Verifying Web Services Composition Based on Hierarchical Colored Petri Nets

YanPing Yang
Computer School, National University of Defense Technology
Changsha, Hunan, P.R.China
yanpingyang@nudt.edu.cn

QingPing Tan
Computer School, National University of Defense Technology
Changsha, Hunan, P.R.China
eric6508@21cn.com

Yong Xiao
PDL, National University of Defense Technology
Changsha, Hunan, P.R.China
yongxiao@nudt.edu.cn

ABSTRACT
Current Web services composition proposals, such as BPML, BPEL, WSCI, and OWL-S, provide notations for describing the control and data flows in Web service collaborations. However, such proposals remain at the descriptive level, without providing any kind of mechanisms or tool support for verifying the composition specified in the proposed notations. In this paper, we present to verify Web services composition by using CP-nets. CP-nets combine the strengths of Petri nets with the expressive power of high-level programming and have sound mathematical semantics. These services composition proposals can be transformed by transformation rules into CP-nets, which can be used to analyze the performance and to investigate behavioral properties such as deadlock or livelock by CP-nets tools.

Categories and Subject Descriptors
.D.2.4 [**Software Engineering**]: Software/Program Verification – *Formal methods, Reliability, Validation.*

General Terms
Reliability, Experimentation, Languages, Verification.

Keywords
Web service, Composition, Verification, Transformation, CP-net.

1. INTRODUCTION
Web Services receive significant research recently from both academia and industry due to its broad applications and flexible architecture supporting re-composition and reconfiguration.

Web services composition is an emerging paradigm for enabling application integration within and across organizational boundaries. Accordingly, a current trend is to express the logic of a composite web service using a business process modeling language tailored for web services. A landscape of such languages such as Business Process Modeling Language (BPML), Business Process Execution Language (BPEL) [17] and Web service Choreography Interface (WSCI) [18] has emerged and is continuously being enriched with new proposals from different vendors and coalitions. Practical experience indicates that the definition of real world Web services composition is a complex and error-prone process. However, all these proposals still remain at the descriptive level, without providing any kind of mechanisms or tool support for verifying the composition specified in the proposed notations.

Therefore, there is a growing interest for the verification techniques which enable designers to test and repair design errors even before actual running of the service, or allow designers to detect erroneous properties (such as deadlock and livelock) and formally verify whether the service process design does have certain desired properties (such as consistency with the conversation protocols of partner service).

In this paper, we want to use Colored Petri Nets (CP-nets) [1] analysis and verification technique to raise the reliability of Web Services composition. CP-nets were formulated by Jensen [1] as a formally founded graphically oriented modeling language. CP-nets are useful for specifying, designing, and analyzing concurrent systems. In contrast to ordinary Petri nets, CP-nets provide a more compact way of modeling complex systems, which makes CP-nets a powerful language for modeling and analyzing industrial-sized systems. This is achieved by combining the strengths of Petri nets with the expressive power of high-level programming languages. Petri nets provide the constructions for specifying synchronization of concurrent processes, and the programming language provides the constructions for specifying and manipulating data values, and the latter point is very important to model business process. For the lack capability of ordinary CP-nets to model the recursive definition among business activities, here we use a kind of high-level CP-net called hierarchical CP-nets.

The basic idea underlying hierarchical CP-nets is to allow the modeler to construct a large model from a number of smaller CP-nets called pages. These pages are then related to each other in a well-defined way as explained below.

In a hierarchical CP-net, it is possible to relate a so-called substitution transition (and its surrounding places) to a separate CP-net called a subpage. A subpage provides a more precise and detailed description of the activity represented by the transition. Each subpage has a number of port places and they constitute the interface through which the subpage communicates with its surroundings. To specify the relationship between a substitution transition and its subpage, we must describe how the port places of the subpage are related to so-called socket places of the

IHIS'05, November 4, 2005, Bremen, Germany.

substitution transition. This is achieved by providing a port assignment. When a port place is assigned to a socket place, the two places become identical. The port place and the socket place are just two different representations of a single conceptual place. More specifically, this means that the port and the socket places always have identical markings. It should be noted that substitution transitions never become enabled and never occur. Substitution transitions work as a macro mechanism. They allow subpages to be conceptually inserted at the position of the substitution transitions – without doing an explicit insertion in the model.

