# Towards Automated Web Service Composition with the Abductive Event Calculus

Onur Aydın[1] Nihan Kesim Cicekli[2] Ilyas Cicekli[3]

[1]Microsoft Corporation, Seattle, U.S.A.
[2]Department of Computer Engineering, METU, Ankara , Turkey
[3]Department of Computer Engineering, Bilkent University, Ankara, Turkey
*onura@microsoft.com*, *nihan@ceng.metu.edu.tr, ilyas@cs.bilkent.edu.tr*

In this paper, the use of a logic programming framework, the event calculus [3], is discussed in the automated composition of web services. The web service discovery problem is beyond of the scope of the paper. Our goal is to show that the event calculus can be used for both definitions of web services composition. That is, it can be used to generate a composite process as the output of planning. It can also be used to define a generic composition and produce a user specific composition (plan) according to the user constraints. Abductive planning of event calculus is used to show that when atomic services are available, composition of services that would yield the desired effect is possible. An abductive planner implementation of the event calculus [4] is extended to be used for plan generation.

**Event Calculus and Abductive Event Calculus**

Event calculus [3] is a logical formalism which is used with domains where events affect and change the world. The formulation of the event calculus is defined in first order predicate calculus. There are actions and effected fluents. Fluents are changing their valuations according to effect axioms defined in the theory of the problem domain. Each event calculus theory is composed of axioms. The axioms that define whether a fluent holds starting from the initial state are as follows.

$HoldsAt(F, T) \leftarrow Initially(F) \wedge \neg Clipped(T_0, F, T)$

$HoldsAt(\neg F, T) \leftarrow Initially(\neg F) \wedge \neg Declipped(T_0, F, T)$

Axioms below are used to deduce whether a fluent holds or not at a specific time.

$HoldsAt(F, T) \leftarrow Happens(E, T_1), Initiates(E, F, T_1), T_1 < T , \neg Clipped(T_1, F, T)$

$HoldsAt(\neg F, T) \leftarrow Happens(E, T_1), Terminates(E, F, T_1), T_1 < T, \neg Declipped(T_1, F, T)$

The predicate *Clipped* defines a time frame for a fluent that is overlapping with the time of an event which terminates or releases this fluent. Similarly *Declipped* defines a time frame for a fluent which overlaps with the time of an event that initiates or releases this fluent.

Abduction is logically the inverse of deduction. It is used over the event calculus axioms to obtain partially ordered sets of events. Abduction is handled by a second order logical prover which is defined as an abductive theorem prover (ATP) in [4]. ATP tries to solve the goal list by proving the elements one by one. During the resolution, abducible predicates, < (temporal ordering) and *Happens*, are stored in a residue to keep the record of the narrative. The narrative is a sequence of time-stamped events, and the residue keeping a record of the narrative is the plan.

**Web Services Composition with Abductive Planning**

The event calculus can be used for planning as it is theoretically explained in [4]. The planning problem in the event calculus is formulated in simple terms as follows: Given the domain knowledge (i.e. a conjunction of *initiates, terminates*), the Event Calculus axioms (i.e. *HoldsAt* ) and a goal state (e.g. *HoldsAt*(*f,t*)), the abductive theorem prover generates the plan which is a conjunction of *Happens* and temporal ordering predicates. ATP returns a valid sequence of time stamped events that leads to the resulting goal. Multiple solutions are thought to be as different braches of a more general plan and they are obtained with the help of backtracking.

In the event calculus framework, the web services are modeled as events with input and output parameters. For instance, a web service, which returns the availability of a flight between two locations, can be described as:

*Happens*(*getFlights*(*Orgn, Dest, FlDate, FNL*), $T_1$, $T_1$) ←
*Ex_getFlights*(*Orgn, Dest, FlDate, FNL*).

The predicate *Ex_getFlights* is used as a precondition for the event and it is invoked anytime it is added to the plan to populate the input parameters.

It is also possible to create generic compositions in the event calculus. ATP can then be used to generate a plan which corresponds to the user specific execution of the composite service. Composite services correspond to compound events in the Event Calculus [2]. An OWL-S to event calculus translation scheme is presented to show that OWL-S composition constructs can be expressed as event calculus axioms [1].

**Conclusions**

The Event Calculus framework can be used for the solution of web service composition problem. When a goal situation is given, the event calculus can find proper plans as web service compositions with the use of abduction technique. It is possible that the solutions that are generated by the event calculus can be compiled into a graph like composition for the satisfaction of the goal situation [1]. The Event Calculus can also be used to create generic compositions and ATP can be used to generate a plan which corresponds to the user specific execution of the composite service.

As a future work, the results expressed in this paper will be implemented in a real web environment. Common structures of compositions will be expressed as *meta* event calculus constructs. Another improvement might be on queries which are known a priori for the compositions. Queries can be entered in a natural language and then translated into the event calculus goals.

**References**


1. Aydin, O., Automated web service composition with the event calculus, M.S. Thesis, Dept. of Computer Engineering, METU, Ankara, 2005.
2. Cicekli N. K., Cicekli I. Formalizing the specification and execution of workflows using the event calculus, to appear in Information Sciences.
3. Kowalski R. A., Sergot M. J.A Logic-Based Calculus of Events. New Generation Computing, Vol. 4(1), pp. 67--95, 1986.
4. Shanahan M.P. An abductive event calculus planner. Journal of Logic Programming, Vol. 44(1-3), pp. 207--240, July 2000.