2010 IEEE 6th World Congress on Services

# Live Instance Migration with Data Consistency in Composite Service Evolution

Jianing Zou, Xudong Liu, Hailong Sun, Jin Zeng
School of Computer Science and Engineering, Beihang University
Beijing, China
{zoujn, liuxd, sunhl, zengjin}@act.buaa.edu.cn

***Abstract*—Composite service evolution is one of the most important challenges to deal with in the field of service composition. And how to migrate live instances to the evolved definition is a critical issue for correct service evolution. However, most of current instance migration approaches only consider control flow problem while ignoring data flow correctness during migration. In particular, this paper presents a data consistent approach to instance migration. We first propose a data dependence graph model to represent the data flow information in a composite service. Then, we propose a set of compliance criteria which relax the traditional compliance notion in instance migration through analyzing the data flow of the composite service. In this way, the number of migratable instances is increased, reducing the cost of redoing the case or service compensation. Finally, we design the DCIM algorithm to implement the data consistent instance migration, and an extensive set of simulations are performed to evaluate the algorithm.**

***Instance migration; service composition; composite service evolution; data consistency***

## I. INTRODUCTION

Service composition is widely considered as an efficient approach to building complex applications through composing loosely coupled component services [1]. In general the main exhibitions of composite service evolution are component service changeability and structural adaptation of process models [2]. Here we mainly consider the structural adaptation issue. The structural adaptation aims at improving flexibility through changing the control flow or data flow of the implemented process model. How to dynamically migrate instances that have already begun execution before evolution is a challenging issue in composite service evolution, especially for long running business applications, e.g. banking systems and medical care systems.

For live instance migration in composite service evolution, instance compliance notion [6] is used as the main criterion to determine valid instance migration. There are two approaches to determining instance compliance [6]: graph equivalence [7,8] and trace equivalence [3,9-14]. Graph equivalence approach will reduce migratabe instance numbers significantly, therefore not solving instance migration problem well. Trace equivalence based instance migration approaches [3, 9-14], currently, seldom take data flow information into consideration, thus leaving the possibility of data inconsistency problem.

Data flow correctness preservation is the same critical issue as control flow correctness preservation in real application. Although Rinderle-Ma et al. have also taken the data flow correctness problem [13] into consideration during instance migration, they solve the problem by defining the pre-conditions of each dynamic change operations. However because the operation type set can be large when differentiating the operating scope (such as node/edge/data element) and the operating action (such as add/delete and so on), the description of pre-conditions can be too long-winded to be expressed formally.

In this work, we solve this problem through analyzing the data dependence between component services in the definition, and instance migratability concerning the data can be efficiently calculated without using any pre-defined constrains. Compared with existing approaches, our algorithm can support more live instances safely migrating to a new definition. This is achieved through relaxing the traditional instance compliance criteria to permit safe dynamic structural changes in the past region of a live instance.

Major contributions of this paper are as follows:

- We propose a data dependence graph model to represent the data dependence relationship between component services in a composite service.
- We define a set of instance compliance criteria to judge whether an instance is migratable or not. Especially, with our criteria, we can support the migration of those instances whose past region is changed in the new composite service definition, which is not supported in [13].
- We provide an instance migration algorithm called DCIM, basing on our proposed relaxed compliance criteria to calculate a both control flow and data flow correct instance migration resolution in composite service evolution.

The rest of the paper is organized as follows. In Section 2, we discuss the related work. Section 3 introduces the preliminaries of this paper. In section 4, we give scenarios of data operating bugs in dynamic change operations which will break data consistent status during instance migration. In Section 5, we give three relaxed compliance criteria based on data dependence analysis and then propose the instance migration algorithm concerning data in composite service evolution. Section 6 describes case studies and experiments. Finally, we wrap up this paper with some conclusions and future work in Section 7.

978-0-7695-4129-7/10 $26.00 © 2010 IEEE
DOI 10.1109/SERVICES.2010.76

## II. RELATED WORK

To deal with instance migration problem, two major types of solution have been presented until now: graph equivalence and trace equivalence [6]. An instance can be migrated to a graph equivalent definition without causing control flow error. To describe graph equivalence between the WF schema before and afterwards, WF Net [7] proposed an inheritance relationship definition, while [8] defined a prefix relationship between instance graph and the structural schema after evolution. But, flaws of graph equivalence methods are unacceptable sometimes, because they either sacrifice the dynamic change power to ensure valid migration(e.g. WF Net [7] forbid order-change operations) or disallow any migration of instances whose past region has been different in the new definition (e.g. [8] ). However these prohibited situations are very common in composite service evolution, therefore increasing the cost of compensation or redoing unnecessary works.

