Self-composition of services with chemical reactions

Francesco De Angelis
University of Geneva, ISS
Route de Drize 7 Carouge
SWITZERLAND
francesco.deangelis@unige.ch

Jose Luis Fernandez-Marquez
University of Geneva, ISS
Route de Drize 7 Carouge
SWITZERLAND
joseluis.fernandez@unige.ch

Giovanna Di Marzo Serugendo
University of Geneva, ISS
Route de Drize 7 Carouge
SWITZERLAND
giovanna.dimarzo@unige.ch

Keywords

Self-composition, chemical reactions, services, context-awareness, dynamic environment

1. INTRODUCTION

Next generation of socio-technical infrastructures will be characterized by the presence of complex networks of pervasive systems, composed of thousands of heterogeneous devices consuming and producing high-volumes of interdependent data. Smart-cities represent an example of these future digital scenarios: by using wide area mobile ad-hoc networks (MANETs), data will be shared among applications placed on cars or running on several devices such as smartphones, tablets, public displays and sensors placed at the edges of the roads; moreover, all these devices will access traditional remote web-services. Smart-cities depict the emergence of new open-infrastructure pervasive systems, where scalability and dependability will be achieved by developing and adapting (at run-time) applications through compositions of customized services.

The static character of traditional approaches for composition of services, such as orchestration and choreography, has been recently challenged by so-called dynamic service composition approaches, involving semantic relations [1], or AI planning techniques to generate process automatically based on the specification of a problem [4]. All these approaches turn to be unfeasible for being adopted in future pervasive systems because of their restricted scalability due to the centralization of the composition process, limited support for context-awareness and slow reactivity to sudden appearance or disappearance of services.

To tackle these limitations this paper introduces a new approach for service composition named self-composition. Compositions are realized in a pervasive ecosystem [5] without the presence of coordination units, through incremental distributed light-weighted interactions among services, inspired by chemical reactions bindings. As well as improving scalability by adopting a distributed bio-inspired model for interactions, compositions can be also empowered by the use of contextual information gathered by sensors and inserted in the ecosystem.

SAC'14 March 24-28, 2014, Gyeongju, Korea.

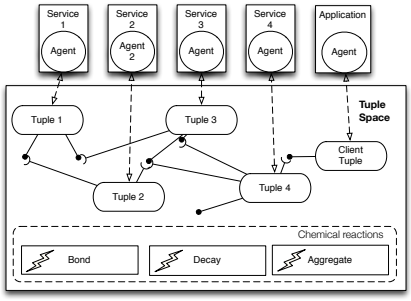

Figure 1: Reference model. Dotted lines represent interaction between agents and their tuple in the tuple space; solid lines represent interactions among tuples.

2. REFERENCE MODEL

The model considered is an abstraction of the SAPERE active tuple space model [2, 5] inspired by chemical reactions; the system is composed of four entities: *tuples*, *chemical reactions*, *agents* and *services/applications* (Figure 1).
Tuples: passive entities located in a shared container named tuple space. Tuples are vectors of properties *(name=value)* used to describe and represent services, applications and contextual information.
Chemical reactions: functions defined on the set of tuples defining rules used to manipulate, update and delete them. They are provide the system with an automatic way for tuples interactions and they deliver notifications to the agents associated with tuples being modified.
Agents: external active entities represented in the tuple space by a tuple implementing the interfaces between services/applications and the tuple space. An agent interacts with the tuple space updating-deleting tuples and receiving a notification each time an interaction is performed on its tuple by a chemical reaction.
Services: entities producing some data as result of a computation performed on data passed as input. Services that want to request or provide information create an agent used as an interface to exchange data within the tuple space. Services can be represented by using a precondition-postcondition notation. For example a tuple $T_1 = (city =?, weather =!)$ represents a service W producing a result of type $weather$ (i.e., weather forecast) given an input value of type $city$.

We define two basic chemical reactions: *Bond* and *Decay*.
Bond: A bond is a relationship between two distinct tuples containing the same property names and it is realized by inserting a question mark "?" as property value. For example, a bond will be created between service W injecting $T_1 = (city =?, weather =!)$ and $T_2 = (city = \text{“}NewYork\text{”})$ because of property *city*. When a bond is established, an event of activation is delivered to the agent exhibiting the value "?" in one of the properties of its tuple. In this case, the service W introduced before will be invoked, generating the weather forecast for New York. This will result in the generation of a tuple, e.g. $T_3 = (city = \text{“}NewYork\text{”}, weather = \text{“}sunny\text{”})$.

Decay. It provides a mechanism to free resources by removing tuples from the space.

