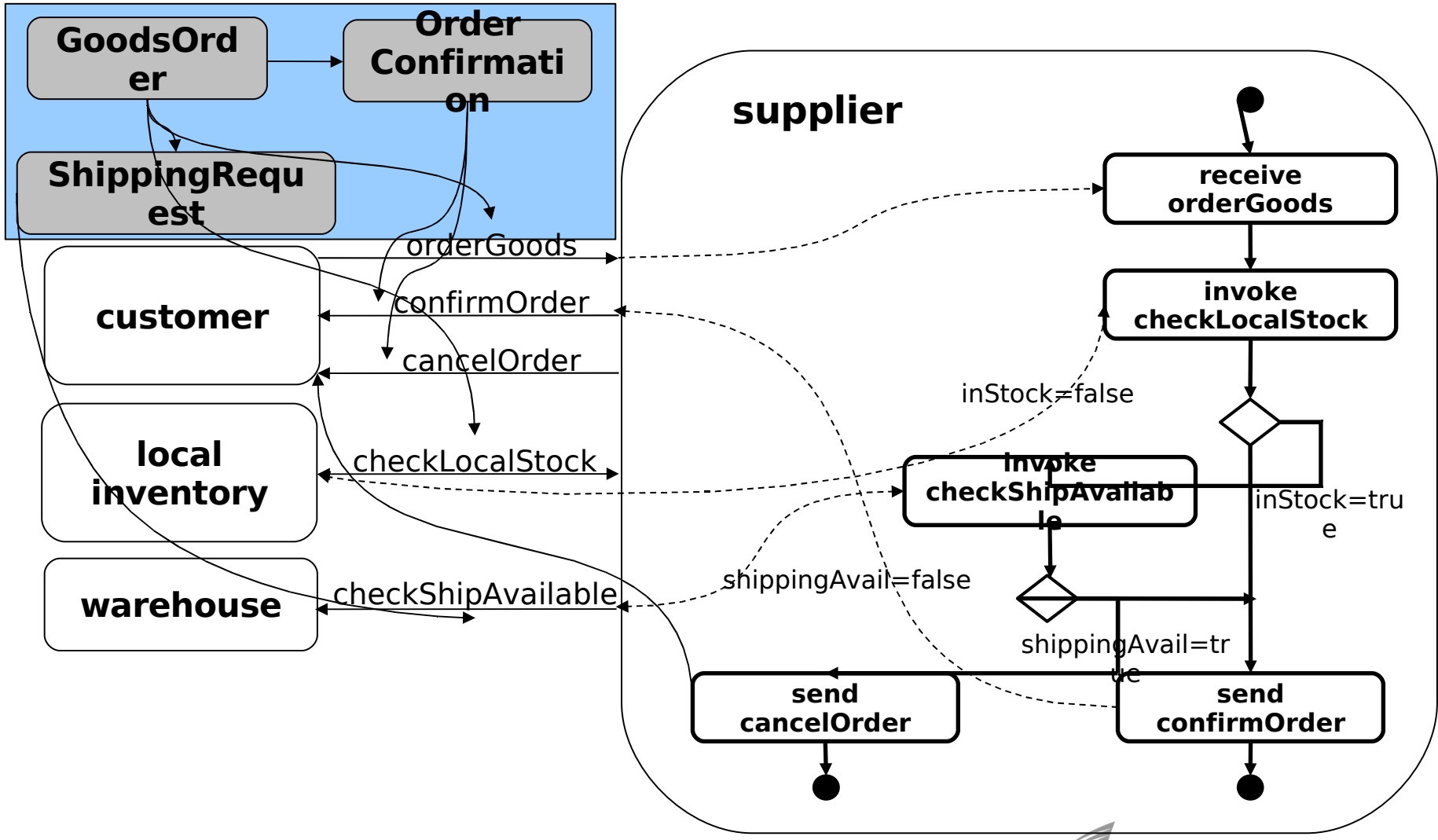
Functional level:



