A Hybrid Instance Migration Approach for Composite Service Evolution

Jianing Zou, Hailong Sun, Xudong Liu, Kun Fang, Jingjing Lin
School of Computer Science and Engineering, Beihang University, Beijing, China
{zoujn, sunhl, liuxd, fangkun, linjj}@act.buaa.edu.cn

***Abstract*—Composite service evolution is one of the most important challenges faced in the field of service composition. And how to migrate running instances to the evolved definition is a critical issue for correct service evolution. In this paper, we proposed a hybrid instance migration approach in the aim of both increasing migration efficiency and flexibility. Based on Single Entry Single Exit fragments and process structure tre, we reduce the change region calculation algorithm's time complexity from exponential time to linear time. Moreover, our instance migration approach also includes data dependence analysis to avoid data flow problems during migration. This makes our approach more practical since data flow correctness preservation is critical in real world application. Finally, prototype system is given and experiments are carried out to prove the feasibility of our approach.**

Keywords- Instance migration; service composition; composite service evolution

I. INTRODUCTION

Service composition is widely considered as an efficient approach to building complex applications through composing loosely coupled component services [1]. Due to the highly dynamic network environment and ever-changing user requirements, it is imperative to maintain the flexibility of composite services. Thus, composite service evolution, which provides an appropriate solution to enhancing flexibility, has attracted many researchers' attention in recent years. Composite service evolution includes component service changeability and structural adaptation of process models [2]. In this paper, we focused on structural adaptation issue in composite service evolution.

How to migrate running instance during composite service evolution is an important challenge e.g., under the scenario that imperative government policy or business law changes happen. Besides, when the evolved composite service has a long running lifecycle, it is not acceptable to use other techniques such as version management to deal with running instances' evolution problem. For example, for a mortgage composite service whose execution cycle lasts several decades, it is unreasonable to maintain each instance's execution on its original version when the composite service changes each month. This will result in too many versions existing in the system, and make management quite complicated. However, live instance migration can be adopted to lower the complexity of runtime management as well as enhance the composite service flexibility in coping with changes. Therefore, instance migration has attracted more and more researchers' attention in service computing.

Instance migration problem includes control flow and data flow correctness preservation. The former one is the main focus of most published research. As what is mentioned in [9], control flow correctness mainly aims for maintaining the soundness during migration i.e., migrated instance won't result in execution deadlock or improper termination. Existing approaches solving this problem can be classified into two streams: one based on change region computation between old and new composite service models [6][9]; The other based on compliance notion to find an equivalent state of the instance on the evolved model [1][12]. Adopting the first approach will reduce the time of migration determination, because all running instance of the same composite service model will share the same change region calculation result. However, it sacrifices migration flexibility by forbidding any migration of an instance entering the change region which may not break the soundness. The more live instances are forbidden to get migrated, the bigger waste of time is needed in dealing with composite service evolution, because all instances should be rolled back and redone and these work are not necessary if the migration approach is flexible enough. Besides, the complexity of change region calculation i.e., $O(n^4(n!)^2)$ is quite high [9]. Thus the calculation of the change region between original and evolved composite service model restricts efficient instance migration during evolution. The second approach based on instance compliance determination is more flexible than the first approach, such as tolerating changes in the loop or deletion changes during migration. But the restriction of this approach is that it inevitably faces the state explosion problem when the composite service model grows complex. When the nodes number of the model exceeds 50, it takes minutes to determine whether valid instance migration exists. Considering the possibility of existing large number of live instances in a system whose underlining model is evolving, the total migration time may become really long. Therefore, we propose a hybrid instance migration approach by combining the advantages of these two mainstream approaches in order to increasing the migration efficiency as well as its flexibility. We adopt the control flow analysis approach of decomposing the composite service model into a Single Entry Single Exit (SESE) fragment set and constructing a process structure tree (PST) based on that set. Chang region is calculated by identifying the changed SESE fragments in the PST, thus reducing the algorithm complexity to linear time [5]. Only instance running inside of a change region need individual migration determination. And its instance log is replayed on the local reachability graph of the entered change region, thus reducing the

 ISBN: 978-1-61208-105-2

possibility of state explosion since the input model size of reachability graph calculation algorithm is much smaller.

