A Case Study of The Intelligent Process Decentralization Method

[1]Faramarz Safi Esfahani, [2]Masrah Azrifah Azmi Murad, [2]Md. Nasir Sulaiman, [2]Nur Izura Udzir

***Abstract*— *several* researches have been conducted to decompose business processes in Service Oriented Architecture (SOA). There exist several methods that encapsulate each activity of a business process in one agent, while other methods focus on fragmenting a business process and encapsulate each fragment in an agent. As the mentioned approaches decompose a business process without considering the adaptability of a process with run-time environment, the intelligent business process decentralization (IPD) has been presented that uses a process mining approach. This novel approach detects the frequent paths of a business process and encapsulates the most relevant activities as agents. Being disseminated on a network, the agents are able to communicate with each other through a middleware. This essay shows how IPD algorithm works and detects the frequent paths of a loan taking process to decompose it.**

***Index Terms*—Adaptive Systems; Service Oriented Architecture; Distributed Orchestrate Engine, Business Process Decomposition, Frequent Path Mining.**

I. INTRODUCTION

According to SOA stack[1], business process logic is divided to orchestration and choreography layers. From business process distribution point of view, choreography layer is instinctively distributed to several distinct business processes communicating with each other and normally run on different workflow engines, whereas orchestration layer is workflow engine centric. It means business processes are executed by an orchestrate engine that is responsible for running the activities of a process. A single engine is usually applied to manage a business process and scalability is satisfied by replicating orchestration engines which do not obviate the problems of centralized engines completely [2].

On one hand, several researches have been done to distribute a business process, but unfortunately there is no strict pattern to distribute a business process accordingly. To be more specific, the question is that how we can distribute a business process and what criteria and patterns can be used to contribute business process distribution. In our previous work [3], three methods of business process distribution have been identified, that are Fully, Semi and Intelligent Process Distribution or FPD, SPD and IPD, respectively.

Fully Process Distribution (FPD) is already introduced in [2, 4] and there also exist several researches [2, 4] to fully distribute a BPEL process to its building activities. Having broken a process to its activities, we are able to encapsulate them into agents whose interactions are handled through a middleware. Fully process distribution, though, improves average execution time, throughput and service delay. Negatively, the huge number of produced agents as well as the number of messages for communication will swamp a run-time environment. As a matter of fact, FPD puts each activity in one agent which is the lowest granularity which results in there being a lot of agents communicating through a middleware. The run-time system also can move these small and light weight agents and put them beside their required resources and decrease the amount of bandwidth that can be occupied by the interaction of agents and resources and it increases the system adaptability.

According to [3], Semi Process Distribution (SPD) contains all methods of process distribution that use different criteria for partitioning a process such as [5] that encapsulates segmented activities together. SPD on one hand, results in more coarse-grained agents that reduce the number of 1) produced agents, and 2) agent interactions. On the other hand, this pattern does not consider the adaptability of a business process with run-time environment. To put it in another way, SPD degrades the adaptability of the system owing to the fact that we cannot put together either relevant agents or an agent along with its required resources due to coarse granularity. SPD, though, increases the resource usage such as memory or processor usage because of the increased size of agents.

Intelligent Process Distribution (IPD) introduced in [3, 6, 7], proposes a process mining approach in which some patterns have been introduced to encapsulate business activities in agents, depending on the previous behavior of process instances. The recommended IPD approach will improve three aspects of system quality. One; is the amelioration of business process adaptability with run-time environment, another; choosing the best agent granularity based on detecting most relevant activities or frequent paths and encapsulating them in agents, a third; is decreasing the resource usage due to reduced and improved number of produced agents and messages.

In this essay we show how IPD method detects the most frequent paths of a business process using a mining approach. The size of a frequent path and subsequently its

[1]Faramarz, Safi Esfahani is with the Department of Software Engineering of Islamic Azad University, Najaf Abad Branch, Esfahan, Iran. He is also a PhD Candidate in the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM), 43400, Selangor, Malaysia. Email: fsafi@acm.org

[2]Assistant Prof. Dr. Masrah Azrifah Azmi Murad, Associate Prof. Dr. Md. Nasir Sulaiman, and Assistant Prof. Dr Nur Izura Udzir are with the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM), 43400, Selangor, Malaysia. Emails: {masrah, nasir, izura}@fsktm.upm.edu.my

equivalent produced agent depends on the level of granularity and minimum required frequency that come from either a Service Level Agreement (SLA) document [8] or run-time environment feedback. Obviously, a larger agent demands more memory without communication messages among its building activities. The detection of relevant activities or correspondingly frequent paths stem from an intelligent method based on business process execution log mining. Finally, the algorithm will be imposed on a loan taking business process as a case study.