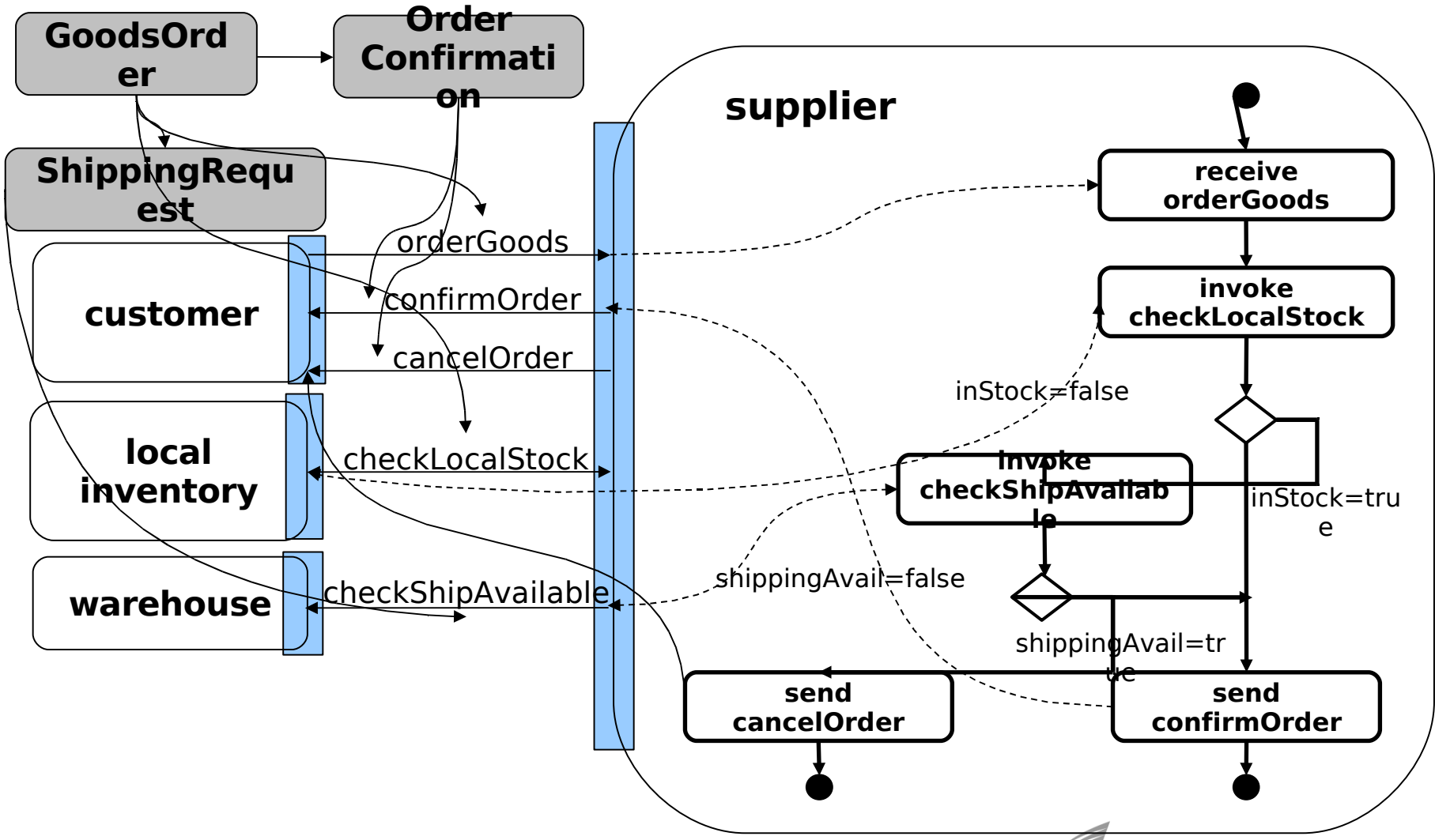
Process level:



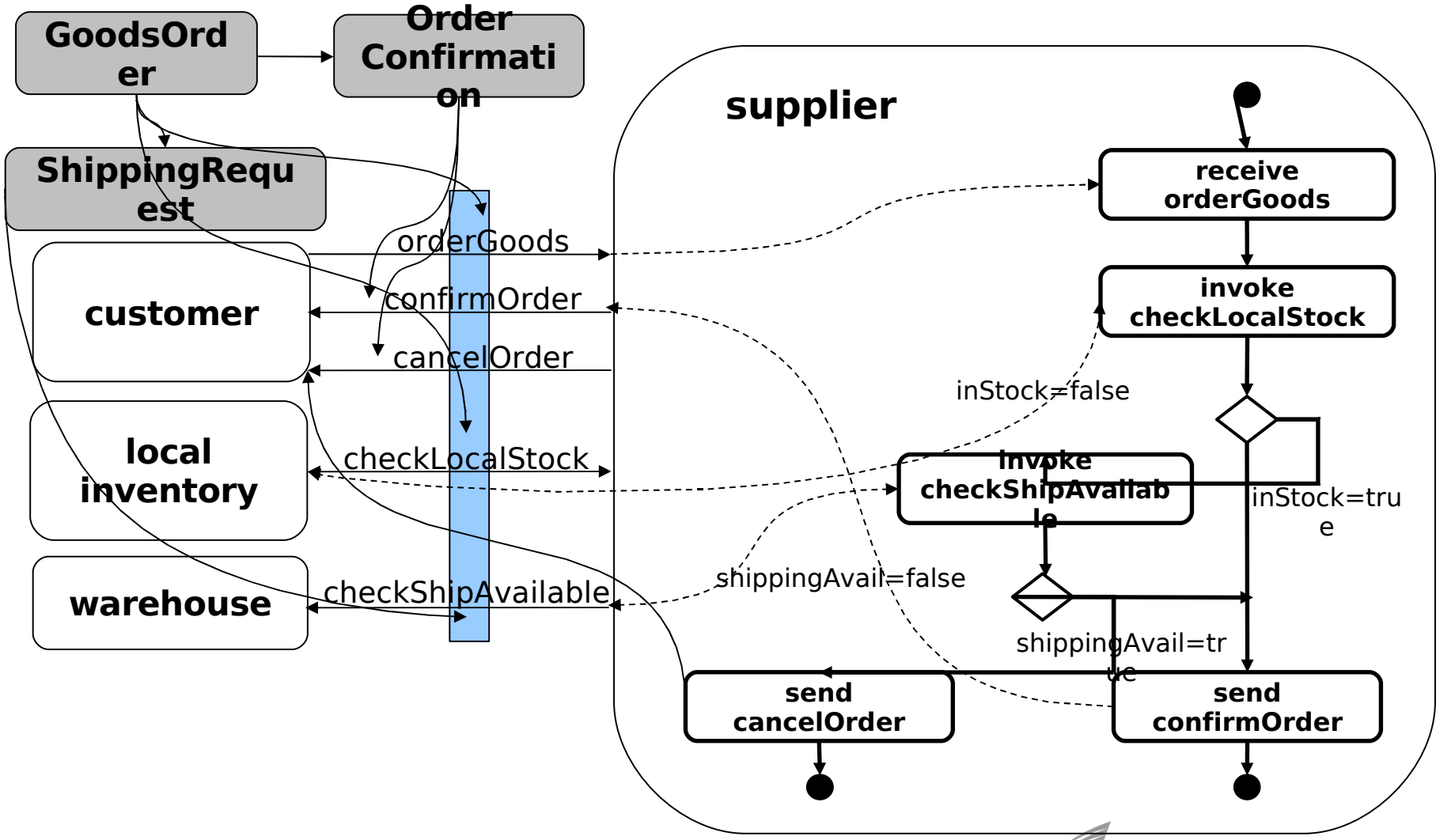
What is needed for minimizing the cost of connecting service modules and mazimizing reuse: Common Data Encoding, Data Semantics