Since Web service modeler may be unfamiliar with CP-nets, we provide translation rules of composition language into CP-nets and the techniques to analyze and verify them for investigating behavioral properties and techniques to simulate them to evaluate the performance of system.

The formal verification of Web Services composition based on transformation requires not only that the target model has the ability to verify the properties we need, but also that the transformation can transform the source model correctly and even effectively. So we verify the transformation itself based on some recognized criteria.

The rest of this paper is organized as follows: our verification approach is presented in section 2; a composition language transformation example is shown in section 3; section 4 describes the verification of transformation itself; section 5 gives the related work and the paper is concluded in section 6.

2. VERIFICATION APPROACH

Figure 1 shows our analyses and verification approach. Web services composition description are translated into CP-nets, the input of Design/CPN[1] or CPN tools[2] , which are two outstanding tools with a variety of analysis techniques and computing tools for CP-nets. The formalization mainly concerns with the translation of composition specification into CP-net models. This is particularly important in discussions with Web services modelers unfamiliar with CP-nets.

The formal verification of Web Services composition based on transformation requires not only that the target model has the ability to verify the properties we need, but also that the transformation can transform the source model correctly and even effectively. So we verify the transformation itself based on the recognized criteria.

CP-net models are executable. This implies that it is possible to investigate the behavior of the system by making simulations of the CP-net model. Simulations can well serve as a basis for investigating the performance of the considered system.

The state space method of CP-nets makes it possible to validate and verify the functional correctness of systems. The state space method relies on the computation of all reachable states and state changes of the system, and is based on an explicit state enumeration. By means of a constructed state space, behavioral properties of the system can be verified. The properties of CP-nets to be checked include boundness, deadlock-freedom, liveness, fairness, home, and application specific properties. The application specific properties are expressed as reachability of CP-nets. The properties of CP-nets mentioned above have their specific meaning in verifying Web services composition:

- ✓ Reachability—The possibility of reaching a given state. By formulate application specific properties as reachability of CP-net models, we can validate whether the models of the composition process can achieve the desired result.
- ✓ Boundness—The maximal and minimal number of tokens which may be located on the individual places in the markings. As the prototype of the composition process, if a place of it is *Control Place,* then the number of tokens it contains is either o or 1, otherwise this indicates errors in the process. If a place of it is a *Message Place*, then boundedness can be used to check whether the buffer overflows or not.
- ✓ Dead Transitions — The transitions which will never be enabled. There are no activities in the process that cannot be realized. If initially dead transitions exist, then the composition process is bad designed.
- ✓ Dead Marking — Markings having no enabled binding element. The final state of process instance is one of dead marking. If the number of dead markings reported by state space analysis tool is more than expected, then there must be errors in the design.
- ✓ Liveness—A set of binding elements remaining active. It is always possible to return to an activity if we wish. For instance, this might allow us to rectify previous mistakes.
- ✓ Home — About markings to which CP-net is always possible to return. It is always possible to return to a state before. For instance, to compare the results of applying different strategies to solve the same problem.
- ✓ Fairness — How often the individual transitions occur. Fairness properties can be used to show the execution

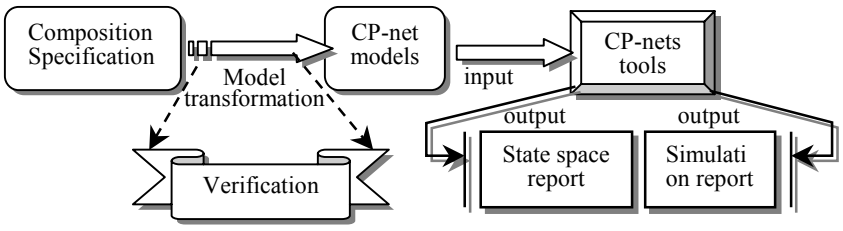

Figure 1. Verification and analysis approach

[1] http://www.daimi.au.dk/designCPN/

[2] http://www.daimi.au.dk/CPNtools/

numbers in each process. We can find the dead activity that will never be executed.

