Chapter 49
Semantic Web Service Automatic Composition Based on Discrete Event Calculus

Kai Nie, Houxiang Wang and Jiao He

Abstract A semantic Web service automatic composition method based on discrete Event Calculus is proposed aiming at the issues of service AI planning composition such as large number of services and the confine of sequence composition process. Firstly, the extension of EC to DEC is present. Then the eight basic semantic Web service composition processes and their IOPE are modeled based on the actions, fluent and axioms of DEC. The service composition process is divided into two steps, abstract service planning and instance execution. And the service automatic composition framework is introduced. Also the abduction DEC planning method and semantic matching method of instance execution are given. The comparison indicate the superiority of this method: it solves the confine of sequence composition process of classic AI planning composition method with Event Calculus' presentation of compound action, concurrent action, continuous action, knowledge of the agent and the predicate number of the DEC is much smaller than EC which speed the service discovering and composition.

Keywords Semantic web service · Automatic service composition · Intelligent planning · Discrete event calculus · Abduction

49.1 Introduction

Emerging semantic Web services standards as OWL-S [1] enrich Web service standards like WSDL and BPEL4WS with rich semantic annotations to facilitate flexible dynamic web services discovery, invocation and and composition. The semantic

K. Nie (✉) · H. Wang · J. He
College of Electronic Engineering, Naval University of Engineering,
No 717, Jiefang Dadao Road 430033 Wuhan, People's Republic of China
e-mail: 1999104133@163.com

description of function interface and behaviour of Web service can be provided by OWL-S, but it does not contain automated reasoning mechanism and does not support automatic service discovery and composition of semantic Web services. The key technology of automatic service composition is using semantic match and automated reasoning mechanism. The automatic semantic Web service composition methods contain method based on workflow, method based on intelligent planning, method based on formal description and so on [2]. The intelligent planning method shortening as AI is very popular and mature in Web service composition and can be implemented in some tools such as JSHOP and Prolog. The AI composition methods contains Hierarchy Task Network (HTN) [3], Situation Calculus [4], PDDL [5], dynamic, description logic [6], theorem prover [7] and so on. But they have two shortcomings: large number of services and the confine of sequence composition process.

Event Calculus is one of the convenient techniques for the automated composition of semantic web services. The Event Calculus (EC) is a temporal formalism designed to model and reason about scenarios described as a set of events whose occurrences have the effect of starting or terminating the validity of determined properties of the world. The one that will be used in this paper is the one defined by Shanahan [8]. The Event Calculus is based on first-order predicate calculus, and is capable of representing actions with indirect effects, actions with non-deterministic effects, compound actions, concurrent actions, and continuous change. Agarwal [9] presents a classification of existing solutions into four categories (approaches): interleaved, monolithic, staged and template-based composition and execution. Okutan [10] proposes the application of the abductive Event Calculus to the web service composition and execution problem about the interleaved and template-based type. Ozorhan [11] presents a monolithic approach to automated web service composition and execution problem based on Event Calculus. But they all have large numbers of predicates and the compose speed of them is slow. The discrete Event Calculus (DEC) has been proven to be logically equivalent with EC, when the domain of time points is limited to integers [12]. The DEC can eliminate the triply quantified axioms that lead to an explosion of the number of predicates.

In this paper, the Abductive Planning of the discrete Event Calculus is used to show that when atomic services are available. The service composition process is divided into two steps, abstract service planning and instance execution. And the service automatic composition framework is introduced and semantic matching method of instance execution is given.

49.2 Discrete Event Calculus

49.2.1 Event Calculus

The Event Calculus is based on first-order predicate calculus, and is capable of representing a variety of phenomena, including actions with indirect effects, actions with non-deterministic effects, compound actions, concurrent actions, and

Table 49.1 Event calculus predicates in classical logic

Predicate	Meaning
Happens (E, T)	Event E occurs at time T
Initially$_P$ (F)	Fluent F holds from time 0
Initially$_N$ (F)	Fluent F does not hold from time 0
Holds At (F, T)	Fluent F holds at time T
Initiates (E, F, T)	Event E initiates fluent F at time T
Terminates (E, F, T)	Event E terminates fluent F at time T
Clipped (F, T_0, T_1)	Fluent F is terminated some time in the interval [T_0 T_1]
Declipped (F, T_0, T_1)	Fluent F is initiated some time in the interval [T_0 T_1]
Releases (E, F, T)	Fluent F is not constrained by inertia after event E at time T

continuous change. The basic ontology of the Event Calculus comprises actions or events (or rather action or event types), fluents and time points. Table 49.1 introduces the language elements of the Event Calculus.

