Formal Functional Description of Semantic Web Services:
the Logic Description Method (LDM)

Ye, Li sagi.ye@gmail.com
Chen, Junliang chjl@bupt.edu.cn

Switching and Intelligent Control Research Center,
State lab of Switch Technology and Telecommunication Networks,
Beijing University of Posts & Telecommunications (BUPT).

Functional Properties of Semantic Web Services: “IORPE”

- Input:
  - To define the data-type of input parameters;
- Output:
  - To define the data-type of output parameter;
- IO-Relation:
  - To define the logic relation between input/output parameters;
- Precondition:
  - The “world states” required for the successful execution of the service;
- Effect:
  - The change of the “world states” due to the execution of the service.

All in logic predicates

- Parameter: *Type* (*name*)
  - *integer* (*input_1*),
  - *zip-code* (*input_2*),
  - *weather-data* (*output*);
- World state and its change:
  - *logged-in* (*user-id*),
  - message-sent (*address*, *content*),
  - *account-charged* (*account*, *amount*);
- IO-Relation:
  - *has-Id* (*user*, *user-id*),
  - *has-Name* (*town*, *name*) & *located* (*town*, *location*).

Why IO-Relation?

- If a service is defined as:
  - input is "*integer*", output is "*integer*",
  - that is "*integer -> integer*",

  can the function be figured out?
- Example, IO-Relation for the following two services:
  - S1: increases the input by 1

    IO-Relation = *Successor* (*input*, *output*);
  - S2: doubles the input

    IO-Relation = *Double* (*input*, *output*);

[The "*Successor*" and "*Double*" are pre-defined semantic terms.]

LDM: render into a FOL formula

Services as a first-order logic (FOL) formula:

```
All x1 x2 ... xn (
    T1(x1) &...& Tn(xn) & P(x1,x2,...,xn)
    ->
    (exists r (Tr(r) & R(x1,x2,...,xn,r)))).
```

where:

- `x1...xn:` represents the input of the service;
- `r:` represents the output of the service;
- `T*(k):` means ‘k’ has the type of ‘T*’;
- `P(x1,...,xn):` means the input satisfy precondition ‘P’;
- `R(x1,...,xn,r):` means service has the result ‘R’.

Note: IO-Relation and Effect are different in concept. But they share the same representing form and are both input/output related. So they are combined as the result of service.

The “air temperature” example

- `Centigrade(x):` x is a value of temperature in Centigrade unit.
- `Fahrenheit(x):` x is a value of temperature in Fahrenheit unit.
- `Loc(x):` x is a value of location data.
- `WeatherData(x):` x is a value of weather data.
- `MobileStation(x):` x is the identifier of a mobile station.
- `MsUser(x):` x is the identifier of a mobile station user.
- `C2F(x,y):` x in Centigrade unit and y in Fahrenheit unit represent the same physical temperature.
- `F2C(x,y):` reversal of `C2F`.
- `WeatherInfo(x,y):` Location x has the weather information of y.
- `At(x,y):` Object x is at the location of y.
- `Carries(x,y):` x carries the object y.
- `WeatherTemp(x,y):` y is the air temperature in Centigrade unit extracted from the weather data x.
- `TempInfoC(x,y):` y is the air temperature in Centigrade unit of location x.
- `TempInfoF(x,y):` y is the air temperature in Fahrenheit unit of location x.

The “air temperature” example

Available Web Services:

- TempC2F: convert from Centigrade to Fahrenheit

```
All tc (Centigrade(tc) ->
(exists tf (Fahrenheit(tf) & C2F(tc, tf)))).
```

- TempF2C: convert from Fahrenheit to Centigrade

```
All tf (Fahrenheit(tf) ->
(exists tc (Centigrade(tc) & F2C(tf, tc)))).
```

- Weather : supply weather information

```
All l (Loc(l) ->
(exists w (WeatherData(w) & WeatherInfo(l,
 w)))).
```

- Location : supply location information

```
All ms (MobileStation(ms) ->
(exists l (Loc(l) & At(ms, l)))).
```

The “air temperature” example

User request [ReqService]:

A mobile station user want to get the air temperature (in Fahrenheit unit) of his current location.

```
ReqService: All user (MsUser(user) ->
(exists tf (Fahrenheit(tf) & R(user,
 tf)))).
```

And R(user, tf) is defined as:

```
KR: All user tf
(exists l (At(user, l) & TempInfoF(l, tf))
 -> R(user, tf)).
```

The “air temperature” example

Necessary knowledges:

- K1: A object carrier has the same location of this object;
  - `All container object (Carries(container, object) -> (all l (At(object, l)->At(container, l)))).`
- K2: A mobile station user carries a mobile station;
  - `All user (MsUser(user) -> (exists ms (MobileStation(ms) & Carries(user, ms)))). [GetMS]`
- K3: Weather information includes air temperature;
  - `All w (WeatherData(w) -> (exists tc (Centigrade(tc) & WeatherTemp(w, tc)))). [GetTemp]`
  - `All w tc l (WeatherTemp(w, tc) & WeatherInfo(l, w) -> TempInfoC(l, tc)).`