Data flow correctness preservation is another inevitable aspect of instance migration problem. However its solution is seldomly discussed until now. This makes current instance migration solution fail to avoid the potential data flow flaws such as data missing or data mismatch. Recent work [1] points out that control flow change may also bring in data flow change during composite service evolution. However it does not illustrate the problem that dynamic instance migration may also result in data flow problems. In this paper, we elaborated the potential data flow problems during instance migration and propose a solution of combining change regions based on data dependence analysis. In this way data flow problems during migration can be avoided.

System administrators are the users of this approach, because when they are in charge of maintaining the whole system, they have to deal with the model change problem. This approach will greatly reduce their work of rolling back running instances as well as redoing the existing work. However, the data flow analysis in this paper is still not sophisticated enough for real life application. Concrete data structure analysis should be carried out in order to deal with different types of application in different scenarios. However, the data dependence analysis in this paper can effectively help reducing data flow problems, such as data missing or data mismatch. These problems are all critical problems in applying instance migration into real world application.

The rest of the paper is organized as follows. In Section 2, we discuss the motivation example of this paper. Section 3 introduces the preliminaries. In Section 4, we give the design overview of our hybrid instance migration approach, which is illustrated in detail in Section 5. Section 6 describes prototype demo and experiments. Finally, we wrap up this paper with some conclusions and future work in Section 7 and 8.

II. MOTIVATING EXAMPLE

To motivate our example, we refer to a real business loan application demo cited from IBM company web page. The application's Business Process (BP) model described in BPMN [2] is shown in Figure 1. a After the loan officer receive an loan applicant's detail information, including loan amount, repay plan, personal information, income information, and credit information, the process will split according to the loan amount. If the amount is less than 10,000$, the application is directly passed by a fast track approval. Otherwise, credit check and employment check have to be done before the application is recommended to his loan manager. Then the loan manager will firstly review monthly loan sales activity (i.e., company's current loan sale activity) and then loan history (i.e., past loan sales activity) to assess the bank company's running situation. After that, a loan approval decision will be made, based on the bank's business status as well as the applicant's information. If result gets passed, the notification and loan contract will be generated. In the last step, a reply email will be sent to the loan applicant. During loan composite service evolution, the process model later generates a new version shown in Figure 1. b There are three changes carried out between the two versions of loan model. Firstly, employment check and credit check are now required to proceed in parallel to reduce overall processing time. Secondly, a loan plan adjustment activity is inserted, allowing the loan officer to modify applicant's loan amount or repayment plan during the loan approval time after communicating with the applicant. Thirdly, in order to lower repayment failure, a third party risk check service is inserted before the loan manager manually makes the loan approval decision.

The earlier an instance can be migrated to an evolved composite service model, the more advantage of the new model such as performance improvement and functional adjustment it can enjoy in its future execution. However, not every point in the process model is safe for instance migration. For example, if transferring an instance on an unsafe migration point in Figure 1.a, it will result in execution deadlock after migration, whereas safe migration points won't cause this problem. Therefore, it is critical to find as many as possible safe migration points to enhance composite service flexibility while calculation complexity should not be too high.

III. PRELIMINARIES

A. SESE Fragments and Process Structure Tree

In this paper, we used a process graph V = (N, E) to represent the BP model. A process graph has a finite node set N and control flow set E. N is classified in two types, action nodes and control nodes. Action nodes are in charge of concrete work implementation, such as Service Task in BPMN; whereas control nodes control the execution flow, such as Gateways or Start Event and End Event in BPMN

Process Graph, in general, can be decomposed into Single Entry Single Exit fragments [5]. One decomposition approach is to ensure all composed SESE fragments not overlapping each other on the same hierarchical level. This type of SESE fragment is called *canonical fragment*. Canonical fragments can be organized in a hierarchical way, i.e., a canonical fragment can be divided into child canonical fragments or compose a higher level parent canonical fragment. All Canonical fragments of process graphs in Figure 1. a and Figure 1. b are visualized by a surrounding of dotted lines.