Needed: Common Interfaces, Service Semantics



Needed: Common communication protocols and patterns



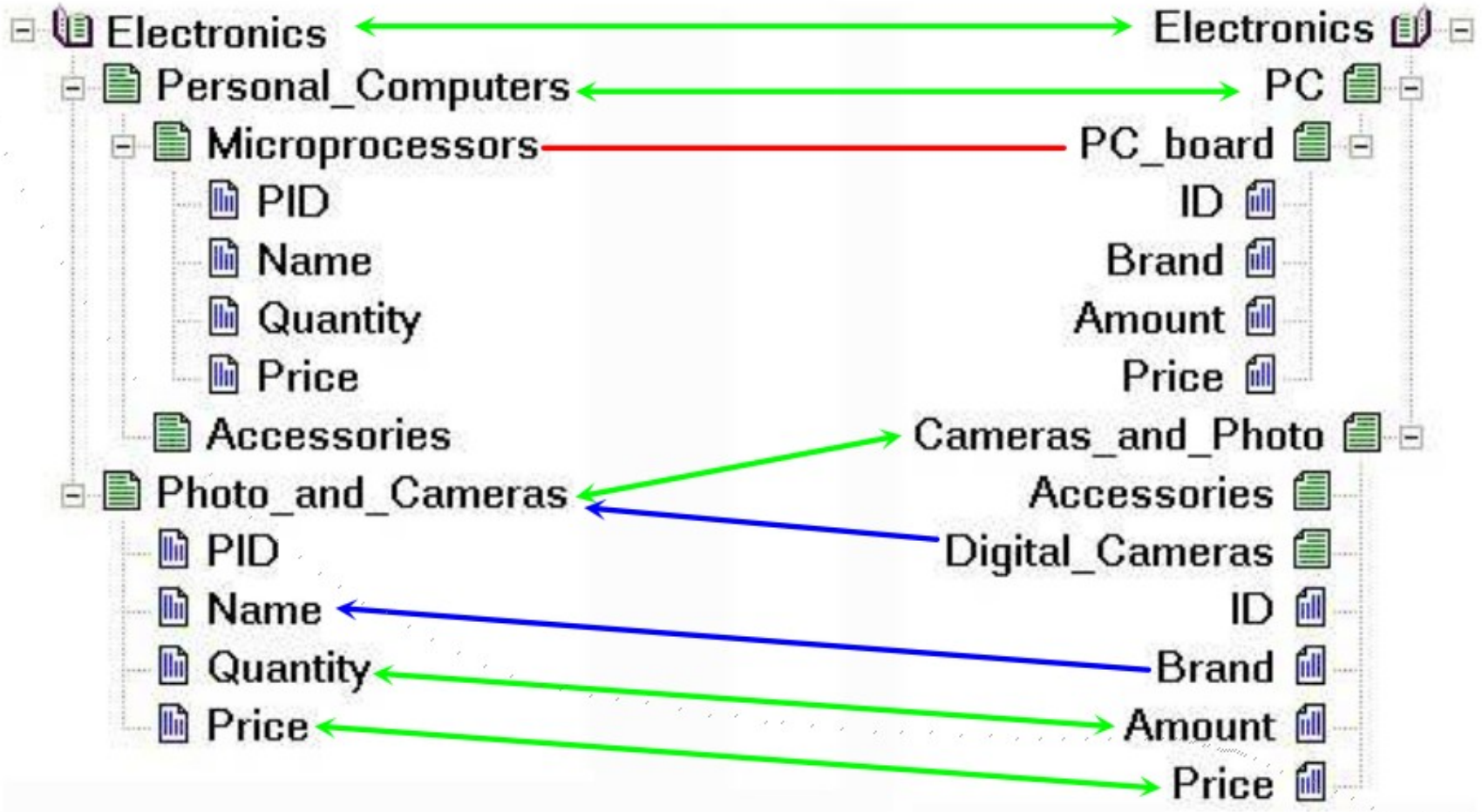
Schema matching

Schema matching

- The ontology matching problem:
  - Resources are being expressed in different ways must be reconciled before being used
- Mismatch
  - Different languages
  - Different terminologies
  - Different way of modelling

(Shvaiko & Euzenat, 2005)

Example: XML Schema



(Shvaiko & Euzenat, 2005)

Reducing heterogeneity

- Reducing heterogeneity in 2 steps
  - Match (determine the alignment)
  - Process the alignment (merge, transform, etc.)
- Does not have to be totally automatic!
  - e.g. transforming using logical rules

Mapping

- Mapping ***M*** is a 5-uple <id,e,e',R,n>
  - *id* is a identifier of a mapping element
  - *e* and *e'* are entities (e.g. XML elements)
  - *R* is a relation (e.g. equivalence, more general, disjoint) or a rule etc.
  - *n* is a confidence measure in some mathematical structure (e.g. [0,1])

(Shvaiko & Euzenat, 2005)

