Using Temporal Business Rules to Synthesize Service Composition Process Models

Jian Yu[1], Jun Han[2], Paolo Falcarin[1], Maurizio Morisio[1]

[1] Department of Automation and Information, Politecnico di Torino, 10129 Turin, Italy
{jian.yu, paolo.falcarin, maurizio.morisio}@polito.it
[2] Faculty of ICT, Swinburne University of Technology, 3122 Hawthorn, Australia
jhan@ict.swin.edu.au

Abstract. Based on our previous work on the conformance verification of service compositions, in this paper we present a framework and associated techniques to generate the process models of a service composition from a set of temporal business rules. Dedicated techniques including path-finding, branch structure introduction, and parallel structure introduction are used to semi-automatically synthesize the process models from the semantics-equivalent Finite State Automata of the rules. These process models naturally satisfy the prescribed behavioral constraints of the rules. With the domain knowledge encoded in the temporal business rules, an executable service composition program, e.g. a BPEL program, can be further generated from the process models.

1 Introduction

The service-oriented computing paradigm, which is currently highlighted by Web services technologies and standards, provides an effective means of application abstraction, integration and reuse with its loosely-coupled architecture [1]. It prompts the use of self-describing and platform-independent services as the fundamental computational elements to compose cross organizational business processes. Executable Service composition languages including BPEL [2] and BPMN [3] have been created as effective tools for developing applications in this paradigm.

In the process of developing service-oriented applications, it is essential to ensure that the service composition being developed possesses the desired behavioral properties specified in the requirements. Unexpected application behaviors may not only lead to mission failure, but also may bring negative impact on all the participants of this process.

One of the typical solutions to this problem is through verification: by formally specifying the behavioral properties and then applying the model checking technique to ensure the conformance of the application to these properties. A bunch of research works have been published on the verification of service compositions in BPEL BPEL [4, 5, 6]. We also have proposed a pattern based specification language PROPOLS and used it on verifying BPEL programs [7]. One of the significant features of our approach is that PROPOLS is an intuitive, software practitioner accessible language that can be used by business experts to express temporal business rules.

Synthesis is the process of generating one specification from another at an appropriate level of abstraction, while some properties of the source specification are kept in the target one. Comparing with verification, the synthesis approach gives further benefits to the developers: Except for ensuring the property conformance, part of the application design and programming work is done automatically.

In this paper, we propose a synthesis framework and associated techniques to generate service composition process models from a set of PROPOLS temporal business rules. The PROPOLS rules prescribe the occurrences or sequence patterns of business activities in a business domain. The behavioral model of a set of rules can be achieved by translating each rule in the set into a semantics-equivalent Finite State Automaton and then composing them into another FSA with the logical operators defined upon FSA. A set of process models of the targeting service composition can be synthesized by analyzing the acyclic acceptable paths of the resulting FSA. Dedicated techniques include path-finding, branch structure introduction, and parallel structure introduction. Because of the "looseness" of the temporal business rules specification, which means the specification is incomplete, some of the generated process models are either trivial or meaningless to the developer. At this time, the developer can introduce new pertinent business rules to get a more precise result of the process models. When a satisfactory process model is generated, it can be further transformed into a BPEL program by discovering reusable Web services based on the ontology information encoded in the business activities. In a word, our synthesis framework offers an "intuitive specification" and then "correct by auto-construction" solution bring benefits to either a novice or an expert software developers.

The rest of the paper is organized as follows. Section 2 presents an overview of our synthesis framework. Section 3 explains the synthesis process and techniques in detail by an example from the e-business domain. Section 4 discusses the related work and we conclude the paper in Section 5.

2 Overview of the Synthesis Framework

Fig. 1 summarizes the main components of our synthesis framework. The shadowed ovals indicate the three major phases, specification, synthesis, and transformation, where iterations between specification and synthesis are usually necessary to get a precise process model. The focus of this paper is on the synthesis phase.