In this way, a process graph can be represented by a process structure tree (PST) [5]. The root of PST is the entire process graph which contains all the canonical fragments. We use *parent(f)* to denote the fragment in PST which directly contain fragment f. Besides, fragments in the process graph have order relation in position between one another. Through a depth first graph searching algorithm, the precedence relation of fragment positions is determined. We use *precede(fx,fy,BP)* to denote that fx's position is always before fy's position in the BP graph regardless of the different depth first searching tree. And *paths(fx,fy,BP)* denotes all possible control flow paths that connect fx and fy in the BP graph.

 ISBN: 978-1-61208-105-2

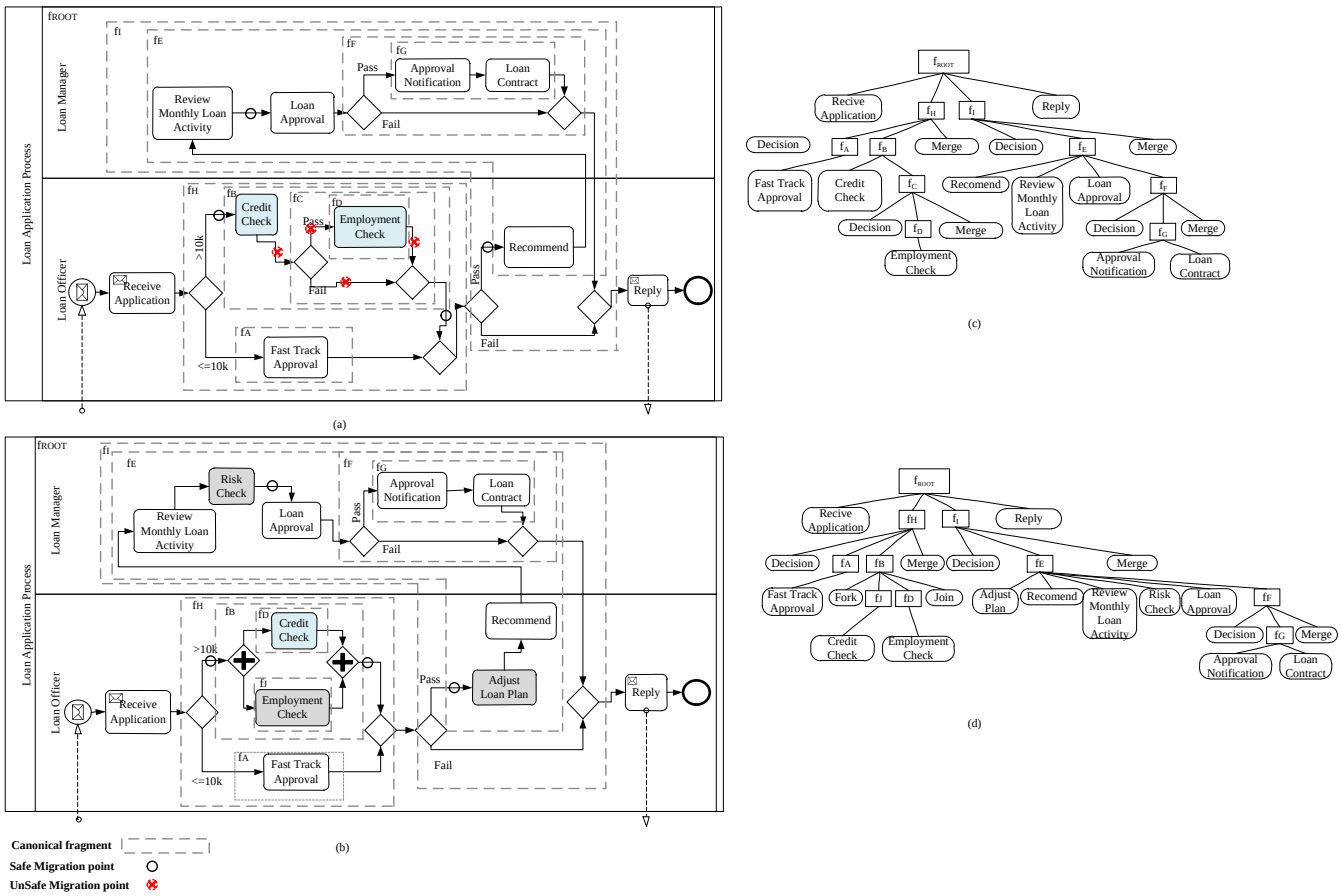

Figure 1. Versions V_1 and V_2 of bank loan business process modeled in BPMN and corresponding PST tree

B. *Soundness of Process Graph and SESE Fragments*

Similar to the soundness definition of a workflow graph, a sound process graph can be easily defined [5]. Process graph soundness ensures the execution correctness of the BP model, including liveness criterion which says each execution can be completed normally and safeness criterion which says each completion of a run is terminated properly with no tokens left inside the graph. Therefore sound process graph is free of execution deadlock or lack of synchronization. Soundness analysis of a process graph is made easier by calculating its PST and component canonical fragment set. According to the theorem 2 in [5], a process (workflow) graph is sound if and only if all its child fragments are sound and the process (workflow) graph that is obtained by replacing each child fragment with an activity is sound. And what's more strictly, if a fragment that is of any of the three types -- well-structured, unstructured concurrent and unstructured sequential fragment -- is sound then all its child fragments are sound. Therefore, we only decompose the process graph in a way that all component fragments are one of those three fragment types. In this way, every component fragments in the PST is guaranteed to be sound.

IV. DESIGN OVERVIEW

Figure 1. shows an overall view of our hybrid instance migration approach. Our approach is divided into two parts, static analysis and runtime analysis. Static analysis generates the changed region between two versions of process model, and mainly includes three steps. Firstly, change operation set is automatically calculated through comparison between the versions of a process model, using the approach proposed in [4]. Secondly, the change region set is calculated by identifying the affected region by each change operation. Thirdly, changed fragments in the changed region are combined according to the data dependence relation among them to avoid data flow flaws during migration.

