

Synthesizing Service Composition Models on the Basis of Temporal Business Rules

Jian Yu^{1,2} (喻 坚), Yan-Bo Han³ (韩燕波), Jun Han⁴ (韩 军), Yan Jin⁴ (金 岩), Paolo Falcarin¹ and Maurizio Morisio¹

¹*Department of Automation and Information, Politecnico di Torino, Torino 10129, Italy*

²*School of Computer Science, The University of Adelaide, SA5005, Australia*

³*Grid and Service Computing Research Center, Institute of Computing Technology, Chinese Academy of Sciences Beijing 100190, China*

⁴*Faculty of ICT, Swinburne University of Technology, Hawthorn 3122, Australia*

E-mail: jian.yu@polito.it; yhan@ict.ac.cn; jhan@ict.swin.edu.au; yjin@ict.swin.edu.au; paolo.falcarin@polito.it
maurizio.morisio@polito.it

Received November 14, 2007; revised June 9, 2008.

Abstract Transformational approaches to generating design and implementation models from requirements can bring effectiveness and quality to software development. In this paper we present a framework and associated techniques to generate the process model of a service composition from a set of temporal business rules. Dedicated techniques including path-finding, branching structure identification and parallel structure identification are used for semi-automatically synthesizing the process model 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. A running example in the e-business domain is used for illustrating our approach throughout this paper.

Keywords service composition, composition synthesis, behavioral model, temporal patterns

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^[1,2]. It promotes 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^[3] and BPML^[4] have long been effective tools for developing applications in this paradigm. Recent trends in user-centric service creation leveraged by Web 2.0 technologies also drive people to vary BPEL to facilitate the creation of a new breed of Web applications called mashups^[5,6], which are built by aggregating Web contents and services like Google maps and Flickr images.

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 to 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 papers have been published on the verification of service compositions in BPEL^[7–9]. We also have proposed a pattern-based specification language PROPOLS and used it for verifying BPEL programs^[10]. 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.

Regular Paper

This work is supported by the European IST-FP6 Project OPUCE under Grant No. 34101, the National Natural Science Foundation of China under Grant No. 60573117, the National Basic Research 973 Program of China under Grant No. 2007CB310804, and the Australian Research Council under Grant No. LP0775188.

Synthesis is the process of generating one specification from another at an appropriate level of abstraction, while significant properties of the source specification are kept in the target one. Compared with verification, the synthesis approach gives further benefits to developers: except for ensuring the property conformance — part of the application design and programming work is done automatically. This kind of transformational implementation can bring effectiveness and quality to software development^[11]. Expectedly, this approach would be very helpful to novice developers including end-users who want to create their own service applications^[12].

On the basis of our previous work^[13], in this paper, we propose a synthesis framework and associated techniques to generate the service composition process model from a set of temporal business rules which prescribe the occurrences or sequence patterns of business activities in a business domain. We use PROPOLS^[10] as the rule specification language. 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 (FSA) and then composing them into another FSA with the logical operators defined upon FSA. A set of possible execution paths of the targeting service composition can be generated by finding the acceptable paths of the resulting FSA. Pattern-based branching structure identification and parallel structure identification techniques are used for generating a process model upon one of these paths. Because the temporal rules are specified at business level with easy comprehension of properties, they are loose constraints^[14] and usually insufficient to characterize solutions precisely enough; therefore high level manual interactions including introducing new rules and choosing suitable paths are necessary for producing a satisfying process model. It is worth noting that these manual interactions are simple enough for people without any knowledge of formal methods to use. Finally, the generated process model can be further transformed into a BPEL program by discovering reusable Web services based on the ontology information encoded in the business activities.

In general, the core benefit of our approach is that it provides an approach to semi-automatically generating design models (the process model) from requirement models (the temporal rules) featuring intuitive specification and correct-by-construction. All the involved human activities are rule specification and result elicitation, which make our approach easy to access.

The rest of the paper is organized as follows. Section 2 presents an overview of our synthesis framework. Section 3 and Section 4 explain in detail the two major

phases in the synthesis framework, specification and synthesis, with a running example. Section 5 is the discussion and related work, and finally we conclude the paper in Section 6.

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. Iterations between specification and synthesis phases are usually necessary to fine-tune the process model.

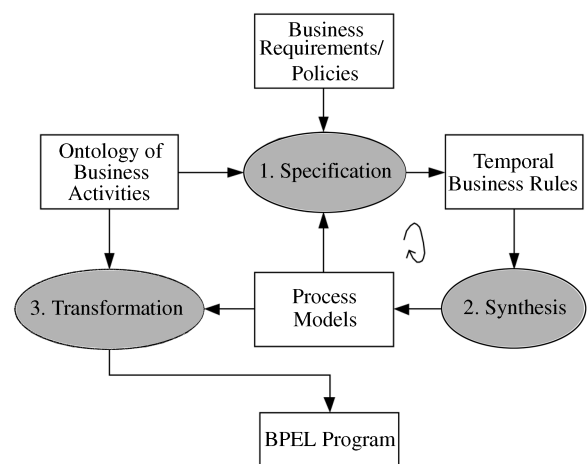


Fig.1. Overview of the synthesis framework.

