UniLFS: A Unifying Logical Framework for Service Modeling and Contracting

RuleML 2103: 7th International Web Rule Symposium

July 11-13, 2013

Dumitru Roman[1] and Michael Kifer[2]

[1]*SINTEF / University of Oslo, Norway*

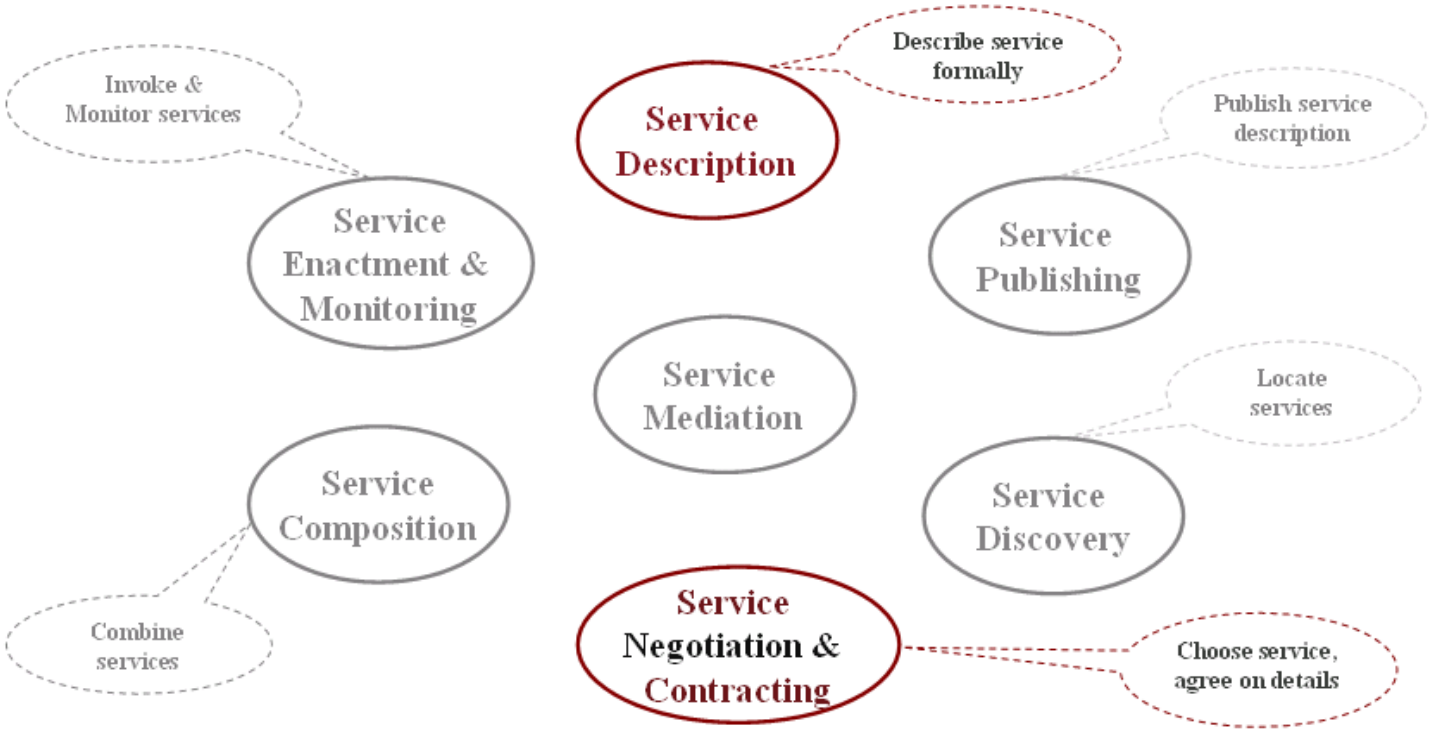
[2]*State University of New York at Stony Brook, New York, U.S.A.*

dumitru.roman@sintef.no, kifer@cs.sunysb.edu

Outline

- Context and motivation: Automated service contracting
- **UniLFS**: Modeling and reasoning about service behavior for automated service contracting
  - Elements of service contracts
  - Expressivity of the modeling language for service contracts
  - Reasoning technique for service contracting
- Related work
- Conclusions and outlook