After getting the change region set during evolution, we enter the runtime analysis part. Every instance on the same process model can use the same static analysis result, i.e., the changed region set, to implement the first step in runtime analysis. Firstly, instance state is collected from the instance log repository. It is compared with the changed region to decide whether the instance has entered a change region. If not so, the instance can resume execution directly on the new model. Otherwise it is required to carry out a compliance determination step which is the second step. In this step, new model's reachability graph is generated (using the approach mentioned in [10]) and a path which is identical with the

migrated instance's log is searched on the reachability graph. If search succeeds, then the instance's state is transformed to the new state where the search stops. This is the third step of runtime analysis. Otherwise, it means the instance is not compliant with the evolved model and will be postponed migration until it steps out of the changed region. This is the last step of runtime analysis

V. HYBRID INSTANCE MIGRATION APPROACH

Dynamic instance migration can easily break the correctness of execution, such as the dynamic change problems mentioned in [6] and data flow flaws mentioned later in this paper. These problems are caused by transferring an instance's running state on one process model to a different process model. From another point of view, this equals to running an instance on a merged process model. The merged model connects the old process faction which is between its start point to the instance migration point and the new process fraction which is between the instance migration point and its end point together. Thus the migrated instance's underlying model correctness (including the control flow and data flow correctness) is very vulnerable to be broken if migration point selection is not controlled. In this section, to ensure the merged process model's correctness, we propose a safe migration point selection approach based on change region set between old and evolved process models. Instances are only allowed to be migrated on these safe points in the aim of avoiding any error of live instance migration.