In the specification phase, we use PROPOLS to describe temporal business rules which state the occurrence or sequencing constraints between business activities prescribed by the business requirements or policies. Here the business activities represent reusable services in a business domain which are either coarse-grained services exposed beyond organization boundary or fine-grained services extracted from function libraries. A taxonomy or an ontology can be used to organize business activities for effective browsing and searching. Each rule is composed of two parts: temporal pattern and scope, where a temporal pattern specifies what business activity must occur and a scope specifies when the temporal pattern must hold. For example, if we want to state the precondition relationship between two business activities P and Q in a global sense, we can specify a temporal rule with the form P precedes Q globally. In the next section of this paper, we will briefly introduce PROPOLS language and show how to use it to specify temporal business rules for an example scenario.

The synthesis phase can be further illustrated in Fig.2.

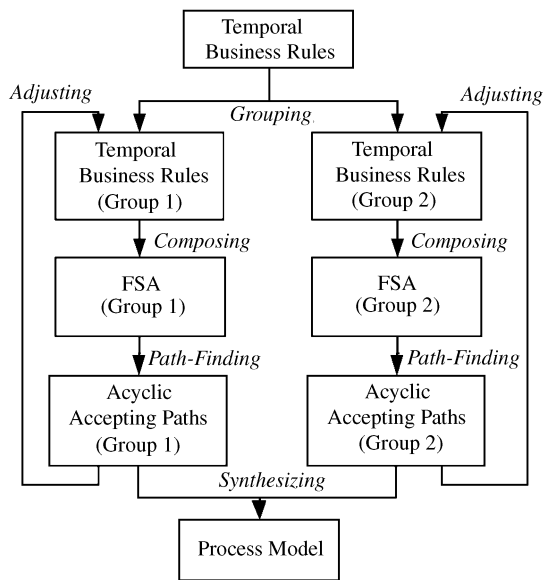


Fig.2. Synthesis process.

First the temporal business rules created in the specification phase can be classified into different groups based on the ontology of business activities. 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 like *Place Order*, *Check Order* and *Conform Order* are classified under *Process Order* goal, then all the temporal rules defined upon these business activities are also automatically put 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.

Next the semantics of a group of temporal rules are captured by an FSA. This FSA is generated by composing the corresponding FSA of each rule in the group (see next section for detail). Every string/path in the accepting language of the resulting FSA (called *accepting path*) is a justified execution path of the related business activities, which conforms to all the rules in the group^[10].

In fact, FSA can have accepting paths that are infinite because of the loop transitions between states. We just find all the *Acyclic Accepting Paths* (AAPs, see Section 4 for detail). The rationale of this approach is that we are in general looking for shortest solutions.

Every AAP is a sequence of business activities satisfying the group of rules. If the set of the generated AAPs cannot satisfy the user's expectation, e.g., its size is too big or it only contains trivial solutions, the user can always refine the resulting AAPs by adjusting temporal rules or introducing new rules. The last

step of the synthesis phase is to synthesize a process model from AAPs. The user is asked to pick one AAP from each group and decide an order between them. Techniques that automatically identify branching and parallel structures then are used for generating the final process model.

The transformation phase will transform the resulting process model 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.

To better illustrate our approach, we present a scenario from the e-business domain as a running example. This is an online purchasing scenario and a brief description is as follows. An e-business merchant can accept online orders and then processes them. Before accepting an order, he must check this order for validity, and the customer who places the order will receive either a confirmation or cancellation of the order based on the checking result. A so-called "Hard-Credit" rule is used for protecting the interests 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 both the goods and the invoice. A third-party trustee, e.g., a bank, is necessary to implement this rule: first the customer deposits the payment to the bank, and then the payment is transferred to the provider if the customer received the desired goods. The next two sections will illustrate the specification and the synthesis phases based on this scenario.

3 Specifying Business Temporal Rules with PROPOLS

PROPOLS is a high-level temporal constraints specification language. The main constructs of PROPOLS are property patterns^[15,16] 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.3 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 event 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,

Temporal Patterns			Scopes
<i>P</i> precedes <i>Q</i> <i>P</i> leads to <i>Q</i> <i>P</i> is absent <i>P</i> is universal <i>Q</i> <i>P</i> exists [at most / at least] <i>n</i> times	}	×	globally before <i>S</i> after <i>R</i> between <i>R</i> and <i>S</i> after <i>R</i> until <i>S</i>

Fig.3. Basic patterns.

leads to means cause-effect relationship, *P is absent* means *P* cannot 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.

Composite patterns are constructed by the logical composition of basic patterns which enrich the expression capability 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*.

In the example scenario, the left part of Fig.4 shows a taxonomy of business activities classified by goals/sub-goals. The right part is the PROPOLS temporal rules defined upon them. These rules are automatically grouped based on the class/type of the involving business activities: if all the business activities of a rule are of the same class, then this rule is also classified under this class, e.g., Rules A.1~A.4. Note that if the business activities of a rule are from different classes, then this rule will not be grouped automatically, e.g., Rule AH.1. We call such rules *cross-group rules*, which are used for verifying the order between AAPs during the synthesis phase (see Section 4 for detail).

