The Journal of Systems and Software 84 (2011) 1591–1617

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Adaptable Decentralized Service Oriented Architecture

Faramarz Safi Esfahani[a,b,*], Masrah Azrifah Azmi Murad[b], Md. Nasir B. Sulaiman[b], Nur Izura Udzir[b]

[a] Department of Computer Science, Islamic Azad University, Najafabad Branch, Esfahan, Iran
[b] Faculty of Computer Science and Information Technology, University of Putra Malaysia (UPM), 43400, Selangor, Malaysia

ARTICLE INFO

Article history:
Received 7 April 2010
Received in revised form 12 March 2011
Accepted 13 March 2011
Available online 24 March 2011

Keywords:
Dynamic software
Self-* systems
Self-adaptive systems
Service Oriented Architecture
Workflow Decentralization
Distributed orchestration engine
BPEL
Process mining

ABSTRACT

In the Service Oriented Architecture (SOA), BPEL specified business processes are executed by non-scalable centralized orchestration engines. In order to address the scalability issue, decentralized orchestration engines are applied, which decentralize BPEL processes into static fragments at design time without considering runtime requirements. The fragments are then encapsulated into runtime components such as agents. There are a variety of attitudes towards workflow decentralization; however, only a few of them produce adaptable fragments with runtime environment. In this paper, producing runtime adaptable fragments is presented in two aspects. The first one is *frequent-path* adaptability that is equal to finding closely interrelated activities and encapsulating them in the same fragment to omit the communication cost of the activities. Another aspect is *proportional-fragment* adaptability, which is analogous to the proportionality of produced fragments with number of workflow engine machines. It extenuates the internal communication among the fragments on the same machine. An ever-changing runtime environment along with the mentioned adaptability aspects may result in producing a variety of process versions at runtime. Thus, an Adaptable and Decentralized Workflow Execution Framework (ADWEF) is introduced that proposes an abstraction of adaptable decentralization in the SOA orchestration layer. Furthermore, ADWEF architectures Type-1 and Type-2 are presented to support the execution of fragments created by two decentralization methods, which produce customized fragments known as Hierarchical Process Decentralization (HPD) and Hierarchical Intelligent Process Decentralization (HIPD). However, mapping the current system conditions to a suitable decentralization method is considered as future work. Evaluations of the ADWEF decentralization methods substantiate both adaptability aspects and demonstrate a range of improvements in response-time, throughput, and bandwidth-usage compared to previous methods.

1. Introduction

Business processes might be very large, geographically dependent, long running, carry a vast number of calculations, manipulate a huge amount of data and will eventually be realized as thousands of concurrent process instances. Authors in Li et al. (2010) refer to the applications of business processes in industries such as chain management, online retail, or health care to consist of complex interactions among a large set of geographically distributed services deployed and maintained by various organizations. In addition, an electronic manufacturer is also reported in Li et al. (2010) that employs business processes to conduct its operations, which include component stocking, manufacturing, warehouse, order management, and sales forecasting. There exist geographically distributed parties such as a number of suppliers, several organizational departments, a dozen of sales centers, and many retailers. Indeed, any process in an organization is likely to receive a number of requests from different parties. On the other hand, new software paradigms introduced by the cloud computing (Dimitris, 2011; Ahson and Ilyas, 2011) such as software, platform and infrastructure as services are targeted to receive a huge number of requests. Particularly, in the case of orchestration engine as a service, a large number of workflow instances are requested from various clients all around the world. Consequently, regardless of the type of process, it might be requested from different parties, which may result in creating thousands of concurrent executing instances. In order to handle the requests, different number of machines and also resources must be employed. This paper proposes an adaptable and distributed workflow engine in an ever-changing environment.

According to the SOA stack (Bruni et al., 2006), the business logic consists of orchestration and choreography layers. Choreography layer is intrinsically distributed, whereas the orchestration layer is workflow engine centric. Indeed, a single engine (WMC,

* Corresponding author at: Department of Computer Science, Islamic Azad University, Najafabad Branch, Esfahan, Iran. Tel.: +989131288242.
E-mail addresses: fsafi@iaun.ac.ir (F. Safi Esfahani), masrah@fsktm.upm.edu.my (M.A. Azmi Murad), nasir@fsktm.upm.edu.my (Md.N.B. Sulaiman), izura@fsktm.upm.edu.my (N.I. Udzir).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.03.031

1998) is usually applied to execute a business process and the scalability is naturally addressed by replicating orchestration engines, which is not a final solution for the scalability issue of centralized engines (Li et al., 2010; Silva Filho et al., 2003; Viroli et al., 2007). The decentralization of business processes has been introduced as an alternative solution, which is currently based on system analyst and designers' opinion at design time. However, it did not pay attention to the fact that ever-changing runtime environment raises special requirements, on which there is no information at design time.