- ✓ Conservation－Tokens never destroyed. Certain tokens such as resources maintained in the system are never destroyed.

2.1 Verification Examples

Our approach is essentially independent of the language describing composition, so we use UML Activity diagram as platform independent model to illustrate Web Services composition. Figure 2 shows a typical Web services composition example for handling a purchase order within a virtual enterprise comprising a Customer, an InvoiceProvider, a ShippingProvider and a SchedulingProvider.

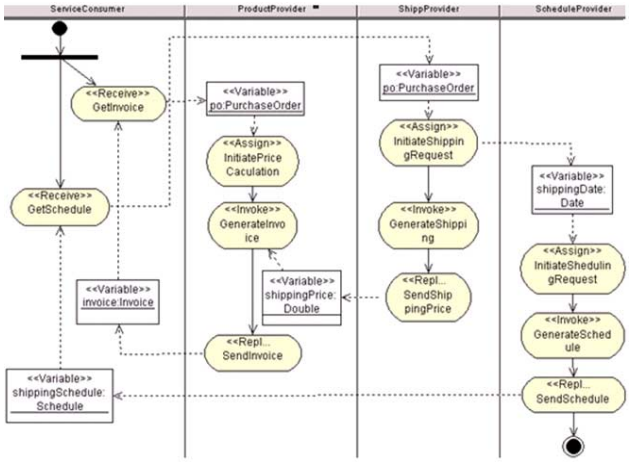

Figure 2. Purchase Order example of composition

On receiving the purchase order from a customer, the process initiates three tasks concurrently: calculating the final price for the order, selecting a shipper, and scheduling the production and shipment for the order. While some of the processing can proceed concurrently, there are control and data dependencies between the three tasks. In particular, the shipping price is required to finalize the price calculation, and the shipping date is required for the complete fulfillment schedule. When the three tasks are completed, invoice processing can proceed and the invoice is sent to the customer.

The corresponding CP-nets model of Purchase Order process is illustrated in Figure 3. The syntactical elements of a CP-net consist of places (drawn as ellipses), transitions (drawn as rectangles), arcs (connecting places and transitions), and inscriptions (text strings associated with places, transitions, and arcs). Places are used to model the state in a system. A state in the context of CP-nets is called a marking which represent a distribution of data values (called tokens) on the places of the CP-net. Each place has a color set (or type) which specifies the colors (or values) of the tokens that the place can hold. The initial distribution of tokens is called the initial marking. Transitions are used to model the dynamics or actions in the system. Arcs pointing to a transition are called input arcs, while arcs pointing from a transition are called output arcs. The arc expressions on input arcs determine what tokens have to be present on input places to enable the transition. The arc expressions on output arcs specify the tokens that will be added to the output places when the transition occurs. In other words, when a transition is enabled, it may occur and thereby remove tokens from the input places as specified by the expressions on the input arcs, and add the tokens to the output places as specified by the expressions on the output arcs.