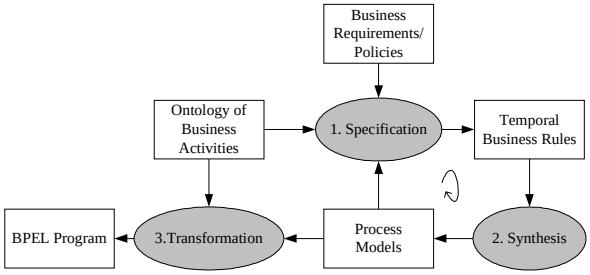


Figure 1. Overview of the Synthesis Framework

Specification

Temporal business rules state the occurrence or sequencing orders between business activities prescribed by some business requirements or policies. Business activities represent reusable services in a business domain, either coarse-grain services exposed beyond organization boundary or fine-grain services extracted from function libraries. A taxonomy or ontology can be used to organize business activities for effective browsing and searching.

We use PROPOLS [7] to specify temporal business rules. PROPOLS is a high-level temporal constraints specification language. The main constructs of PROPOLS are property patterns [8, 9] abstracted from frequently used temporal logic formulas. A logical composition mechanism allows the combination of patterns to express complex requirements. Below we briefly describe the key constructs and semantics of PROPOLS.

PROPOLS has two main constructs: basic patterns and composite patterns. Fig. 2 shows the form of basic patterns. The constructs on the left are temporal patterns and those on the right are scopes. A temporal pattern specifies what must occur and a scope specifies when the pattern must hold. The P, Q, R, and S in the figure denote events parameters (or business activities in this work) and n is a natural number.

Every temporal pattern has the intuitive meaning by its name, e.g. “precedes” means precondition relationship, “leads to” means cause-effect relationship, “p is absent” means p can not occur, “p is universal” means only p can occur, and “exists” defines the occurrence time of an event. Scope “globally” refers to the whole execution period of an application. Scope “before S” refers to the portion before the first occurrence of S, and so on.

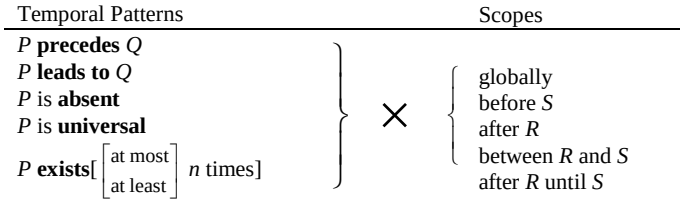
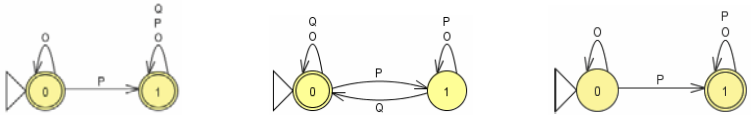


Figure 2. Basic Patterns

Every basic pattern has a one-to-one relationship with a predefined FSA which precisely expresses its semantics. E.g. Fig. 3 illustrated the corresponding FSA of basic patterns “precedes”, “leads to”, and “exists” with “globally” scope, where the symbol O denotes any other events than the named events. Fig. 3(a) says that before P occurs, an occurrence of Q is not accepted. Fig. 3(b) says that if P has occurred, an occurrence of Q is necessary to drive the FSA to a final state. And Fig. 3(c) says that only the occurrence of P can make the FSA reach a final state.



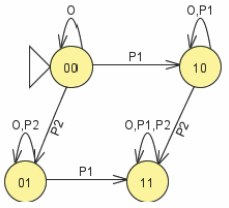
(a) P **precedes** Q **globally** (b) P **leads to** Q **globally** (c) P **exists globally**

Figure 3. FSA Semantics of Basic Patterns

Composite patterns are constructed by the logical composition of basic patterns. The syntax of composite patterns in BNF is:

Pattern = basic pattern | composite pattern
Composite pattern = not Pattern | Pattern and Pattern | Pattern or Pattern | Pattern xor Pattern

The semantics of composite patterns can be expressed by the logical composition defined upon FSA [10]. E.g. Fig 4 describes the logical composition between two basic patterns: “P1 exists globally” and “P2 exists globally”. The states are the Cartesian production of the two FSA and the final states are determined by the logical operator used.