II. BACK GROUND AND RELATED WORK

BPEL: The Business Process Execution Language or BPEL briefly supports web services relationships from the following aspects including: message exchange correlation for long running message exchanges, parallel processing of activities, the mapping of data between partner interactions and consistent exception and recovery handling. BPEL activities can be classified as basic activities that perform some primitive operations and structured activities that define the control flow. The key BPEL basic activities are Invoke, Receive, Reply, Assign, Compensate, Compensate-Scope, Empty, Exit, Throw, Re-throw, Validate and Wait whereas the structured BPEL activities are Flow, For-Each, If, Pick, Repeat-Until, Scope, Sequence and While, respectively. In addition, two BPEL models have been identified which are block and graph based. Several prominent companies have implemented block-based model, while graph-based BPEL has been implemented by some of them. The introduced algorithm in this paper supports the block- based style of BPEL[2, 9, 10].

Service Level Agreement: Combining functionalities is not the only requirement for e-Business integration. Non-functional quality requirements must also be met. Service Level Agreements (SLA) capture the mutual responsibilities of the provider of a service and its client with respect to the non-functional properties [8]. SLAs are gaining their importance due to the increasing number of companies conducting business over the Internet, requiring the position of SLAs at organizational boundaries to provide a basis on which to emulate the electronic equivalents of a contract based on business management practices. In addition, in[11], a monitoring method was implemented to show how we can monitor SLAs in an heterogeneous environment. Our work can use this method to control SLAs in run time of an orchestrated workflow engine as future work. Work [12] is a FPD method that has no control on compile time of producing agents and uses a cost function to disseminate agents on network at run-time. Our work produces agents according to the execution history of previous business processes at compile time. Run-time management of our method is in future work and is not comparable to [12] at the moment. Work [3] studied an SLA-driven business process distribution and showed how different distribution policies including FPD, SPD and IPD affect system non-functional factors. Finally, in this paper, an algorithm is presented to detect frequent paths of a business process to decompose it to coarser agents based on granularity level and minimum support that come from a SLA.

Frequent Path Detection and Process Mining: A variety of mining algorithms have been developed to detect frequent paths in different data structures such as graphs and trees, however, none of them have considered mining approaches to business process decomposition and the adaptability of business process with run-time environment as well. Our work also mines process log information to detect frequent paths of a process using a mining method. The final result would be frequent paths and infrequent activities in terms of granularity level (G) and minimum support that both come from an SLA. We use G to provide granular agents commensurate with run-time requirements.

Also, several researches have been conducted towards building models without a priori knowledge, called Process Mining, based on sequences of events. Using process mining, one can look for the presence or absence of certain patterns and deduce some process models from it [13]. The main difference with our work is that we already know the business process description and the structure of executed business process log files as well.

BPEL Decomposition and Interaction Middleware: NINOS [2] uses a Publish/Subscribe[2, 14, 15] messaging service to handle the interaction of agents. In this work, a distributed agent-based orchestration engine is presented in which each activity of a business process encapsulated in an agent and collaborates with other agents in order to execute the whole process. In [4], a LINDA platform [16] used to wrap each activity of a BPEL process in agent and Linda Tuple Space concept is applied to realize the cooperation of agents. These methods are called Fully Process Distributed (FPD) in [3]. A different approach is Semi Process Distribution (SPD) that collapses a business process to partitions according to a variety of criteria. Work[5] partitions a business process so that each partition can be enacted by a different participant. In fact [5] disconnects the partitioning itself from the design of the business process. Furthermore SOA stack supports messaging and [5] uses SOA messaging protocols and WSDL to wire decomposed components. In [17], each partition is detected according to the BPEL roles. In [12], a Control Flow Graph has been used to automatic partitioning of a BPEL process similar to program partitioning in multiprocessors.