Here is part of state space analysis result of Purchase Order process generated by State Space tools of CPN tools.

```
Boundedness Properties
-----------------------------------------
  Best Integers Bounds    Upper      Lower
  CustomerSubPage'CS1 1   0          0
  CustomerSubPage'CS2 1   0          0
  CustomerSubPage'CS3 1   0          0
  CustomerSubPage'INV 1   1          1
  CustomerSubPage'PO 1    0          0
  CustomerSubPage'Purchasing__End 1  1    1
  CustomerSubPage'Purchasing__Start 1 0  0
  CustomerSubPage'SCH 1   1          1
  InvoiceSubPage'IS1 1    0          0
  InvoiceSubPage'IS2 1    0          0
  ScheduleSubPage'SD1 1   0          0
  ShipSubPage'SP1 1       0          0
  SuperPage'FL1 1         0          0
  SuperPage'FL2 1         0          0
  SuperPage'FL3 1         0          0
  SuperPage'FL4 1         0          0
  SuperPage'FL5 1         0          0
  SuperPage'FL6 1         0          0
  SuperPage'Input_send_Invoice 1     0     0
  SuperPage'Input_send_PurchaseOrder 1  0  0
  SuperPage'Input_send_Schedule  1   0     0
  SuperPage'Input_send_ShippingPrice 1  0  0
  SuperPage'Input_send_ShippingSchedule1 0 0
  SuperPage'SE1 1         0          0
  SuperPage'SE2 1         0          0

Home Properties
-----------------------------------------
  Home Markings:  All

Liveness Properties
-----------------------------------------
  Dead Markings:  None
  Dead Transitions Instances: None
  Live Transitions Instances: All

Fairness Properties
-----------------------------------------
   No infinite occurrence sequences.
```

Next, consider an example process model that shows a faulty behavior (Figure 4). The model consists of six activities numbered from 1 to 6. The activity 5 has two input control links, coming from the activities 2 and 3, and a join condition of AND. It further has an output data link to the activity 6. The activity 6 has a control link from the activity 4 and two data links from 4 and 5. According to the operational semantics, the activity 5 can be fired when the two input control links have definite values and also the join condition becomes true. The activity 5, after the completion of its internal activity body, puts out some data on the data link to the activity 6.

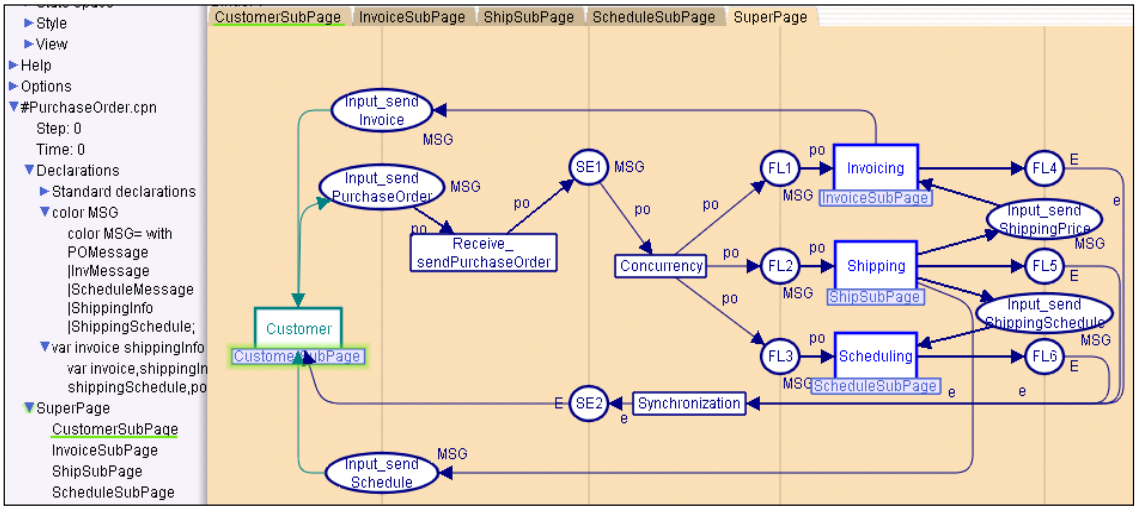

Figure 3. Purchase Order SuperPage

Imagine a situation that one of the input control links to the activity 5 is false and thus the join condition is not satisfied. The activity 5 is not put into execution, and the data link input to the activity 6 does not have a definite value. It results in a faulty situation where the activity 6 waits for the value indefinitely. CP-net tools can detect the faulty situation as a deadlock.