In this paper, we extend instance compliance criterion to make more instances migratable during composite service evolution. Besides, basing on a formal modeling of data flow relationship between component services in the definition, we also tackle the problem of maintaining data consistency during instance migration, which is vital for real world applications to implement runtime flexibility.

## III. PRELIMINARIES

In this paper, we adopt trace equivalence based approach to ensure valid instance migration. Once we can replay the instance trace on the new process model, we are sure that an instance is compliant with the new composite service definition and thus can be continued from a new status in the evolved process schema without leading to control flow flaw or data inconsistent problem after migration.

In order to represent the data flow information of the composite service, we use WFD-net (Workflow-net with Data) [15] to formally represent the underlain process model. WFD-net is obtained by include data flow information in a workflow net [17], which is a special kind of Petri-net.

***Definition 1 (WFD-net)***: *A tuple <P,T,D,CF,DF> is a workflow net with data (WFD-net) iff:*

- *<P,T,CF> is a WF-net, with place set P, transition set T and control flow arc set CF;*
- *D is a set of data elements;*
- *DF is the set of data flow arcs between data elements and transitions;*
- *Read DF: D→T represent that transition read data element as input parameter;*
- *Write DF : T→D represent that transition write data element as output parameter;*

In order to ensure data consistency during instance migration, we have to record the data operating behaviors of each finished transition in the instance trace repository. We extend the trace definition given in [13] to include the finished data operations of an instance, in the aim of providing the basis for further data consistency analysis during dynamic migration.

***Definition 2 (Data-extended Trace):*** *Let $\delta_S$ denote the set of all possible traces producible on a WFD-net CS=<P,T,D,CF,DF> . A data-extended trace $\sigma_I^S \in \delta_S$ is defined as $\sigma_I^S =< e_1,....,e_k >$ ( $e_i \in \{Start(t)^{(d_1,v_1),...,(d_n,v_n)}, end(t)^{(d_1,v_1),...,(d_m,v_m)}\}$ where tuple $(d_i, v_i)$ describes a read/write access of transition t on data element $d_i \in D$ in CS with associated value $v_i (i = 0...k)$).*

The temporal order of $e_i$ in $\sigma_I^S$ reflects the order in which transitions were started and/or completed over *CS*. And $\sigma_I^S$ can indicate well the current instance status of both control flow and data flow.

## IV. ANALYSIS OF DATA INCONSISTENCY IN SERVICE EVOLUTION

During composite service structural evolution, process definition is changed by carrying out dynamic change operations. Change patterns are proposed [16] to abstractly describe all the change situations in the real world. Although some high level change patterns can be really complicated (e.g. move a process block), they can be implemented through combination of three basic dynamic change operations: add, delete, and order change. In each dynamic change operation, the data flow correctness can be easily broken, because almost every component service needs to input and output data elements. And, when those changes happen in the past region of an instance, data consistency of finished transitions can be damaged too. This situation is called "change the past"(CP) problem [6] and is prevented or still stay critical in most of the current flexible systems supporting structural evolution. We found that, why CP problem is very dangerous and avoided by most of current applications is because they lack of data flow analysis in their implementation. Therefore, the data depended on by instances cannot be ensured to stay in the correct status after dynamic modification. But, actually if input data state of already finished transitions under the new definition can be guaranteed to be the same with what is in the original definition, and the state of the data depended by the instance is correctly updated in time before resuming execution, we can safely migrate a live instance whose finished region is changed in the new definition. We call these data related problems data operating bugs in dynamic change operations.

## V. RELAXED COMPLIANCE CRITERION CONCERNING DATA FLOW CORRECTNESS

Traditional compliance criterion will unnecessarily restrict the number of migratable instances if the intended structural change affects the past region of those instances. However, if data consistency of each finished transition in an instance is not broken (i.e. the input data state of finished transitions stay the same) during structural modification, that instance can still be safely migrated. Thus tra-

ditional compliance criterion can be relaxed by allowing migrating instances whose past region may have changed in the new definition.

In the following, we relax traditional compliance notion by allowing deleting, order-change, and adding transition in the past region of an instance under the premise of not breaking data consistency during migration.

Our discussion bases on the scenario that, a WFD-net based composite service *CS* is evolved to a new definition *CS'*. And an instance *I*'s data-extended trace is $\sigma_I^S$.

### A. Delete Tolerant Compliance Criterion

Instance *I* can be migrated to a valid status in the evolved composite service *CS'* without causing data inconsistent problem if the transitions deleted in *CS'* (denoted as $T_{del}$) are not relied in data flow by any finished transition in *I* and $\sigma_{I\ dp}^S$ which represent the instance trace purging transitions in $T_{del}$ from $\sigma_I^S$ can be replayed on *CS'*. In this case, input data state of every finished transition in $\sigma_I^S$ will stay the same after migration.

