

Supporting Dynamic Service Composition at Runtime based on End-user Requirements

Eduardo Silva, Luís Ferreira Pires, Marten van Sinderen

Centre for Telematics and Information Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente

November 24th 2009



UNIVERSITY OF TWENTE.

Outline

- 1 Dynamic Service Composition
- 2 User Types
- 3 DynamiCoS Framework
- 4 Conclusions and Future Work

Outline

- 1 Dynamic Service Composition
- 2 User Types
- 3 DynamiCoS Framework
- 4 Conclusions and Future Work

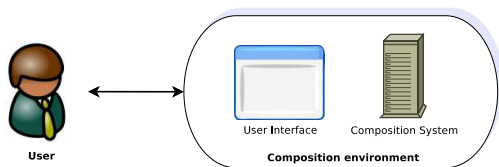
Dynamic Service Composition

Dynamic Service Composition (DSC) is the process of creating of a new service on demand based on a set of requirements that a user has at a given moment.

DSC properties:

- **User is central** to the composition process;
- System performs several **tasks automatically** on behalf of the user (many types of users cannot handle complexity of composition process!);
- Highly suitable for **runtime service composition**.

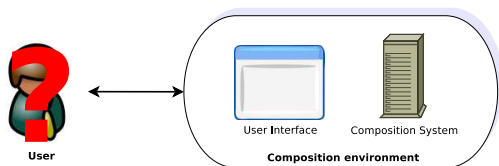
User and Supporting System in DSC environments



- User has to be supported by the system;
- ...But Users do not always have the same characteristics;
- ...This implies that supporting systems have to be shaped according the user being supported!

But how to identify and characterise different users?

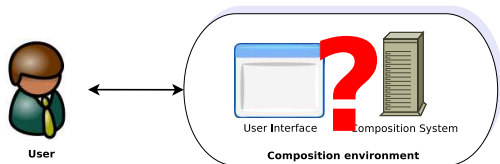
User and Supporting System in DSC environments



- User has to be supported by the system;
- ...But Users do not always have the same characteristics;
- ...This implies that supporting systems have to be shaped according the user being supported!

But how to identify and characterise different users?

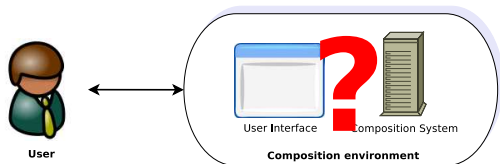
User and Supporting System in DSC environments



- User has to be supported by the system;
- ...But Users do not always have the same characteristics;
- ...This implies that supporting systems have to be shaped according the user being supported!

But how to identify and characterise different users?

User and Supporting System in DSC environments



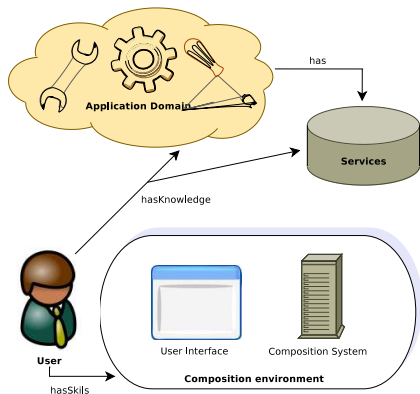
- User has to be supported by the system;
- ...But Users do not always have the same characteristics;
- ...This implies that supporting systems have to be shaped according the user being supported!

But how to identify and characterise different users?

Outline

- 1 Dynamic Service Composition
- 2 User Types**
- 3 DynamiCoS Framework
- 4 Conclusions and Future Work

User Knowledge



- User has (or may have) **Application Domain knowledge**;
- User has (or may have) **skills on the supporting composition environment**.

Types of Users

Type of End-user	Domain Knowledge	Technical Knowledge
<i>Layman</i>	No	No
<i>Domain Expert</i>	Yes	No
<i>Technical Expert</i>	No	Yes
<i>Advanced</i>	Yes	Yes

- Different types of user will require different supporting environments!

We present **DynamiCoS**, which at the moment supports users with Domain Knowledge (*Domain Experts* and *Advanced Users*).