As shown in Fig. 1, business process decentralization methods can be studied from three aspects including fragmentation, enactment, and adaptability. A number of decentralized workflow engines have emerged to support these aspects of decentralization. The most challenging area is adaptability that is the ability of system to reconfigure its component to refrain from system bottlenecks such as throughput, response-time, and bandwidth-usage. In order to achieve adaptability, the system must be able to react to runtime circumstances and reconfigure itself. A Fully Process Decentralization (FPD) method is applied by Li et al. (2010), Viroli et al. (2007), Fortino et al. (2006a), Guo et al. (2005), Daniel Wutke (2008), Alonso et al. (1995) and Muthusamy et al. (2009), which decentralizes a business process to activity level fragments. The fragments are encapsulated in runtime components and a third-party middleware is applied to support the communication among fragments. Adaptability also comes about by locating/relocating the runtime components based on system conditions. The FPD negatively produces a number of fragments, in which their message passing and resource usage will eventually result in swamping the runtime environment. The main reason is that the fragments are statically produced without considering runtime circumstances. In contrast, there are several dynamic fragmentation methods (Cai et al., 1996; Tan and Fan, 2007; Atluri et al., 2007), which produce fragments at runtime without considering the runtime circumstances. Thus, the produced fragments may cause violating system thresholds.

In our opinion, the SOA orchestration layer suffers from the lack of decentralization adaptability with runtime environment. Two aspects of runtime adaptable decentralization are studied in this paper. The first aspect is the *frequent-path (frequent paths)* adaptability, which is equal to detecting closely interrelated activities and encapsulating them in the same fragment to omit the communication cost among the activities. The *proportional-fragment (fragment proportionality)* is another adaptability aspect, which is analogous to the proportionality of the number of produced fragments with the number of machines to extenuate the internal communication among fragments on the same machine.

An ever-changing runtime environment along with the mentioned adaptability aspects may result in producing different decentralized versions of a business process at runtime. Considering such dynamicity in workflow execution raises several problems, which can be classified into two categories. The first problem is lack of suitable software architectures to support the execution of different decentralized versions of a business process. Moreover, there is no customizable, adaptable and dynamic workflow decentralization for fragmenting workflows to satisfy adaptability from frequent paths and fragment proportionality points of view.

Having an abstract layer in the SOA architecture to consider the adaptability of decentralization with runtime environment can be a solution for the mentioned problems. Indeed, the main objectives of this work can be categorized as follows. First, to design suitable architectures in order to support the execution of different decentralized versions of business processes. Secondly, to present dynamic and adaptable decentralization methods to produce customized fragments.

In addition, the contributions of this work can be classified as follows:

- Introducing an adaptable and Decentralized Workflow Execution Framework (ADWEF) that presents an abstract layer to isolate design-time and run-time workflow decentralization.
- Presenting two types of ADWEP architectures to support dynamic process decentralization.
- Introducing the *frequent-path* and *proportional-fragment* adaptability aspects.
- Representing two customizable decentralization methods including HPD and HIPD to support the adaptable workflow decentralization.
- Improving response-time, throughput, and bandwidth-usage of workflow fragments by considering the frequent paths and fragment proportionality aspects of adaptability.

Evaluations of the decentralization methods along with the mentioned adaptability aspects demonstrated a range of improvements compared to the previous methods: (A) the frequent-path adaptability along with variable-request-rate/constant-message-size improved response-time and throughput about [71–99%] and [9–54%] in Type-1, [44–87%] and [4–50%] in Type-2; (B) the frequent-path adaptability along with constant-request-rate/variable-message-size improved response-time and throughput about [16–99%] and [1–99%] in Type-1, [47–96%] and [16–66%] in Type-2; (C) both adaptability aspects along with variable-request-rate/constant-message-size, resulted in response-time and throughput improvement [62–99%] and [2–61%] in Type-1, [10–88%] and [5–50%] in Type-2; (D) both adaptability aspects along with constant-request-rate/variable-message-size resulted in response-time and throughput improvement [16–99%] and [1–99%] in Type-1, [26–99%] and [5–93%] in Type-2.

The scope of this work is limited to the shaded area of Fig. 1, which shows three aspects of workflow decentralization including workflow fragmentation, enactment, and adaptability. It is also worth mentioning that some aspects of workflow management systems are not included in this work such as governance of workflow fragments, inter-organizational workflows and security, managing runtime state of workflow fragments, runtime workflow reconfiguration, long-running workflows, transaction, exception handling, and sharing workflow fragments interior variables. These are normally implemented by a distributed middleware (Cai et al., 1996; Tan and Fan, 2007; Atluri et al., 2007) or from the dynamic process decentralization view; they demand more attention in future work as well.