The axioms of the Event Calculus relating the various predicates together are as follows:

$$holdsAt(F,T) \leftarrow initially_P(F) \wedge \neg clipped(F,0,T) \tag{49.1}$$

$$\begin{aligned} holdsAt(F,T) \leftarrow {} & happens(E,T_0) \wedge initiates(E,F,T_0) \wedge T_0 < T \\ & \wedge \neg clipped(F,T_0,T) \end{aligned} \tag{49.2}$$

$$\begin{aligned} clipped(F,T_0,T_1) \leftrightarrow {} & \exists E,T\, happens(E,T) \wedge T_0 \le T \wedge T \le T_1 \\ & \wedge terminates(E,T,F) \end{aligned} \tag{49.3}$$

$$\begin{aligned} declipped(F,T_0,T_1) \leftrightarrow {} & \exists E,T\, happens(E,T) \wedge T_0 \le T \wedge T \le T_1 \\ & \wedge initiates(E,F,T) \end{aligned} \tag{49.4}$$

49.2.2 DEC

The axioms of DEC utilize a subset of the EC elements (Table 49.1), that is happens, holds At, initiates and terminates. The axioms that determine when a fluent holds, are defined as follows:

$$holdsAt(F,T+1) \leftarrow happens(E,T) \wedge initiates(E,F,T) \tag{49.5}$$

$$holdsAt(F,T+1) \leftarrow holdsAt(F,T) \wedge \neg\exists E\; happens(E,T) \wedge terminates(E,F,T) \tag{49.6}$$

Compared to EC, DEC axioms are defined over successive timepoints. Additionally, the DEC axioms are quantified over a single time point variable. Therefore the number of predicates is substantially smaller than EC. Axioms (49.6), however, contain the existentially quantified variable E. Each of these axioms will be transformed into $2^{|\varepsilon|}$clauses. Moreover, each clause will contain a large number of disjunctions. To overcome the creation of $2^{|\varepsilon|}$clauses, we can employ the technique of subformula renaming, as it is used in [12]. According to this technique, the subformula $happens(E,T) \wedge initiates(E,F,T)$ in (49.5), is replaced by a utility predicate that applies over the same variables, e.g. $startAt(E,F,T)$. A corresponding utility formula, i.e. $startAt(E,F,T) \leftrightarrow happens(E,T) \wedge initiates(E,F,T)$, is then added to the knowledge base.

In order to eliminate the existential quantification and reduce further the number of variables, we adopt a similar representation as in [12], where the arguments of initiation and termination predicates are only defined in terms of fluent and time points—represented by the predicates initiated At and terminated At respectively. As a result, the domain-independent axioms of DEC presented above are universally quantified over fluents and time points. The axioms that determine when a fluent holds are thus defined as follows:

$$holdsAt(F,T+1) \leftarrow initiatedAt(F,T) \tag{49.7}$$

$$holdsAt(F,T+1) \leftarrow holdsAt(F,T) \wedge \neg terminatedAt(F,T) \tag{49.8}$$

49.3 Service Automatic Composition Method Based on DEC

49.3.1 The Basic Service Processes Translation to DEC

In OWL-S the composite processes are composed of sub-processes and specify constraints on the ordering and conditional execution of the sub-processes. The minimal set of control constructs according to includes Sequence, Split, Split +Join, Any-Order, Choice, If–Then-Else, Repeat-While and Repeat-Until. These constructs are translated automatically into compound events in our framework.

Translations of the sequence control construct:

$$\begin{aligned} &axiom(happens(pSequenceExample(Inputs, Outputs), T_0, T_m), \\ &[happens(pProcess_1(Inputs_1, Outputs_1), T_1, T_2), happens(pProcess_2(Inputs_2, Outputs_2), T_3, T_4), \ldots, \\ &happens\big(pProcess_n(Inputs_n, Outputs_n), T_x, T_y\big), before(T_0, T_1), before(T_1, T_3), \ldots, before(T_x, T_m)]) \\ &Input_i \subseteq Inputs \cup \bigcup_{j=1}^{i-1} Outputs_j for\ i = 1, \ldots, n\ Outputs \subseteq \bigcup_{i=1}^{n} Outputs_i \end{aligned}$$