Types of Users

Type of End-user	Domain Knowledge	Technical Knowledge
<i>Layman</i>	No	No
<i>Domain Expert</i>	Yes	No
<i>Technical Expert</i>	No	Yes
<i>Advanced</i>	Yes	Yes

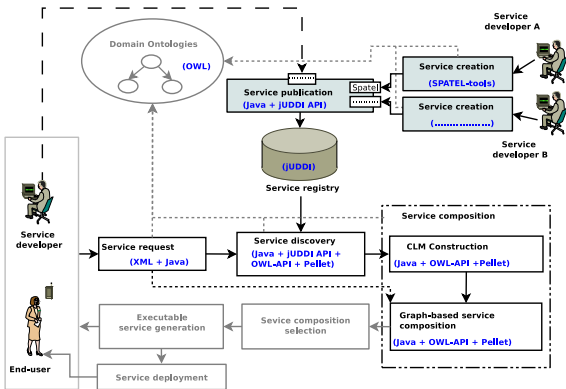
- Different types of user will require different supporting environments!

We present **DynamiCoS**, which at the moment supports users with Domain Knowledge (*Domain Experts* and *Advanced Users*).

Outline

- 1 Dynamic Service Composition
- 2 User Types
- 3 DynamiCoS Framework**
- 4 Conclusions and Future Work

DynamiCoS Framework



- Stakeholders: **End-users** and **Service developers**;
- Two main flows: creation and publication of **new services** and **composition** of existing services;
- Use **ontologies** to automate parts of the composition process.

Running Example

Scenario and Services:

- **E-Health scenario**, aiming at provide support to users with some health problem and requiring assistance;
- Example of **services**: *FindHospital*, which finds the nearest hospital given a location; *FindDoctor*, which finds a doctor given a hospital and a medical speciality; *LocateUser*, which locates a user given his telephone location, *MakeMedicalAppointment*, which makes an appointment between a patient and a doctor of a given hospital, etc.;

Example of Service Request: *"Make a medical appointment in the nearest hospital"*

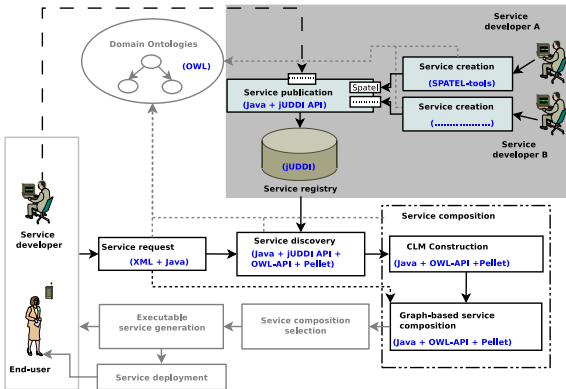
Running Example

Scenario and Services:

- **E-Health scenario**, aiming at provide support to users with some health problem and requiring assistance;
- Example of **services**: *FindHospital*, which finds the nearest hospital given a location; *FindDoctor*, which finds a doctor given a hospital and a medical speciality; *LocateUser*, which locates a user given his telephone location, *MakeMedicalAppointment*, which makes an appointment between a patient and a doctor of a given hospital, etc.;

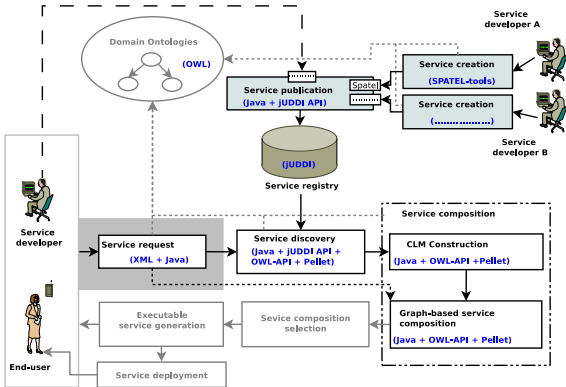
Example of Service Request: *“Make a medical appointment in the nearest hospital”*

Service Creation and Publication



- Services are possibly **created by different developers**, described in **different languages**;
- For each language there must be an **interpreter**;
- **Service representation** in the framework:
 $s = \langle ID, I, O, P, E, G, NF \rangle$.

Service Request



- Allows users to **specify declaratively** the desired service $\rightarrow (G, IOPE, NF)$ of the service;
- $(G, IOPE, NF)$ are **semantic references** to the framework ontologies;
- Service request **interface can vary**, as long as it collects the required information;
- We define a **XML-based representation** of the service request.