A. *Soundness Preservation*

If the merged process model with joints on a migration point set between old and new process model is sound, then migration on those points can avoid deadlock or lack of synchronization problems. Analogous to what is mentioned in [9], if migration points are outside the changed region during evolution, then the soundness of merged process model is guaranteed; otherwise migration is quite likely to end in execution error. We propose in this paper that, the changed region that guarantees control flow soundness during migration is the changed canonical fragment set during evolution. Because this approach does not need instance runtime information, it facilitates valid migration determination to be done once for all instances running on the same process model to be migrated, thus saving a lot migration time. When an instance is running outside the changed region, its state can directly mapped onto the new process model without any transformation, and continue execution with the new model without breaking control flow soundness.

The time to carry out migration thus is controlled with the help of *safe migration points* which are outside the changed region. If an instance's running stage does not step on safe migration points, it will continue execution until it is on. Then, the instance will transfer to the new process model and finish execution in the end.

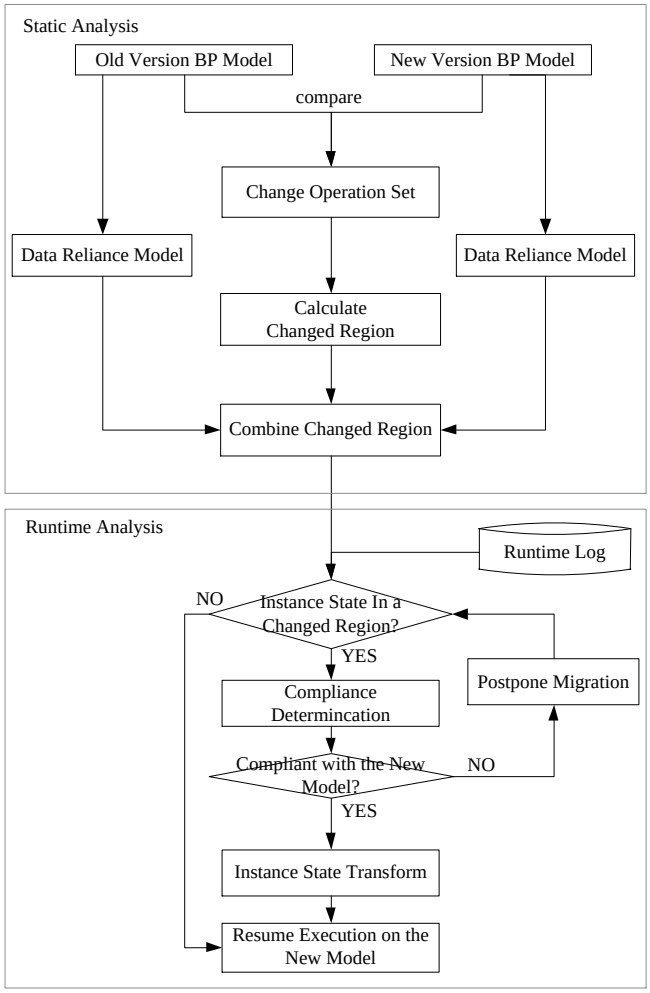

Figure 2. Overview of the hybrid instance migration approach.

B. *Computation of Changed Region*

Changed region calculation is implemented through analyzing change operation set which converts the old process model to the new one. To get the change operation set during evolution, change log can be recorded intentionally or automatically calculated through the approach mentioned in [4]. Because each type of change operation has a specified change effect area, the changed region set between two versions of the process model can be calculated by iterating each change operation in the change log and determining its corresponding effect region. According to what is mentioned in the previous section, the effect region of each change operation will always be the set of canonical fragments in order to guarantee control flow soundness during migration.