3. SELF-COMPOSITION ALGORITHMS

Compositions are based on expected input and output types. A composition spontaneously arises when the output type of a service matches the input type of another service.

3.1 Executing all services

A service request is activated by injecting a tuple containing the initial input values in the system, specifying the expected output type. This input causes all services able to process that value to execute and to possibly each produce an output value. This will prompt other services to react and execute, and so on until one of those outputs matches the expected output type. This algorithm does not require any plan of service execution and it produces all possible compositions that can be generated starting from a set of initial values of types $P_1, \ldots, P_n$.

Service request: a client application encodes a service request by creating a tuple $T = (P_1 = \text{“}p1\text{”}, \ldots, P_n = \text{“}p_n\text{”}, R =?)$ containing the list of initial input values p_i and the question mark "?" for the expected output of type R.

Composition execution: every service agent satisfying a precondition property of tuple T is activated by means of a bond: it reads all values contained in T, invokes the service and inserts the result values of type $R_1, \ldots, R_n$ in a new tuple. This in turn activates other services. At the end of this process, if the composition can be computed by using a set of existing services, a tuple containing a value of the desired type R appears in the tuple space. This tuple will then bond with the ? in the original request tuple T.

3.2 Designing all compositions

This approach permanently (re-)builds all possible compositions designs among agents, depending on arriving or departing services, independently of queries. All services contribute in maintaining a graph of interdependencies [3]; through progressive bonds every service identifies the set $\mathcal{P}$ of all the input types that, if satisfied, will eventually trigger its own execution. This set is then used by the client agent in order to select a sequence of services to invoke to produce a specific composition.

Design process: A client agent S_n injects a request indicating an expected output of type R, which will then bond with all services able to provide output R. It also obtains a set of preconditions that trigger those services. Progressively, a sequence of service names $\mathcal{L} = (S_1, \ldots, S_n)$ is built, such that: service S_1 provides input values of type $P_1, \ldots, P_n$; the postcondition (output values) of S_i is the precondition (input values) of S_{i+1}; and service S_n provides output value of type R. In other terms, $\mathcal{L}$ is a sequence of services that generates type R starting from input values of type $P_1, \ldots, P_n$.

Composition execution: The client agent inserts the tuple $T = (P_1 = \text{“}p1\text{”}, \ldots, P_n = \text{“}p_n\text{”}, \mathcal{L}, R =?)$, which actually starts the execution process. Only services that are part of $\mathcal{L}$ will be progressively triggered through successive bonds. When this process stops, the final value of type R appears in the tuple space.

Services updates: The graph of interdependencies is rebuilt at prefixed interval times by using the decay chemical reaction.

Additional details concerning the proposed algorithms can be found in [3].

4. CONCLUSION

We defined a model and two algorithms for self-composition of services by defining a pervasive ecosystem built on top of a chemical tuple space. The proposed approaches present several benefits for scalability and context-awareness: the composition is automatically produced with no effort by the application's developer in terms of synchronization and execution planning; the computation is implicitly distributed and it involves the parallel execution of services as soon as tuples with intermediary results are generated; no coordination entities is needed. The shared space provides a natural way for caching intermediary and temporary values, preventing identical service requests from being executed several times. Our approaches are then particularly suitable for context-aware applications, where components (services and applications) interact by dealing with contextual information injected in the tuple space by sensors.

Acknowledgment

This work has been supported by the EU-FP7-FET Proactive project SAPERE Self-aware Pervasive Service Ecosystems, under contract no.256873.

5. REFERENCES

[1] M. Beek, A. Bucchiarone, and S. Gnesi. A survey on service composition approaches: From industrial standards to formal methods. In *Technical Report 2006TR-15, Istituto*, pages 15–20. IEEE CS Press, 2006.

[2] G. Castelli, M. Mamei, A. Rosi, and F. Zambonelli. Pervasive middleware goes social: The sapere approach. In *Proceedings of the 2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops*, SASOW '11, pages 9–14, Washington, DC, USA, 2011. IEEE Computer Society.

[3] F. De Angelis, J. L. Fernandez-Marquez, and G. Di Marzo Serugendo. Self-composition of services. Technical Report TR.WP2.2013.2, SAPERE Project, 2013.

[4] Z. Wu, A. Ranabahu, K. Gomadam, A. Sheth, and J. Miller. Automatic composition of semantic web services using process and data mediation. In *Proc. of the 9th Intl. Conf. on Enterprise Information Systems*, pages 453–461, 2007.

[5] F. Zambonelli et al. Self-aware pervasive service ecosystems. *Procedia Computer Science*, 7:197 – 199, 2011.