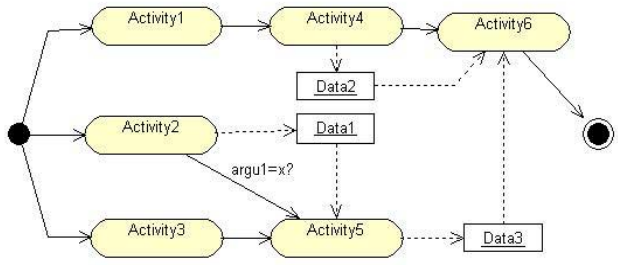

Figure 4. Process with Deadlock

3. TRANSFORING WSCI TO CP-NETS

Our approach is essentially independent of the language describing composition. As an example, to show the effectiveness of our technique, we pick up WSCI and present the transformation from WSCI to CP-nets.

The aim of this section is to provide a transformation from WSCI specification to CP-nets in a constructive way. The overview of the transformation idea can be concluded as follows:

- ✓ The *interface* element describes the WSCI view of a Web service participating in a choreographed, long-lasting and stateful message exchange with other services. Each *interface* is represented by a CP-net called Interface Net (I-Net).
- ✓ Messages are represented by tokens. Different message type can be represented by the products type of the component part type of messages.
- ✓ A WSCI *activity* is usually mapped to a CP-net transition. We do so for several reasons. First, mapping subactivities into places poses the following problem: if a place represents a subactivity state, when the actor returns the subactivity will start again. Thus, to represent the leaving point where a subactivity continues would be impossible. Secondly, the hierarchical modeling technique is by means of substitution transitions, and therefore if a transition represents a subactivity, there always remains the possibility of decomposing it into various actions (other transitions) and resting points (places) that enable interruptions and returns. Thirdly, modeling subactivities by transitions allows us to model data flow in the places of the subactivity flow more clearly. The detailed transformations of activity can be found in [15].
- ✓ The control flow relations between activities specified by WSCI semantics are captured with CP-nets token firing rules and the arc inscriptions and transition guard expressions.
- ✓ Each WSCI *model* is represented by a hierarchical CP-net called Composition Net (C-Net).

3.1 Interface Net

WSCI aims to describe how Web services participate in choreographed, long-lasting and stateful message exchanges. The focus of the described behavior is on the temporal and logical dependencies among the messages the Web service exchanges with one or more other services in the context of a given scenario. WSCI maps this description to the notion of an interface. We transform an interface into an I-Net.

I-Net is a hierarchical CP-net where:

- ✓ *Places are of three different types, specifically control places, referred to as CP, input message places, referred to as IMP, and output message places, referred to as OMP; let us define MP = IMP ∪OMP and P = CP ∪MP;*
- ✓ *Transitions are of three different types. The fist type is auxiliary transition, referred to as AUT, which is used to implement composite construct such as loop or*

conditional fork. The second type is substitution transition, referred to ST, which is abstract representation of subpages modeling sub processes. The third type is activity transition, referred to as ACT; let us define T= AUT ∪ST ∪ACT;

- ✓ *Tokens placed on control (input/output message) places are referred to as control (message, respectively) tokens;*
- ✓ *Arc connected with control (input/output message) places are referred to as control (message, respectively) arcs.*
- ✓ *Each place ∈MP is labeled with a message, i.e., a function mess: MP →ε is defined (ε is the set of messages specified for the Web service, according to the formalization provided in [20])*

3.2 Composition Net

WSCI describes the coordination by means of the WSCI *Model*. The global *Model* is described by a collection of interfaces of the participating services, and a collection of links between the operations of communicating services. Links between operations indicate that the respective services will exchange messages across those links.

