Report June 26th
Partial Coverage Descriptor

This document proposes and describes an extension to the former Partial Coverage Descriptor (PCD) definition to include SLAs. Additionally, some question about the former implementation and rewriting are posed.

1. Former PCD definition

The PCD definition proposed in [2] consists in a tuple $\langle S, h, \varphi, G, Def, has\_opt \rangle$, where:

S is a concrete service;

h are mapping between inner terms in the concrete service definition;

φ are mapping between terms from the abstract composition to terms in the concrete service definition;

G is a set of abstract services names and quality constraints covered by S;

Def is a set of quality constraints that are not covered by S alone; and

has_opt is a boolean flag to indicate that some abstract service in the definition of S has been used in G and has an optional parameter.

2. PCD extension

Based on this former PCD definition, the extension to incorporate SLA measures is a tuple $\langle S, SLA_s, Qpref, h, dep, \varphi, G, Def, has\_opt \rangle$, where:

S is a concrete service;

SLA$_s$ is a set of SLA clauses (key-value) associated to the service S;

Qpref is a set of user quality preferences (key-value)from the query definition;

h are mapping between inner terms in the concrete service definition;

dep is a set of abstract services in the definition of S that are inter-dependents;

φ are mapping between terms from the abstract composition to terms in the concrete service definition;

G is a set of abstract services names and quality constraints covered by S;

Def is a set of quality constraints that are not covered by S alone; and

has_opt is a boolean flag to indicate that some abstract service in the definition of S has been used in G and has an optional parameter.

3. Remarks and Questions

In the proposed extension three new parameters are added to the tuple: SLA_s, $Qpref$ and *dep*. The idea behind the SLA_s and $Qpref$ parameters is to express the SLA clauses and the user quality preferences.

Considering the *dep* parameter, the purpose is to express dependencies between abstract services in the concrete service definition. For example, consider the query:

$Q(x) :- a1(x,y), a2(y,z), a3(z,w)$

And the concrete services:

$S1(a,b) :- a1(a,b)$

$S2(a,b) :- a2(a,c), a4(c,d), a3(d,y)$

The concrete service $S2$ can solve $a2$ and $a3$, but on its definition there is another service $a4$ that should be execute after $a2$. In the current implementation this kind of dependency is not covered, and while trying to execute this example an error occurs.

While testing theses dependencies, I found some interesting that I did not understand why. Consider that we have a query:

$Q(x) :- e1(x,y), e2(y,x), e3(x,z)$

And concrete services:

$V1(a,b) :- e1(a,b)$

$V2(a,b) :- e2(a,b)$

$V3(a,b) :- e3(a,b)$

As expected the rewriting result is $V1(x,y), V2(y,x), V3(x,z)$. However, if we proceed an small change in the definition of $V2$ to:

$V2(a,b) :- e2(a,b), e3(a,b)$

Now this service can solve two abstract services in the query. I was expecting as result the rewriting:

$V1(x,y), V2(y,x)$

But the results were:

$V1(x,y), V2(y,x), V2(x,z)$

$V1(x,y), V2(y,x), V3(x,z)$

I would like to understand why because in my opinion these two answer are not good: the first one uses two times $V2$, and the second uses together $V2$ and $V3$, but both of them solve the same abstract service $e3$. Additionally, I also tried to add a dependency between $e2$ and $e3$ in the definition of $V2$ ($V2(a,b) :- e2(a,c), e3(c,b)$), and I also got an error.

Considering the constraints (Q) in the abstract composition definition, I think they should used to express the user's quality preferences that should by matched in the SLAs. In the current implementation, until now, I could find where and how theses constraints are used. I know that they are used in the Algorithm 3 [1] while combining PCDs to rewrite the query, but there is no example about how to use them. Also, Def parameter is always empty what proves that there is no constraints.

Another point is about the *has_opt* flag. There is no example to show how to use it. If I understood well this parameter will indicate that an abstract service in the concrete service definition has an optional parameter on its definition. So, I tried to run the follow example: query $Q(x) :- a1(x,y), a2(y,z)$ and the concrete services: $S1(a,b) :- a1(a,b)$; $S2(a,b) :- a2(a,b,c)$. c would be the optional parameter in the service definition, but this query can not be processed, an error occurs.

Bibliography

[1] Cheikh Ba, Umberto Costa, Mirian Halfeld Ferrari, Rémy Ferre, Martin A. Musicante, Veronika Peralta, and Sophie Robert. Preference-Driven Refinement of Service Compositions. In *CLOSER (4th International Conference on Cloud Computing and Services Science)*, Proceedings of CLOSER 2014, page 8 pages, Barcelona, Spain, April 2014. POSTER presentation.

[2] UmbertoS. Costa, MirianHalfeld Ferrari, MartinA. Musicante, and Sophie Robert. Automatic refinement of service compositions. In Florian Daniel, Peter Dolog, and Qing Li, editors, *Web Engineering*, volume 7977 of *Lecture Notes in Computer Science*, pages 400–407. Springer Berlin Heidelberg, 2013.