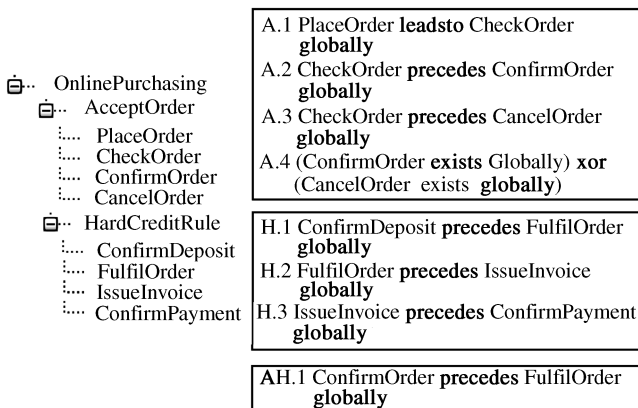


Fig.4. Business activities and temporal rules of the online purchasing example.

Rule A.4 is a composite pattern, which intuitively

means that the order will either be confirmed or canceled, but not both.

The formal semantics of each basic pattern can be described by an FSA. Fig.5 shows the FSA semantics of Rules A.1 and A.2. The symbol \circ in the figure denotes any other events than the named events. Fig.5(a) says if PlaceOrder has occurred, an occurrence of CheckOrder is necessary to drive the FSA to a final state. Fig.5(b) says that before CheckOrder occurs, an occurrence of ConfirmOrder is not accepted.

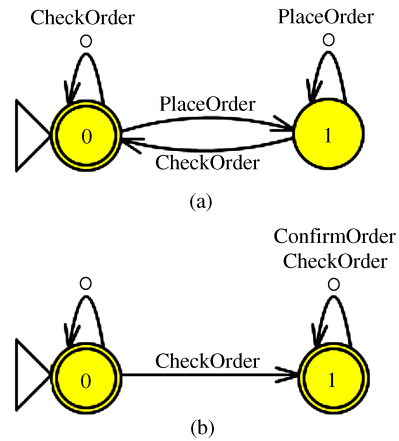


Fig.5. FSA semantics of basic patterns. (a) Rule A.1. (b) Rule A.2.

The semantics of composite patterns can be expressed by the logical composition of FSA^[17]. For example, the FSA on the right part of Fig.6 represents the semantics of Rule A.4, which is the xor composition of the FSA of two basic patterns: *ConfirmOrder exists globally* and *CancelOrder exists globally*. The states are the Cartesian production of the two FSAs and the final states are determined by the logical operator used.

4 Synthesis Process

In this section we detail the synthesis process described in Fig.2.

4.1 Grouping

Grouping is the first step of the synthesis process. We have seen from Fig.4 that the grouping of rules can be done automatically if we have a taxonomy of the business activity. In fact grouping is an optional step; the user can use or skip this step depending on the amount of the rules.

4.2 FSA Composition

Fig.7 shows the FSA generated by the *and.composition* of Rules A.1~A.4 using the verification tool introduced in [10]. If we use *and.composition*

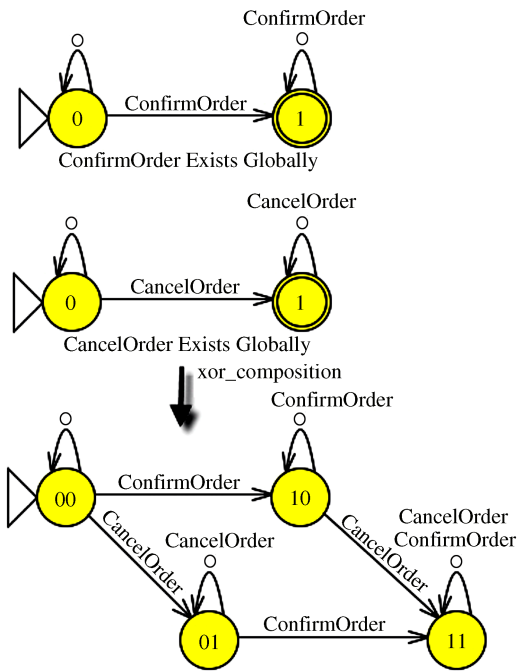


Fig.6. Logical composition of two *exists* basic patterns.

to compose FSA, every path from the initial state to the final states of the composed FSA is also an accepting path of every component FSA, which means that every accepting path of the composed FSA is a valid occurrence of events that satisfies all the involving rules. In Fig.7, every path from initial state 0 to final state 12

or final state 15 is a valid run of business activities, which satisfies every rule in Rules A.1~A.4, e.g., the path *CheckOrder*; *CancelOrder* on top of the figure.