Service Request

Running Example

Running example Service Request: *“Make a medical appointment in the nearest hospital”*

Can be translated to the following **service request**:

```
<ServiceRequest>
  <input>IOTypes.owl#CellNumber</input>
  <output>IOTypes.owl#MedicalAppointment</output>
  <goal>Goals.owl#FindLocation</goals>
  <goal>Goals.owl#FindHospital</goals>
  <goal>Goals.owl#FindDoctor</goals>
  <goal>Goals.owl#MedicalAppointment</goals>
</ServiceRequest>
```

Service Request

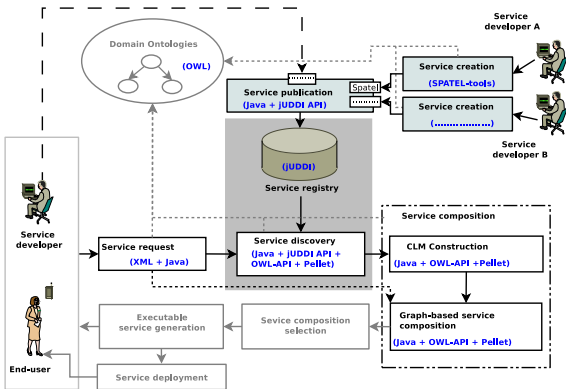
Running Example

Running example Service Request: *“Make a medical appointment in the nearest hospital”*

Can be translated to the following **service request**:

```
<ServiceRequest>
  <input>IOPTypes.owl#CellNumber</input>
  <output>IOPTypes.owl#MedicalAppointment</output>
  <goal>Goals.owl#FindLocation</goals>
  <goal>Goals.owl#FindHospital</goals>
  <goal>Goals.owl#FindDoctor</goals>
  <goal>Goals.owl#MedicalAppointment</goals>
</ServiceRequest>
```

Service Discovery



- Service discovery based on service request parameters ($G, IOPE, NF$);
- Pure Goal-based (G) can be made;
- Partial matches can be returned, e.g.:
RequestedConcept \sqsubseteq *DiscoveredConcept*.

Service Discovery

Running Example

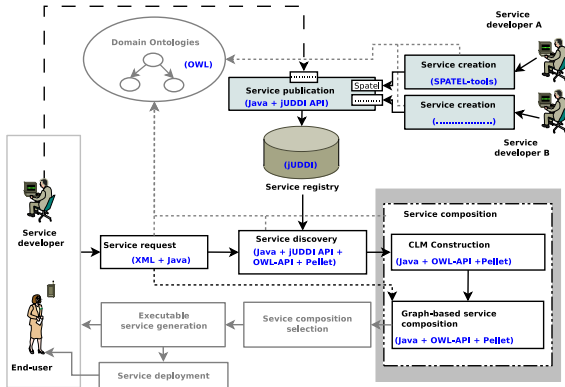
Considering that a **pure goal-based service discovery** is performed, the services that semantically match the goals are retrieved:

```
<goal>Goals.owl#FindLocation</goals>  
<goal>Goals.owl#FindHospital</goals>  
<goal>Goals.owl#FindDoctor</goals>  
<goal>Goals.owl#MedicalAppointment</goals>
```

The following set of **services are discovered**:

Service	Input	Output	Goal
<i>locateUser</i>	IOTypes.owl#CellNumber	IOTypes.owl#Coordinates	Goals.owl#FindLocation
<i>findHospital</i>	IOTypes.owl#Coordinates	Core.owl#Hospital	Goals.owl#FindHospital
<i>findDoctor</i>	IOTypes.owl#MedSpeciality Core.owl#MedicalPlaces	Core.owl#Physician	Goals.owl#FindDoctor
<i>makeMedAppointment</i>	Core.owl#Physician Core.owl#Patient	IOTypes.owl#MedicalAppointment	Goals.owl#MedicalAppointment

Service Composition



- Service **composition** represented as a graph: $G = (N, E)$;
- N are **services**, E **coupling** between services;
- Coupling ($O \rightarrow I$) are **semantic compositions** or **causal links** ($\equiv, \sqsubseteq, \sqsupseteq, \perp$);
- **Two steps composition process**: 1) CLM (Causal Link Matrix) creation; 2) Graph composition algorithm.

Service Composition

Running Example

Discovered Services:

Service	Input	Output
<i>locateUser</i> (S1)	IOTypes.owl#CellNumber	IOTypes.owl#Coordinates
<i>findHospital</i> (S2)	IOTypes.owl#Coordinates	Core.owl#Hospital
<i>findDoctor</i> (S3)	IOTypes.owl#MedSpeciality Core.owl#MedicalPlaces	Core.owl#Physician
<i>makeMedAppointment</i> (S4)	Core.owl#Physician Core.owl#Patient	IOTypes.owl#MedAppoint

Created CLM:

