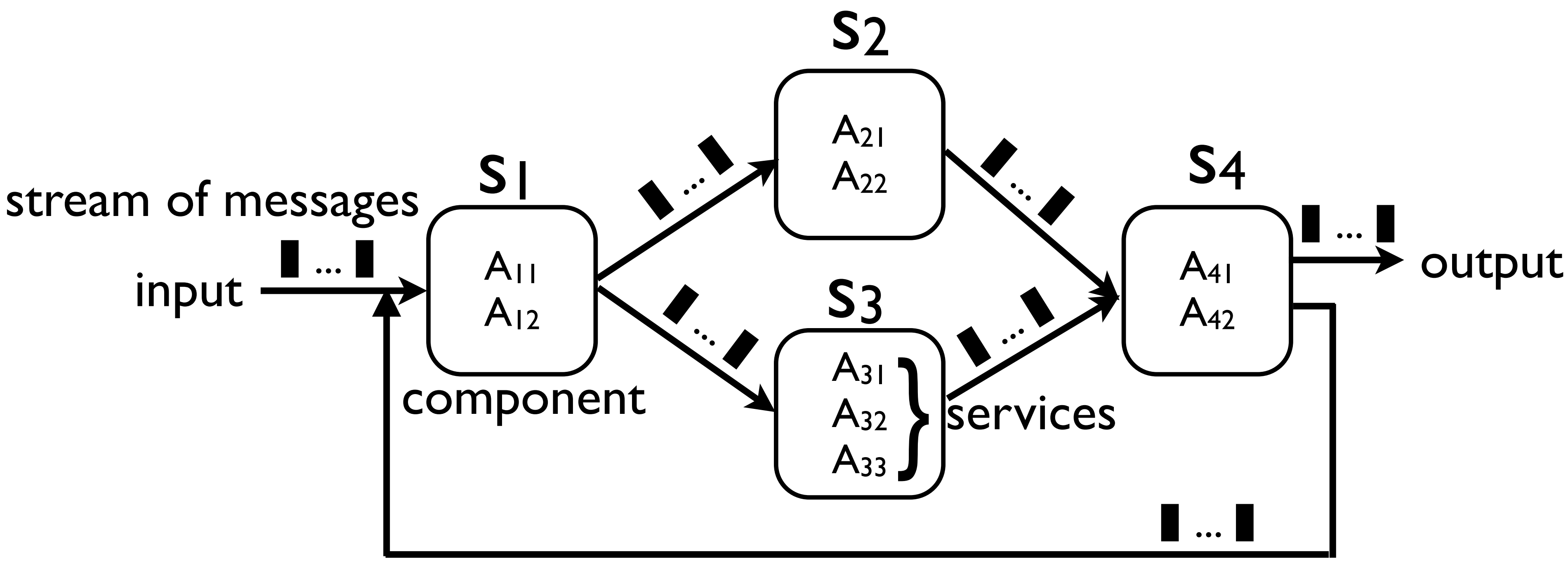
Interface reconciliation in Kahn Process Networks using CSP and SAT

Pavel Zaichenkov
Olga Tveretina
Alex Shafarenko

Compiler Technology and Computer Architecture Group
University of Hertfordshire (UK)