### B. Order-change Tolerant Compliance Criterion

If there exist two finished transition $e_i, e_j$ in $\sigma_I^S$, which are data irrelevant with each other (i.e. the data element set written by $e_i$ has no common part with data element set read by $e_j$), and also the data element written by $e_i$ or $e_j$ are not read by any finished transition in $\sigma_I^S$, then we can switch $e_i, e_j$ in $\sigma_I^S$ to get partially order exchanged trace $\sigma_{I\ poe}^S$. If $\sigma_{I\ poe}^S$ can be replayed on the evolved composite service *CS'*, then *I* can be migrated to a valid status in *CS'* without causing data inconsistent problem.

### C. Add Tolerant Compliance Criterion

If some of the transitions added in the evolved composite service *CS'* (denoted as $T_{add}$) are not relied in data flow by any finished transition in instance *I* and also for each transition $t_{add}$ in $T_{add}$ we can find transition in $\sigma_I^S$ that write the data element read by $t_{add}$, then we can extend $\sigma_I^S$ to $\sigma_{I\ ae}^S$ by inserting transitions contained in $T_{add}$. If $\sigma_{I\ ae}^S$ can be replayed on *CS',* then *I* can be migrated to a valid status in *CS'* without causing data inconsistent problem mentioned.

## VI. Data Consistent Instance Migration

Different from previous instance migration solution, our instance migration approach guarantee the data consistency of both finished and to be executed transitions in an instance. We extend the instance state information that is considered during instance migration to contain the data state information. And also we update the data flow information dynamically and timely during instance evolution. In this way, data inconsistent problems can be effectively avoided during instance migration. We give a concrete implementation of data consistent instance migration -- DCIM Algorithm. It takes the old process definition $WFDNet^0$ and the evolved process definition $WFDNet^N$ as well as instance status $IS^0$ and data-extended trace $\sigma_I^S$ as input. If a data consistent instance migration exists, DCIM Algorithm returns the new instance status $IS^N$ as well as the newly added transitions' input data value set $NTSet$. If data consistent instance migration doesn't exist, it returns nothing.

DCIM Algorithm contains four main parts: marking status transition; implement order-change tolerant compliant migration; implement delete tolerant compliant migration; implement add tolerant compliant migration.

After getting the data consistent instance status $IS^N$ after migration, instance will first append executing newly added transitions in *CS'* with their input data value stored in $NTSet$. Then it can resume execution on the new definition CS'.

## VII. Case Study and Experiment

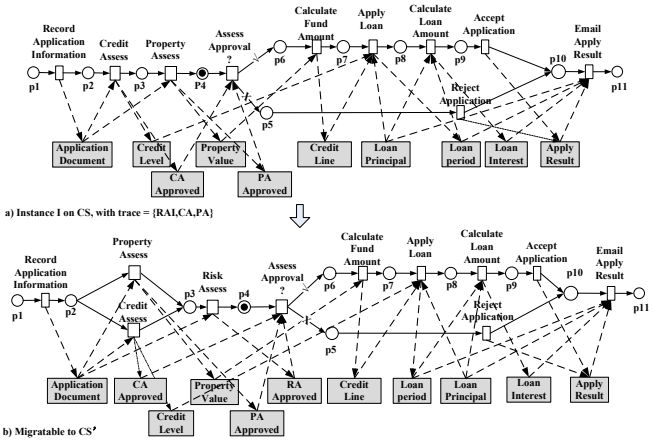


Figure 1. A bank loan application implemented through service composition

Figure 1 shows an evolution scenario of a bank loan application. In order to reduce the application duration and to carry out the business law newly enacted by the government, component services credit assess and property assess in the composite service has been parallelized, and a risk assess component service has been newly added into the composite service. Our approach permit Instance *I* in Figure 1.a being safely migrated to the new definition, which is prohibited in traditional compliance criterion. With appending execution of component service RA before resuming executing on the new definition with marking status – one token in place p4. This migration is data consistent afterwards.

We carried out simulation experiments to demonstrate the performance of our algorithm in practice. The simulation experiment shows the relationship of data flow complexity (represented by the number of data elements and the average number of data operations related to each data element) and migration time. The result is calculated by

averaging the randomly generated data operating conditions with each data element at least being read twice and written twice. When the data element number is fixed, we vary the instance log length (i.e. finished transition number in the instance) from 0 to 18 to get various results in Figure 2. They show that the growth of execution time of our algorithm while increasing the data flow complexity in the definition is quite slow. And while increasing the length of instance log, the execution time growth is exponential. This result matches the DCIM algorithm implementation, because the calculation of reachable graph in DCIM is exponential. Some data point in the growth line is quite high when comparing to the normal trend. This is because some dynamic change operations can cause the instance migration time increasing more rapidly than others.