All these methods do not have any control on the number of produced agents, granularity as well as adaptation of agents to the run-time environment. IPD [3, 6, 7] uses a mining process method to discover useful patterns to provide suitable agents. In [7], some useful IPD distribution patterns for most salient BPEL activities have been shown and proved. In [6], also a methodology along with an algorithm for using IPD has been presented and in [3] IPD has been studied from an SLA point of view and compared to FPD and SPD methods. In fact, the common problem in [3, 6, 7] is that they have not implemented the idea of IPD and only

the IPD abilities have been introduced. In [18] an implementation of the IPD has been shown and the current paper is an improved version of that work along with a new loan taking process case study. The methods [2-7, 12, 17, 18] are the most relevant works to our approach from a business process decentralization point of view.

III. BASIC DEFINITIONS

To be more specific, the idea of IPD is formulated in this section. We introduce set $A=\{a\}$ as a set of activities and E as a set of edges that are used to make a tree of activities. Based on A and E, $BT=(A,E)$ is defined which stands for a BPEL tree and is described by BPEL language. Furthermore, we consider ET as a tree built from a BPEL execution history log file and definitely $ET \subseteq BT$.

A path is defined as a sequence of activities starting from process root to a leaf. Also, a frequent path is a path that all of its activities are frequent. To realize the concept of frequent path we consider *min_sup* value that shows the minimum value of iteration for each activity to be a frequent activity.

To categorize the activities as well as paths in a BPEL tree several concepts are required including Frequent Activity (FA), Frequent Path (FP), Infrequent Activities (IA) and Infrequent Path (IP) that will be introduced, respectively.

FA is a set of frequent activities as: $FA=\{fa\}=\{a \mid a\in A \wedge (frequency(a)\geq \min\_sup)\}$

The frequency of each activity is calculated depends on the type of activity that is as follows:

$$frequency(a)=\begin{cases}\text{if } chilren(a)=\Phi, \text{ return activity iteration number}\\ \text{else, return max } (frequency(children(a)))\end{cases}$$

FP includes just frequent activities from the root to the leaves of a process tree. We would also like to granulate a $fp\in FP$ in terms of granularity degree, G, which is defined as follows:

$$G=\{G_i \mid G_{Depth-1}>...>G_i>...>G_{root}=G_0\}$$

Accordingly, FP_G is defined as a set of granular frequent paths, fp_G , starting from level *G* of the tree to level TreeDepth-1. Also function *level(G)* returns the nodes in level G of the tree. Should a path include just one frequent activity, it is not considered as a frequent path.

$$FP_G=\{fp_G\}=\{fp(a_i) \mid 0\leq G\leq Depth-1, \forall a_i\in level(G), \mid fp(a_i)\mid >1\}$$

Function $fp(a_i)$ searches for subsequent frequent activities of a node and put them in set fp_{ai} which is according to the following definition:

$$fp(a_i)=\begin{Bmatrix}if\ (children(a_i)=\Phi),\ fp_{ai}=fp_{ai}\cup a_i\\ if\ (children(a_i)\neq\Phi),\ fp_{ai}=fp_{ai}\cup fp(children(a_i))\end{Bmatrix}$$

IA also is a set of infrequent activities as: $IA=\{a_i \mid a_i\in A, frequency(a_i)<\min\_sup\}$.

In addition IP is the union of infrequent activities as well as frequent paths that their cardinality is equal to one: $IP=IA\cup\{fp(a_i) \mid \forall a_i\in A, \mid fp(a_i)\mid =1\}$. Those frequent paths that include just one activity are behaved as infrequent paths as well. It is worth mentioning that all the paths in IP and FP sets are encapsulated in their own dedicated agents, afterwards.

IV. IPD METHOD

In this part, the essential steps of IPD method, node frequency calculation and frequent path mining algorithm to business process decomposition are proposed.

A. IPD BASIC PHASES

The basic steps of the IPD method are introduced in this section.

Phase0 (SLA Driven Initialization): Users are able to define their requirements through *SLAs* including: 1) the minimum frequency or minimum support for all activities. 2) Determining the level of granularity for each frequent path.

Phase1 (Pre-processing): In this phase all noise data must be removed from produced log file.

Phase2 (Tree Construction) includes: 1) the construction of the process tree from a BPEL file. 2) Marking the executed activities according to log file information. Each node's visited counter is incremented on each visit.

Phase3 (Frequency Calculation) includes the calculation of all activities frequency.

Phase4 (Frequent Path Detection and Agent construction) this phase depends on the required granularity *G* and minimum support stem from a *SLA*. It starts from the level G and finds the frequent paths in subsequent layers.