The “air temperature” example

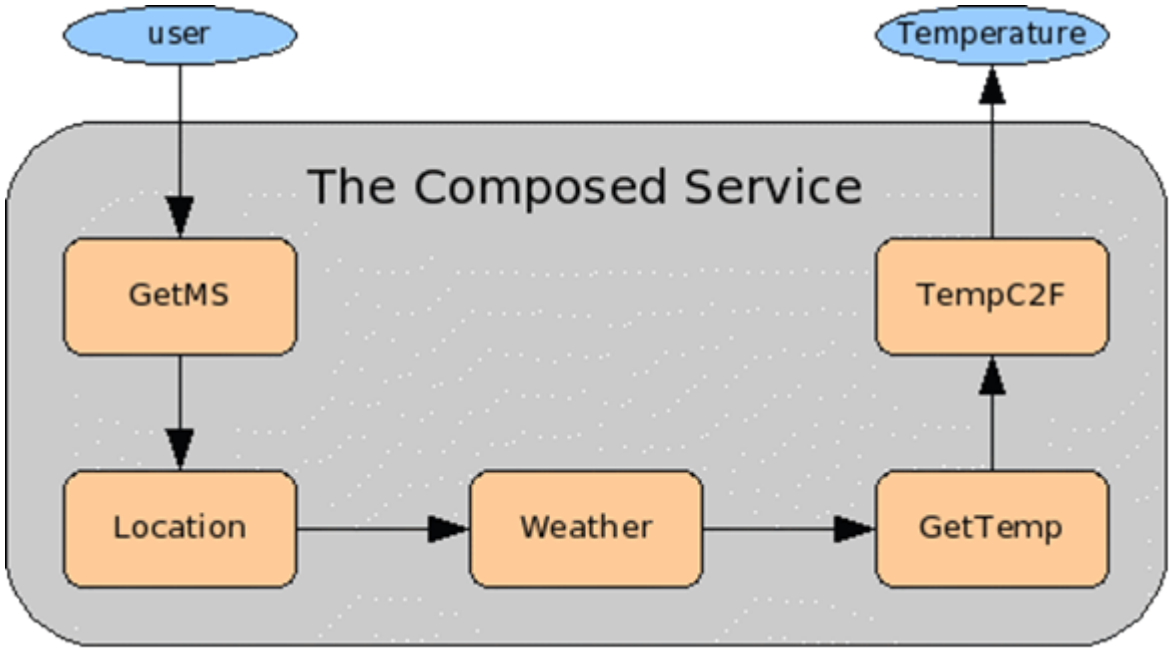
Necessary knowledges:

- K4: One temperature can be represented in different units while still has the same physical meaning.
  - `All l tc tf (TempInfoC(l, tc) & C2F(tc, tf) -> TempInfoF(loc, tf)).`
  - `All l tc tf (TempInfoF(l, tf) & F2C(tf, tc) -> TempInfoC(loc, tc)).`

The “air temperature” example

Compose Result:

```
ReqService(user) =
TempC2F(GetTemp(Weather(Location(GetMS(user)
  )))).
```



Evaluate

- First-Order Logic is already a reliable, mature and well studied mathematics system. It is formal, precise and concise.
- The semantic of the services and knowledges are self-explaining. Just reading the FOL formula’s meaning in logic way.

Evaluate

- The composed service can have firmly, precisely inferred semantics. It is determinate and explainable.
- The result can be easily checked for correctness. This is a key feature for the automatic mechanism to be used in a composition task because it makes the result believable.

The “air temperature” example

Verify process:

1)MsUser(user); ***by input***

2)ms = GetMS(user) => MobileStation(ms) & Carries(user, ms) [a]; by ***K2***

3)l = Location(ms) => Loc(l) & At(ms, l) [b]; ***by Location***

4)w = Weather(l) => WeatherData(w) & WeatherInfo(w, l) [c]; ***by Weather***

5)tc = GetTemp(w) => Centigrade(tc) & WeatherTemp(w, tc) [d]; ***by K3***

6)tf = TempC2F(tc) => Fahrenheit(tf) & C2F(tc, tf) [e]; ***by TempC2F***

7)[a]&[b] => At(user, l); [f] ***by K1***

8)[c]&[d] => TempInfoC(l, tc); [g] ***by K3***

9)[g]&[e] => TempInfoF(l, tf); [h] ***by K4***

10)[f]&[h] => R(user, tf). ***By KR***

Evaluate

- The knowledge-base of LDM is like a ontology of semantic web which based on FOL instead of DL. And FOL is far more powerful the DL.

Thanks.

Any Question?