This study is also an extension of our previously published papers, which have introduced dynamic criteria for business process decentralization. In Safi Esfahani et al. (2008, 2009a), merely idea and motivations of using a mining method for Intelligent Process Decentralization (IPD) were introduced and the improvement of only response-time was *mathematically* shown for several sample BPEL processes. Moreover, Safi Esfahani et al. (2009b) showed an SLA-driven aspect of the IPD as well. The current paper in comparison to Safi Esfahani et al. (2008, 2009a,b) is based on a real implementation of two dynamic decentralization criteria for the loan application business process and several empirical results are shown as well. There is no intersection among the mentioned papers and the current one except for introducing the idea of the IPD. Furthermore, Hierarchical Process Decentralization criterion (HPD) and its composition with the IPD for two different case studies were shown in Safi Esfahani et al. (2009c,d) without any empirical result to substantiate the presented methods. In this current study in comparison to Safi Esfahani et al. (2009c,d) an improved algorithm called Hierarchical Intelligent Process Decentralization (HIPD) is demonstrated along with several experiments to show the abilities of two decentralization methods for the loan application BPEL process

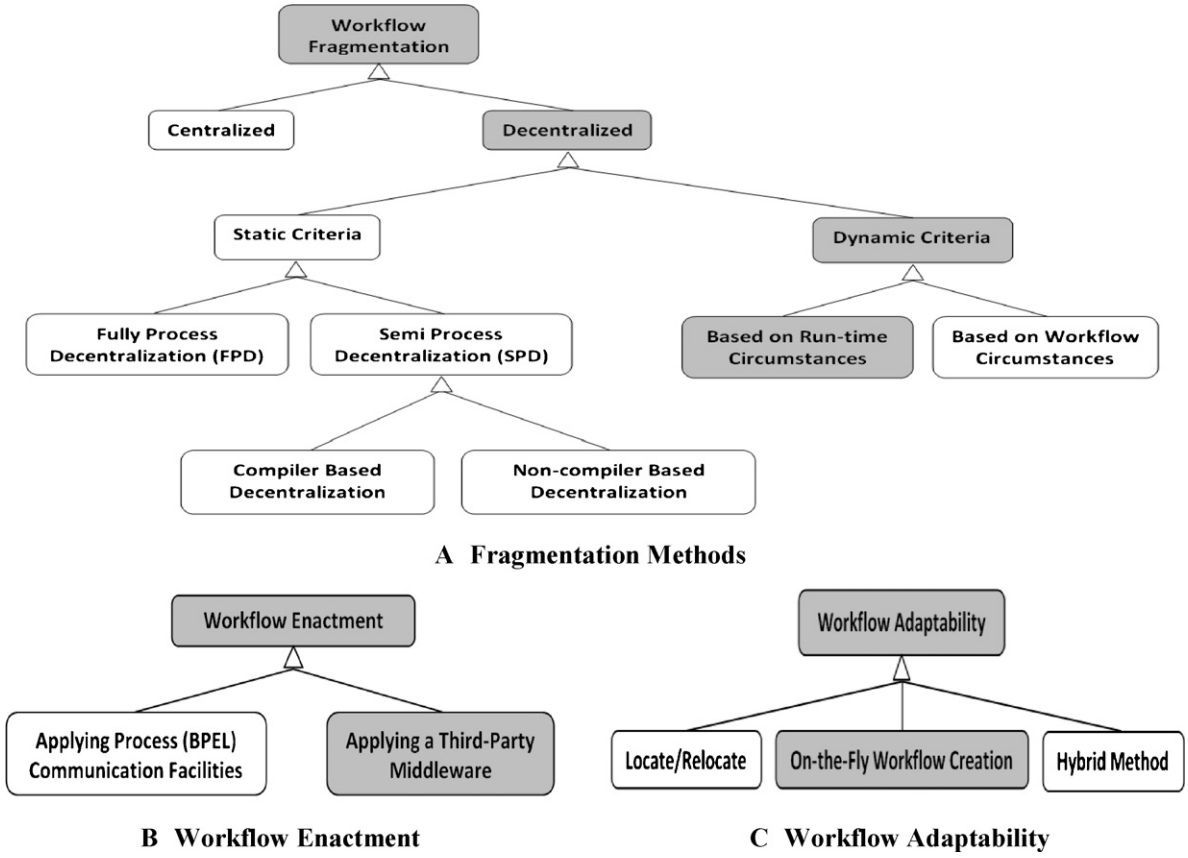

Fig. 1. (A–C) Three aspects of workflow decentralization.

as well. The decentralization methods are also compared to the Centralized and Fully Process Decentralization (FPD) methods. Meanwhile, this paper proposes the idea of adaptable and dynamic decentralization as well as fragment proportionality of decentralization methods, which were never included in the previous works.