Each *Model* can be represented as an Composition Net, which is a specific net connecting at least two I-Nets, and specifying the routing of messages and the act of passing the task of the orchestration from an organization to another one.

Composition Net (C-Net) is a hierarchical CP-net where:

- ✓ *Places are of one type, specifically message places. Each* place *will be assigned to input/output message places belong to different I-Nets.*
- ✓ Transitions *are also of one type, specifically organization transitions. Each transition is an abstract representation of I-Net in the superpage .They are labeled with an organization; the availability of a token in an place connecting the organization transition means that the task of the composition of the overall process is currently assigned to the organization labeling the transition.*
- ✓ *For each place p, there exist omp∈ OMP and imp∈ IMP* such *that p, omp and imp are members of one fusion set, meanwhile omp and imp necessarily belong to different I-Nets.*

4. VERIFYING TRANSFORMATION

The formal verification of Web Services composition based on model transformation requires not only that the target model language has the ability to verify the properties we need, but also that the transformation can transform the source model correctly and even effectively. In [2], the fundamental properties of a correct transformation are summarized as that the transformation should be complete, unique, syntactic correct, semantic correct and could terminate.

- ✓ Syntactic correctness, i.e., to guarantee that the generated model is a syntactically well–formed instance of the target language. Syntactic completeness is to completely cover the source language by transformation rules, i.e., to prove that there exists a corresponding element in the target model for each construct in the source language.
- ✓ Termination, i.e., to guarantee is that a model transformation will terminate.
- ✓ Uniqueness (Confluence, functionality): As non-determinism is frequently used in the specification of model transformations, we must also guarantee that the transformation yields a unique result.
- ✓ Semantic correctness (Dynamic consistency): In theory, a straightforward correctness criterion would require to prove the semantic equivalence of source and target models. However, as model transformations may also define a projection from the source language to the target language (with deliberate loss of information), semantic equivalence between models cannot always be proved. Instead we define correctness properties (which are typically transformation specific) that should be preserved by the transformation.

And for a transformation for verification, the performance requirement that the transformation should be effective should also be considered, i.e. the contents that need to be verified of the source model should be transformed to some properties of the target model that can be analyzed. This additional requirement is correlated with the semantic correctness, because the semantics are usually one part of the contents to be verified.

Because we transform the composition specification through traversing the xml document in a constructive way, the termination, completeness and uniqueness of the transformation can be easily proved. The syntactic correctness can be preserved by checking the generated nets against CP-net metamodel. For the verification technique of the semantic correctness and effectiveness of transformation, we propose to exploit Information Capacity presented in [19] to verify that there is no semantic information lost.

5. RELATED WORKS

Most of existing approaches [3-7] to verify business process are based on model checking techniques.

Nakajima [3] describes how to use the SPIN model checker to verify web service orchestration. In order to do the verification using SPIN, business processes are first translated into Promela, the specification language provided by SPIN. The language used to compose Web Services is the Web Services Flow Language (WSFL)[16] which is one of BPEL's predecessors.

Table 1. The comparison between some existing approaches to business process verification

Researchers	Specification	Formal Model	Tools	Transformation verification?
Koshkina	BPEL	Labelled Transition System	CWB	No
Foster	BPEL	FSP-processes	LTSA toolkit	No
Karamanolis	Abstract business process	FSP-processes	LTSA toolkit	No
Nakajima	WSFL	Promela	SPIN	No
Koehler	Abstract business process	Nondeterministic Automata	NuSMV	No
Stahl	BPEL	Petri net	LoLA	No
Martens	BPEL	Petri net	Wombat4ws	No
Narayanan	DAML-S	Petri net	KarmaSIM	No
Our work	BPEL, WSCI	CP-nets	CPN tools	yes

In [4], Karamanolis and his group translate business processes into FSP processes and use the LTSA toolkit [3] for model checking. The LTSA toolkit allows the user to specify properties in terms of deterministic FSP-processes. Similarly, Foster and his group [5] describe a BPEL plug-in for the LTSA toolkit. They translate BPEL program into FSP-processes and subsequently use the LTSA toolkit to verify the FSP-processes.