Tasks to be automated when dealing with Services

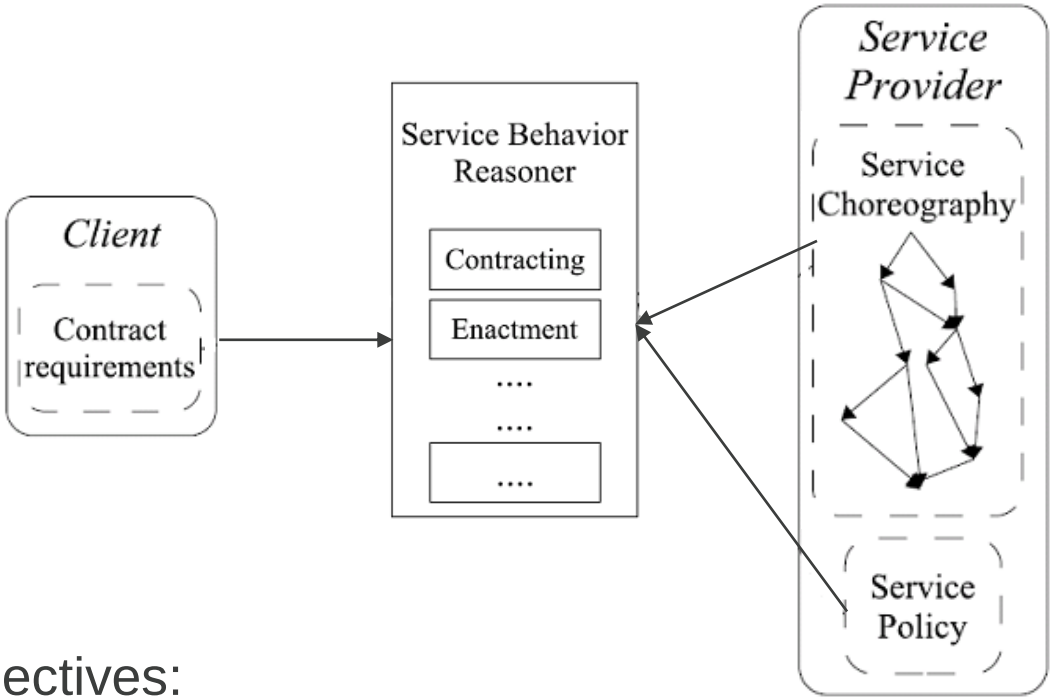


Automated service contracting

- Overall aim: Automate service contracting
  - Problems with traditional service contracting
    - High costs for contract establishment and management
    - Slow for rapidly changing business situations
  - Automated service contracting can potentially
    - Lower contract establishment and management costs
    - Speed up and improve contract establishment and management
- Current status: Limited expressivity of modeling frameworks and lack of sound and complete contract establishment techniques
- General approach
  1. Focus on those aspects of contracting that constrain interactions in processes
  2. Define the conceptual framework
  3. Formalization, representation, and reasoning

Service contracting – overview and objectives

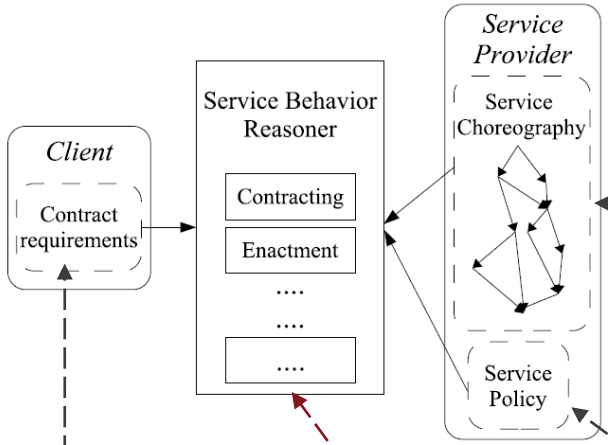
Service Contract = Choreography + Service Policies + Client Contract Requirements

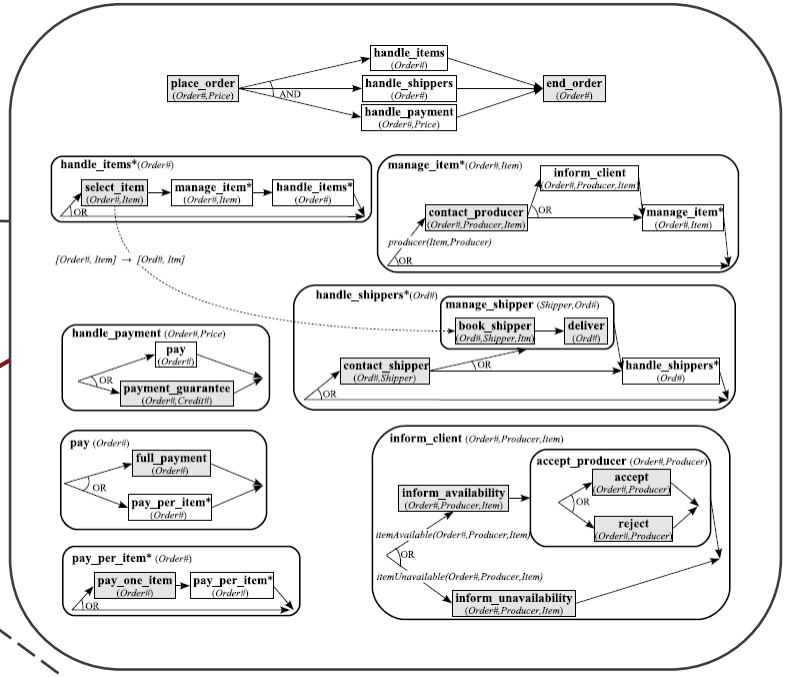


Main objectives:

1. An expressive language for contracting
   - Service choreography
   - Service policies
   - Client contract requirements
2. A reasoning mechanism to decide if service contracting is possible

Service contracting – example overview





Is this contract's execution possible?

Client contract requirements

4. All items in the same order must be shipped at the same time.
5. If full payment is chosen by the client, then it must happen only after all purchased items are delivered.
6. Before the client purchases items, the service must book a shipper.

Service policy

1. A shipper is booked only if the user accepts at least 7 items.
2. If pay per item is chosen by the user, then the payment must happen immediately before each item delivery.
3. Payment guarantee must be given before the client is informed about the availability of items.