Logic Operator	Final States
And	11
Or	01,10,11
Xor	01,10
Imply	00,01,11

Figure 4. FSA Semantics of Basic Patterns

Synthesis

Fig. 5 describes the steps of synthesis.

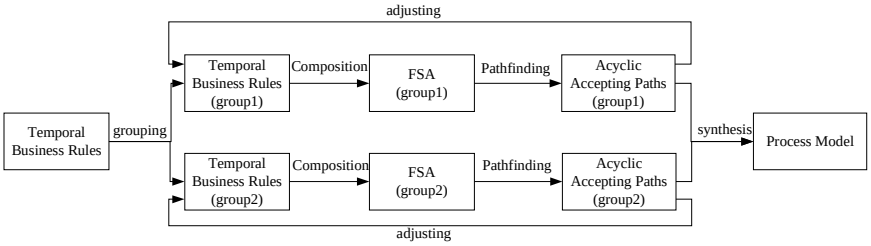


Figure 5. Synthesis Process

The main purpose of grouping is to separate concerns. One grouping strategy is by the goals/sub-goals of the business activities involved. E.g. if a set of business activities is classified under “ProcessOrder” goal, then all the temporal rules defined upon these business activities are in one group. Grouping can reduce the number of temporal rules that should be considered at a time, which reduces the complexity of the total synthesis process.

Based on a group of temporal rules, we get the corresponding semantic-equivalent FSA of each rule and then compose them into one FSA based on the logical composition operators defined upon FSA [10]. Usually, the “and” operator is used because we want all the rules be satisfied, in this situation, the resulting FSA precisely describe the all-satisfying semantics of this group of rules. Every string in the accepting language of the resulting FSA is a justified execution path of the related business activities, which conforms to all the rules in the group (Yu et al., 2006b).

In fact Many accepting paths are infinite because of the loop in the resulting FSA. We just find all the acyclic accepting paths (AAPs in short), because a loop can’t introduce new final state to the path.

Every AAP is a sequence of business activities satisfying the group of rules. If the generated AAPs can't satisfy the user's expectation, e.g. the number is too big or only contains trivial solutions, the user can refine his requirements by introducing additional temporal rules between business activities to get more precise AAPs.

The last step is to synthesis all the generated AAPs into a process model. First the user should pick one AAP from each group and connect them manually. Then techniques that automatically introduce branch and parallel structures will be used to generate a process model. Temporal rules defined between groups also will be used to check the validity of the process model.

Transformation

The resulting process model is transformed into the control flow constructs, e.g. "sequence", "switch", and "flow", in BPEL. The ontology of business activities will be used to discover reusable Web services and transformed into the "invoke" action in BPEL.

3 The Synthesis Process

In this section we describe our synthesis method and techniques in detail by an example. This example is adapted from a frequently appeared online purchasing scenario in the e-business domain. Our scenario accepts online orders and then processes them by the "Hard-Credit" business rule. To accept an order, this order must be checked for validity, the customer who places the order will receive either a confirmation or cancellation of the order based on the checking result. The purpose of the "Hard-Credit" rule is to protect the benefits of both the customer and the business provider. This rule states that the customer MUST pay when the order is fulfilled, and the payment is made only after the customer has received the goods and invoice. A third-party trustee, e.g. a bank, is necessary to implement this rule, first the customer deposit the payment to the bank, then the payment is transferred to the provider if the customer received the desired goods.

Specification

Fig. 6 shows the business activities solicited from the above-stated scenario and the temporal business rules defined upon them. Business activities and temporal rules are classified by two sub-goals: "AcceptOrder" and "HardCreditRule". Note that there is also one temporal rule, AH.1, defined between groups.

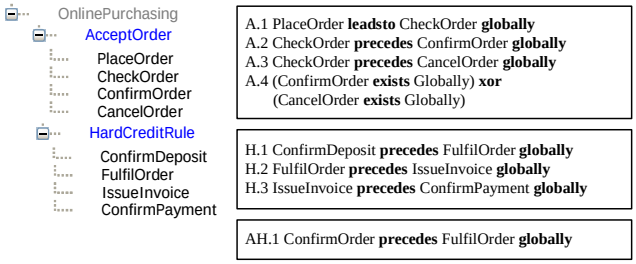


Figure 6. Business Activities of the Online Purchase Example

Fig. 7 shows the FSA generated by the and-composition of A.1~A.4 using the verification tool introduced in [7]. Every path from the initial state (0) to the final states (12 and 15) is a valid run that satisfies rule A.1 to rule A.4.

