Use of Semantics for Service Composition

Mariano Belaunde (MAPS/MEP/UED)

Metadata Registries Workshop, May 30th 2012
Berlin

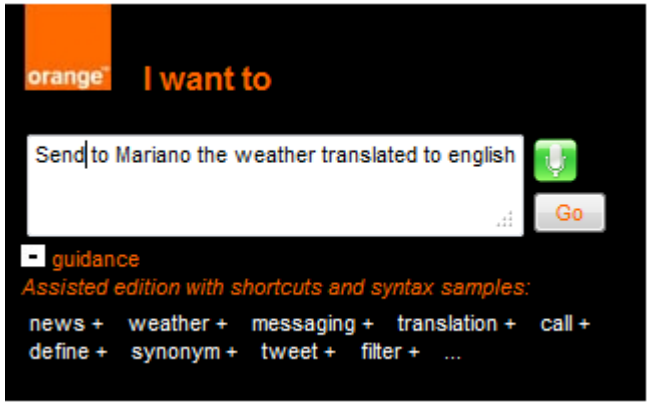
Outline

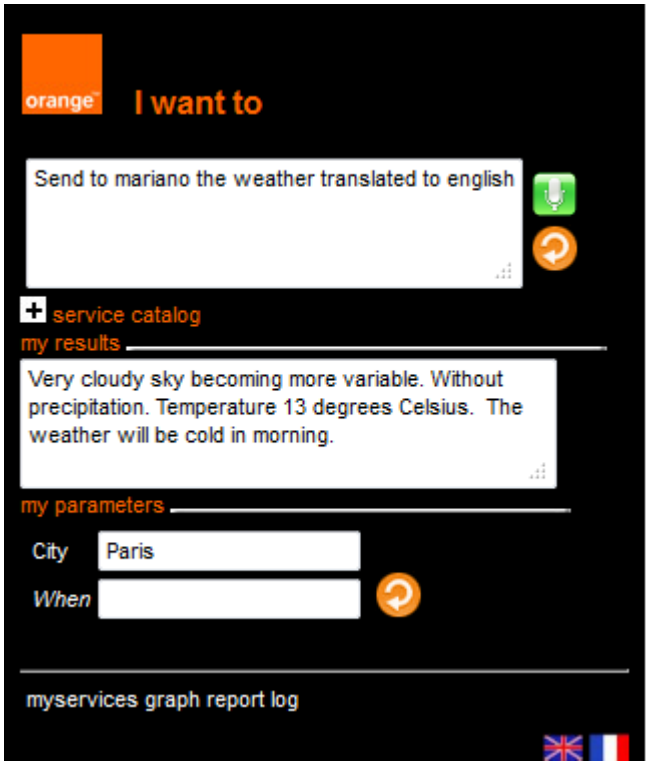
- Semantics for Service Composition exploiting Natural Language
- Semantics for Service Composition with recommendations
- Conclusions:
  - Three layer architecture

Semantics for Service Composition exploiting Natural Language

NL interpretation in Natural Mashups

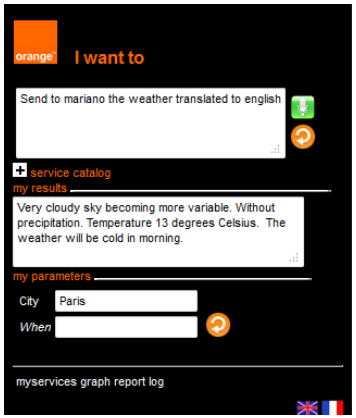
- "Natural Mashups" Goal
  - Make as simple as possible the creation of personalized added-value services that can be executed immediately or saved for further usage.
    - Using a restricted form of natural language