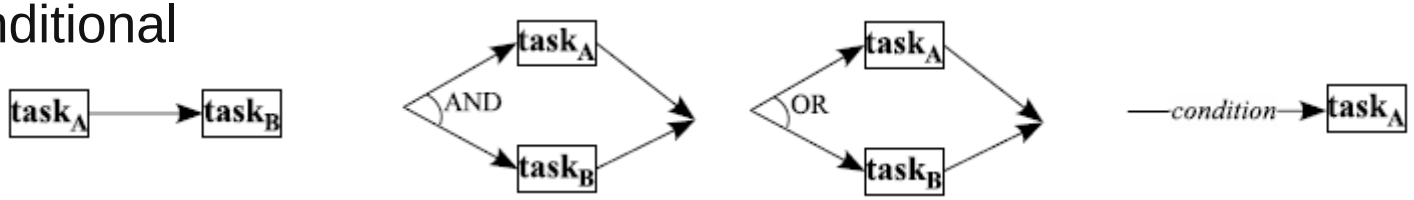
UniLFS – A language for service contracts

- Aim
  - An expressive representational framework for service contracts: combining procedural and declarative aspects
- Common requirements
  - Service choreographies
    - Conditional control flow: Sequential, concurrent, non-deterministic, and iterative interactions
    - Data flow: Local and non-local data passing between interactions
  - Service Policies and Client Contract Requirements
    - Temporal, conditional, and data constraints

Modeling service choreographies

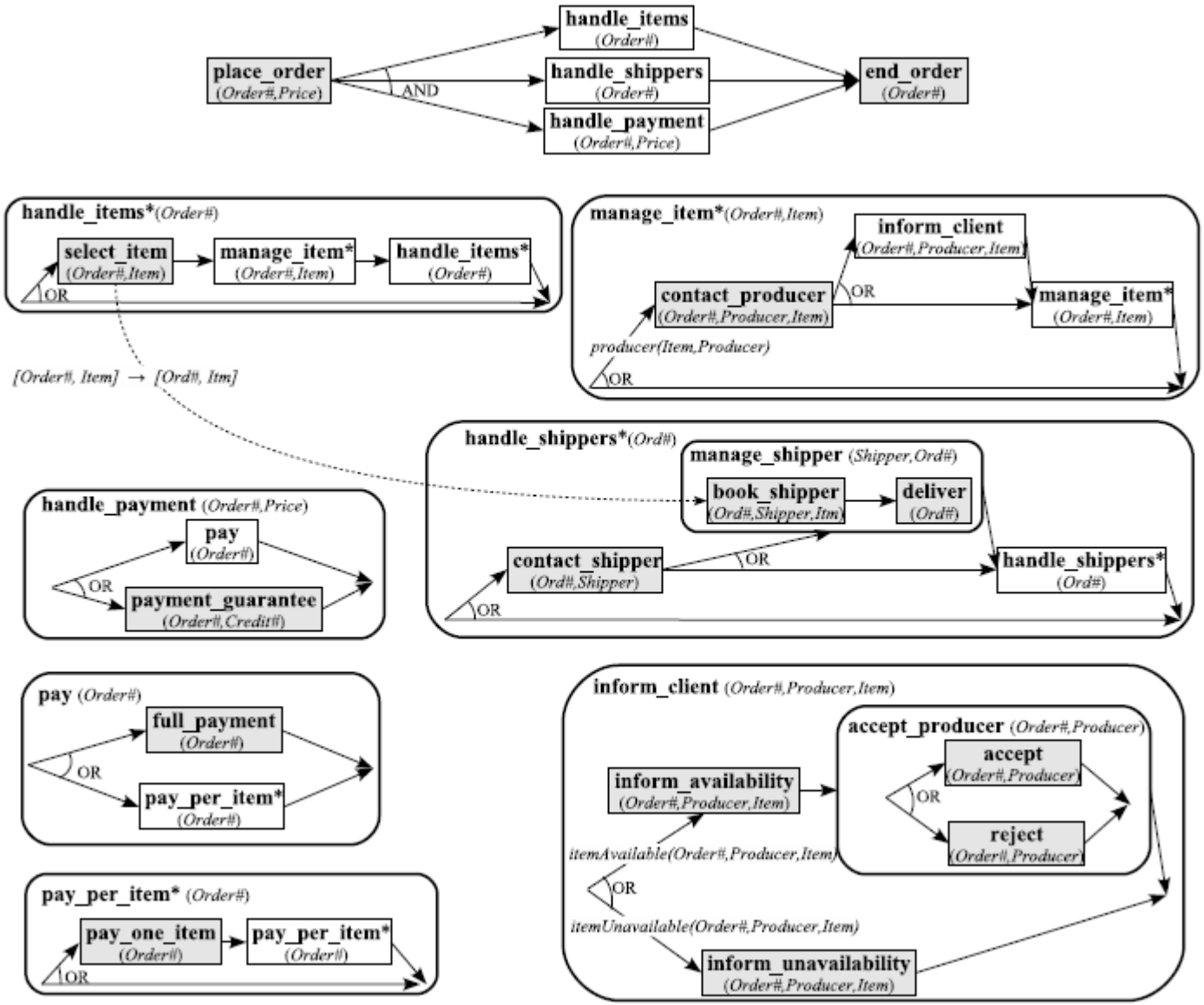
- Tasks: Task atoms and task definitions (hierarchical representations)