4.3 Path-Finding

Because FSA has only one initial state, all the AAPs of an FSA form a path tree with the initial state as the root. In Fig.8, a variation of the Depth-First-Search algorithm^[18] for graphs is used for generating this AAPs tree. The original DFS algorithm generates only one node on the tree for every state. In our case, if a state has *N* non-reflexive incoming edge, it should be on *N* branches of the tree. We modify the DFS algorithm to accommodate this change. In the last line of our algorithm, a state is marked unvisited when a new branch is to be visited so that the state can also have a node on this branch.

Applying this path-finding algorithm to the FSA in Fig.7, we can get all the acyclic paths starting from the initial state. Fig.8 is an excerpt of the path tree, showing some of the paths containing final states.

There are totally 8 AAPs found by the algorithm:

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*,

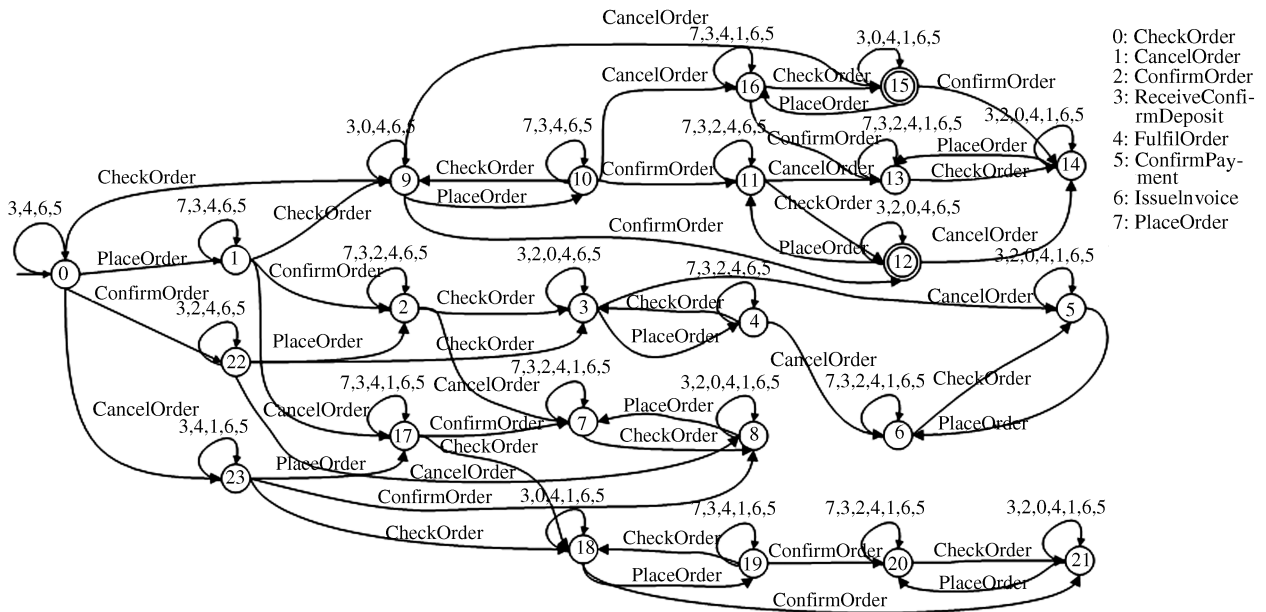


Fig.7. *and_composition* of the FSA of Rules A.1~A.4.

^①“PlaceOrder” is shortened as “Place” if no ambiguity is introduced. The same rule is applied to other business activities.

- 7. Check; Cancel,
- 8. Check; Confirm.

```

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 starting with the initial
  state**/
  searchC(0); }
Procedure searchC(int v) {
  ord[v] = Visited;
  AdjacentList A = G.getAdjacentList(v);
  for (Node t = A.begin(); !A.end(); t = A.next()){
    if (ord[t.v] == NotVisited){
      addEdge2Tree(v,t);
      searchC(t.v);
      /**mark the state as not-visited when move to a
      new branch**/
      ord[t.v] = NotVisited;}}}}
    
```

Fig.8. Algorithm for FSA acyclic path-finding.

Clearly, not every AAP fits the user's need. At this time, the user can introduce new rules to remove some unrelated AAPs. For example, if one extra rule, *PlaceOrder precedes CheckOrder globally*, is introduced, the number of the above AAPs will be reduced to 4, only AAPs 1~4 are left.

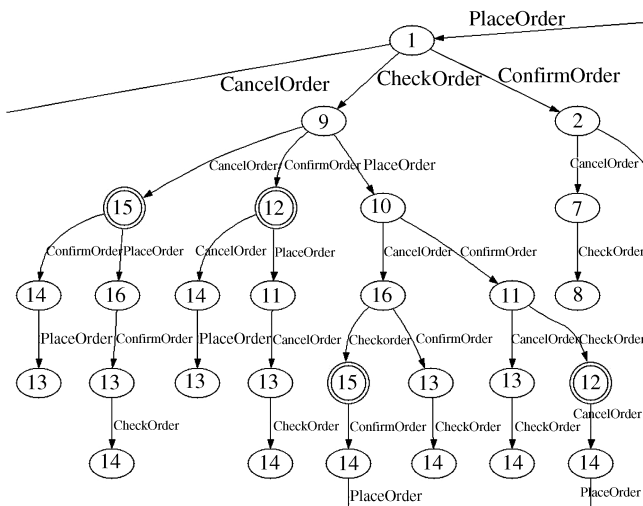


Fig.9. Exception of the path tree.