Translations of the Split control construct:

$$\begin{aligned}&axiom(happens(pSplitExample(Inputs, Outputs), T_0, T_m),\\ &[happens(pProcess_1(Inputs_1, Outputs_1), T_1, T_2), happens(pProcess_2(Inputs_2, Outputs_2), T_3, T_4), \ldots,\\ &happens(pProcess_n(Inputs_n, Outputs_n), T_x, T_y), before(T_0, T_1), before(T_0, T_3), \ldots, before(T_0, T_x)])\end{aligned}$$

$$Inputs_i \subseteq Inputs\ for\ i = 1, 2, \ldots, n\ Outputs \subseteq \bigcup_{j=1}^{n} Outputs_j$$

Translations of the Split–Join control construct:

$$\begin{aligned}&axiom(happens(pSplitJoinExample(Inputs, Outputs), T_0, T_m),\\ &[happens(pProcess_1(Inputs_1, Outputs_1), T_1, T_2), happens(pProcess_2(Inputs_2, Outputs_2), T_3, T_4), \ldots,\\ &happens(pProcess_n(Inputs_n, Outputs_n), T_x, T_y), before(T_0, T_1), before(T_0, T_3), \ldots, before(T_0, T_x),\\ &before(T_2, T_m), before(T_4, T_m), \ldots, before(T_y, T_m)])\end{aligned}$$

Translations of the If–Then–Else control construct:

$$\begin{aligned}&axiom(happens(pIfThenElseExample(Inputs, OutPuts), T_0, T_m),\\ &[happens(jpl\_\ pIfCondition(Inputs_1), T_1, T_2), happens(pThenCase(Inputs, Outputs), T_3, T_4),\\ &before(T_0, T_1), before(T_2, T_3), before(T_4, T_m)])\end{aligned}$$

$$\begin{aligned}&axiom(happens(pIfThenElseExample(Inputs, OutPuts), T_0, T_m),\\ &[happens(jpl\_\ pElseCondition(Inputs_1), T_1, T_2), happens(pElseCase(Inputs, Outputs), T_3, T_4),\\ &before(T_0, T_1), before(T_2, T_3), before(T_4, T_m)])\end{aligned}$$

Translation of the Repeat-While control construct:

$$\begin{aligned}&axiom(happens(pRepeatWhileExample(Inputs, OutPuts), T_0, T_m),\\ &[happens(jpl\_\ pLoopCondition(Inputs_1), T_1, T_2), happens(pWhilecase(Inputs, Outputs), T_1, T_3),\\ &before(T_0, T_1), before(T_1, T_2), before(T_1, T_3), before(T_3, T_m)])\end{aligned}$$

49.3.2 Web Service Automatic Composition Framework

The Web service automatic composition framework is in Fig. 49.1. The framework is divided into two steps: abstract service planning and instance execution. The steps of the workflow are: convert OWL-S descriptions and user inputs and outputs to the discrete Event Calculus axioms; execute the abductive theorem prover to generate plans; convert the generated plans to graphs; convert the selected graphs to OWL-S service files; execute the selected OWL-S service composition.

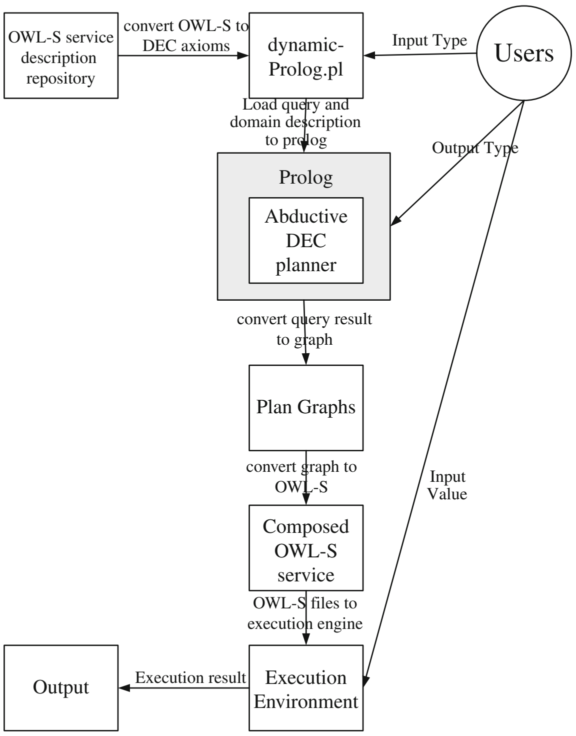

Fig. 49.1 The service automatic composition framework based on DEC

49.3.3 Abstract Service Planning

49.3.3.1 Translation of IOPE to DEC Axioms

The input and output of Web service translation to DEC are as follows:

$$\begin{aligned}&axiom\big(initiatedAt\big(web\_service\_event\big(I_1, I_2, \ldots, I_k, O_1, O_2, \ldots, O_j\big), known(out\_parameter\_name_i, O_i), T\big),\\ &[holdsAt(known(in\_parameter\_name_1, I_1), T), \ldots, holdsAt(known(in\_parameter\_name_k, I_k), T),\\ &holdsAt(precondition_1, T), \ldots, holdsAt\big(precondition_p, T\big)\,])\end{aligned}$$

The precondition of Web service translation to DEC is as follows:

$$\begin{aligned}&axiom(happens(pPreconditionExample(Input_1, Input_2), T_1, T_N),\\ &[jpl\_pPrecondition(Input_1, Input_2), atom\_number(Input_1, Arg_1),\\ &atom\_number(Input_2, Arg_2), Arg_1 < Arg_2, [Other\ Events\ and\ Temporal\ Orderings\ ]\ldots])\end{aligned}$$

The effect of Web service translation to DEC is as follows:

$$\begin{aligned}&axiom\big(initiatedAt\big(web\_service\_event\big(I_1, I_2, \ldots, I_k, O_1, O_2, \ldots, O_j\big), effect\_e_i, T\big),\\ &[holdsAt(known(in\_parameter\_name_1, I_1), T), \ldots, holdsAt(known(in\_parameter\_name_k, I_k), T),\\ &holdsAt(precondition_1, T), \ldots, holdsAt\big(precondition_p, T\big)\,])\end{aligned}$$

Here *effect_e$_i$* is the Prolog translation of the effect name taken from the *has effect* section of the service profile.

49.3.3.2 Abductive DEC Planning Algorithm

Abduction is used over the Event Calculus axioms to obtain partially ordered sets of events. Abduction is handled by a second order Abductive Theorem Prover (ATP) in [13]. The ATP tries to solve the goal list by proving the elements one by one. During the resolution, abducible predicates are stored in a residue to keep the record of the narrative. The narrative is a sequence of time-stamped events. In the definition of the predicate abduce, GL denotes the goal list, RL represents the residue list, NL represents the residue of negated literals, G is the axiom head and AL is the axiom body.

$$AH \leftarrow AB_1 \wedge AB_2 \wedge \ldots \wedge AB_N \quad axiom(AH, [AB_1, AB_2, \ldots, AB_N])$$

$$abduct(GL, RL) \leftarrow abduct(GL, \square, RL, \square) \;\; abduct(\square, RL, RL, NL)$$

$$\begin{aligned} abduct([holdsAt(F,T)|GL], CurrRL, RL, NL) \leftarrow axiom(initially(F), AL),\\ irresolvable(clipped(0,F,T), CurrRL, NL)\,, append(AL, GL, NewGL),\\ abduct(NewGL, CurrRL, RL, [clipped(0,F,T)|NL]) \end{aligned}$$

$$\begin{aligned} abduct([holdsAt(neg(F),T)|GL], R1, R3, N1, N4) \leftarrow axiom(initially(neg(F)), AL),\\ irresolvable(declipped(0,F,T), CurrRL, NL), append(AL, GL, NewGL),\\ abduct(NewGL, CurrRL, RL, [declipped(0,F,T)|NL]) \end{aligned}$$

49.3.4 *Generating Output Graphs from the Results of the Planner*

The Service-Oriented C^{4}ISR system on the naval fields is taken as an example for generating output graphs from the results of the planner. Fig. 49.2 is the generating graph. There eight services are as follows: the surface target detection service W_1, the underwater target detection service W_2, the information process service W_3, the fire control compute service W_4, the missile launch service W_5, the torpedo launch service W_6, the efficiency evaluate service W_7, the repeat attack service W_8, *St* and *En* are the start and end services. The W_1 and W_2 are concurrent, then the W_3 is in sequence. The W_5 and W_6 are choice relation. The W_8 is loop with W_4, W_5, W_6 and W_7, and k is a small integer. The abstract service of the process is corresponding to an instance service set, and the QoS of the instance services are different according their equipments such as execute time, cost, and accuracy.