In TABLE I. , we enumerate 7 basic types of change operation (that is the same as what is mentioned in [4]) and their corresponding effect region function. Most of the effect region functions are straightforward to understand, such as the insertAction operation will only affect the region on its insertion point, though some may need more detailed explanation. The moveAction(V,x,a,b) operation, for

 ISBN: 978-1-61208-105-2

TABLE I. CHANGE OPERATION AND ITS EFFECT REGION

Change Type	Change Operation	Explain	Effect Region (ER)
Action-oriented	InsertAction(V,x,a,b)	Serially insert a new fragment x between two succeeding nodes a and b in process V. Changes effect is restricted to the insertion point.	ER=paths(a,b)
	DeleteAction(V,x)	Delete an existed action node x in V. Changes effect is restricted to the deletion point.	ER=x
	MoveAction(V,x,a,b)	Move an existing fragment x to the point between two succeeding nodes a and b in V. Change affected region is extended from x to x's new position.	if x < a, ER = paths(x,a) If x > b, ER = paths(b,x)
Fragment-oriented	InsertFragment(V,f1,a,b,f2) The generic operation InsertFragment is realized by: •InsertParallelFragment •InsertAlternativeFragment •InsertSequentialFragment •InsertCyclicFragment •InsertUnstructuredConcurrentFragment •InsertUnstructuredSequentialFragment •InsertComplexFragment	Insert a new fragment f1 between two succeeding nodes a and b in process model V, copying the structure of f2, and reconnection of control flow. Insertion type can vary according to fragment type of f2, such as parallel insertion or sequential insertion. Change affected region is f2 which is the parent fragment of f1 in the new process model.	ER = f2
	DeleteFragment(V,f1)	Delete fragment f1 from process model V.	ER = f1
	MoveFragment(V,f1,a,b)	Analogous to moveAction operation.	if f1 < a, ER = paths(f1,a) If f1 > b, ER = paths(b, f1)

instance, has an effect region extended from the action x's original position to its new position between a and b. If x is moved to a upstream position, then the ER function should equals all the fragments on the path from b to x; otherwise equals all the fragments on the path from x to a. Each change operation and its change effect are elaborated in detail in TABLE I. .

C. Data Flow Correctness Preservation

Uncontrolled instance migration can also bring in data flow flaws, such as data missing, data mismatch. This is because changed fragments may have data dependent relations, like downstream actions reads data written by upstream ones. If this data dependent relation is not considered, migration outside the changed region may cause data flow flaws. Action nodes in the merged model may read data not written by any nodes nor initialized, resulting in a missing data error. In other circumstances, data definition may be changed in the new process model, causing a mismatch between activities who write and read the same data. Thus, in order to ensure data flow correctness during migration, data dependent fragments in the change region set should be combined, i.e., to include data dependent fragments and the area between them in a larger changed fragment. Migration is only allowed to be taken outside the combined changed region in the aim of avoiding any data flow flaw. Data dependence relation between fragments in the changed fragment set is calculated by leveraging the data flow analysis technique as described in [7].

D. State Transformation in Instance Migration

Our hybrid approach allows instance migration in a change region if instance's execution log is compliant with the change region's new process model. If there is one execution path in the new process model that is equivalent with the instance log, then the instance is called compliant with the new model. Based on compliance, migration flexibility is enhanced, thus reducing the cost of aborting and redoing finished works. However, under this circumstance, instance state cannot be directly mapped to the new model and state transformation during migration is necessary [12].

VI. PROTOTYPE AND EXPERIMENT

First, a snapshot of our prototype tool is given in Figure 3. It shows migrating a running instance of bank loan composite service in Figure 1. from model V_1 to a new model version V_2. First of all, the process structure trees of two process models is calculated and shown in the PSTView (in the bottom area) of the prototype tool. Then, change region set is calculated and is depicted by the rectangle box in the right middle of the prototype tool. After that, each instance is determined whether it can be safely migrated with the help of the change region set. For example, the state of the instance in Figure 3. is currently executing exclusive gateway of the process model. This instance is safe to be directly migrated because its state is outside the change regions (i.e., the red boxes). And its new state on the new process model is correctly calculated.