From now on, a review of the related work is proposed in Section 2. An Adaptable and Decentralized Workflow Execution Framework (ADWEF) is introduced in Section 3. It is supported by two types of architecture called Type-1 and Type-2. The ADWEF is followed by two decentralization methods, namely HPD and HIPD in Section 4. The setup of experiments environment on the WADE/JADE multi-agent platform (introduced in Appendix A) along with the evaluations of both decentralization methods executed by the architecture Type-1 are presented in Section 5. Furthermore, several boundary-condition experiments as well as the evaluation of the architecture Type-2 are studied in Appendices B and C. Section 6 contains some concluding remarks and future works.

2. Background and related work

This section introduces research activities, which are more relevant to this study. First, block-structured and graph-structured business processes are introduced and the main reasons of focusing on block-structured processes are articulated. Secondly, different methods of business process fragmentation and enactment are elaborated along with the adaptability of decentralization. Finally, due to adapting the workflow fragments with previously behavior of workflows, relevant research studies on detecting frequent paths and process mining are described.

2.1. Block/graph-structured BPEL

Business Process Execution Language or BPEL (OASIS, 2007; ActiveVOS, 2011; Vanhatalo et al., 2009) supports web services relationships and their interactions in business transactions in a loosely couple way to quickly adapt with business environment changes. Briefly, BPEL describes message exchange correlation of long running processes, parallel processing of activities, mapping of data between partner interactions, consistent exception, and recovery handling.

Table 1 classifies BPEL activities (OASIS, 2007; ActiveVOS, 2011) into basic activities to perform some primitive operations and

Table 1
Summary of BPEL activities.

Activity name	Description	Basic/structured
Assign	Manipulating state variables.	Basic
Compensate	Handling faults or exceptions.	Basic
Invoke	Synchronous or asynchronous Web service call.	Basic
Receive	Blocking and wait for a message to arrive.	Basic
Reply	Responding to synchronous operations.	Basic
Terminate	Terminating a process instance.	Basic
Throw	Indicating a fault or exception.	Basic
Wait	Delaying execution for a duration or deadline.	Basic
Flow	Executing activities concurrently.	Structured
Pick	Conditional execution based on events.	Structured
Sequence	Sequential execution of a set of activities.	Structured
Switch	Conditional execution based on instance state.	Structured
While	Looping construct.	Structured

structured activities to define control flow. The key basic activities are Assign, Compensate, Invoke, Receive, Reply, Terminate, Throw, and Wait whereas the key structured activities are Flow, Pick, Sequence, Switch (If), and looping constructs such as While.

Conceptually, workflow languages can be classified according to notions of blocks and graphs (Mendling et al., 2008). In block-structured languages, control flow is defined similar to existing programming languages by using block structures such as If or While in C language. This kind of modeling provides clear semantics to every modeler familiar with basic computer programming languages and transforming it to an executable code is more convenient. In contrast, process control flow in graph-structured workflow languages is defined through explicit control links between activities and is very similar to business process modeling languages such as BPMN (OMG, 2010). Based on Mendling et al. (2008), Kopp et al. (2008) and Garca-Banuelos (2008), all graph-structured models can be mapped to block-structured models and vice versa. Several research studies have tried to convert the graph-structured specified business processes to the block-structured ones (Mendling et al., 2008; Kopp et al., 2008; Garca-Banuelos, 2008). Accordingly, WS-BPEL is essentially a hybrid language derived from a block-structured ancestor XLANG (Vanhatalo et al., 2009; Microsoft, 2001) and a graph-structured ancestor WSFL (IBM, 2001). It allows users to choose between both approaches and even possible to mix both concepts freely (Kopp et al., 2008). Nonetheless, due to the simplicity of block-structured BPEL and its similarities to programming languages several pioneer companies such as Oracle and Sun Microsystems do not support graph-structured BPEL (ActiveVOS, 2011). Indeed, blocked-structured BPEL is worth being a good start point to the decentralization of business processes.

2.2. Business process fragmentation, enactment and adaptability

Business process decentralization methods can be studied from three aspects including fragmentation, enactment, and adaptability. The most relevant parts of each aspect are highlighted in Fig. 1, which are focused on them in this section.