- Composition of tasks: Sequential, concurrent, non-deterministic, and conditional



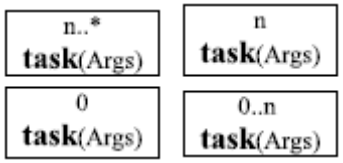
- Data passing between tasks
  - Data-passing through **shared arguments** is possible between a task and its direct successors, or within the definition of a composite task
  - Data-passing through **shared data space** is used when passing data is not possible through shared arguments due to the difference in scope of the arguments

Example

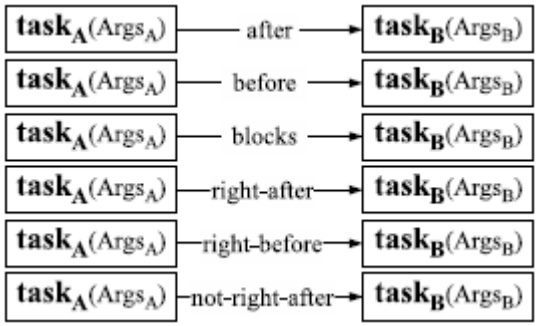

handle_items
(Order#)
place_order
(Order#,Price)
AND
handle_shippers
(Order#)
end_order
(Order#)
handle_payment
(Order#,Price)
handle_items*(Order#)
select_item
(Order#,Item)
manage_item*
(Order#,Item)
handle_items*
(Order#)
OR
manage_item*(Order#,Item)
inform_client
(Order#,Producer,Item)
contact_producer
(Order#,Producer,Item)
OR
manage_item*
(Order#,Item)
producer(Item,Producer)
OR
[Order#, Item] → [Ord#, Itm]
handle_shippers*(Ord#)
manage_shipper (Shipper,Ord#)
book_shipper
(Ord#,Shipper,Itm)
deliver
(Ord#)
contact_shipper
(Ord#,Shipper)
OR
handle_shippers*
(Ord#)
OR
handle_payment (Order#,Price)
pay
(Order#)
OR
payment_guarantee
(Order#,Credit#)
pay (Order#)
full_payment
(Order#)
OR
pay_per_item*
(Order#)
inform_client (Order#,Producer,Item)
accept_producer (Order#,Producer)
accept
(Order#,Producer)
OR
reject
(Order#,Producer)
inform_availability
(Order#,Producer,Item)
itemAvailable(Order#,Producer,Item)
OR
itemUnavailable(Order#,Producer,Item)
inform_unavailability
(Order#,Producer,Item)
pay_per_item* (Order#)
pay_one_item
(Order#)
pay_per_item*
(Order#)
OR

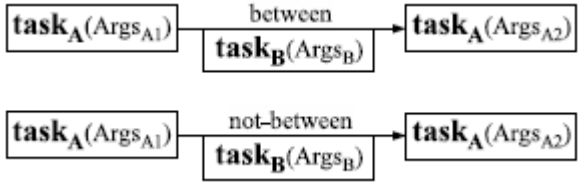
Modeling policies as constraints

- Existence constraints

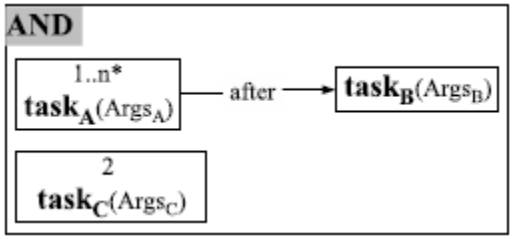


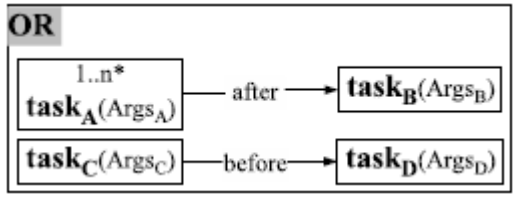
- Serial constraints





- Complex constraints – if C1 and C2 are constraints, then so are
  - C1 and C2
  - C1 or C2