4.4 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. Suppose the chosen AAP for Rules A.1~A.4 is σ_A : *PlaceOrder; CheckOrder; ConfirmOrder*, and the chosen AAP for Rules H.1~H.3 is σ_H : *FulfilOrder; IssueInvoice; ConfirmPayment*, the user can either connect σ_A before or after σ_H . At this time, cross-group rule AH.1 is used for verifying the order between AAPs with the tool introduced in [10]. So the verification will not pass if σ_H is put before σ_A . Note that this step is necessary only when grouping is applied on the rules.

A heuristic method is used for introducing branching 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 correctly 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^[19]. Based on this assumption, suppose we have two AAPs σ_1 and σ_2 , σ_1 has the form $\alpha PQ\beta$ and σ_2 has the form $\alpha QP\beta$, where α and β are two path strings, *P* and *Q* are events, we can conclude that *P* and *Q* can be put in a parallel structure because the occurrence sequence of *P* and *Q* has nothing to do with the execution consequence. For example, 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 candidate synthesized process model based on σ_A and σ_H is shown in Fig.10.

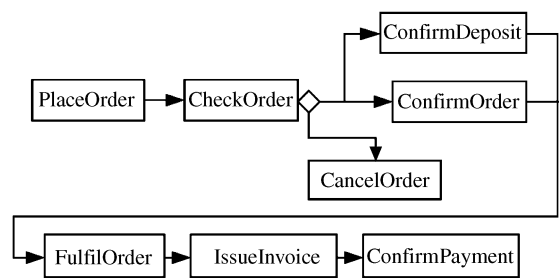


Fig.10. Candidate synthesized process model.

5 Transformation

The transformation phase utilizes both the abstract process model generated in the synthesis phase and the ontology information associated with the business activity to create a skeleton of the target BPEL program. An ActivityAppContext ontology is used for specifying the application context of business activities. Fig.11

is a graphical representation of the ActivityAppContext ontology which is generated by OntoViz, a plug-in for ontology editor Protégé. For every application context of an activity, there are 4 main properties: goal, provider, user and interactionType. A request must be initiated from a user to a provider. The interactionType property specifies how the user interacts with the provider, that is, synchronous or asynchronous.

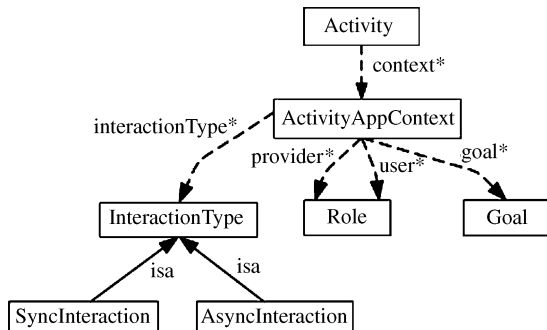


Fig.11. ActivityAppContext ontology.

A corresponding application context description for the PlaceOrder activity can be found in Fig.12.

```

<!DOCTYPE rdf:RDF [ <!ENTITY Act
"http://softeng.polito.it/you/activity.owl#" > ]>
<rdf:RDF
xmlns="http://softeng.polito.it/you/PlaceOrder.owl#"
xml:base="http://softeng.polito.it/you/PlaceOrder.owl"
xmlns:Act="http://softeng.polito.it/you/activity.owl#"
<Act:Activity rdf:ID="PlaceOrder">
  <Act:context rdf:resource=" #E-Business"/>
</Act:Activity>
<Act:ActivityAppContext
  rdf:ID="E-Business">
  <Act:provider rdf:resource="&Activity;Merchant"/>
  <Act:user rdf:resource="&Activity;Customer"/>
  <Act:goal rdf:resource="&Activity;AcceptOrder"/>
  <Act:interactionType rdf:resource="&Activity;
  SyncInteraction"/>
</Act:ActivityAppContext></rdf:RDF>
    
```

Fig.12. Application context description for activity PlaceOrder.

From the application context description of an activity, we can determine the partners involved in the activity, and also the interaction mode between them. We can use this information to create partnerLinkType, partnerLink and invoke/receive activities of a BPEL program. For example as shown in Fig.13, on the basis of above defined context description, we can create a BPEL partnerLinkType between Customer and Merchant, and a BPEL receive activity (because the process

is running on the merchant side, and the message is sent from Customer to Merchant) that relates to the Customer-Merchant partnerLinkType.

```

<plnk:partnerLinkType name="CustomerMerchantLinkType">
  <plnk:role name="customer"><plnk:portType
    name="customer:CustomerPortType"/>
  </plnk:role>
  <plnk:role name="Merchant">
  <plnk:portType
    name="tns:MerchantPortType"/>
  </plnk:role></plnk:partnerLinkType>
  <partnerLink name="CustomerMerchantLink"
    partnerLinkType=
      "tns:CustomerMerchantLinkType"
    partnerRole="customer"
    myRole="Merchant"/>
  <receive partnerLink="CustomerMerchantLink"
    operation="PlaceOrder"/>
    
```

Fig.13. ActivityAppContext ontology.