2.2.1. Fragmentation methods

Fragmentation can be centralized or decentralized based on Fig. 1(A). The *centralized fragmentation* of a business process means considering the whole process as a single fragment (WMC, 1998). For *decentralized fragmentation*, there are three approaches. One is applying *compiler techniques* to detect parallel and sequential parts of a process to decentralization. Another is using designer-inspired *static* criteria to business process decentralization at compile/design time. A third is the *dynamic* decentralization based on dynamic criteria, which are determined at/by runtime.

Compiler Based Process Decentralization. In Muth et al. (1998) a centralized workflow-like activity diagram is transformed to several fragments. The main goal is parallelizing the execution of different activities based on compiler-inspired methods. In Nanda and Karnik (2003) and Nanda et al. (2004), a compiler-based method based on program analysis is also introduced. A Control Flow Graph (CFG) is used to automatically partition a BPEL process into parallel routines, which is similar to program partitioning in multiprocessors. Generally, the mentioned compiler-inspired studies can be complementary to our work in that these methods analyze and improve a business process and after that the process can be used in our method to be dynamically decentralized based on runtime conditions.

Static process decentralization methods are classified as *Fully Process Decentralization (FPD)* and *Semi Process Decentralization (SPD)*. The criterion of the FPD methods (Li et al., 2010; Viroli et al., 2007; Fortino et al., 2006a,b; Guo et al., 2005; Daniel Wutke, 2008; Alonso et al., 1995; Muthusamy et al., 2009) is decentralization in the softest activity-level granularity. Having fragmented a process to its building activities, a workflow engine is able to encapsulate the activities into runtime components whose interactions are handled through a middleware. The FPD negatively produces a number of fragments where their message passing and resource usage will eventually result in swamping the runtime environment. The criteria of the SPD methods (Zhai et al., 2007; Daniel Wutke and Leymann, 2009; Khalaf and Leymann, 2006; Fdhila et al., 2009; Sadiq et al., 2006; Khalaf and Leymann, 2008; Yildiz, 2007a,b) for decentralization are varied based on process designers' opinion. The SPD on one hand results in producing coarser runtime components, which reduces the number of (1) Produced fragments and (2) Inter-fragment interactions. On the other hand, this pattern is not runtime adaptable due to static fragments produced at design/compile time. The SPD may also increase memory and/or processor usage due to the increased size of fragments.

Dynamic Process Decentralization means on-the-fly fragmentation of workflows at runtime. An abstract and conceptual dynamic workflow fragmentation method applying Petri net is introduced by Tan and Fan (2007). The presented method partitions a centralized process into fragments step by step while the process is executed. The created fragments can migrate to proper servers where tasks are performed and new fragments are then created and forwarded to other servers to be executed. This work is different from our work in that the decentralization in this method is based on the workflow condition at runtime; however, not runtime environment circumstances. Nonetheless, this method is similar to our work in that the fragments are created on-the-fly at runtime. In Atluri et al. (2007), a decentralized workflow model is presented, which is also a realized and inter-organizational version of the research study (Tan and Fan, 2007). In the presented model, a workflow is divided into partitions called self-describing workflows. A self-describing workflow carries enough information to be handled by a light weight workflow management component called workflow stub located at each organization. This approach partitions a workflow as it progresses with its execution based on a precondition set. As a result, the partitions are sent to an agent only when needed. In addition, this work provides on-the-fly fragments, which is similar to our work; however, it is different in that the partitioning is based on process runtime conditions; while we consider runtime environment circumstances. Dartflow (Cai et al., 1996) project shows the use of mobile agents in distributed workflow execution. A workflow model is fragmented dynamically and the partitions are carried by mobile agents and sent to different sites which are responsible for them. Generally, these research studies establish good points for dynamicity of creating fragments at runtime, managing workflow states, variable sharing and exception handling in a dynamic workflow system, which can be a complementary to our work as well.

2.2.2. Workflow enactment methods

Two workflow enactment methods are shown in Fig. 1(B). The SPD fragmented workflows are usually using business process communication facilities, while other methods use a third-party middleware such as agent based platforms, which is more focused in this section.

Third-Party Middleware Centric Approaches apply other applications to manage the fragments without embedding them in a decentralized process. PADRES publish/subscribe middleware is used in Fidler et al. (2005) to handle the communications of a workflow system. Based on the PADRES, a workflow engine is emerged in Li et al. (2010) namely NINOS. It fragments a business process to the lowest activity level granularity. For each business process based on the used activity type, its respective agent is created and deployed to network machines. As the system is based on the finest gran-

ularity level; therefore, the system is considerably flexible from different aspects. Authors in Muthusamy et al. (2009) try to make NINOS (Li et al., 2010) adaptable with runtime environment and SLA documents. When an agent is under high load, there is a possibility to clone this agent and submit it to another machine. Even, when the number of network machines is incremented or decremented, the system is able to move/remove agents to/from machines.