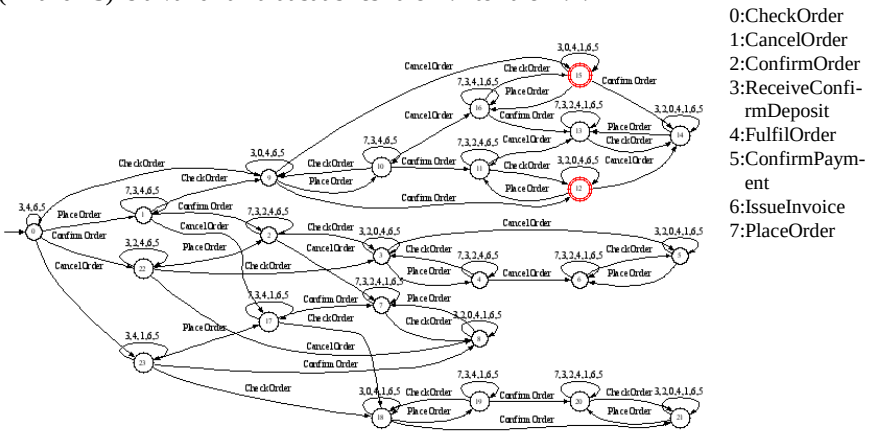


Figure 7. FSA Composed from A.1~A.4

Path-Finding

The algorithm of finding all the acyclic paths in a FSA is described in Fig. 8. This is a variation of the Depth-First-Search algorithm [11]. The most significant modification is that a state can be visited N times if it has N non-loop input edges (is on N different non-loop paths starting with the initial state). For example, state 9 in Fig. 7 has 2 non-loop input edges, so it will be visited 2 times when searching.

```
Global var: int counter, int[] order;
Procedure fsaAcyclicPath(FSA G) {
   counter = 0;
   order = new int[G.numberOfStates];
   for (int t = 0; t < G.numberOfStates; t++){
      order[t] = NotVisited;}
//search all the paths start with the initial state
      searchC(0);
}
Procedure searchC(int v) {
   order[v] = Visited;
   AdjacentList A = G.getAdjacentList(v);
   for (Node t = A.begin(); !A.end(); t = A.next()){
       if (order[t.v] == NotVisited){
           addEdge2Tree(v,t);
           searchC(t.v);
//mark the state as not-visited when move to a new path
           order[t.v] = NotVisited;}}}
```

Figure 8. Algorithm for FSA Acyclic Path-finding

Using the above path-finding algorithm to the FSA in Fig. 7, we can get all the acyclic paths starting from the initial state. Fig. 9 is an excerpt of the path-tree where the concentric circles are the final states.

From the generated path-tree, we can totally get 8 AAPs: 1.(Place, Check, Cancel)[1] , 2.(Place, Check, Confirm), 3.(Place, Check, Place, Cancel, Check), 4.(Place, Check, Place, Confirm, Check), 5.(Check, Place, Cancel, Check) , 6.(Check, Place, Confirm, Check) , 7.(Check, Cancel), 8.(Check, Confirm). Clearly not every AAP fits the user's need. At this time, the user can add rules to get more precise AAPs. E.g. if one extra rule, "A.5 PlaceOrder precedes CheckOrder globally", is introduced, the number of the above AAPs will be reduced to 4, only 1~4 are left.