CSPSAT 2015
August 31, 2015

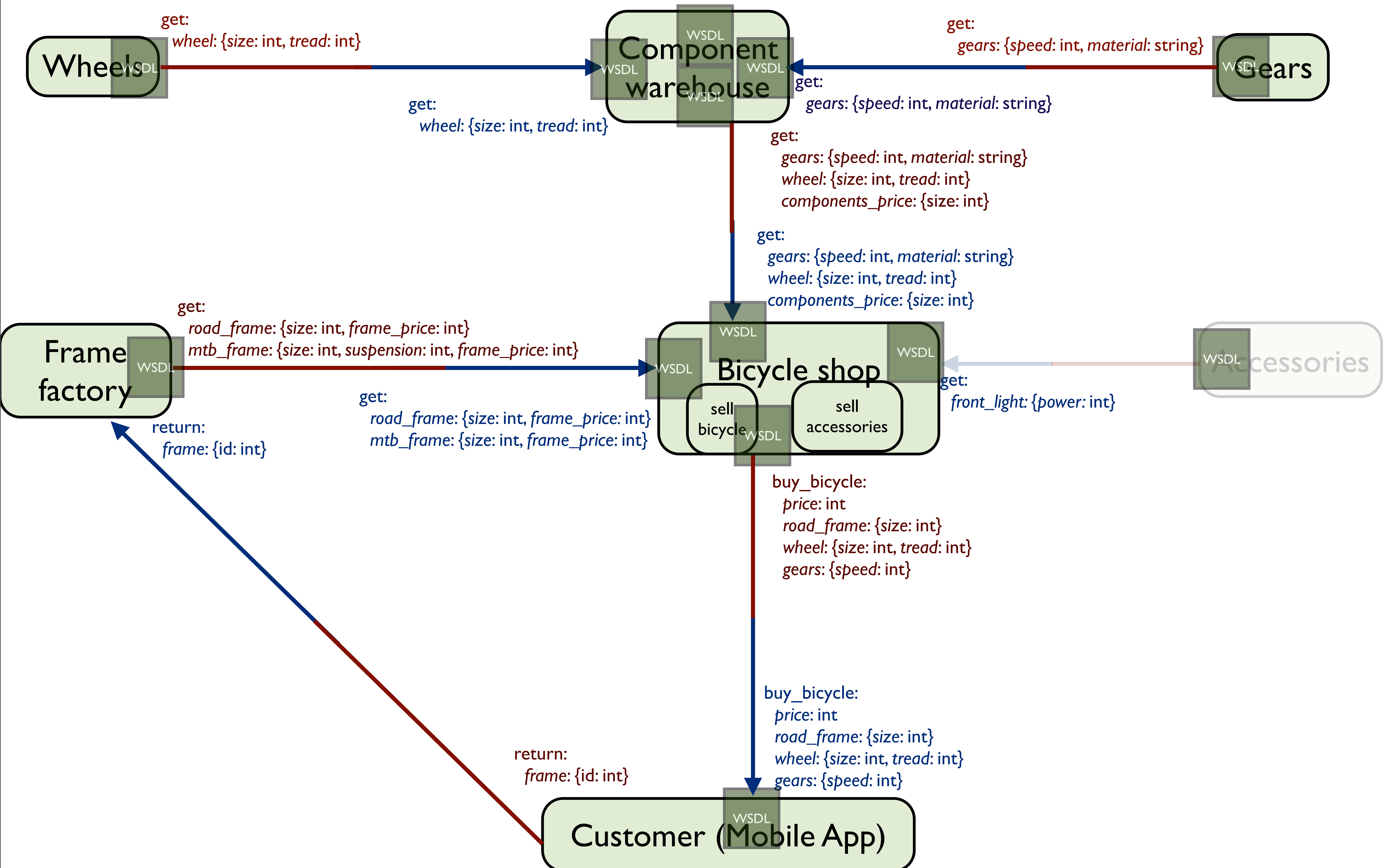
Kahn process networks (KPNs)