Alignment

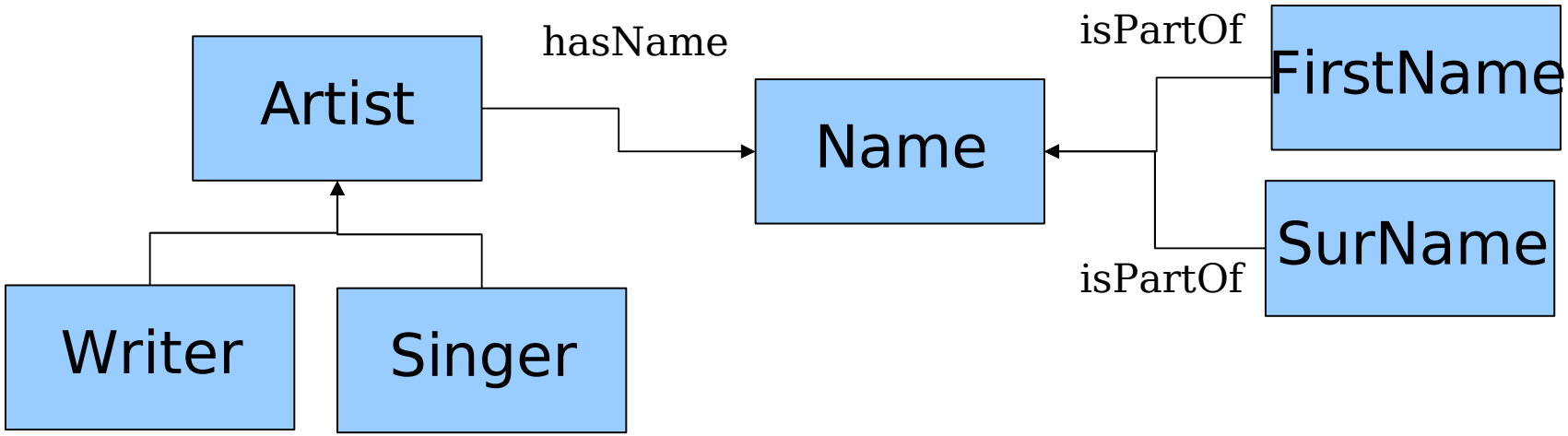
- Alignment **A** is a set of mapping elements
  - Depending on the schemas / ontologies being matched
- Alignment can be used to merge, transform etc. the data

(Shvaiko & Euzenat, 2005)

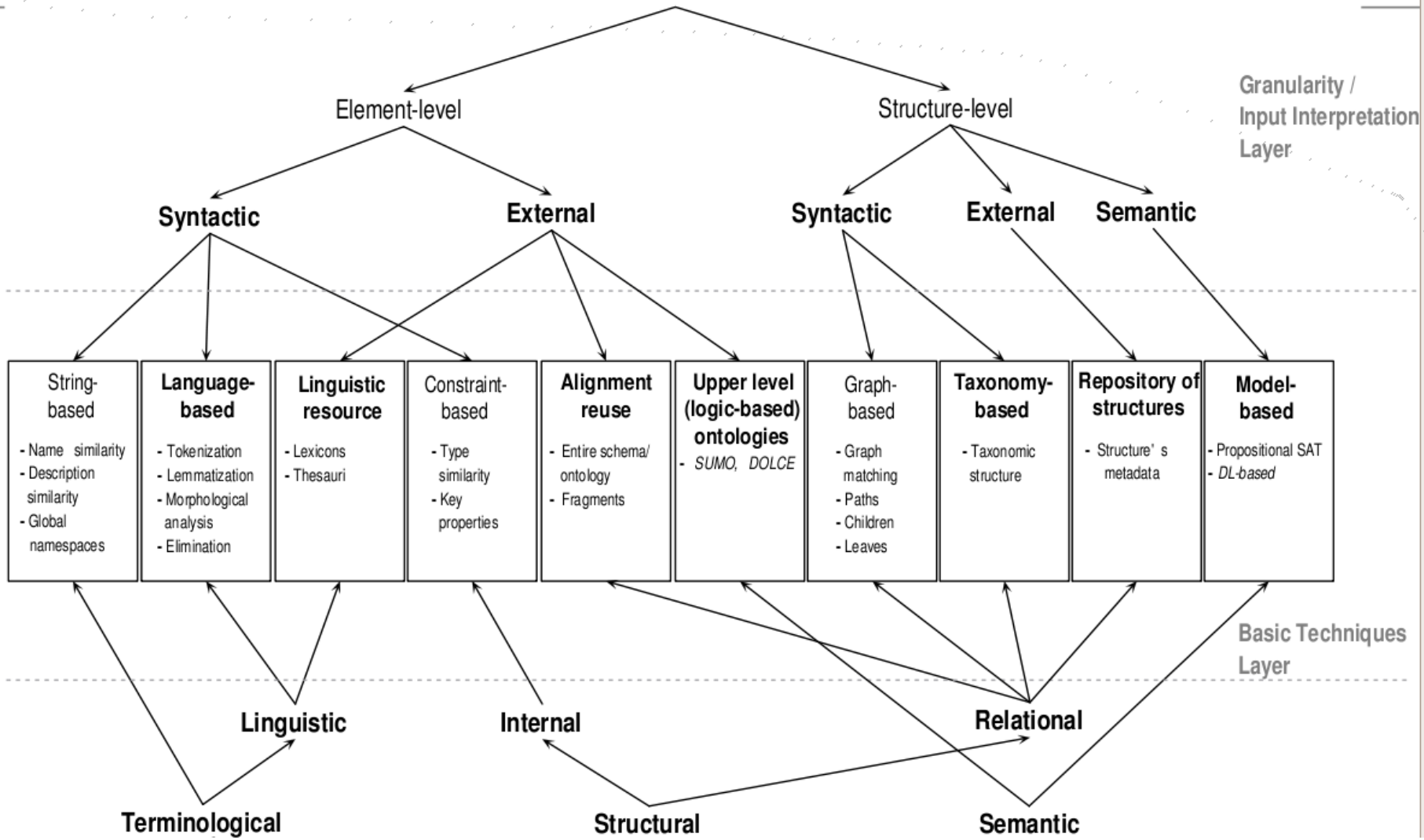
The Need for Semantics: Data Semantics

- Suppose we have the data object “Book” that has the fields “Writer name”, “Book name”
- What happens if someone else expects an object with fields “Artist first name”, “Artist surname”and “Work name”
- If the fields were semantically annotated, a rulebase (mapping elements) could be used to automatically align data objects to each other