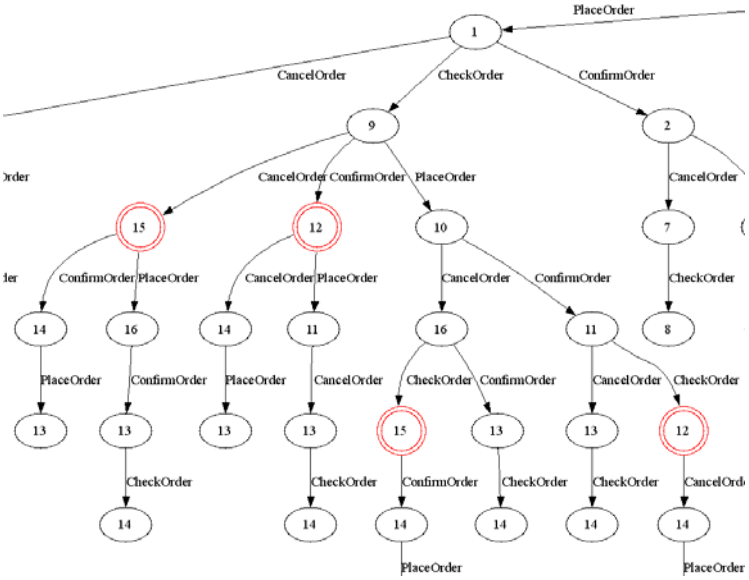


Figure 9. Excerpt of the Path-Tree

Synthesis

After all the satisfying AAPs are generated, the user can pick one AAP from each group and connect them manually to build the initial process model which only contains sequence structures. Temporal rules defined between groups will be used to check the validity of such connections.

A heuristic method is used to introduce branch structures into the initial process model: If a rule has the form like "P exists globally xor Q exist globally", e.g. rule A.4, we introduce a branch between P and Q. The justification of this method is that the process model with the branch will be verified correct against the temporal rule.

The introduction of parallel structures is based on interleaving assumption, which states that two events are concurrent if their occurring order does not change the consequence (Milner, 1989). Based on this assumption, if we have two business activities P and Q, P next to Q in one AAP and Q next to P in another AAP, which means the

[1] "PlaceOrder" is shorten as "Place" if no ambiguity is introduced. The same rule applied to other business activities.

occurring order of P and Q has nothing to do with the execution consequence, we are sure that P and Q can be put in a parallel structure. E.g. if we compose all the rules in Fig. 6, we can find that “ConfirmOrder” and ConfirmDeposit” can go in parallel.

Using the above-stated methods and techniques, a possible synthesized process model is shown in Fig. 10.

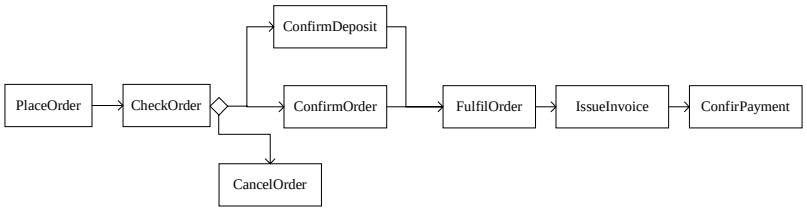


Figure 10. A Possible Synthesized Process Model

4 Related Work

A body of work has been reported on generating process models in the area of service oriented computing. Berardi et. al. use situation calculus to model the actions of Web services, and generate a tree of execution paths [13], they also use FSA to model the actions of individual services and then synthesis the service composition FSA [14]. In [15], Wu et. al. discuss how to synthesis Web service compositions based on DAML-S using an AI planning system SHOP2. Duan et. al. synthesis a BPEL abstract process from the precondition and post-condition of individual tasks [16]. Most of the above works are based on AI planning. One problem with AI planning synthesis is that planning focuses on generating a sequential path for conjunctive goals and does not consider generating process constructs like conditional branching and parallel execution.

More generally, there are also some approaches on generating formal behavioral models from another formalism. E.g. Beeck et. al. use Semantic Linear-time Temporal Logic to synthesis state charts [17]. Uchitel et. al. use Message Sequence Charts to synthesis Finite Sequential Processes[18]. The most significant difference between our approach and theirs is that our process model is more close to the final program, while their models are more abstract and suitable for reasoning the general properties of the system.