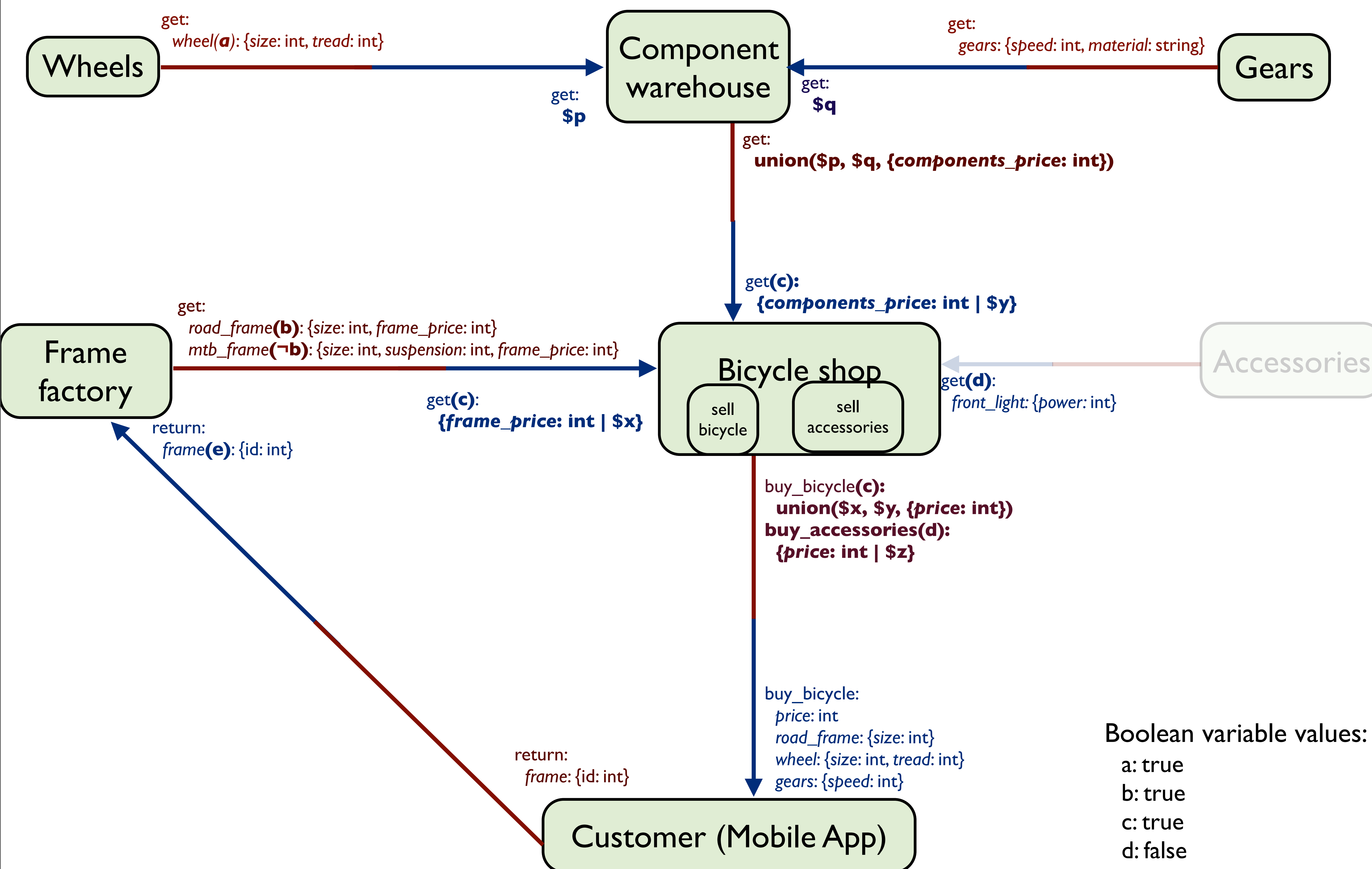


- components are connected by FIFO data channels that transfer self-contained messages (no references or pointers)
- each component contains a set of services. A service is a sequential process that implements some domain-specific algorithm
- the components are developed in a decontextualised manner: a component contains numerous services that represent algorithms compatible with various contexts
- the services are "black boxes". We assume that services only expose their interfaces
- assumption: the network topology is defined statically (in contrast to actor model or MPI)
- application: web-services in the Cloud (Service-Oriented Architecture)

Web services: state of the art

- Many web services are built using Service-Oriented Architecture (SOA)
- A service contract is specified in Web Service Description Language (WSDL)
- WSDL (XML) is a representational glue for services: responsible for combining and formatting the data
- Lack of flexible representational glue that connects services together


get:
wheel: {size: int, tread: int}
Wheels
WSDL
Component warehouse
WSDL
WSDL
WSDL
WSDL
get:
gears: {speed: int, material: string}
Gears
WSDL
get:
wheel: {size: int, tread: int}
get:
gears: {speed: int, material: string}
get:
gears: {speed: int, material: string}
wheel: {size: int, tread: int}
components_price: {size: int}
get:
gears: {speed: int, material: string}
wheel: {size: int, tread: int}
components_price: {size: int}
get:
road_frame: {size: int, frame_price: int}
mtb_frame: {size: int, suspension: int, frame_price: int}
Frame factory
WSDL
Bicycle shop
WSDL
WSDL
WSDL
sell bicycle
WSDL
sell accessories
Accessories
WSDL
get:
front_light: {power: int}
get:
road_frame: {size: int, frame_price: int}
mtb_frame: {size: int, frame_price: int}
return:
frame: {id: int}
buy_bicycle:
price: int
road_frame: {size: int}
wheel: {size: int, tread: int}
gears: {speed: int}
buy_bicycle:
price: int
road_frame: {size: int}
wheel: {size: int, tread: int}
gears: {speed: int}
return:
frame: {id: int}
Customer (Mobile App)
WSDL