With ontologies (defining classes and relationships) and transformation rules, it is possible to encode data semantics in a common manner, and translate between encodings



Mapping element:
Entities: FirstName, surName
Rule: Name:=concat(FirstName, SurName)
Confidence = 1 (boolean model)


Granularity / Input Interpretation Layer
Element-level
Structure-level
Syntactic
External
Syntactic
External
Semantic
String-based
- Name similarity
- Description similarity
- Global namespaces
Language-based
- Tokenization
- Lemmatization
- Morphological analysis
- Elimination
Linguistic resource
- Lexicons
- Thesauri
Constraint-based
- Type similarity
- Key properties
Alignment reuse
- Entire schema/ ontology
- Fragments
Upper level (logic-based) ontologies
- SUMO, DOLCE
Graph-based
- Graph matching
- Paths
- Children
- Leaves
Taxonomy-based
- Taxonomic structure
Repository of structures
- Structure' s metadata
Model-based
- Propositional SAT
- DL-based
Basic Techniques Layer
Linguistic
Internal
Relational
Terminological
Structural
Semantic

Service composition

The Need for Semantics: Service Semantics

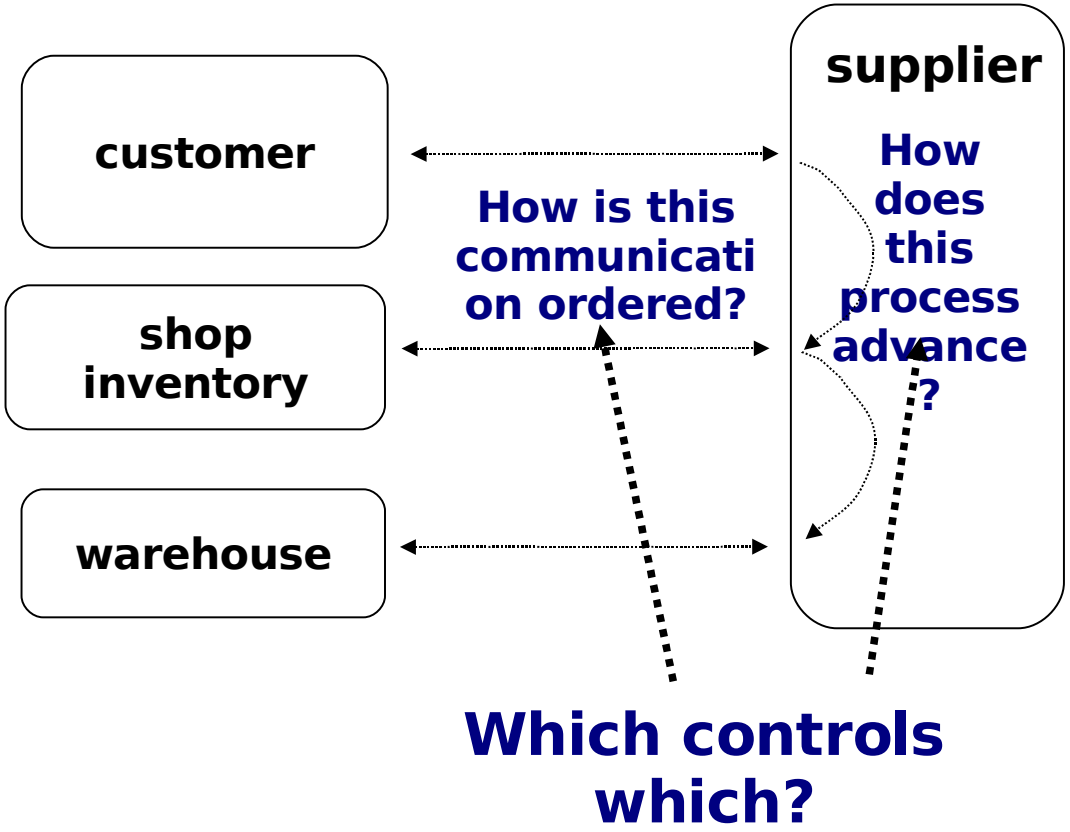
- Currently, we have a service that takes in a number, two strings of text, and returns a number. The number is termed “amount”, the strings “currency1” and “currency2”
- Linking data semantics to interface variables solves the problem of what the data actually means
- But we'd also like to know what the service *does*
- So, semantically annotate that this is a currency converter service (with e.g. STRIPS)
- Formally: With the prequisites of recognized currencies and an amount, the output will be the amount in currency one transformed into currency two

Automatic Web Service Composition

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

Introduction and Example: Buying skateboots

- In a Web Services environment, there is a need for combining the functionality provided by Web Services into a composite service. This is called composition.
- Depending on the messages arriving and sent within the partner network, we should be able to decide what to do next.
  - Process description dominant: **Orchestration**
    - One peer orchestrates
  - Communication pattern dominant: **Choreography**
    - “peer-to-peer scenario”