Phase5 (Wiring frequent and infrequent agents): In this phase all agents are being wired so that they can communicate through a middleware. Wiring is not our main concern at the moment.

At the first execution of the algorithm, a tree is built from the execution log of the process and is stored in memory. Then, the frequency of all activities is calculated and later on, according to the pre-determined granularity level and minimum support that come from an SLA, the tree is mined. Just those children of a node that their frequency is smaller or equal to the pre-determined minimum frequency are then selected. The output of the algorithm is the distinct groups of activities which are encapsulated in distinct agents.

```
name: CalculateNodeFrequency;
input: node;
output: node frequency;

If (node.childrenNumber >0)
  begin
       maxFrequecny = 0;
       for each child ∈ node.children()
       begin
            tempFrequency =
            CalculateNodeFrequency(child);//Recursive Call
            if (tempFrequency > maxFrequency)
            begin
                maxFrequency = tempFrequency;
            end.
       end.
       node.frequency = maxFrequency;
       return maxFrequency;
  end.
else
  begin
       node.frequency = node.visitNumber;
       return node.frequency;
  end.
```

Figure1) Node Frequency Calculation Algorithm

B. CALCULATING THE FREQUENCY OF ACTIVITIES

In order to detect all frequent patterns of a business process, we have to know the execution frequency of the business process activities. The starting point to detect the frequency of activities is to count the number of visits for each activity. To achieve this goal, log files produced by a BPEL engine are used to calculate the number of visits for each node (*visit number*). According to Figure1, the frequent path detection is based on the presumption that if one simple activity is visited *n* times, so its frequency is *n*. While, a complex activity contains a number of simple and complex activities, therefore, its frequency is equal to the frequency of the maximum frequent child. To implement the algorithm a recursive calculate node frequency method has been implemented. It traverses a node to reach its children recursively and finally returns the frequency of the most frequent child.

```
name: LevelNth ;
input: tree, n // Nth level of the tree ;
output: A list containing the nodes of Nth level ;
 travers the tree using Bread First Algorithm
 return a list including nodes in level n;

name: FrequentPathMining ;
input: min_sup ;
output: A list containing groups of frequent paths

calculateNodeFrequency(root);
for each node ∈ levelNth(granularity)
begin
     if (node.frequency >= min_sup)
     begin
        create a new group i;
        //Traversing the node's sub trees
         for each stNode ∈ sub_tree(node)
            if(stNode.frequency >= min_sup);
                add the stNode to group_i;
         if group_i.size()<2 remove the group_i;
     end;
end;
return all groups ;
```

Figure2) Frequent Path Mining Algorithm

C. FREQUENT PATH MINING ALGORITHM

The frequent path mining algorithm is based on two factors including the degree of granularity, G, and the minimum support of the agents that shows the minimum frequency required for an activity to be included in a frequent path. According to Figure2, to obtain the frequent paths in a BPEL log tree; 1) the *frequency* of each node is calculated; 2) The nodes in level *G* is listed using levelNth function; 3) For each sub-tree of nodes in level *G* a new *group* is created and finally; 4) Each sub-tree of nodes in level *G* is *traversed* and those activities that their *frequency* is equal or larger than *min_sup* is selected and added to a relevant *group*.

D. ALGORITHM ANALYSIS

For a business process tree including *n* nodes, the frequent path mining algorithm is consist of three steps including calculating the frequency of the nodes, returning the nodes in level G of the tree and finally traversing the sub trees of the nodes in level G of the tree.

In order to calculate the complexity of the algorithm, the complexity of each step must be calculated individually. In the mentioned steps the traversing of the tree is based on a breadth first algorithm. So, the complexity of the node frequency algorithm is a function of $O(n^{Depth})$. Similarly, the complexity of finding the nodes in level G of the tree is as function of $O(n^{G})$ due to the fact that all the nodes must be traversed to reach the level G. After obtaining the nodes in level G, in the worst case all the nodes in level G are frequent nodes and therefore the entire sub trees have to be traversed and consequently it would be a function of $O(n^{G} \times n^{Depth-G})$. As a result, the final complexity would be $O(n^{Depth})$ in the worst case.