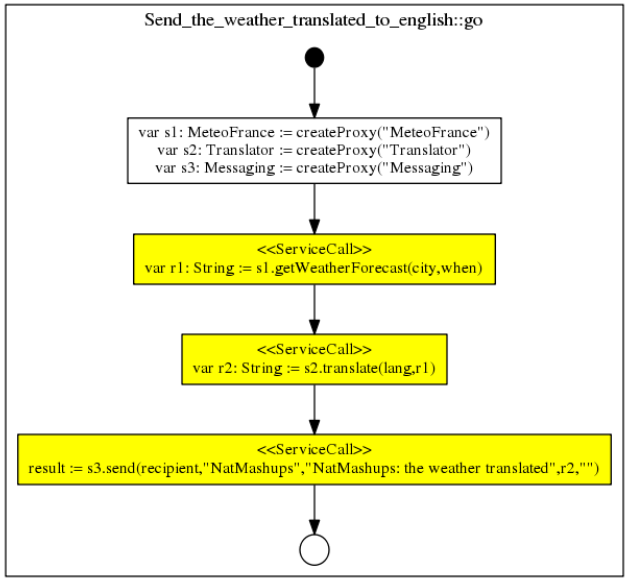


Usage of Semantics in the interpretation system

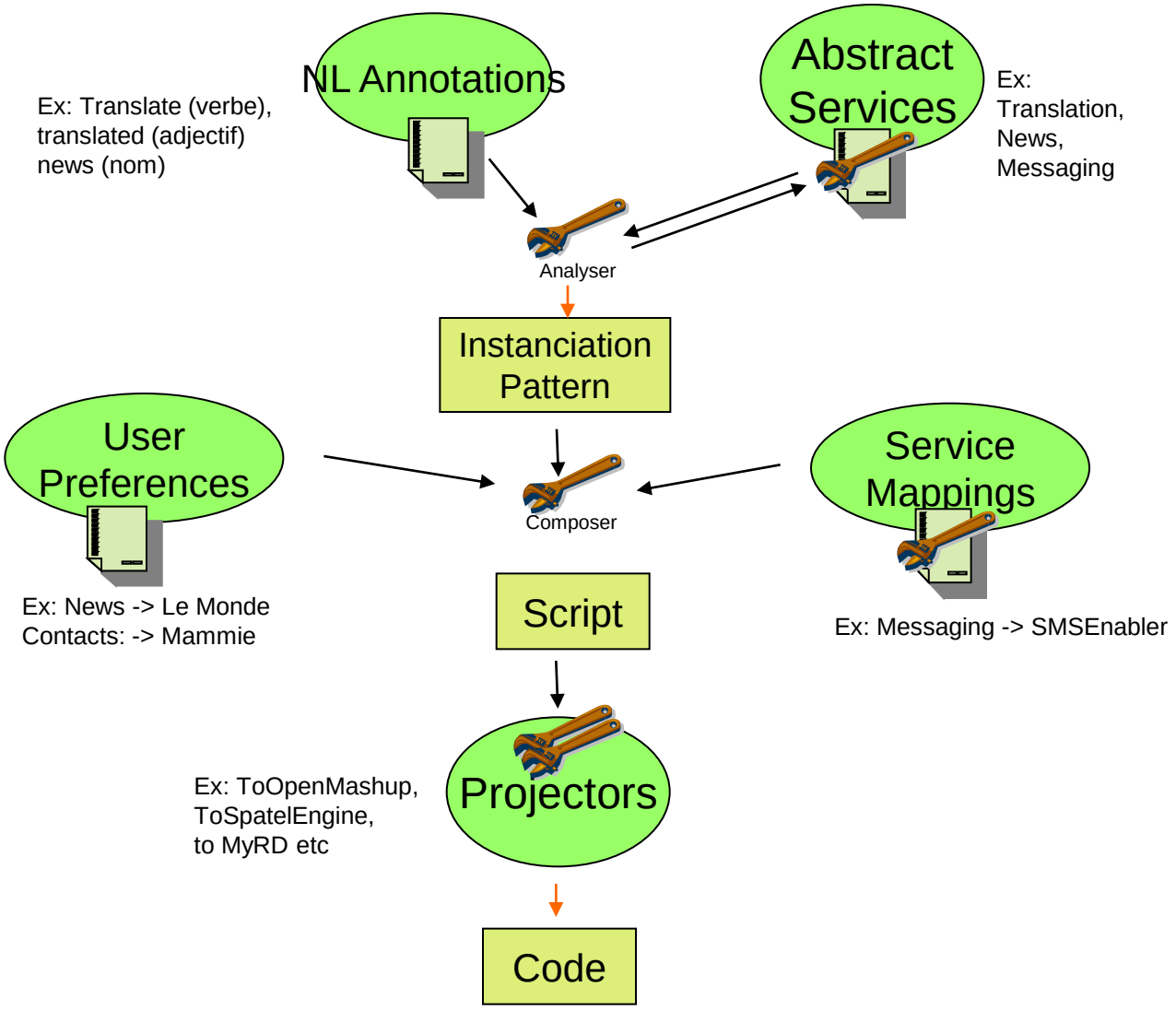
- Core design principles
  - Interpretation of a request implies reasoning on **semantic abstractions**. In our case the core semantic data are service operations (message sending, translation of a text, and so on).
  - Then these abstractions are annotated by **syntax patterns** (NL Annotations) to activate NL
  - Then **mappings** from abstraction to concrete services need to be defined
- Advantages
  - Using abstractions avoid replication of NL annotations (Rather than annotating "send SMS by Orange", and "send SMS by Telefonica" we annotate generically: "Messaging/send").
  - More stable definitions on which reasoning can be done
  - We can easily switch from one mapping choice to another mapping choice. Mapping could be done statically or dynamically depending on user context.
- How to represent our semantic data (functions)
  - Use service-based representation => ***Abstract Services*** containing generic service operations**.** No formalism gap between abstract and concrete level.
    => Example "Messaging" abstract service representrs the semantics
  - => Language datatype represents the type of parameter
  - => An Ontology is generated from service-based representation