Fig.14 illustrates the structure of the final BPEL program skeleton and its interactions with the partners.

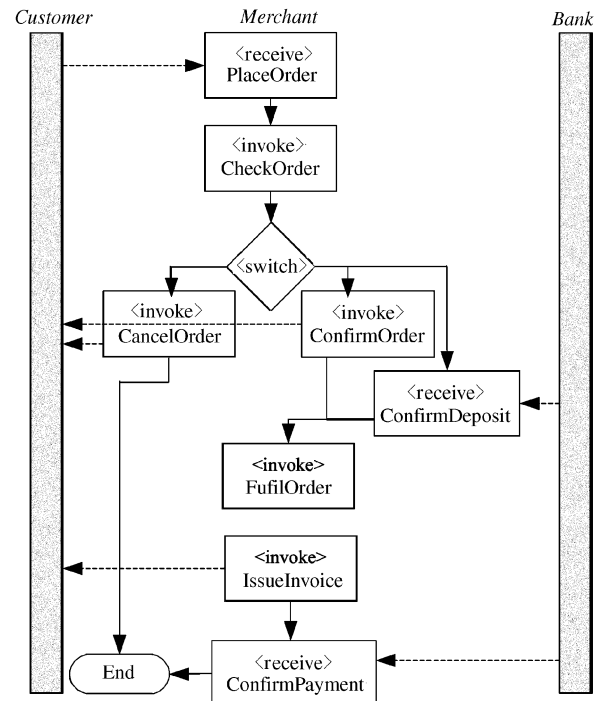


Fig.14. Structure of the final BPEL program.

Finally, to make the generated BPEL skeleton a full-fledged executable BPEL process, concrete Web services should replace the business activity placeholders which can be fulfilled either by implementing from scratch or discovering reusable services based on the

semantics of the business activities. Other tasks like variables and port type definition are also needed. Such a discussion is out of the scope of this paper.

6 Application and Evaluation

A test-bed for our service composition synthesis approach is the EU FP6 funded research project OPUCE (Open Platform for User-Centric Service Creation and Execution)^[20]. One of the major purposes of OPUCE is to develop an advanced service creation platform for end-users to create their own value-added services using well-encapsulated telecom and IT base services. In this context, there are two sets of business rules which are defined between base services. One set of rules are specified by base service providers and the platform administrators to regulate the behavior of base services. The other set of rules are specified by end-users to partially express and record their requirements to the desired application. These two sets of rules together are used for synthesizing a service composition model. Based on the user response, the programming work became an interesting trial-and-error process. The users also reported that when too many candidate paths are generated, instead of introducing new rules, reducing the number of base services by refining goals into sub-goals is a more effective way.

To test the performance of our synthesis approach, we organize 5 groups of rules whose corresponding composed FSA has state number of 30, 100, 150, 250 and 500 respectively and record the time used to finish the synthesis work on each group. Fig.15 is the result of this experiment on a notebook with AMD Turion64 2.0GHz CPU and 896MB memory running Windows XP. As we can read, when the state number is 30, the same size as the one by composing Rules A.1 to A.4 in Section 4, it only takes less than 200ms to finish the synthesis. But if the number of states reaches 500, it will take about 5 seconds to finish the task. If we take 2 seconds as the upper limit of user's acceptable time on a synthesis task, the threshold of state number is at about 200, i.e.,

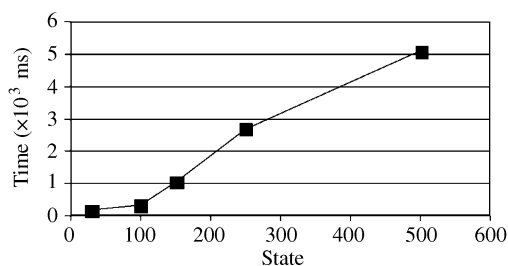


Fig.15. Performance results of the synthesis framework.

the state number of an FSA by composing 5 to 6 basic rules. This result is also compatible with the response of the user experience: our synthesis approach works best by dividing a large group of rules into smaller size.

7 Discussion and Related Work

One limitation of our work is that the synthesis framework cannot generate loops in the process model because all the accepting paths in the path-tree have finite length. In a general sense it may seem a big drawback. But in the context of user-centric service creation, looping is not allowed inherently in mainstream platforms including Microsoft Popfly^[21] and Yahoo Pipes^[22], and also in the OPUCE platform^[20]. We believe that the reason why looping is banned is based on the consideration of usability. Admittedly, end users can still compose versatile services using the above-mentioned platforms; also it is not always an easy task for a non-technically experienced end user to configure a correct loop.

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^[23]. They also use FSA to model the actions of individual services and then synthesize the service composition FSA^[24]. Wu *et al.* discuss how to synthesize Web service compositions based on DAML-S using an AI planning system SHOP2^[25]. Duan *et al.* synthesize a BPEL abstract process from the pre-condition and post-condition of individual tasks^[26]. Most of the above work is 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.