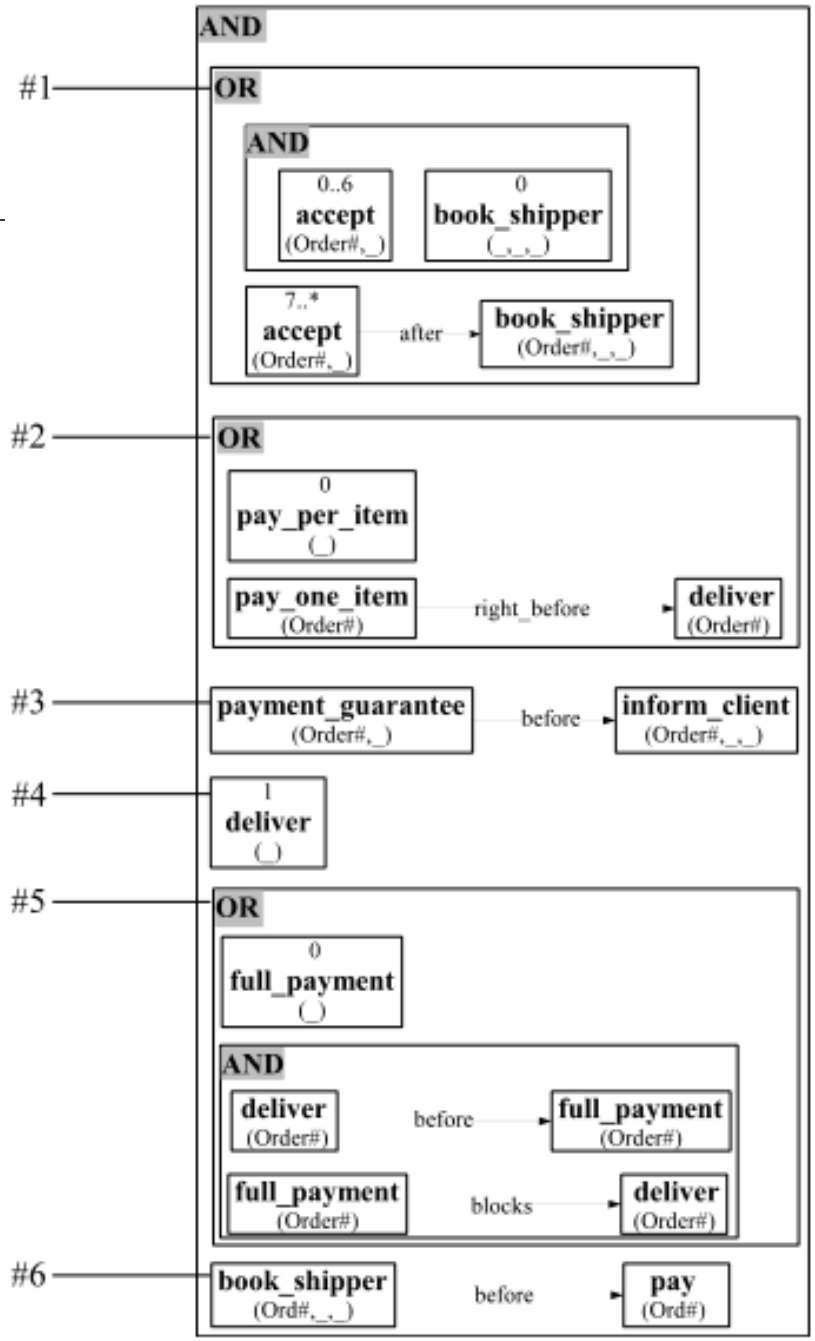
Example

Service policy

1. A shipper is booked only if the user accepts at least 7 items.
2. If pay per item is chosen by the user, then the payment must happen immediately before each item delivery.
3. Payment guarantee must be given before the client is informed about the availability of items.

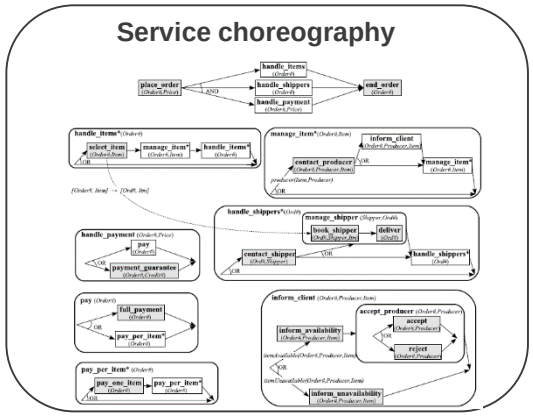
Client contract requirements

4. All items in the same order must be shipped at the same time.
5. If full payment is chosen by the client, then it must happen only after all purchased items are delivered.
6. Before the client purchases items, the service must book a shipper.



The Problem of Service Contracting and Enactment

- **Service contracting**: Decide if the contract is satisfiable
- **Service enactment**: If it is, find an enactment



What is the actual order of interactions if service execution possible?

Is this contract's contracting is possible?

Client contract requirements

4. All items in the same order must be shipped at the same time.
5. If full payment is chosen by the client, then it must happen only after all purchased items are delivered.
6. Before the client purchases items, the service must book a shipper.

Service policy

1. A shipper is booked only if the user accepts at least 7 items.
2. If pay per item is chosen by the user, then the payment must happen immediately before each item delivery.
3. Payment guarantee must be given before the client is informed about the availability of items.

Solution overview

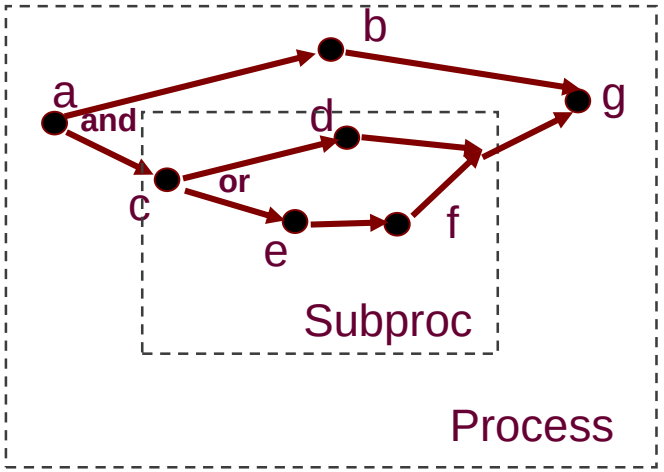
1. Concurrent Transaction Logic (CTR) – our formalism for
   - Choreographies
   - Service policies
   - Client contract requirements
2. Extend CTR when necessary
3. Prove correctness