Boolean variable values:
a: true
b: true
c: true
d: false
e: true

Challenges

- cross-service interaction is essentially local (pairwise)
- processing graph is not taken into account
- cycles make direct constraint resolution impossible

Research problem

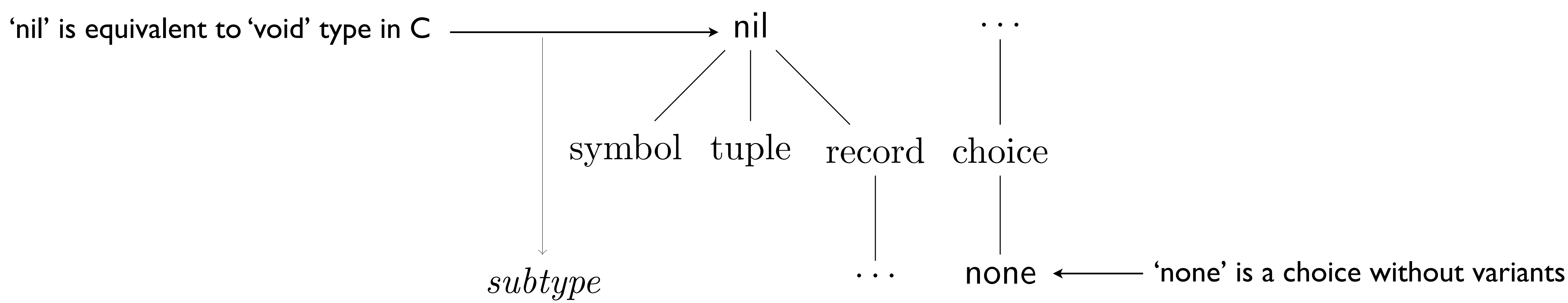
- a static mechanism that establishes compatibility of services and works out necessary conditions
- support of structural subtyping in the interfaces is needed
- inheritance that propagates requirements over the processing graph is needed

Contribution

- Designed a Message Definition Language that supports subtyping, inheritance and Boolean flags (MDL can be used instead of XML-based WSDL)
- Developed a mechanism for interface reconciliation based on CSP and SAT
- Designed and implemented a communication protocol for services specified in C++

Message Definition Language

- describes a message format as a term algebra
- the subtyping relation forms two semilattices for term sorts



- support of inheritance for
  - records (similar to 'structs' in C, but don't take an order of elements into account)
    intuitively, a record that contains more elements is a subtype
  - choices (similar to 'unions' in C, but don't take an order of elements into account)
    intuitively, a choice that contains less elements is a subtype

Constraint Satisfaction Problem

The seniority relation $\sqsubseteq$ represents the subtyping relation on terms. If a term t' describes the input interface of a service, then the service can process any message described by a term t, such that $t \sqsubseteq t'$.

Definition 2 (CSP-KPN). *For each $t \sqsubseteq t' \in \mathcal{C}(\mathsf{G})$ find a vector of Boolean values $\vec{b} = (b_1, \ldots, b_l)$, vectors of ground terms $\vec{t} = (t_1, \ldots, t_m)$, $\vec{t'} = (t'_1, \ldots, t'_n)$ such that*

$$t[\vec{f}/\vec{b}, \uparrow\vec{v}/\vec{t}, \downarrow\vec{v}/\vec{t'}] \sqsubseteq t'[\vec{f}/\vec{b}, \uparrow\vec{v}/\vec{t}, \downarrow\vec{v}/\vec{t'}],$$

where $\vec{f} = (f_1, \ldots, f_l)$, $\uparrow\vec{v} = (\uparrow v_1, \ldots, \uparrow v_m)$, $\downarrow\vec{v} = (\downarrow v_1, \ldots, \downarrow v_n)$. The tuple $(\vec{b}, \vec{t}, \vec{t'})$ is called a solution.