In Alonso et al. (1995), a process is fully fragmented to the finest level of granularity in an activity level and then distributed to a number of nodes. In this paper, a distributed architecture known as Exotica/FMQM for workflow systems is introduced. This architecture does not keep process state information in a centralized database. However, a kind of message is introduced called persistent message to store state information of business processes. Although this work can be a good model for our future implementation of workflow state management, our focus is on dynamic and runtime based workflow fragmentation.

METEOR project (Miller et al., 1996a) proposes a distributed workflow decentralization architecture. It mostly focuses on system architecture and implementation techniques based on specific communication mechanisms without paying attention to the model fragmentation issues. METEOR2 (Sheth et al., 1996; Das et al., 1997) also introduces a method of process scheduling among various tasks. The scheduling information is distributed among different task managers. Each task manager passes the control to its immediate neighbor once the task it controls terminates. The task manager along with a task activation, task invocation, error handling, and recovery components are all implemented based on CORBA components. In these approaches, a workflow is pre-partitioned in a central server; thus, the partitions are made statically in the central server and distributed to each execution agent. Our work is different in that the fragmentation mechanism is based on runtime feedbacks and cannot be static at all.

A method is proposed by Discenza (2001) to integrate business processes based on a bottom up view and assumes that fragments are already exist. Communication of fragments is handled by providing several event points using a publish/subscribe model. In our work, fragments do not exist previously and are produced dynamically.

SELF-SERV (Benatallah et al., 2003; Sheng et al., 2002) presents a method to implement inter-organizational workflows. In fact, in this method the invocation of composed services are implicitly encoded within processes. State diagrams with transitions and several specific rules called ECA are used to model a workflow. In addition, lightweight coordinators manage the state transition, i.e. when to enter or to finish the state. The composite service state diagram is translated into XML document and control information of each state of the composite state chart is extracted into a routing table and uploaded to hosts of component services. The routing table is used as a knowledge base at runtime by each of the coordinators and contains information of the location, peers, and control flow routing policies. In comparison to our work, the composite service is not fragmented but rather the routing information is extracted for orchestrating the execution of each service.

INCA (Barbara et al., 1996) applies an INCA object to workflow decentralization. An INCA contains the context of a workflow instance including an execution state of the process and distribution of the objects among processing stations. The INCA is then routed among the stations according to the control flow specification stored in the INCA as well as the processing station. The INCA object in each station is executed as a centralized workflow and the lack of parallelism is a drawback of INCA. In our method, there is a dynamic fragmentation and in some cases it might also be possible to create a centralized workflow based on the system circumstances.

Aalst extends WF-net model to the inter-organizational paradigm (Aalst, 2000) based on a top-down view and a global workflow model is made up of several local ones. By transferring an inter-organizational workflow into a WF-net, the correctness issue can be solved easily. In contrast, our work is limited to local workflows and dynamic fragmentation method.

In MOBIS (Baresi et al., 2005; Maurino and Modafferi, 2005), the BPEL4WS schema is an UML activity diagram, which is transformed to a new activity diagram equipped to control delegation rules to coordinate process controllers. The new diagram is divided into sub diagrams that are distributed to mobile devices to be run. This method also uses a static fragmentation before deciding where the fragments run; however, our work prefers runtime based decentralization.

Authors in Viroli et al. (2007) introduce a LINDA (Carriero and Gelernter, 1989) based platform in a conceptual level to wrap each activity of a BPEL process in an agent and the concept of Tuplespace is applied to realize the cooperation of agents. In Viroli et al. (2007) and Fortino et al. (2006a,b), an architecture is also introduced to implement distributed workflow enactment using JADE. In Daniel Wutke and Leymann (2008), Fabio et al. (2001) and Bellifemine et al. (2008), a model and infrastructure based on tuplespace is presented for the same idea. In Guo et al. (2005), a multi agent platform to decentralize business workflows is introduced as well. All of these methods use activity level fragments and are considered equal to the FPD.