Short introduction to CTR

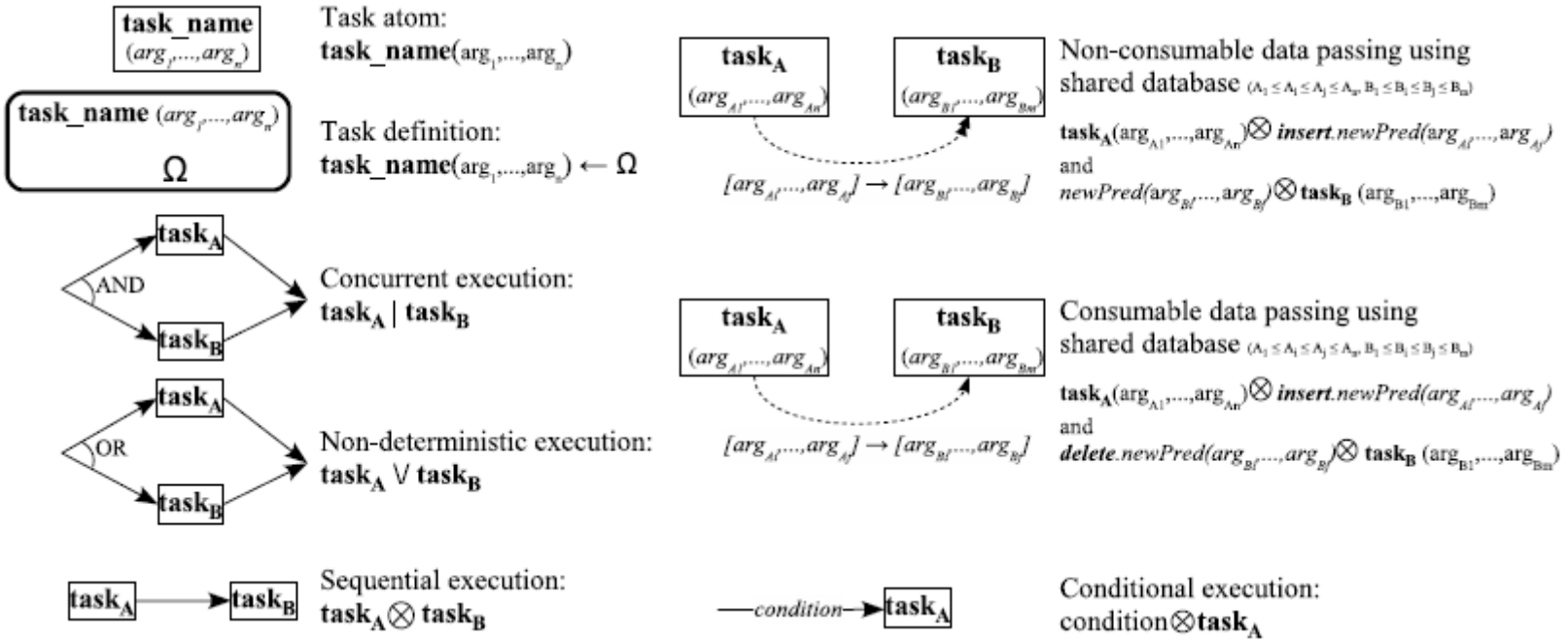
- Why CTR?
  - Logic for specification and execution of transactional processes
  - Integrates queries, updates, and transaction composition
  - Has proof procedure for concurrent Horn formulas: *proof* =*execution*
  - Can express a wide variety of constraints
  - Has been applied in the area of workflow verification and scheduling with promising results
- Atomic CTR formulas: same as in classical logic
- More complex formulas: built using connectives
  $a \otimes b$, $a \mid b$, $a \wedge b$, $a \vee b$, $\neg a$, $\odot a$