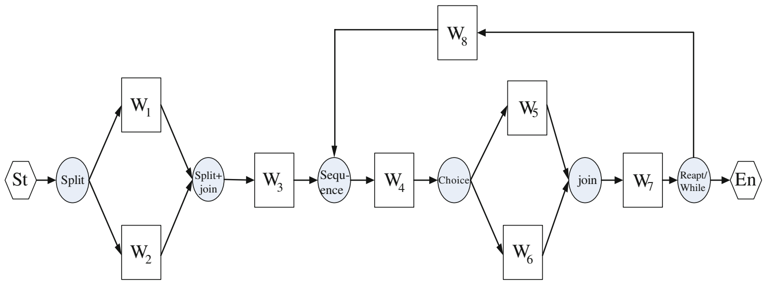

Fig. 49.2 Graph model of service-oriented C^4ISR system on the naval fields

49.3.5 The Instance Service Matching Method

The instance service matching process is divided into two steps: the local semantic matching and the global QoS-ware matching. The local semantic matching is not only considering the input and output matching, but also the local semantic matching. The global QoS-ware matching is translated into a multi-objective services composition optimization with QoS constraints. The multi-objective estimation of distribution algorithm based on Independent Component Analysis is utilized to produce a set of optimal Pareto services composition with constraint principle by means of optimizing various objective functions simultaneously.

49.4 Conclusion

A semantic Web service automatic composition method based on discrete Event Calculus is proposed aiming at the issues of service AI planning composition method such as large number of services, the confine of sequence composition process. The eight basic semantic Web service composition processes and their IOPE are modeled based on the actions, fluent and axioms of DEC. The service composition process is divided into two steps, abstract service planning and instance execution. And the service automatic composition framework is introduced. Also the abduction DEC planning algorithm and semantic matching method of instance execution are given. The comparison indicate the superiority of this method: it solves the confine of sequence composition process of classic AI planning composition method with DEC' presentation of compound action, concurrent action, continuous action, knowledge of the agent, the predicate number of the DEC is much smaller than EC which speed the service discovering and composition.

References

1. Martin D, Burstein M, McDermott D et al (2007) Bringing semantics to web services with OWL-S. World Wide Web J 10(3):243–277
2. Charif Y, Sabouret N (2006) An overview of semantic web services composition approaches. Electron Notes Theoret Comput Sci 33–41
3. Sirin E, Parsia B, Wu D et al (2004) HTN planning for Web service composition using SHOP2. J Web Sem 1:377–396
4. Mcilraith S, Son T (2002) Adapting go log for composition of semantic web services. In: Proceedings of the eighth international conference on principles of knowledge representation and reasoning, France
5. Klusch M, Gerber A, Schmidt M (2005) Semantic web service composition planning with OWLS-XPlan. In: Proceedings of the 1st international AAAI fall Symposium on agents and the semantic web. Arlington, pp 117–120
6. Wan Changlin, Han Xu, Niu Wenjia et al (2010) Dynamic description logic based web service composition and QoS model. Acta Electronica Sinica 38(8):1923–1928 (in Chinese)
7. Rao JH, Kungas P, Matskin M (2006) Composition of semantic web services using linear logic theorem proving. Inf Syst 4–5:340–360
8. Shanahan M (1999) The event calculus explained. In: Artificial intelligence today: recent trends and developments. Springer, Berlin 409–430
9. Agarwal V, Chafle G, Mittal S et al (2008) Understanding approaches for web service composition and execution. In: Proceedings of the first Bangalore annual compute conference. ACM, New York, pp 1–8
10. Okutan C, Cicekli NK (2010) A monolithic approach to automated composition of semantic web services with the event calculus. Knowl-Based Syst 23:440–454
11. Ozorhan EK, Kuban EK, Cicekli NK (2010) Automated composition of web services with the abductive event calculus. Inf Sci 180:3589–3613
12. Mueller ET (2008) Event calculus. In: Handbook of knowledge representation, vol 3. Elsevier, Amsterdam, pp 671–708
13. Shanahan MP (2000) An abductive event calculus planner. J Logic Prog 44:207–240