Formal proofs are available in the full paper (http://arxiv.org/abs/1503.00622)

Solution algorithm

$B_0 \subseteq B_1 \subseteq \cdots \subseteq B_s$ are sets of Boolean constraints.

$\vec{a}^{\uparrow}$ and $\vec{a}^{\downarrow}$ are vectors of semiground terms called *conditional approximations* of the solution.

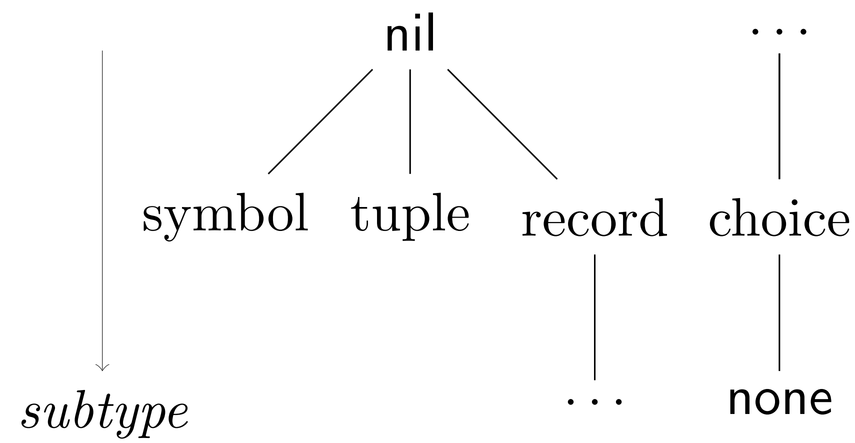
We seek the solution as a fixed point of a series of approximations in the following form:

$$(B_0, \vec{a}^{\uparrow}_0, \vec{a}^{\downarrow}_0), \dots, (B_{s-1}, \vec{a}^{\uparrow}_{s-1}, \vec{a}^{\downarrow}_{s-1}), (B_s, \vec{a}^{\uparrow}_s, \vec{a}^{\downarrow}_s), \qquad (1)$$

where for every $1 \leq k \leq s$ and a vector of Boolean values $\vec{b}$ that is a solution to $\mathsf{SAT}(B_k)$ (by $\mathsf{SAT}(B_k)$ we mean a set of Boolean vector satisfying B_k):

$$\vec{a}^{\uparrow}_{k-1}[\vec{f}/\vec{b}] \sqsubseteq \vec{a}^{\uparrow}_k[\vec{f}/\vec{b}] \qquad \text{and} \qquad \vec{a}^{\downarrow}_k[\vec{f}/\vec{b}] \sqsubseteq \vec{a}^{\downarrow}_{k-1}[\vec{f}/\vec{b}], \qquad (2)$$

where the elements of the vectors are compared pairwise. The starting point is $B_0 = \emptyset$, $\vec{a}^{\uparrow}_0 = (\mathsf{none}, \dots, \mathsf{none})$, $\vec{a}^{\downarrow}_0 = (\mathsf{nil}, \dots, \mathsf{nil})$ and the series terminates as soon as $\mathsf{SAT}(B_s) = \mathsf{SAT}(B_{s-1})$, $\vec{a}^{\uparrow}_s = \vec{a}^{\uparrow}_{s-1}$, $\vec{a}^{\downarrow}_s = \vec{a}^{\downarrow}_{s-1}$.



Formal proofs are available in the full paper (http://arxiv.org/abs/1503.00622)

Structural subtyping and inheritance

an example for records (messages)

an interface, which is derived from the service

frame factory

{frame_price: int,
color: int, *size*: int}

{*frame_price*: int | \$y} → {*price*: int | \$y}

customer

{*price*: int, *size*: int}

after specialisation the interface becomes

{*frame_price*: int, *size*: int} → {*price*: int, *size*: int}

size is inherited from the input to the output; *color* is ignored

an example for choices (collections of messages)

warehouse

an interface, which is derived from the service

customer

(: *get_component*: ...,
rent_bicycle: ...,
~~*replace_component*: ...~~ :)