V. EXPERIMENTAL RESULTS

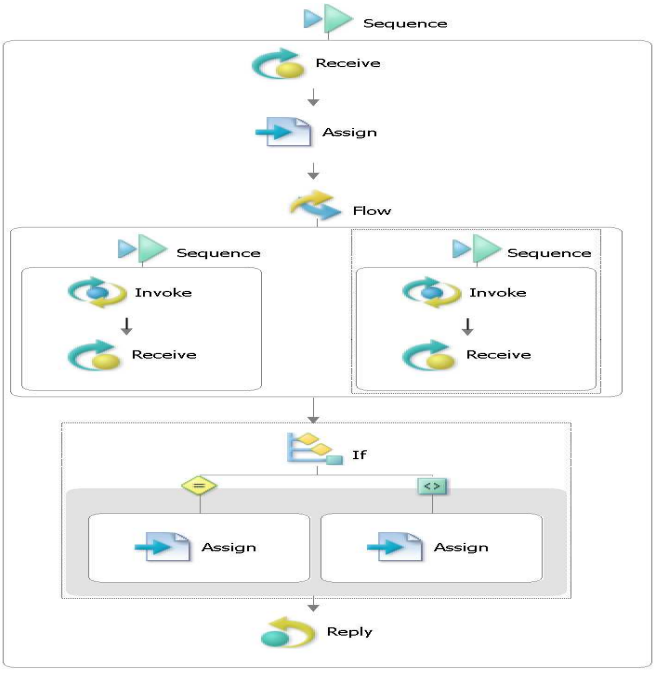
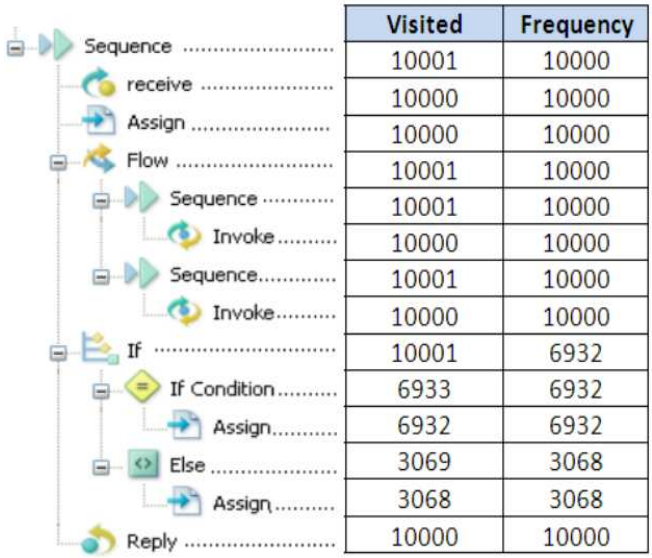


Figure 3) a Loan Taking BPEL Process

```
/process
/process/sequence
/process/sequence/receive
/process/sequence/assign[@name='InputAssign']
/process/sequence/flow
/process/sequence/flow/sequence[3]
/process/sequence/flow/sequence[2]
/process/sequence/flow/sequence[3]/invoke[2]
/process/sequence/flow/sequence[2]/invoke
/process/sequence/flow/sequence[3]
/process/sequence/flow/sequence[2]
/process/sequence/flow
/process/sequence/if
/process/sequence/if/if-condition
/process/sequence/if/if-condition/assign[@name='Yes']
/process/sequence/if/if-condition
/process/sequence/reply
/process/sequence
/process
```

Figure 4) a Log Sample of Loan Process in ActiveBPEL



Visited	Frequency
10001	10000
10000	10000
10000	10000
10001	10000
10001	10000
10000	10000
10001	10000
10000	10000
10001	6932
6933	6932
6932	6932
3069	3068
3068	3068
10000	10000