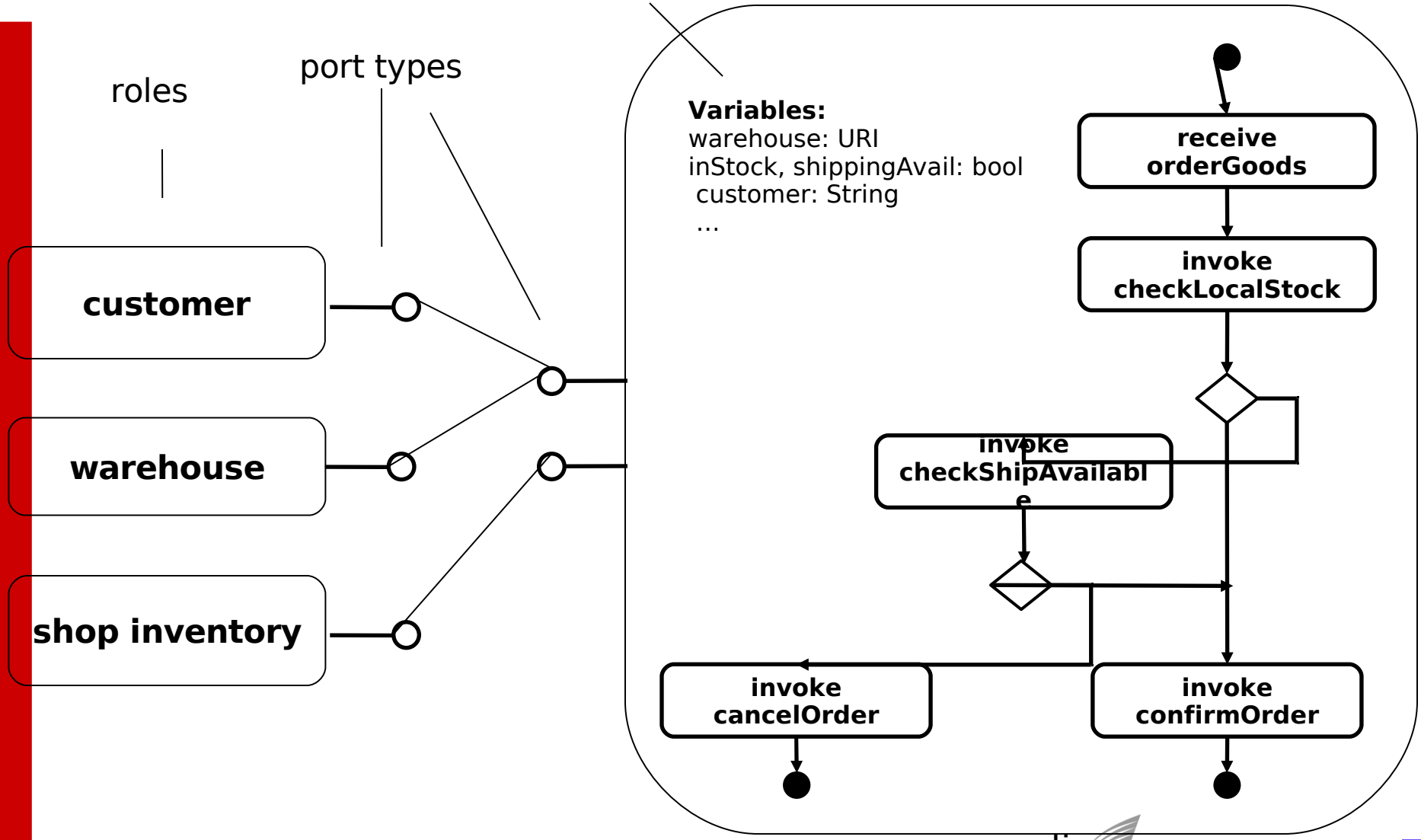
Orchestration

- In a well controlled environment there is a need for a simple process control language that only knows how to consume services and recover from error states (supplier controls the process, partners control any subprocesses).
- Orchestration provides a separated process control for pre-defined services
- Easier maintenance because we just have to reconfigure the process description to change the application logic
- Simple language but enough expression power to handle the workflow execution

Back to the real world: WS-BPEL / BPEL4WS

- Business Process Execution Language for orchestration
- WS-BPEL is the new 2.0 standard (minor changes to the current de-facto-standard)
- OASIS standard
- Originally developed by IBM and Microsoft
- Multiple implementations available from major vendors such as Oracle, IBM, BEA, Microsoft etc...

Abstract and/or executable process
orchestration,
variables and data transfers,
exception handling,
correlation information (for instance routing)



(Alonso & Pautasso, 2004)

Basic Elements of BPEL (Alonso & Pautasso, 2004)

PROCESS

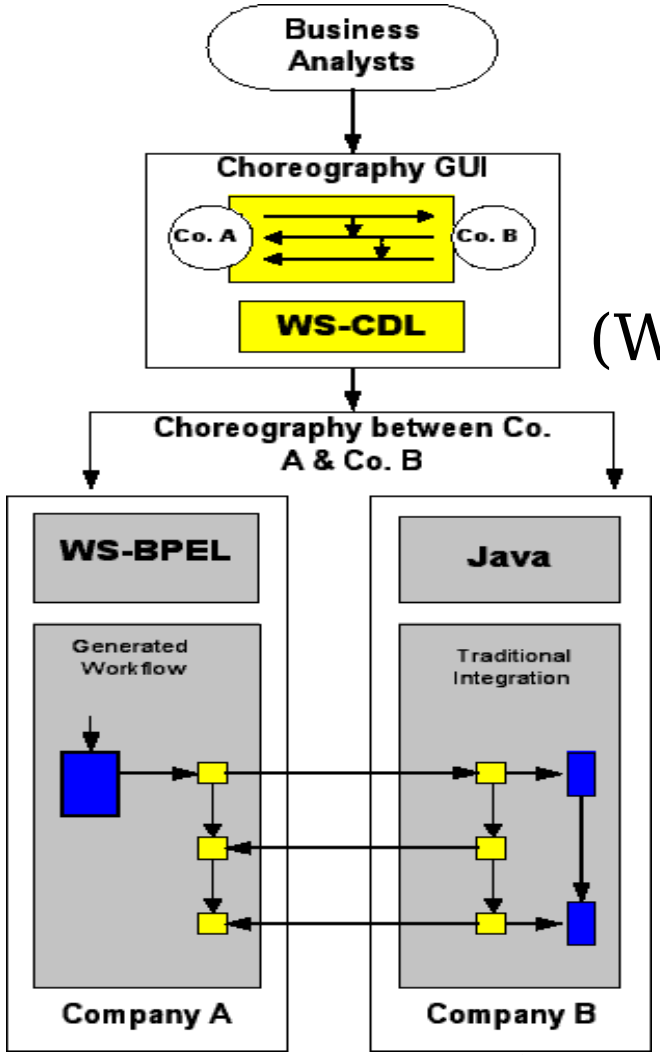
- **PARTNERS: Web services taking part in the process**
- **VARIABLES: the data used by the process**
- **CORRELATION SETS: constructs used to deal with conversations**
- **FAULT HANDLERS: what to do in case of errors (exceptions)**
- **COMPENSATION HANDLERS: what needs to be done to undo an activity**
- **EVENT HANDLERS: what to do when an event arrives**
- **ACTIVITIES: what the process does**