$Process \leftarrow a \otimes (b \mid Subproc) \otimes g$

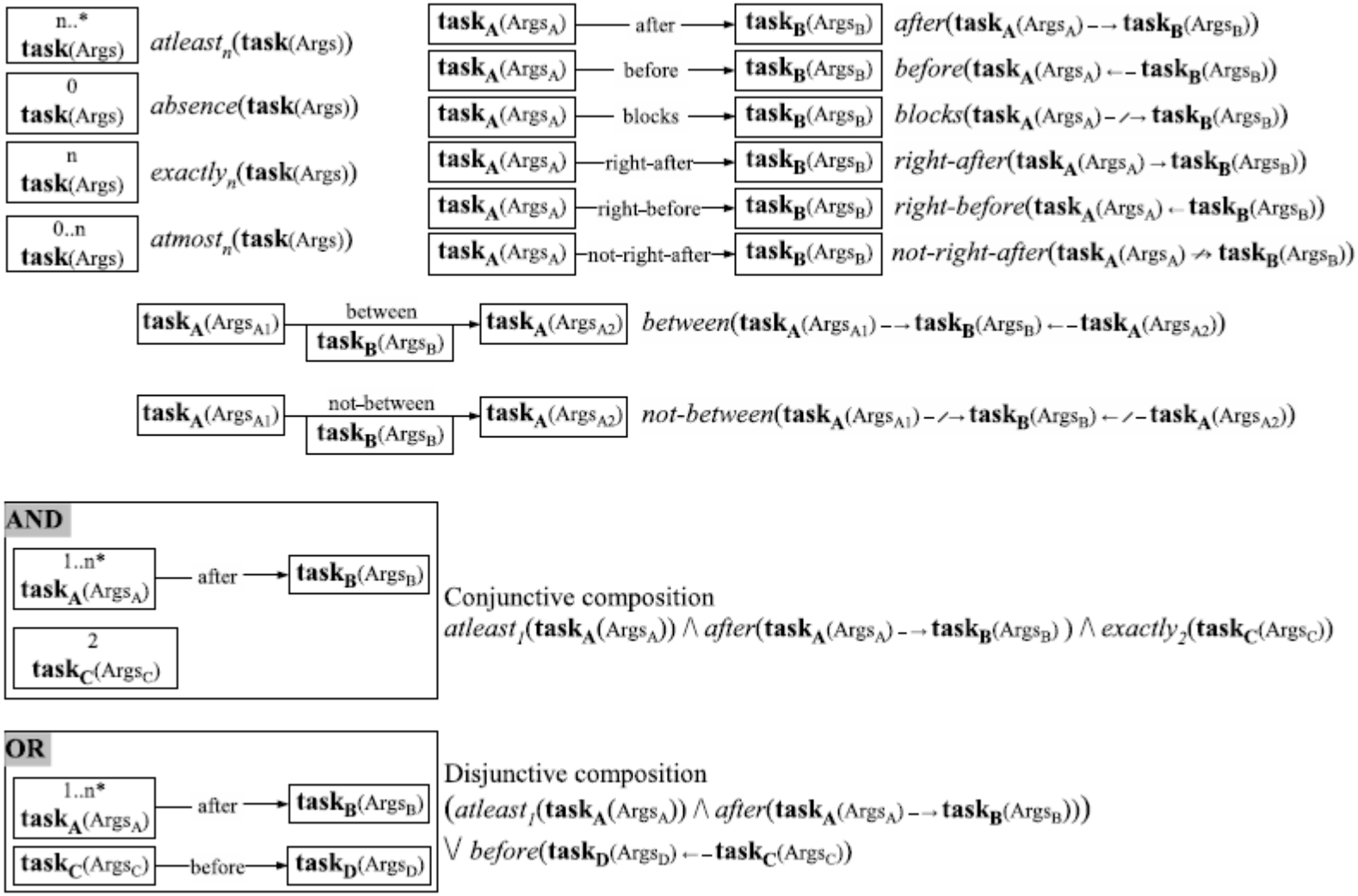
$Subproc \leftarrow (c \otimes (d \vee (e \otimes f)))$



Graphical representation of a service choreography in UniLFS



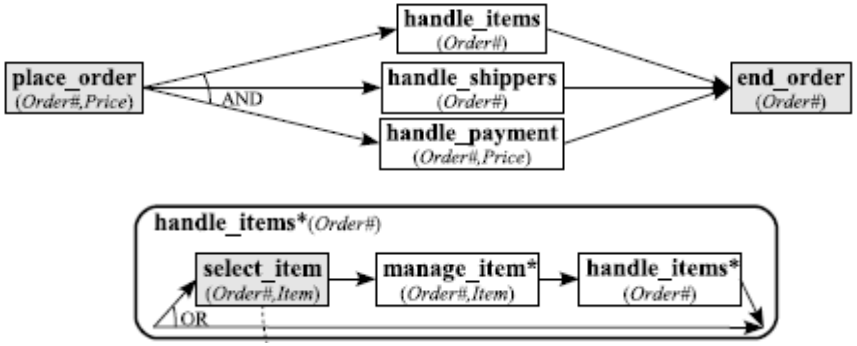
Graphical notation for constraints in UniLFS



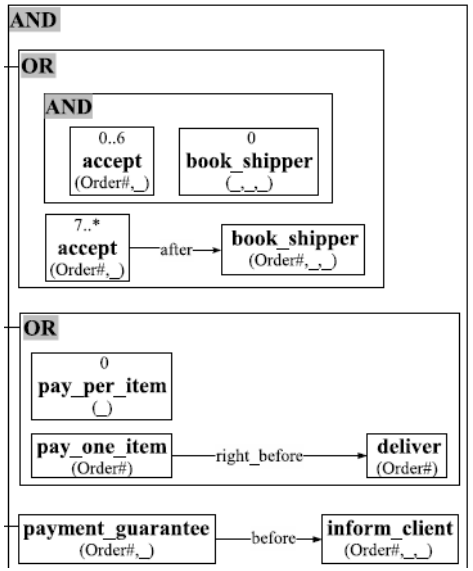
More examples of constraints that can be captured in UniLFS

- if *p* is executed, then *q* must also execute (before or after *q*)
- if *p* is executed, then *p* must also be executed, and vice versa
- every occurrence of task *p* must be followed by an occurrence of task *q* with the same argument and there must be an occurrence of *p* before every occurrence of *p* and their arguments must be the same
- if task *p* is executed then *q* must execute after it, and before that *q* there can be no other *p*
- if task *q* is executed, it has to be preceded by an occurrence of *p*; the next instance of *q* can execute only after another occurrence *p*
- tasks *p* and *q* must alternate
- …

Service contract formalization with CTR – Example



. . .