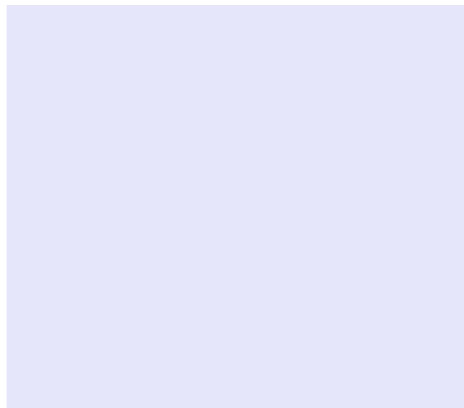
	cellNumber	Coordinates	MedSpeciality	MedPlaces	Patient	Physician	MedAppoint
CellNumber	0	S1,≡	0	0	0	0	0
Coordinates	0	0	0	S2,⊆	0	0	0
MedSpeciality	0	0	0	0	0	S3, ≡	0
MedPlaces	0	0	0	0	0	S3, ≡	0
Patient	0	0	0	0	0	0	S4,≡
Physician	0	0	0	0	0	0	S4,≡

Service Composition

Running Example

Service Composition Process:

Cellnumber



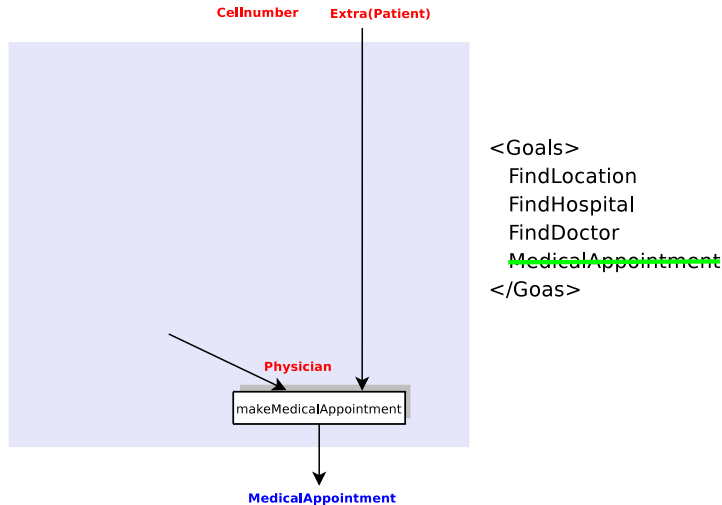
```
<Goals>  
  FindLocation  
  FindHospital  
  FindDoctor  
  MedicalAppointment  
</Goals>
```

MedicalAppointment

Service Composition

Running Example

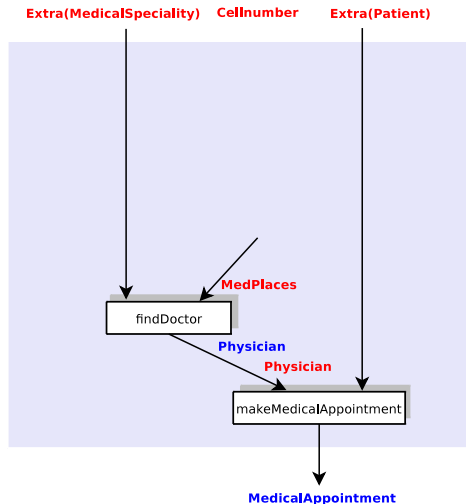
Service Composition Process:



Service Composition

Running Example

Service Composition Process:

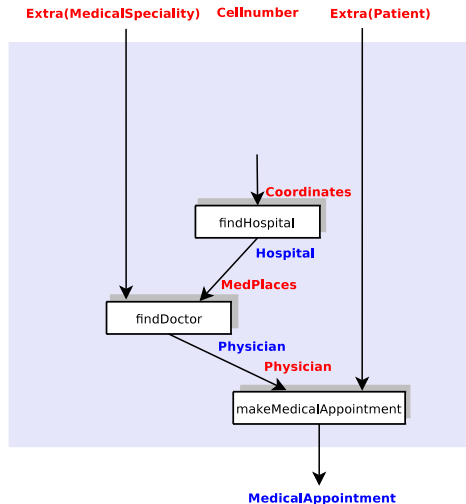


```
<Goals>  
  FindLocation  
  FindHospital  
  FindDoctor  
  MedicalAppointment  
</Goals>
```

Service Composition

Running Example

Service Composition Process:

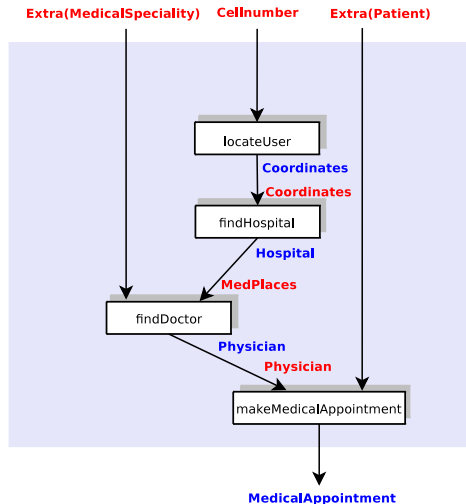


```
<Goals>  
FindLocation  
FindHospital  
FindDoctor  
MedicalAppointment  
</Goals>
```

Service Composition

Running Example

Service Composition Process:



<Goals>

~~FindLocation~~

~~FindHospital~~

~~FindDoctor~~

MedicalAppointment

</Goals>

<http://dynamics.sourceforge.net>

(soon prototype code, web interface and use case examples)

Outline

- 1 Dynamic Service Composition
- 2 User Types
- 3 DynamiCoS Framework
- 4 Conclusions and Future Work**

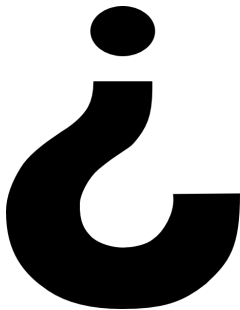
Conclusions

- Dynamic Service Composition is required to achieve **user-centric service delivery**;
- It is difficult to generalise supporting environments since **users are different**;
- We propose a **classification of user types**, characterising users according to their domain and technical knowledge;
- We present **DynamiCoS**, which at the moment allows to support users with domain knowledge.

Future Work

- **More personalisation** by using user context and preferences;
- Extend DynamiCoS to support other types of users. This can be achieved by including **guiding mechanism** in the composition process, by interacting with the user to capture his requirements and deliver necessary knowledge so that he can make a decision;
- Create **better User Interfaces (UI)**.

Questions?



web: www.cs.utwente.nl/goncalvesem

email: [e.m.g.silva\[at\]cs.utwente.nl](mailto:e.m.g.silva@cs.utwente.nl)

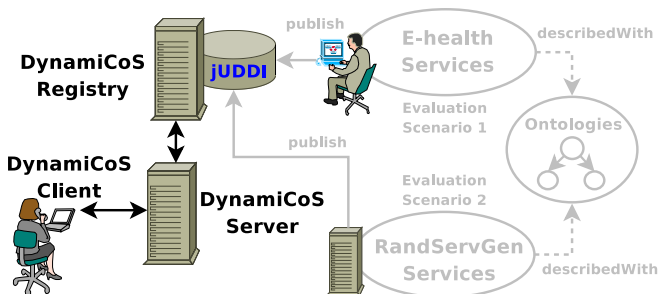
Outline

5 Evaluation

6 Prototype

Performance Evaluation

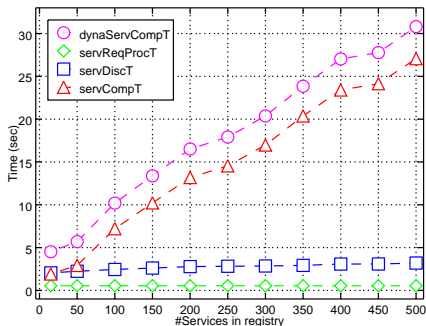
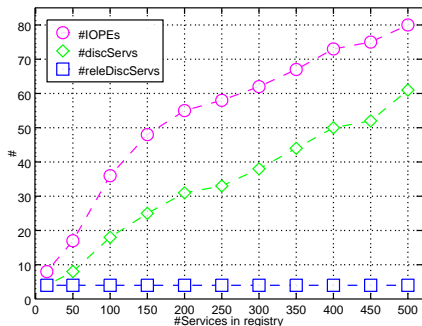
Evaluation Setup



We have defined **manually** a set of services (in the E-Health domain), and then we have used the same ontologies to derive **automatically** a set of services to test the behaviour of the approach when more services were available (even though these services are not meaningful).

Performance Evaluation

Evaluation Results



- Number of *IOPEs* and discovered services increase (→ more services are discovered). Relevant discovered services are constant (→ many discovered services are “residual”);
- Service request time is constant (→ the same service request). Service discovery time increases slightly (→ more services are discovered). Composition time increases considerably (→ many services/*IOPEs* handled in the semantic reasoning).

Outline

5 Evaluation

6 Prototype

Service Composition

Running Example