Equivalent to declarations in a normal programming language. It defines the way services are to be called, which data is to be used and which data is to be treated as stateful

These elements establish what the process does, how it reacts under different circumstances (errors, message arrivals, events, etc.), and how data moves from one step to the next

Choreography

- In an open environment there is a need for a description language that describes the services and waits for someone to negotiate and consume (e.g. Each subprocess is a software agent that may participate).
- Choreography defines the composition of interoperable collaborations between any type of party regardless of the supporting platform or programming model used by the implementation of the hosting environment
- Extends the orchestration by defining the abstract communication model



(W3C.org)

Planning

- Planning is a key ability for intelligent systems to
  - increase their autonomy and flexibility through the construction of sequences of actions to achieve their goals
  - representation of actions and world models
  - reasoning about the effects of actions
  - techniques for efficiently searching the space of possible plans

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

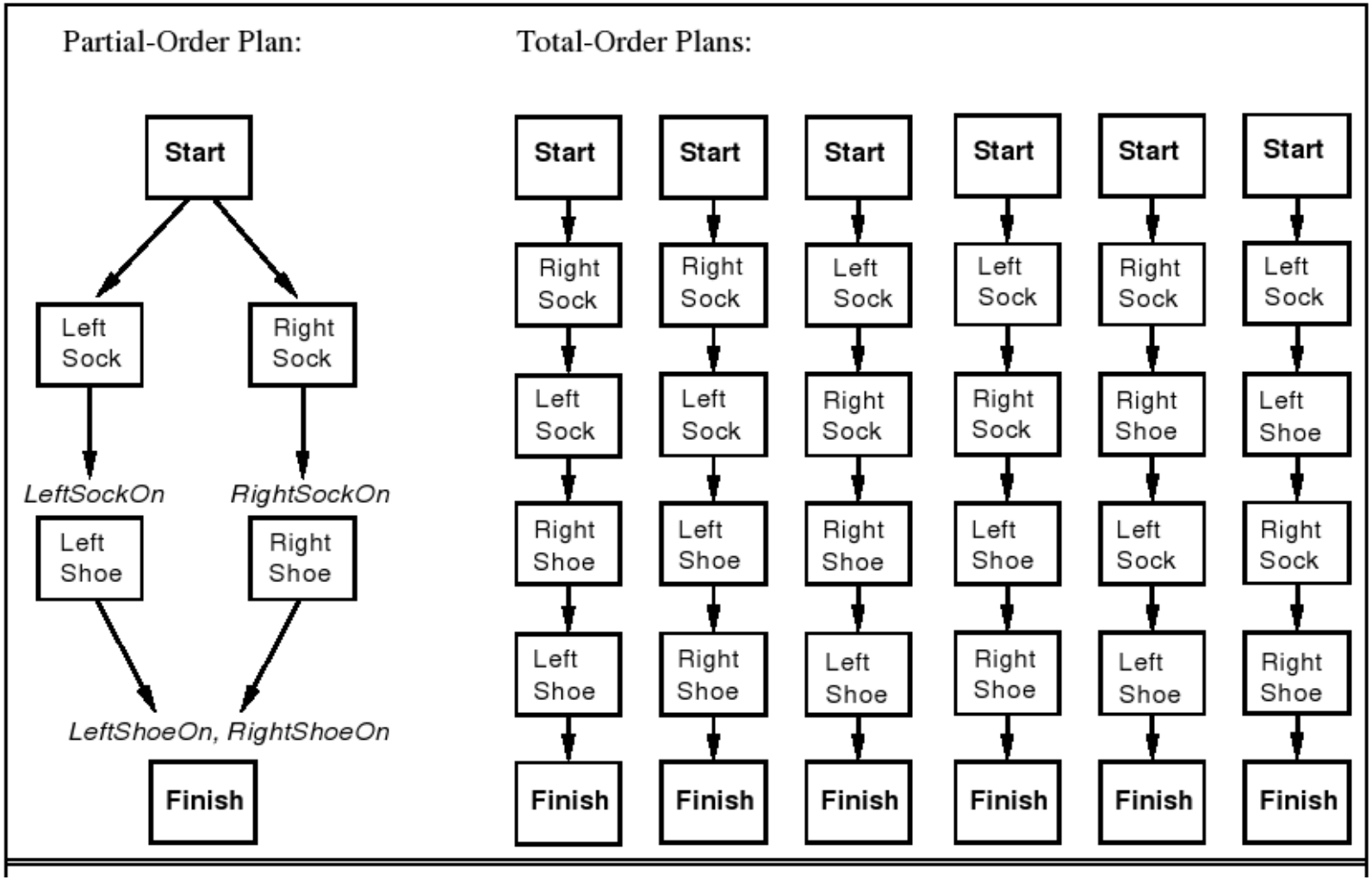
- Actions are represented with IOPEs
  - Inputs
  - Outputs
  - Preconditions
    - In which state the action may be performed
    - Conjunction of facts (only positive literals)
  - Effects
    - What are the changes in the universe according to the outputs of the action
- For example:
  - Action Bake(Cake)
    - 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:
Start
Left Sock
Right Sock
LeftSockOn
RightSockOn
Left Shoe
Right Shoe
LeftShoeOn, RightShoeOn
Finish
Total-Order Plans:
Start
Right Sock
Left Sock
Left Shoe
Right Shoe
Finish
Start
Right Sock
Left Sock
Right Shoe
Left Shoe
Finish
Start
Left Sock
Right Sock
Left Shoe
Right Shoe
Finish
Start
Left Sock
Right Sock
Right Shoe
Left Shoe
Finish
Start
Right Sock
Left Sock
Right Shoe
Left Shoe
Finish
Start
Left Sock
Right Sock
Left Shoe
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