In [6], Koehler, Kumaran and Tirenni model business processes as nondeterministic automata with state variables and transition guards. These automata are subsequently translated into the input language of the model checker NuSMV[4]. Koehler et al show how NuSMV can be exploited to detect termination of business processes.

In [7], Koshkina shows how to exploit an existing verification tool CWB[5] supporting techniques like model checking, preorder checking and equivalence checking to model and verify Web Services composition. Similarly, in [8], Schroeder presents a translation of business processes into CCS. Subsequently, the existing verification tool CWB can be used for verification.

Using Petri nets to model and verify business processes is another choice. For the works of modeling business processes by means of Petri nets, we refer the reader to Van der Aalst [9], Martens [10], Narayanan [11] and Stahl [12].

In [9], workflow nets, a class of Petri nets, have been introduced for the representation and verification of workflow processes.

In [10], Axel Martens translate BPEL to a Petri Net semantic. Due to the mapping into Petri nets, several analysis methods are applicable to BPEL processes models: the verification of usability of one Web service, the verification of compatibility of two Web services], the automatic generation of an abstract process model for a given Web service], and the verification of simulation and consistency. All presented algorithms are implemented within the prototype Wombat4ws[6].

In [11], Narayanan and his group take the DAML-S ontology for describing the capabilities of Web services and define the semantics for a relevant subset of DAML-S in terms of a first-order logical language. With the semantics in hand, they encode service descriptions in Petri Net formalism and provide decision procedures for Web service simulation, verification and composition.

In [12], Christian Stahl and his group translate BPEL business process into a pattern-based Petri net semantic. Then they used the tool LoLA [21] for validating the semantic as well as for proving relevant properties of the particular process.

In [13], Adam and his group develop a Petri net-based approach that uses several structural properties for identifying inconsistent dependency specification in a workflow, testing for its safe termination, and checking for the feasibility of its execution for a given starting time when temporal constraints are present. However, the approach is restricted to acyclic workflows.

As we say above, because CP-nets combine the strengths of Petri nets with the expressive power of high-level programming languages, we claim that using the colored token of CP-nets to model different message and event of business process are more natural. Nevertheless, these previous approaches all does not refer to the verification of transformation. Table 1 shows the comparison between some existing approaches to business process verification.

6. CONCLUSIONS

In this paper, we introduce an approach to verify and analyze Web Services composition based on transformation composition specification to CP-nets. These generated CP-net models can be analyzed, verified and simulated as prototypes of the former by many existing and specialized analysis and verification tools. As a correct transformation, we regard the semantic correctness and effectiveness as the fundamental requirement for transformation

[3] http://www.doc.ic.ac.uk/jnm/book/ltsa/LTSA.html

[4] http://nusmv.irst.itc.it/

[5] http://homepages.inf.ed.ac.uk/perdita/cwb

[6] http://www.informatik.hu-berlin.de/top/wombat/

for verification, besides the completeness, uniqueness, termination and syntactic correctness.

We have analyze and verify the description written in BPEL and WSCI [14,15] using the approach reported in this paper. As future work, the back-annotation techniques from CP-nets are being considered.As future work, the back-annotation techniques from CP-nets are being considered.

7. ACKNOWLEDGEMENTS

The work reported in this paper is partly funded by National Natural Science Foundation of China under Grant No.90104007, 60233020; the National High-Tech Research and Development Plan of China (863) under Grant No.2001AA113202, 2001AA113190, 2003AA001023.

8. REFERENCES

[1] K. Jensen, “Colored Petri Nets Basic Concepts, Analysis Methods and Practical Use”, Volume 1, 2 and 3, second edition, 1996.