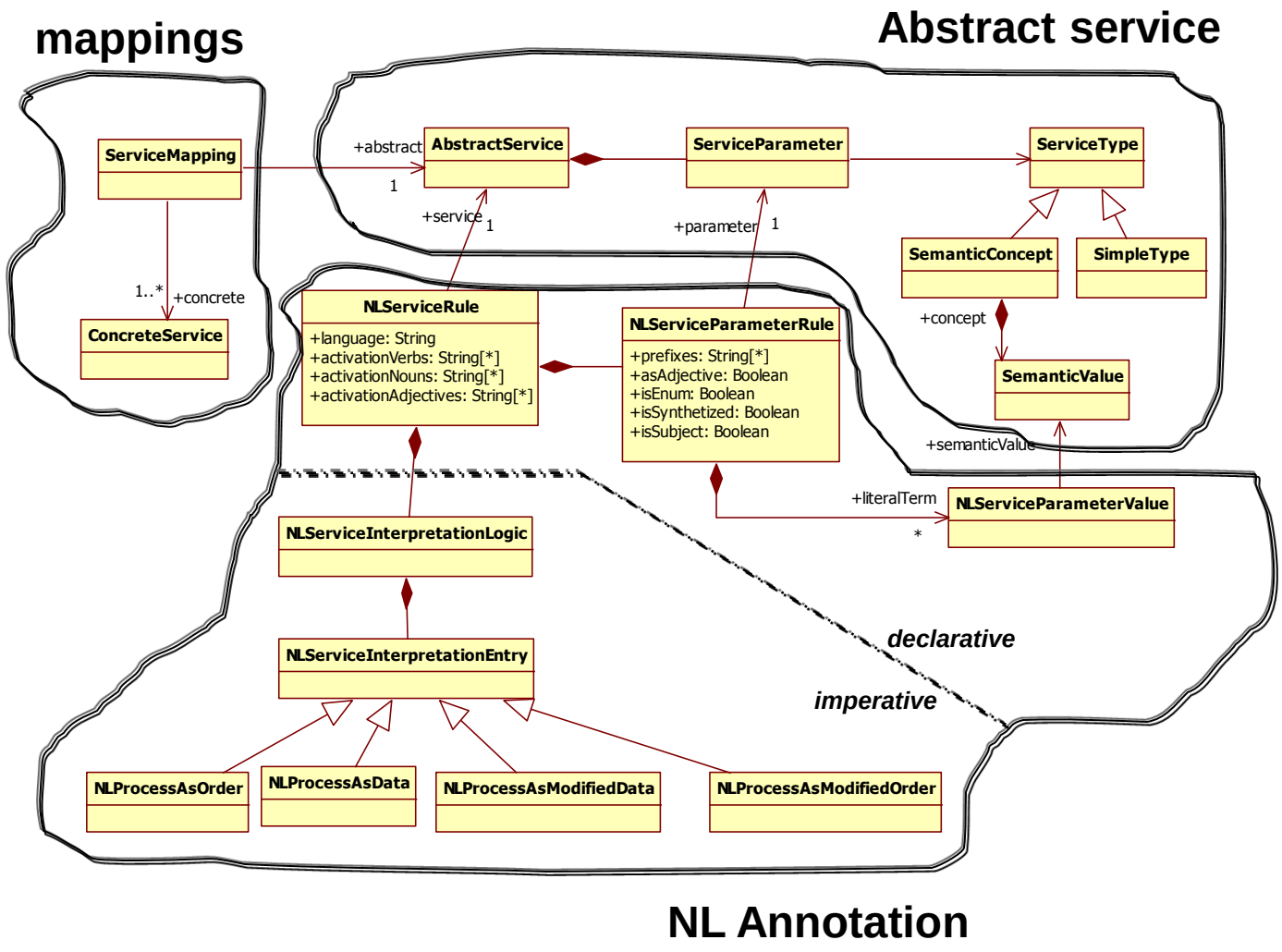

```
sentence: Send by SMS Paris weather translated in Engli
item: Send -> service[verb]: Messaging
item: by -> argprefix
item: SMS -> argvalue: (Messaging) kind="sms"
item: Paris -> argvalue: (Weather) city="'Paris'"
item: weather -> service[noun]: Weather
item: translated -> service[adjective]: Translation
item: in -> argvalue: (Translation) targetLanguage="fre
item: English

*** Data ***
Data 'Weather' (city="'Paris'") (3:5)
ModifiedData 'Translation' (modification="translated",
   text=Data::Weather,targetLanguage="unknown",
   sourceLanguage="french"[preferred]) (3:8)
Order 'Messaging' (kind="sms",message=ModifiedData::Translation) ()
```