Figure 5) Loan Taking BPEL Process, a Tree View

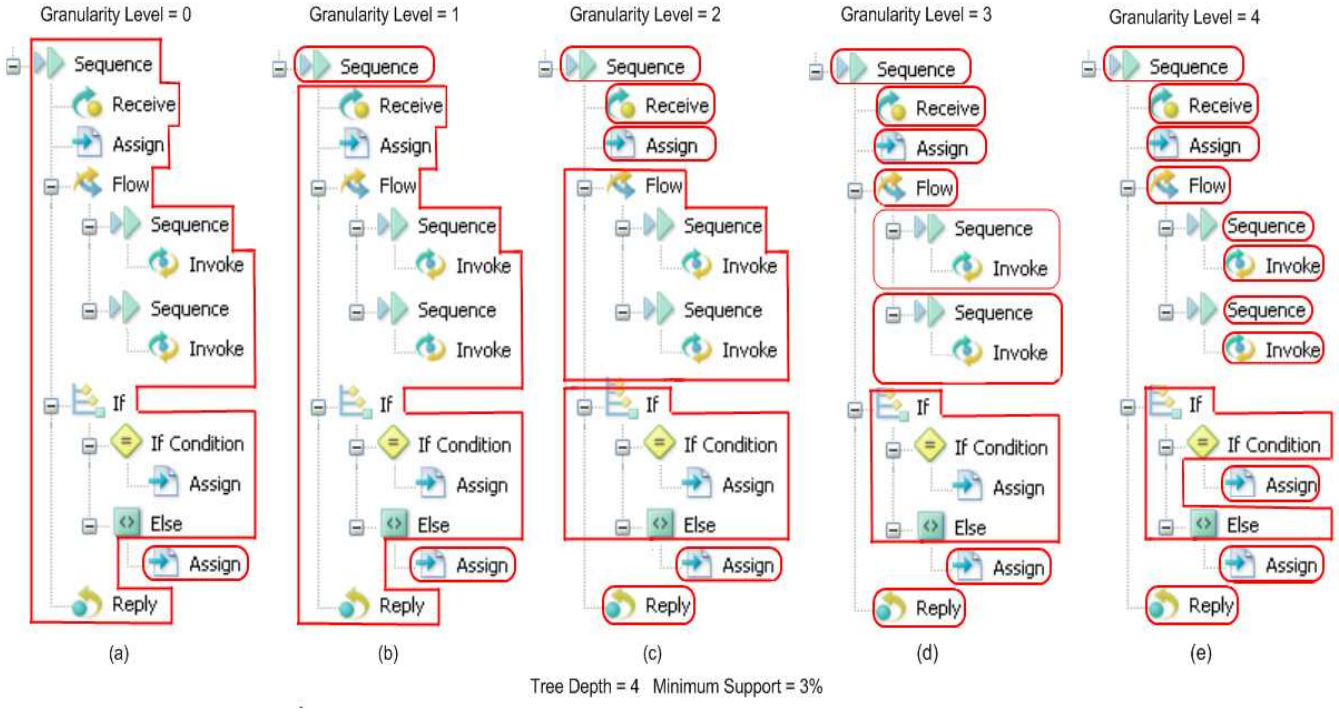


Figure 6) Using Different levels of Granularity

This section illustrates a sample execution of the presented algorithms on a loan taking business process. Receiving a loan request, the process checks the request against two surveyors web services and then upon acceptance of the loan request, issues loan offers. A block-based BPEL illustration of the loan process has been shown in Figure 3 that contains several complex and simple activities including *sequence, flow, if (switch), receive, invoke and assign*.

Precisely, java language was used to implement the algorithms and ActiveBPEL workflow engine[9] and tomcat servlet container [19] to run the loan BPEL process. Preprocessing and omitting the noise information, we got a well formed log file as illustrated in figure 4. Accordingly, each entry in the log file is an activity which is addressed by the hierarchy of the activity.

In this experiment, the business process was called 10000 times so that 70% of the calls resulted in the running of the If-Condition branch of the If activity. Then, the frequent path mining algorithm was run with a minimum-support of 3% and different levels of granularity G, varying from 0 to 4. Figure 5 is the output of the frequent path mining algorithms. Figure 6(a-e) shows how different groups of activities are selected to be encapsulated in agents. Granularity levels 0 to 4 produce two, three, seven, nine and twelve agents, respectively.

VI. CONCLUSION

In this paper, a mining approach has been presented to detect the frequent paths of block-model BPEL specified business processes- the purpose being to decompose a business process which is called Intelligent Processes Decomposition (IPD). Detecting frequent paths is based on the granularity level and minimum support parameters that come from an SLA document, IPD encapsulates them into agents.

The provided agents were expected to be in their best granularity, neither fully distributed nor fully centralized, however commensurate with the run-time behavior of the previously executed business processes.

IPD, though, was tested on a loan taking business process as a case study and resulted in decomposing the loan process to several agents based on the granularity level and minimum support parameters. So, based on different levels of granularity a number of agents were provided.