In a more general context, there are approaches to generating behavioral models from various formalisms. For example, Beeck *et al.* use Semantic Linear-Time Temporal Logic to synthesis state charts^[12]. Uchitel *et al.* use Message Sequence Charts to synthesis Finite Sequential Processes^[27]. 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 properties of the system.

8 Conclusion

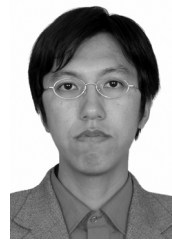
In this paper, we present a framework and associated techniques to semi-automatically synthesis service composition process models from temporal business rules. This framework is supposed to give much help to novice

software practitioners including end-users, because the rule specification language PROPOLS is intuitive and works at the business level, also an auto-verified process model can be generated semi-automatically.

In the future, work is planned on improving the performance of the synthesis framework. We also intend to integrate this framework to some graphical service composition editors, e.g., ActiveBPEL Designer^[28].

References

- [1] Alonso G, Casati F, Grigori K H *et al.* Web Services Concepts, Architectures and Applications. Springer-Verlag, 2004.
- [2] Yu J, Han Y. Service Oriented Computing: Principle and Applications. Tsinghua University Press, 2006.
- [3] Jordan D, Evdemon J *et al.* Web Services Business Process Execution Language, Version 2.0. 2007, <http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.pdf>.
- [4] BPMI. Business Process Modeling Language. 2002, <http://www.bpmi.org/>.
- [5] ProgrammableWeb. <http://www.programmableweb.com>.
- [6] Curbera F, Duftler M, Khalaf R *et al.* Bite: Workflow composition for the web. In *Proc. 6th Int. Conf. Service-Oriented Computing*, Vienna, Austria, LNCS 4749, 2007, pp.94–106.
- [7] Foster H. A rigorous approach to engineering web services compositions [Dissertation]. Imperial College London, 2006, <http://www.doc.ic.ac.uk/~hfl>.
- [8] Stahl C. A petri net semantics for BPEL. Informatik-Berichte 188, Humboldt-Universität zu Berlin, June 2005.
- [9] Fu X, Bultan T, Su J. Analysis of interacting BPEL web services. In *Proc. 13th World Wide Web Conference*, New York, USA, 2004, pp.621–630.
- [10] Yu J, Phan T, Han J *et al.* Pattern based property specification and verification for service composition. In *Proc. 7th Int. Conf. Web Information Systems Engineering*, Wuhan, China, LNCS 4255, 2006, pp.156–168.
- [11] Vienneau L. A Review of Formal Methods. Software Engineering, Computer Society Press, 1996.
- [12] Beek M, Margaria T, Steffen B. A formal requirements engineering method for specification, synthesis, and verification. In *Proc. 8th Int. Conference on Software Engineering Environment*, Washington DC, USA, 1997, pp.131–144.
- [13] Yu J, Wang J, Han Y *et al.* Developing End-User Programmable Service-Oriented Applications with VINCA. The Knowledge Gap in Enterprise Information Flow: Information Logistic Concepts and Technologies for Improving Information Flow in Networked Organizations, Sandkuhl K, Smirnov A, Weber H (eds.), Jönköping University, Ljungby, Sweden, 2005, pp.47–68.
- [14] Yu J, Han J, Falcarin P, Morisio M. Using temporal business rules to synthesize service composition process models. In *Proc. 1st Int. Workshop on Architectures, Concepts and Technologies for Service Oriented Computing*, Barcelona, Spain, 2007, pp.85–94.
- [15] Dwyer M B, Avrunin G S, Corbett J C. Patterns in property specifications for finite state verification. In *Proc. 21st Int. Conf. Software Engineering*, Los Angeles, CA, USA, 1999, pp.411–420.
- [16] Dwyer M B, Avrunin G S, Corbett J C. A System of Specification Patterns.<http://www.cis.ksu.edu/santos/spec-patterns>.
- [17] Yu J, Phan T, Han J, Jin Y. Pattern based property specification and verification for service composition. Technical Report, SUT.CeCSES-TR010, Swinburne University of Technology, 2006, <http://www.it.swin.edu.au/centres/ceceses/trs.htm>.
- [18] Sedgewick R. Algorithms in Java, Third Edition, Part 5: Graph Algorithms. Addison Wesley, 2003.
- [19] Milner R. Communication and Concurrency. Prentice-Hall, 1989.
- [20] OPUCE: Open platform for user centric service creation and execution. <http://www.opuce.tid.es/>.
- [21] Microsoft Popfly. <http://www.popfly.ms/>.
- [22] Yahoo Pipes. <http://pipes.yahoo.com/pipes/>.
- [23] Berardi D, Calvanese D, Giuseppe G *et al.* Automatic composition of e-services that export their behavior. In *Proc. 1st Int. Conf. Service Oriented Computing*, Trento, Italy, 2003, pp.43–48.
- [24] Berardi D, Glancom G, Lenzerini M *et al.* Synthesis of underspecified composite e-services based on automated reasoning. In *Proc. 2nd Int. Conf. Service Oriented Computing*, New York, USA, 2004, pp.105–114.
- [25] Wu D, Parsia B, Sirin E, Hendler J, Nau D. Automating DAML-S web services composition using SHOP2. In *Proc. 2nd Int. Semantic Web Conference*, Florida, USA, 2003, pp.195–210.
- [26] Duan Z, Bernstein A, Lewis P, Lu S. A model for abstract process specification, verification and composition. In *Proc. the 2nd Int. Conference on Service Oriented Computing*, New York, USA, 2004, pp.232–241.
- [27] Uchitel S, Kramer J, Magee J. Synthesis of behavioral models from scenarios. *IEEE Trans. Software Engineering*, 2003, 29(2): 99–115.
- [28] ActiveBPEL Designer.<http://www.activendpoints.com/products/activebpeledes/>.