[2] D. Varró, A. Pataricza, “Automated Formal Verification of Model Transformations”, Proceeding of CSDUML 2003: Workshop on Critical Systems Development with UML, October 2003, Technische Universitat Munchen, pp. 63-78.

[3] S. Nakajima, "Verification of Web service flows with model-checking techniques," presented at First International Symposium on Cyber Worlds, 2002.

[4] C. Karamanolis, D. Giannakopoulou, J. Magee, and S.M. Wheater. “Model checking of workflow schemas”. In Proceedings of the 4th International Enterprise Distributed Object Computing Conference, pages 170–179, Makuhari, Japan, September 2000. IEEE.

[5] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based verification of web service composition," presented at Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on, 2003.

[6] J. Koehler, G. Tirenni, and S. Kumaran. “From business process model to consistent implementation: a case study for formal verification methods”, the 6th International Enterprise Distributed Object Computing Conference (EDOC02), Lausanne, September 2002. IEEE CS, pages 96–106.

[7] M. Koshkina. “Verification of business processes for web services”. Master's thesis, York University, 2003.

[8] M. Schroeder. Verification of business processes for a correspondence handling center using CCS. In A.I. Vermesan and F. Coenen, editors, Proceedings of European Symposium on Validation and Verification of Knowledge Based Systems and Components, pages 1–15, Oslo, June 1999. Kluwer.

[9] W.M.P. van der Aalst. “Verification of workflow nets”. In P. Azema and G. Balbo, editors, Proceedings of the 18th International Conference on Applications and Theory in Petri Nets, volume 1248 of Lecture Notes in Computer Science, pages 407{426, Toulouse, June 1997. Springer-Verlag.

[10] A. Martens. “Distributed Business Processes -- Modeling and Verification by help of Web Services”. PhD thesis, Humboldt-Universit¨at zu Berlin, July 2003. Available at www.informatik.hu-berlin.de/top/download/documents/pdf/Mar03.pdf.

[11] S. Narayanan and S. McIlraith, "Analysis and simulation of Web services," Computer Networks, vol. 42, pp. 675-693, 2003.

[12] Christian Stahl. “Transformation von BPEL4WS in Petrinetze”. Diplomarbeit, Humboldt-UniversitÄat zu Berlin, April 2004.

[13] Adam, N., Alturi, V. & Huang, W.-K. (1998), "Modeling and Analysing of Workflows Using Petri Nets", Journal of Intelligent Information Systems 10(2), 131-158.

[14] Y. Yang, Q. Tan, J. Yu, F. Liu, “ Transformation BPEL to CP-Nets for Verifying Web services Composition”, the International Conference on Next generation Web services Practices (NWeSP'05), IEEE Computer Society, Seoul, Korea, August 2005.

[15] Y. Yang, Q. Tan,, Y. Xiao, Verifying Web Services Composition, eCOMO workshop of the 24th International Conference on Conceptual Modeling (ER2005), Klagenfurt, Austria, October 2005, LNCS , Springer-Verlag. Page 358

[16] F. Leymann etc. Web Services Flow language. Available at http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, May 2001.

[17] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S.Weerawarana. Business Process Execution Language for Web Services (BPEL4WS) version 1.1, May 2003.

[18] Web Services Composition Interface 1.0, Available at http://ifr.sap.com/wsci/specification/wsci-spec-10.htm

[19] R. J. Miller, Y. E. Ioannidis, and R.Ramakrishnan, "The Use of Information Capacity in Schema Integration and Translation," Proceedings of the 19th VLDB Conference, 1993.

[20] M. Mecella, B. Pernici, and P. Craca, “Compatibility of Web services in a Cooperative Multi-Platform Environment”, in Proceedings of VLDB-TES 2001, Rome, Italy, 2001.

[21] Karsten Schmidt. Lola --- a low level analyser. In Nielsen, M. and Simpson, D., editors, International Conference on Application and Theory of Petri Nets, LNCS 1825, page 465. Springer-Verlag, 2000.