At the moment, we are evaluating the IPD against other process decomposition methods. In addition, based on granularity level and minimum support a number of agents are produced. The main question is which granularity level would be the best decomposition based on current system configuration. Indeed, we are extending the idea from compile time to run-time. By run-time we mean IPD will automatically reconfigure the process either based on the changes in SLAs or feedbacks from run-time environment.

REFERENCES

[1]IBM, "SOA terminology overview, Part 1: Service, architecture, governance, and business terms," IBM, 2007, pp. http://www.ibm.com/developerworks/webservices/library/ws-soa-term1/.

[2]V. M. Guoli Li, and Hans-Arno Jacobsen, "NiNos: A distributed service oriented architecture for business process execution," *Technical report, Middleware Systems Research Group*, July 2007.

[3]Faramarz Safi Esfahani, Masrah A. A. Murad, Md.Nasir Sulaiman, and N. I. Udzir, "SLA-Driven Business Process Distribution," in *IARIA/eKnow2009* Mexico: IEEE, 2009.

[4]E. D. Mirkov Viroli, Alessandro Ricci, "Engineering a BPEL orchestration engine as a multi-agent system," *Elsevier*, January 2007.

[5]Rania Khalaf and F. Leymann, "E Role-based Decomposition of Businesses using BPEL," in *IEEE International Conference on Web Services(ICWS'06)*, 2006.

[6]Faramarz Safi Esfahani, Masrah A. A. Murad, Md.Nasir Sulaiman, and N. I. Udzir, "Using Process Mining To Business Process Distribution," in *SAC2009* Hawaii/Honolulu: ACM, 2009, pp. 1876-1881.

[7]Faramarz Safi Esfahani, Masrah A. A. Murad, Md.Nasir Sulaiman, and N. I. Udzir, "An Intelligent Business Process Distribution Approach," *Journal of Theoretical and Applied Information Technology* vol. Vol. 4 No. 12, pp. 1236-1245, 31st December 2008.

[8]J. S. D.Davide Lamanna, Wolfgang Emmerich, "SLAng: A Language for Defigning Service Level Agreements," in *The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, 2003.

[9]Active-Endpoints, "ActiveBPEL Engine - Open Source BPEL Server," 2008, p.

http://www.activevos.com/cec/training/content/a_selfPacedTraining.

[10]OASIS, "Advancing Open Standards for the information society," 2007, pp. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[11]M. Funabashi and A. Grzech, *Challenges of Expanding Internet: E-Commerce, E-Business, and E-Government: E-commerce, E-business, and E-government: 5th IFIP Conference on E-Commerce, E-Business, and E-Government (Monitoring Middleware For Service Level Agreements in the Heterogeneous Environment)(13E'2005), October 28-30, 2005, Poznan, Poland*: Springer-Verlag New York Inc, 2005.

[12]M. G. Nanda, S. Chandra, and V. Sarkar, "Decentralizing execution of composite web services," *ACM SIGPLAN Notices*, vol. 39, pp. 170-187, 2004.

[13]"Process Mining and Monitoring Processes and Services: Workshop Report," in *The Role of Business Process in Service Oriented Architectures*, Eindhoven University of Technology, P.O.Box 513, NL-5600 MB, Eindhoven, The Netherlands., 2006.

[14]H.-A. J. F. Fabret, et al, "Filtering algorithms and implementation for very fast publish/subscribe systems," *In ACM SIGMOD*, 2001.

[15]D. S. R. A. Carzaniga, and A. L. Wolf., "Design and evaluation of a wide-area event notification service," *ACM ToCS*, vol. 19(3):, pp. 332–383, Aug. 2001.

[16]N. Carriero and D. Gelernter, "Linda in context," *Communications of the ACM*, vol. 32, pp. 444-458, 1989.

[17]Y. Zhai, H. Su, and S. Zhan, "A Data Flow Optimization Based Approach for BPEL Processes Partition," 2007.

[18]Faramarz Safi Esfahani, Masrah A. A. Murad, Md.Nasir Sulaiman, and N. I. Udzir, "A Frequent Path Detection Method to Intelligent Business Process Decomposition," in *WORLDCOMP/SWWS'09 - The 2009 International Conference on Semantic Web and Web Services*, Las Vegas, Nevada, USA, 2009.

[19]"The Apache Software Foundation - Apache Tomcat," 2009, p. http://tomcat.apache.org/.