We carried out simulation experiments to demonstrate the performance of our algorithm in practice. In all simulation, we assume that the old and the evolved composite services are sound. The simulation experiment shows the relationship of composite service model complexity (represented by the number of action nodes in the process model) and change region calculation time. We vary the number of action node number in the composite serivce from 12 to 54 with incremental step length of 7. The results are depicted in Figure 4. It is shown that, when process complexity is increased to around 50 action nodes, the

 ISBN: 978-1-61208-105-2

algorithm running time can still be counted by millisecond unit. And the increasing trend is linear, which is consistent with our change region calculation approach complexity. However, we don't give the comparison between change region algorithm in [9] and our approach, because the time complexity of their approach is too high, i.e., $O(n^4(n!)^2)$ and simple process model with only 19 action nodes cost around 0.5 second to compute its change region set. Our change region calculation algorithm is, however, millisecond unit, therefore much more efficient.

VII. RELATED WORK

Existing instance migration approaches focus more on control flow correctness preservation, including change region based approach [6][9] and instance compliance based approach [1][12]. Change region calculation algorithm of former approach is quite slow due to its exponential complexity, thus is the bottleneck of this solution. Approach based on compliance notion has to adopt the reachability graph analysis so that instance log can be replayed and corresponding state can be found on the new definition. But when input model is complex, it will encounter state explosion problem.

Few work until now takes data flow correctness into consideration when dealing with instance migration. Rinderle-Ma et al. give the pre-conditions [1] of each dynamic change operations to protect data flow correctness during composite service evolution. Their description of pre-conditions, however, is too long-winded to express formally and can be quite fault-prone due to manual definition. [13] solves this problem by data dependence analysis, which is similar to our method in this paper.

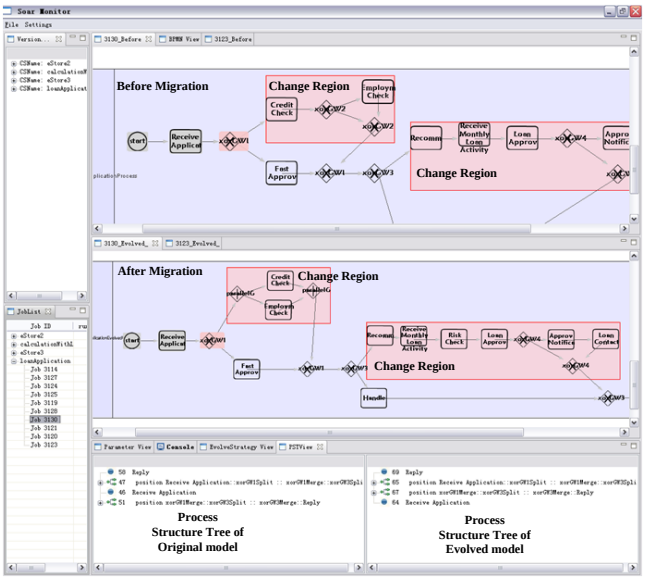

Figure 3. Instance Migration Prototype Tool

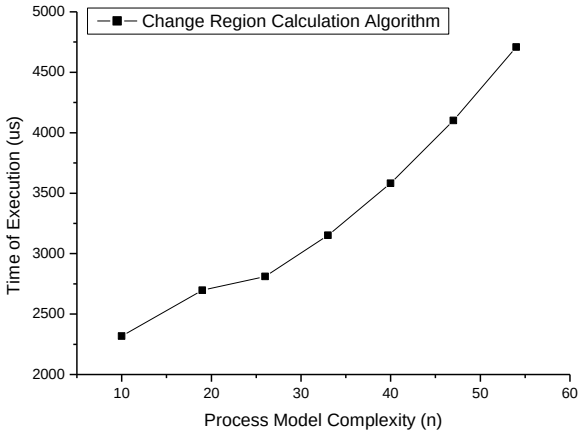

Figure 4. Time Complexity of Change Region Calculation Algorithm.