(: *get_component*: ... | \$x :) → (: *buy_bicycle*: ... | \$x :)

(: *buy_bicycle*: ...,
rent_bicycle: ...,
paint_bicycle: ... :)

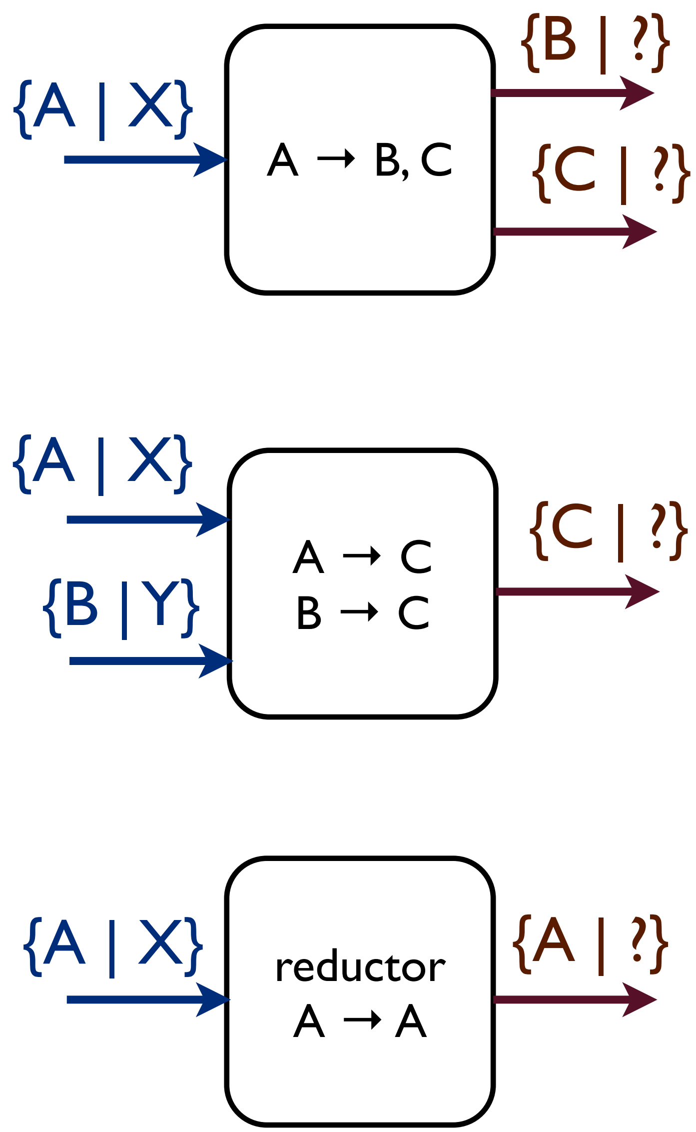
after specialisation the interface becomes

(: *get_component* : ..., *rent_bicycle*: ... :) →
(: *buy_bicycle*: ..., *rent_bicycle*: ...:)

rent_bicycle is inherited from the output to the input; *paint_bicycle* cannot be provided to the customer;
present of *replace_component* in the interface of the producer would cause an error

Inheritance challenges

- How inheritance should work for services with multiple input and output channels?
- What if the data of various sort needs to be inherited from many channels to one?
- How to inherit data in services with accumulating facility
(i.e. a service produces a message in response to a set of messages)?



Implementation

- The CSP solver has been implemented in OCaml
- The solver uses PicoSAT to solve the adjunct SAT problem
- A communication protocol that supports C++ has been implemented. It includes:
  - derivation of interfaces and constraints from the code
  - code specialisation and library generation based on the solution provided by the CSP solver

Summary

- We designed a mechanism that establishes compatibility and specialisation of web-services when they have multiplicity and genericity
- The MDL was introduced to specify flexible service interfaces with support of subtyping, inheritance and Boolean flags
- The problem is represented as a CSP+SAT; the solver has been developed
- Communication protocol for C++ has been developed

Future work

- support of abstract data types (objects)
- performance measurement and optimisation
- improvement of error tracing and reporting

Future work

- support of abstract data types (objects)
- performance measurement and optimisation
- improvement of error tracing and reporting

Thank you!