5 Conclusion

In this paper, we have presented a framework and associated techniques to semi-automatically synthesis service composition process models from temporal business rules. This framework is supposed to give much help to common software practitioners, the rule specification language PROPOLS is intuitive and works at the business level, a “correct” process model can be generated semi-automatically, which facilitates

daily programming work and finally brings benefits to both the novice and the expert software developers.

Currently, we are working on the transformation phase of the framework. In the future, we plan to integrate this framework with some graphical service composition editors, e.g. *ActiveBPEL Designer* [19].

References

1. Alonso, G., Casati, F., Grigori, Kuno H., Machiraju, V.: Web Services Concepts, Architectures and Applications. Springer-Verlag (2004)
2. Arkin, A., Askary, S., Bloch, B., Curbera, F., Goland, Y., Kartha, N., Liu, C.K., Thatte, S., Yendluri, P., Yiu, A.: Web Services Business Process Execution Language Version 2.0 Working Draft. http://www.oasis-open.org/committees/download.php/10347/wsbpelspecification-draft-120204.htm (2004)
3. BPMI: Business Process Modeling Language. http://www.bpmi.org/ (2002)
4. Foster, H.: A Rigorous Approach to Engineering Web Services Compositions. PhD thesis, Imperial College London. http://www.doc.ict.ac.uk/~hf1 (2006)
5. Stahl C.: A Petri Net Semantics for BPEL. Informatik-Berichte 188, Humboldt-Universitat zu Berlin, June 2005 (2005)
6. Fu, X., Bultan T., Su J.: Analysis of Interacting BPEL Web Services. In Proc. 13th World Wide Web Conf. New York, NY, USA (2004) 621-630
7. Yu, J., Phan, T., Han, J., Jin, Y., et al: Pattern Based Property Specification and Verification for Service Composition. In Proc. 7th Int. Conf. on Web Information Systems Engineering. Springer-Verlag, LNCS 4255. Wuhan, China (2006) 156-168
8. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in Property Specifications for Finite state Verification. In Proc. 21th Int. Conf. on Software Engineering. Los Angeles, CA, USA (1999) 411-420
9. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: A System of Specification Patterns. http://www.cis.ksu.edu/santos/spec-patterns (1997)
10. Yu, J., Phan, T., Han, J., Jin, Y.: Pattern based Property Specification and Verification for Service Composition. Technical Report SUT.CeCSES-TR010. CeCSES, Swinburne University of Technology, http://www.it.swin.edu.au/centres/cecses/trs.htm (2006)
11. Sedgewick, R.: Algorithms in Java, Thrid Edition, Part 5: Graph Algorithms. Addison Wesley (2003)
12. Milner, R.: Communication and Concurrency. Prentice-Hall (1989)
13. Berardi, D., Calvanese, D., Giuseppe, G., Lenzerini, M., Mecella, M.: Automatic composition of e-services that export their behavior. In Proc. 1st Int. Conf. on Service Oriented Computing. Trento, Italy (2003)
14. Berardi, D., Glancomo, G., Lenzerini, M., Mecella, M., Calvanese, D.: Synthesis of Underspecified Composite e-Services based on Automated Reasoning. In Proc. 2st Int. Conf. on Service Oriented Computing. . New York, USA (2004)
15. Wu, D., Parsia, B., Sirin, E., Hendler, J., Nau, D.: Automating DAML-S web services composition using SHOP2. In Proc. 2nd Int. Semantic Web Conf. Florida (2003)
16. Duan, Z., Bernstein, A., Lewis, P., Lu, S.: A Model for Abstract Process Specification, Verification and Composition. In proc. of the 2nd Int. Conf. on Service Oriented Computing. New York, USA (2004)
17. Beeck, M., Margaria, T., Steffen, B.: A Formal Requirements Engineering Method for Specification, Synthesis, and Verification. In Proc. 8th Int. Conf. on Software Engineering Environment. Washington, DC, USA (1997)

18. Uchitel, S., Kramer, J., Magee, J.: Synthesis of Behavioral Models from Scenarios. IEEE Trans. On Software Engineering Vol.29 2 (2003) 99-115
19. ActiveBPEL Designer. http://www.activenedpoints.com/products/activebpeldes/ (2007)