VIII. CONCLUSION

In this paper, we introduced a hybrid approach which calculates change region set and implements state transformation using reachability graph to solve the instance migration problem in composite service evolution. First, we introduce the SESE fragment and process structure tree definition and propose a change region calculation approach based on process structure tree comparison. This approach is linear time complexity which is proved by experiment results. Second, data flow problems that may occur during live instance migration is elaborated and data dependence analysis is adopted to solve the problem. Finally, we prototype and experiments are performed to show our approach's feasibility and effectiveness. Our future work includes instance migration in composite service protocol evolution.

ACKNOWLEDGMENT

This work was supported by the National High Technology Research and Development Program of China (863 program) under grant 2007AA010301, 2006AA01A106 and 2009AA01Z419.

REFERENCES

[1] L.-J. Zhang, J. Zhang, and H. Cai, *Services Computing*. 2007: Beijing : Tsinghua University Press.

[2] F-Q, Yang., *Thinking on the Development of Software Engineering Technology*. Journal of Software, 2005. **16**(1): pp. 1-7.

[3] Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification.Technical report, Object Management Group (OMG), February 2009.http://www.bpmn.org/..

[4] J. Küster, C. Gerth, A. Förster, and G. Engels, Detecting and resolving process model differences in the absence of a change log, in: M. Dumas, M. Reichert, M.-C. Shan (Eds.), BPM 2008, LNCS, vol. 5240, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 244–260.

[5] J. Vanhatalo, H. Völzer, and F. Leymann, Faster and more focused control-flow analysis for business process models though SESE decomposition, in: B.J.Krämer, K.-J. Lin, P. Narasimhan (Eds.), ICSOC 2007, LNCS, vol. 4749, Springer, 2007, pp. 43–55.

 ISBN: 978-1-61208-105-2

[6] C Ellis, K Keddara, and G Rozenberg. Dynamic change within workflow systems. Proceedings of the Conference on Organizational Computing Systems, Milpitas, California. ACM SIGOIS. New York: ACM Press, 1995:10–21..

[7] S. Moser, A. Martens, K. Gorlach, W. Amme, and A. Godlinski. Advanced verification of distributed ws-bpel business processes incorporating cssa-based data flow analysis. In SCC 2007, pages 98-105, Salt Lake City, Utah, USA, 2007. IEEE Computer Society.

[8] S. Rinderle-Ma, M. Reichert, and B. Weber. Relaxed Compliance Notions in Adapt ive Process Management Systems. in 27th Internat ional Conference on Conceptual Modeling (ER). 2008.

[9] W.M.P. van der Aalst. Exterminat ing the Dynamic Change Bug: A Concrete Approach to Support Workflow Change. Informat ion Systems Front iers 2001, 3(3): pp. 297–317.

[10] X. Ye, J. Zhou, and X. Song, *On reachability graphs of Petri nets.* Computers & Electrical Engineering, 2003. **29**(2): pp. 263-272.

[11] M. Reichert, S. Rinderle-Ma, and P. Dadam: Flexibility in process-aware informat ion systems. LNCS Transact ions on Petri Nets and Other Models of Concurrency(ToPNoC) 2 (2009) 115-135

[12] J Zeng, JP Huai, HL Sun, T Deng, and X Li. LiveMig: An Approach to Live Instance Migrat ion in Composit e Service Evolut ion. in IEEE International Conference on Web Services. 2008.

[13] LH Lam, Q Tang, ZL Zou, L Fong, and D Frank. Identifying Data Constrained Activities for Migration Planning. in IEEE International Conference on Services Computing. 2009

 ISBN: 978-1-61208-105-2