Architecture Overview



Design Concepts



Semantics for Service Composition exploiting Reccommendations

Illustration of application:
An assistant to build services

- Found the assembly of components that is useful to realize an idea of service
- Partial (or complete) generation of the mashup

Reccommendation

Urbanism Tools

Mopcom-I Service Composition

Provide the main topic of your service

Emergency

START

Recommendations

Select or deselect items, click on arrows to refine the reccommendations

- EmergencyFriends [related]
  - *SocialNetwork::getEmergencyFriends* [prod]
- Doctor [related]
  - *YellowPages::getInterestPoints('doctor')* [prod]
- Location [related]
  - *Geolocation::getUserGeographicalLocation* [prod]

GENERATE SERVICE

- EmergencyFriends [related]
  - *SocialNetwork::getEmergencyFriends* [prod]
    - Authentification [pre]
      - *ExplicitUserAuthentication::authenticateUser* [cons]
    - Contact [use]
      - *CommunicationControl::requestOutgoingCommunication* [cons]
        - InviteAudio [opt]
          - ***Adaptation::convertTextToSpeech*** [prod]
          - ***MessagingManagement::record*** [prod]
      - *MessagingManagement::sendOutMessage* [cons]
      - *SocialNetwork::postToFriend* [cons]
    - EmergencyFriends [out]
      - *CommunicationControl::requestOutgoingCommunication* [cons]
      - *MessagingManagement::sendOutMessage* [cons]

Functional Code

```
emergencyFriends = SocialNetwork::getEmergencyFriends (user,contact);
inviteAudio = Adaptation::convertTextToSpeech (message);
CommunicationControl::requestOutgoingCommunication (inviteAudio,user,contact);
```

Logical Code

```
emergencyFriends = CommunityPlus::getEmergencyFriends(user,contact) ;
inviteAudio = TextToSpeech::text2speech(message,".wav") ;
ClickToCall::createCall(user,contact,inviteAudio) ;
```

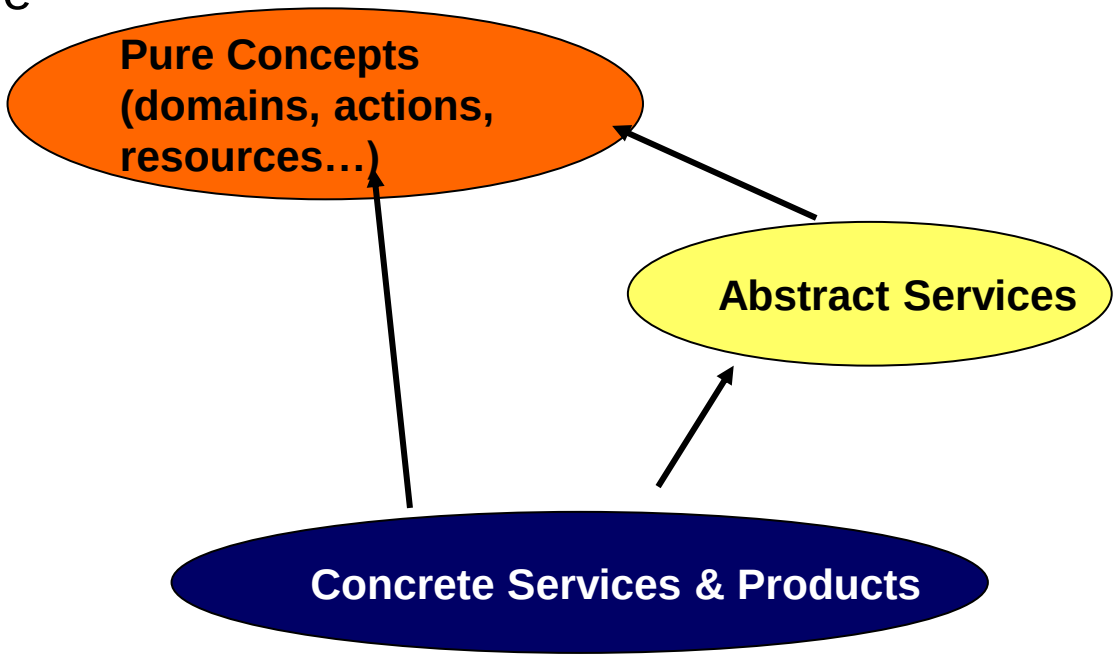
Semantic Usage

- Requirements
  - Various relationships need to be represented:
    - production/consumption of resources,
    - entity substitution,
    - pre/post,
    - outputs/effects
  - Categorization of functionalities (ex: CRUD)
  - Grouping by domain for context resolution

Conclusions

Conclusions

- Three level architecture
  - Concrete Services
  - Abstract Services
  - Pure Concepts



- Meta-modeling and Ontologies are used in complementary way

- Actual work: Tool merging NL, graphical oriented mashup and recommendation.