Jian Yu received his Ph.D. degree in computer software and theory from Peking University, China, in 2002, and his M.Eng. and B.S. degrees in computer applications from the Chinese Academy of Sciences and Zhejiang University, China, in 1998 and 1995, respectively. He is currently a post-doc research fellow in the Software Engineering Group at

the Department of Automation and Information of Politecnico di Torino, Italy, and task leader of Semantic Service Repository Activity in EU FP6 OPUCE Project. He has been software engineer at DigitalChina and researcher at Institute of Computing Technology, Chinese Academy of Sciences from 2002 to 2006. In 2006, he visited Component Software and Enterprise Systems Research Program at Swinburne University of Technology, Australia for half a year. He has published more than 20 articles in refereed international conferences and journals, and one book which is regarded as the first book in Chinese that introduces SOA comprehensively. His current research interests include BPM and Petri Nets, SOA, semantic service repository, and context-aware mobile computing, software engineering.



Yan-Bo Han obtained the Ph.D. degree from the Technical University of Berlin, Germany. He is currently a professor at the Institute of Computing Technology and the Graduate University of Chinese Academy of Sciences, Beijing. His current research interests include distributed systems integration, information fusion, service composition, business

process collaboration and management, situation-aware applications and service personalization, as well as end-user "programming" languages for cooperative applications.



Jun Han received his B.Eng. and M.Eng. degrees in computer science and engineering from the University of Science and Technology of Beijing, China, in 1982 and 1986 respectively, and his Ph.D. degree in computer science from the University of Queensland, Australia, in 1992. He has previously held research and academic positions at the University

of Queensland and Monash University, Australia. He is currently a professor of software engineering at Swinburne University of Technology, Australia, where he directs research into component-based engineering of software and enterprise systems. He is also a research leader with the Australian Cooperative Research Centre in Advanced Automotive Technology (AutoCRC) and the Australian Cooperative Research Centre in Smart Services (Smart Services CRC). He has published over 100 articles in refereed international journals and conferences, including 8 best paper awards at leading international conferences. His research has been supported by the Australian Research Council, the Australian Department of Education, Science and Training, and other government and industry organizations. His current research interests include software architecture design, adaptive software systems, software security engineering, software performance engineering, system integration and interoperability, and services engineering. He is a member of the ACM and IEEE Computer Society.



Yan Jin received his Ph.D. degree in computer science from the University of Adelaide, Australia, in 2004, and his M.Eng. and B.Eng. degrees from Chongqing University, China, in 1997 and 1994, respectively. Until late 2006 he worked as a research fellow in Swinburne University of Technology. He is now an engineer developing software that

allows the user to securely and peer-to-peer sharing their online presence and identity over the hostile Internet. His research interests include software architecture design, component-based software engineering, software performance, and software model checking.



Paolo Falcarin received his M.S. degree in computer science and engineering in 2000, and his Ph.D. degree in software engineering, in 2004, from Politecnico di Torino. He is currently a research assistant in the Software Engineering Group at the Department of Computer Science and Automation of Politecnico di Torino, one of the leading engineering universities in Italy. He was involved in EURESCOM P1109

project, working on different service creation technologies and service description languages. In 2003, he has been visiting Ph.D. candidate at ETH Zurich in the Information and Communication Systems Group. He is currently task leader of service creation environment activities in SPICE project, which is part of the FP6 EU research program. He was the program chair of the Workshop on Telecom Service Oriented Architectures, colocated with ICSOC 2007. His current research interests include automated software engineering, telecom service oriented architectures, software modeling, aspect-oriented programming, and rule-based programming.



Maurizio Morisio received his Ph.D. degree in software engineering (1991) and his M.Sc. degree in electronic engineering (1985) from Politecnico di Torino. He has been researcher at Politecnico di Torino (1991~1998), invited researcher at the Experimental Software Engineering Group at the University of Maryland (1999~2000), associate professor at Politecnico di Torino (2001 current). His current

research interests include experimental software engineering, service engineering, software reuse metrics and models, agile methodologies, and commercial off-the-shelf processes and integration. He is a member of the IEEE Computer Society.