2.2.3. Workflow adaptability

Workflow Adaptability with runtime environment means how system is able to reconfigure itself to refrain from or lessen system bottlenecks such as throughput, response-time, and bandwidth-usage. Fig. 1(C) shows three adaptability categories including locate/relocate (Li et al., 2010), on-the-fly and hybrid methods. The locate/relocate based methods such as Li et al. (2010) and Muthusamy et al. (2009) use the FPD method for fragmentation and look for the best runtime component to execute a fragment. Based on the runtime circumstances, the fragments are then relocated (reconfigured) to avoid violating system thresholds. Alternatively, on-the-fly systems create the fragments based on the current system circumstances and the fragments are then reproduced (reconfigured) on-the-fly to refrain from system threshold violation. This paper is classified in the on-the-fly category as well. The on-the-fly workflow creation can be realized by either compile/recompile of workflows or dynamic workflow creation at runtime.

2.2.4. Brief comparison

A brief comparison of the most relevant methods is shown in Table 2 in terms of process representation, decentralized engine, fragmentation method, and adaptability. Only Li et al. (2010) uses the FPD method and executes the fragments at runtime using a third-party multi-agent platform and consider adaptability at runtime along with Muthusamy et al. (2009). Indeed, the FPD and Centralized methods are considered in our experiments for evaluation purposes because (1) the FPD is a subset of our decentralization methods; (2) the FPD is a basis for current dynamic systems, which uses a third-party platform for implementation and reconfiguration and; (3) the centralized method is a traditional method of workflow execution and can also be chosen by adaptable dynamic decentralization methods based on runtime conditions.

2.3. Frequent path detection and process mining

Several research studies have been conducted towards building models without a priori knowledge of process specification called Process Mining that works based on sequences of events. Using

Table 2
Middleware based decentralization comparison.

	Process representation	Decentralized engine	Static/dynamic fragmentation	Adaptability with runtime environment
Alonso et al. (1995)	Compiled process	Distributed Node Manager	Static Fragmentation (FPD)	No
Das et al. (1997), Miller et al. (1996b), Sheth et al. (1996)	Compiled process	Task Manager	Static Fragmentation	No
Muth et al. (1998), Wodtke et al. (1996)	State chart	Workflow Server Engine (for each node)	Static Fragmentation	No
INCA Barbara et al. (1996)	CORBA component	Processing Agent	No Fragmentation	No
SELF-SERV Benatallah et al. (2003), Sheng et al. (2002)	State chart (composite web service)	Coordinator at each peer	No Fragmentation	No
Nanda et al. (2004), Chafle et al. (2004)	Graph-structured BPEL	Web service host	Static Fragmentation	No
MOBIS Baresi et al. (2005)	UML activity chart	Mobile Peer	Static Fragmentation	No
Atluri et al. (2007)	Graph-structured BPEL	Task Agent	Dynamic Fragmentation based on workflow runtime circumstances	No
NINOS Li et al. (2010)	Graph-structured BPEL	Activity Agent	Static Fragmentation (FPD)	Yes
Muthusamy et al. (2009), Tan and Fan (2007)	Petri net	–	Dynamic Fragmentation based on workflow runtime circumstances	No
Viroli et al. (2007)	Graph-structured BPEL	Linda Object	Static Fragmentation (FPD)	No
Viroli et al. (2007), Fortino et al. (2006a)	Graph-structured BPEL	JADE Performer Agent	Static Fragmentation (FPD)	No
Our approach	Block-structured BPEL	WADE/JADE Performer Agent	Dynamic Fragmentation based on runtime environment feedback	Yes

process mining, one can look for the presence or absence of certain patterns and deduce some process models from it (Van der Aalst et al., 2003). However, none of the process mining methods have considered mining approaches to process decentralization and the adaptability of decentralization with runtime environment. In Dubouloz and Toklu (2005), an algorithm is presented to detect the most frequent paths of graph-structured BPEL processes. Our work and Dubouloz and Toklu (2005) are similar in that both of them know the business process description and each record of the process log file addresses an activity execution in a workflow. In our work, there is no novelty in BPEL mining; therefore, any frequent path detection algorithm can be used. Nonetheless, an algorithm is presented to decentralize a block-structured business process considering both frequent path and granularity. In addition, the mining of graph-structured BPEL processes covers the block-structured BPEL; however, the focus is on both decomposing and mining of processes. In this current study, both HPD and HIPD decentralization methods decompose a block-structured process considering process tree levels and in addition to it, the HIPD method mines process log data to detect frequent paths of business processes. Indeed, more attention is paid to the block-structured BPEL and the graph-structured model will be focused in future work.

According to Han and Kamber (2001), several mining approaches are based on *minimum support (min_sup)* concept, which shows the collocation percentage of items in a dataset of transactions. By using *min_sup* a variety of mining algorithms detect iterative patterns and specify items that are frequently associated together. In the case of this paper, the execution iteration of activities in a workflow is important. A *frequent path* is a sequence of workflow activities, which are iterated together and are encapsulated in the same fragment. Each *infrequent path* includes only one activity and is encapsulated in separate fragments individually. Decentralization, therefore, is analogous to fragmenting frequent as well as infrequent paths. Activities inside a fragment communicate using a shared memory while inter-fragments activities communicate through message exchanging.