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.

..more expressive languages

STRIPS **Language**	ADL **Language**
Only positive literals in states: $Poor \wedge Unknown$	Positive and negative literals in states: $\neg Rich \wedge \neg Famous$
Closed World Assumption: Unmentioned literals are false.	Open World Assumption: Unmentioned literals are unknown.
Effect $P \wedge \neg Q$ means add P and delete Q.	Effect $P \wedge \neg Q$ means add P and $\neg Q$ and delete $\neg P$ and Q.
Only ground literals in goals: $Rich \wedge Famous$	Quantified variables in goals: $\exists x\, At(P_1, x) \wedge At(P_2, x)$ is the goal of having P_1 and P_2 in the same place.
Goals are conjunctions: $Rich \wedge Famous$	Goals allow conjunction and disjunction: $\neg Poor \wedge (Famous \vee Smart)$
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 $(p : Plane)$.

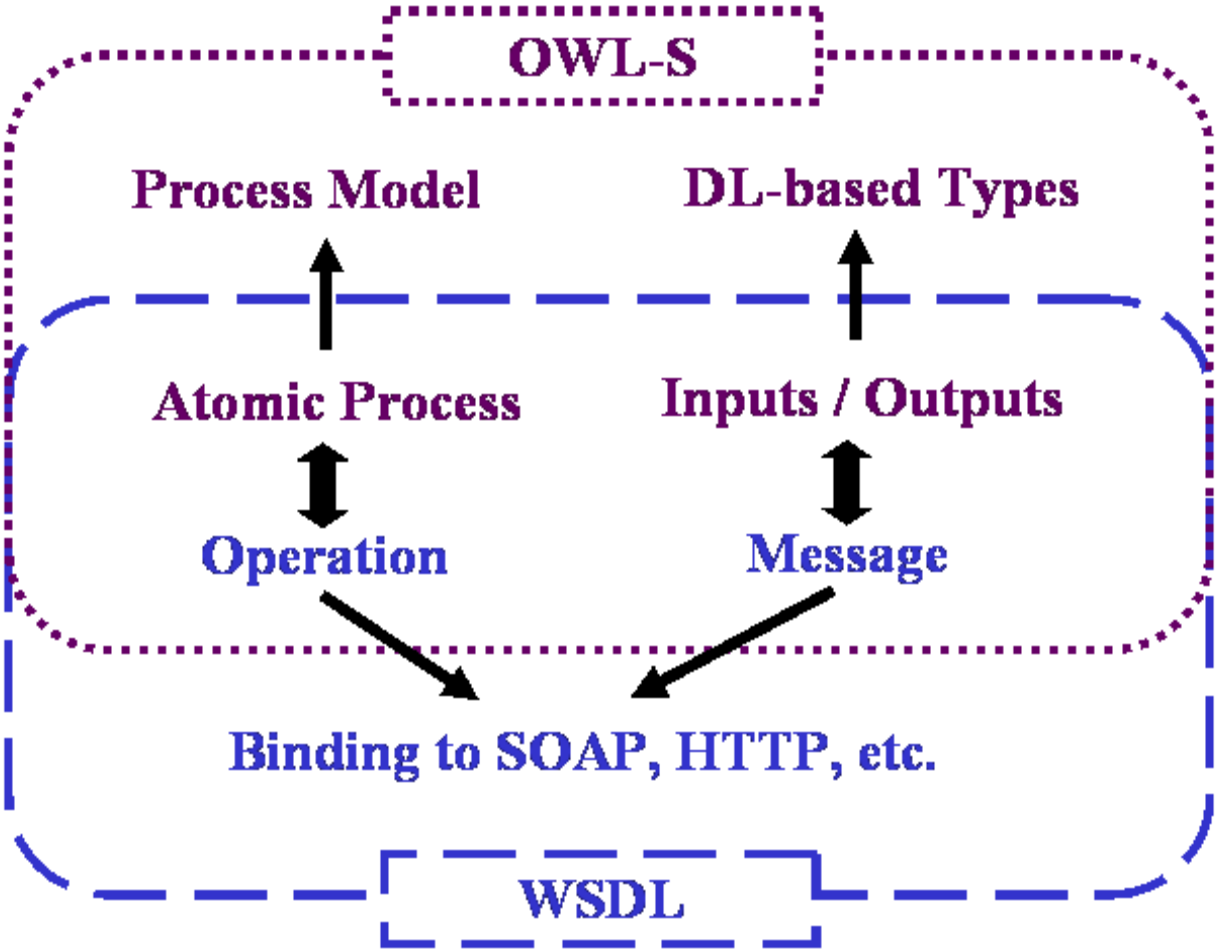
Figure 11.1 Comparison of STRIPS and ADL languages for representing planning problems. In both cases, goals behave as the preconditions of an action with no parameters.

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 actions
  - 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