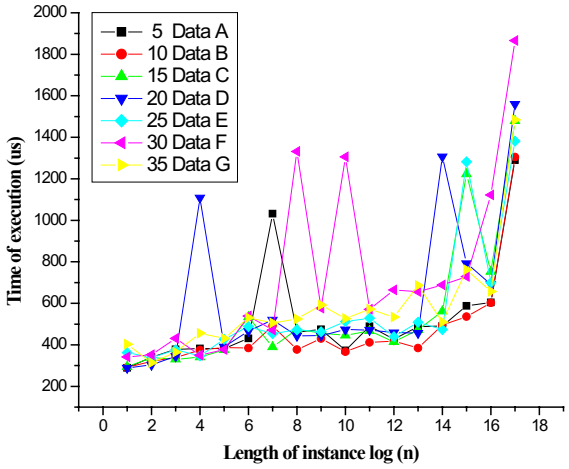


Figure 2. Time VS. Dataflow Complexity

## VIII. CONCLUSION

In this paper, we introduce DCIM, an algorithm to dynamically migrate instance in composite service evolution without breaking data consistency. First, we introduce data operating bugs in dynamic change operations that will damage data consistency preservation during migration and the check criteria to avoid those bugs. Second, based on data dependency analysis we relax traditional instance compliance criteria to allow more instances migratable to the evolved definition. Finally, we give the DCIM algorithm and experiments are performed to show its feasibility and effectiveness. Our future work includes data formats consideration in composite service evolution and instance migration application in business process management area.

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

[2] Fu-Qing, Y., Thinking on the Development of Software Engineering Technology. Journal of Software, 2005. 16(1): p. 1-7.

[3] Jin Zeng, Jinpeng Huai, Hailong Sun, Ting Deng and Xiang Li. LiveMig: An Approach to Live Instance Migration in Composite Service Evolution. in IEEE International Conference on Web Services. 2008.

[4] W M P van der Aalst and S Jablonski. Dealing with workflow change: identification of issues and solutions. International Journal of Computer Systems Science & Engineering. 2000. 15(5): p. 267–276.

[5] Sun, P., C.-j. Jiang, and X.-m. Li, Workflow Process Anal-ysis Responding to Structural Changes. Journal of System Simulation, Apr., 2008. 20(7): p. 1856-1863.

[6] Stefanie Rinderle, Manfred Reichert and Peter Dadam. Correctness criteria for dynamic changes in workflow systems—a survey. Data & Knowledge Engineering, 2004. p. 9-34.

[7] W.M.P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. Theoretical Computer Science 2002. p. 125−203

[8] Mathias Weske, Westf˙alische Wilhelms, and Steinfurter Straße. Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. in Proceedings of the 34th Hawaii International Co- onference on System Sciences. 2001

[9] Manrred Reichert and Peter Dadam. ADEPTflex—Supporting Dynamic Changes of Workflows Without Losing Contro. Journal of Intelligent Information Systems. 1998. p. 93–129

[10] W Song, XX Ma, J Lu and Jisuanji Xuebao. Instance Migration in Dynamic Evolution of Web Service Compositions. Chinese Journal of Computers. 2009. 32(9)

[11] W.M.P. van der Aalst. Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change. Information Systems Frontiers 2001, 3(3): p. 297–317.

[12] Wei Song, Xiaoxing Ma, Wanchun Dou, and Jian Lü. Toward a Model-based Approach to Dynamic Adaptation of Composite Services. IEEE International Conference on Web Services. 2008.

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

[14] Rinderle-Ma, S., M. Reichert, and B. Weber. Relaxed Compliance Notions in Adaptive Process Management Systems. in 27th International Conference on Conceptual Modeling (ER). 2008.

[15] N. Tr˘cka, W.M.P. van der Aalst, and N. Sidorova. Data-Flow Anti-Patterns: Discovering Data-Flow Errors in Workflows. in 21st International Conference on Advanced Information Systems (CAiSE'09), volume 5565 of LNCS. Springer, 2009.

[16] Barbara Weber, Manfred Reichert, and Stefanie Rinderle-Ma. Change patterns and change support features – Enhancing flexibility in process-aware information systems. Data & Knowledge Engineering 2008. p. 438−466

[17] Aalst, W.v.d. and K.v. Hee, Workflow Management Mod-els, Methods, and Systems. 2002, Massachusetts London, England: The MIT Press Cambridge.