. . .

ConcurrentHorn /\ Constraints

$$
\begin{aligned}
&\mathtt{place\_order}(?Order\#, ?Price) \otimes \\
&(\mathtt{handle\_items}(?Order\#) \\
&\quad \mid \mathtt{handle\_shippers}(?Order\#) \\
&\quad \mid \mathtt{handle\_payment}(?Order\#, ?Price)) \otimes \\
&\mathtt{end\_order}(?Order\#)
\end{aligned}
$$

$$
\begin{aligned}
&\mathtt{handle\_items}(?Order\#) \leftarrow \\
&\quad (\mathtt{select\_item}(?Order\#, ?Item) \otimes \\
&\qquad \mathit{insert.selected\_item}(?Order\#, ?Item) \otimes \\
&\qquad \mathtt{manage\_item}(?Order\#, ?Item) \otimes \\
&\qquad \mathtt{handle\_items}(?Order\#)) \\
&\mathtt{handle\_items}(?Order\#) \leftarrow \mathtt{state}
\end{aligned}
$$

. . .

Λ

1. $(\mathtt{atmost}_6(\mathtt{accept}(\_Order\#,\_)) \wedge \mathtt{absence}(\mathtt{book\_shipper}(\_,\_,\_)))$
   $\vee\ (\mathtt{atleast}_7(\mathtt{accept}(\_Order\#,\_))$
   $\wedge\ \mathtt{after}(\mathtt{accept}(\_Order\#,\_) \dashrightarrow \mathtt{book\_shipper}(\_Order\#,\_,\_))$
   where $\mathtt{atmost}_n(p)$ is a shorthand for
   $\mathtt{absence}(p) \vee \mathtt{exactly}_1(p) \vee \mathtt{exactly}_2(p) \vee \ldots \vee \mathtt{exactly}_n(p)$
2. $\mathtt{absence}(\mathtt{pay\_per\_item}(\_))$
   $\vee\ \mathtt{right\_before}(\mathtt{pay\_one\_item}(\_Order\#) \leftarrow \mathtt{deliver}(\_Order\#))$
3. $\mathtt{before}(\mathtt{inform\_client}(\_Order\#,\_,\_) \dashleftarrow \mathtt{payment\_guarantee}(\_Order\#,\_))$

. . .

Contributions

- **Prior work**
  - Original CTR proof theory does not handle constraints
  - Prior work added simple constraints, but
    - No iterative processes
    - No constraints on iterations
    - No data flow
- **New**
  - Add the necessary modeling primitives
  - Extend the CTR proof theory to handle service contracting with iterations, data flow, constraints, etc.
- **As a result**
  - Can capture much of BPMN and WS-BPEL process modeling, extend them with declarative constraints
  - And contracting over them

Reasoning about service contracts with CTR

- Contracting: Find out if an execution of the CTR formula *Choreography /\ Policy /\ ClientRequirements* exists
- Enactment: Find a constructive proof for this execution

Main result: The extended inference system is **sound** and **complete** for proving the above

  - Soundness: Every found execution is correct
  - Completeness: All possible executions can be found

Extended proof theory – An overview

- Sequent: $\mathbf{P}, \mathbf{s} \text{---} \vdash (\exists)\, \psi \wedge \mathcal{C}$
  - Meaning: $(\exists)\, \psi$ can execute starting at state **s** given P in such a way that *C* is satisfied

- Each inference rule looks like this: $\dfrac{sequent_1}{sequent_2}$

  For example:

$$\frac{\mathbf{P}, \mathbf{s} \text{---} \vdash (\exists)\ \psi'\sigma \wedge \mathcal{C}}{\mathbf{P}, \mathbf{s} \text{---} \vdash (\exists)\ \psi \wedge \mathcal{C}}$$

- Has seven inference rules and one axiom

Related Work

- E-contracting
  - Pieces of frameworks, models, architectures and different aspects and phases of e-contracting (negotiation, enforcement, violation detection, monitoring, legal aspects), but no unifying formalism
  - We provide a unifying and general framework for e-contracting
- Workflow/process modeling
  - Most languages are procedural (WS-BPEL, BPMN, YAWL).

    The declarative ones (DecSerFlow) are inexpressive
  - Our framework: very expressive. Integrates conditional control flow, data flow, hierarchical modeling, and complex constraints
- Process verification
  - Most of the existing approaches use temporal logic/model checking. No obvious way to handle certain aspects: data flow, hierarchical modeling
  - We use CTR. More natural modeling language. Sometimes has better complexity.

Conclusions and Outlook

- UniLFS summary
  - Identified main element of service contracts and formally formulated the problem of service contracting
  - *Modeling*: Developed an expressive representational framework for service contracts, combining procedural and declarative elements
  - *Reasoning*: Developed a sound and complete proof theory for deciding if contracting for a service is possible
- Future work
  - Semi-automated extraction of UniLFS specifications from text (e.g. contracts)
  - Complexity study, e.g. subsets of constraints for which the verification problem has a better complexity
  - Implementation

Thank you!

? \/ !