3. Adaptable and Decentralized Workflow Execution Framework

In this section, an *Adaptable and Decentralized Workflow Execution Framework* (*ADWEF*) is introduced. The ADWEF considers both adaptability and decentralization in the orchestration layer of the SOA stack. First, a three-layered architecture for the SOA orchestration layer is presented. Next, the ADWEF architecture is formalized and then realized by two types of architecture.

3.1. Reference business process

Basically, there is no standard business process to benchmark BPEL processes. Nonetheless, a BPEL specified loan application business process is shown in Fig. 3, which has been applied by several research studies such as Li et al. (2010), Nanda et al. (2004) and Khalaf and Leymann (2006) as a reference model. This process is also used as a running example throughout this paper and paves the way for studying simple and structured BPEL activities together. It includes several structured activities such as Sequence, Flow, If (Switch) and simple activities such as Receive, Invoke, Assign, and Reply. The loan process consults with two external web services which send a credit report for the loan applicant. In order to refrain from approbating risky loans, the loan request is accepted when both web service reports confirm the applicant's credit.

3.2. Adaptable SOA orchestration layer

According to the literature, business processes are mostly decentralized to several fragments called *process fragments* based on a number of criteria that are considered by business process designers. After fragmenting a business process, the produced fragments are correspondingly encapsulated in runtime components such as web services, agents, etc. as shown in Fig. 4.

Generally, business designers do not have a prior knowledge on runtime circumstances and consequently are not able to make

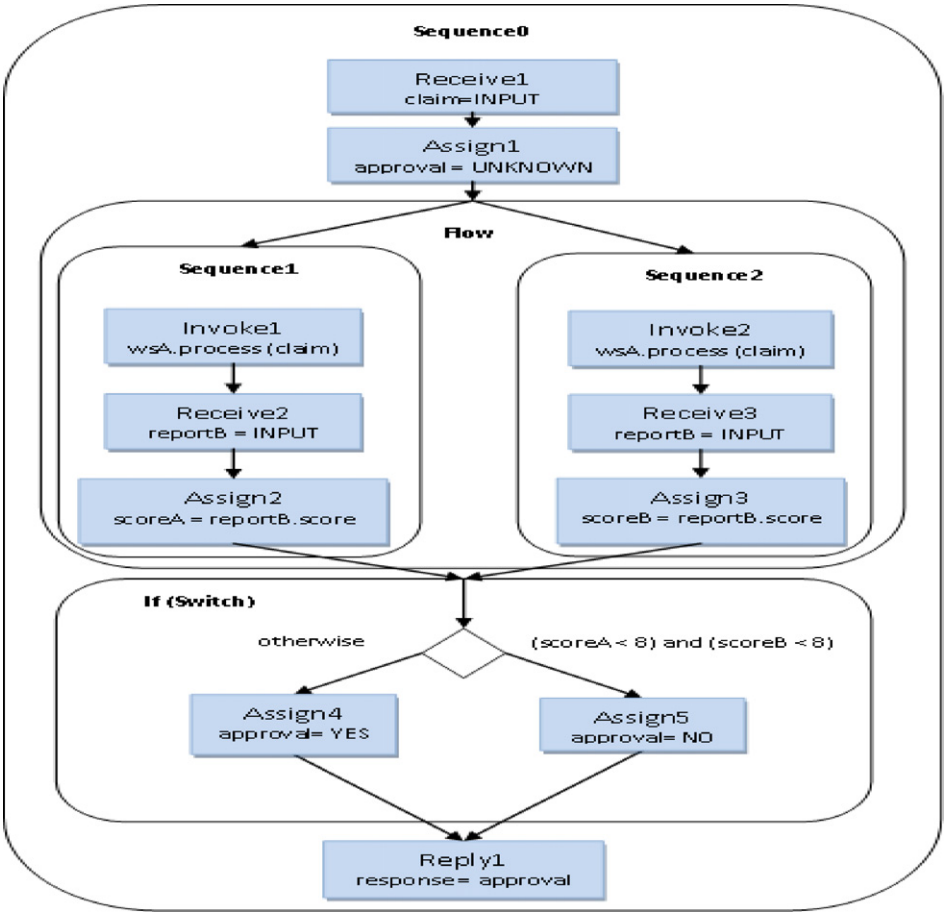

Fig. 2. Recommended abstraction for SOA orchestration layer.

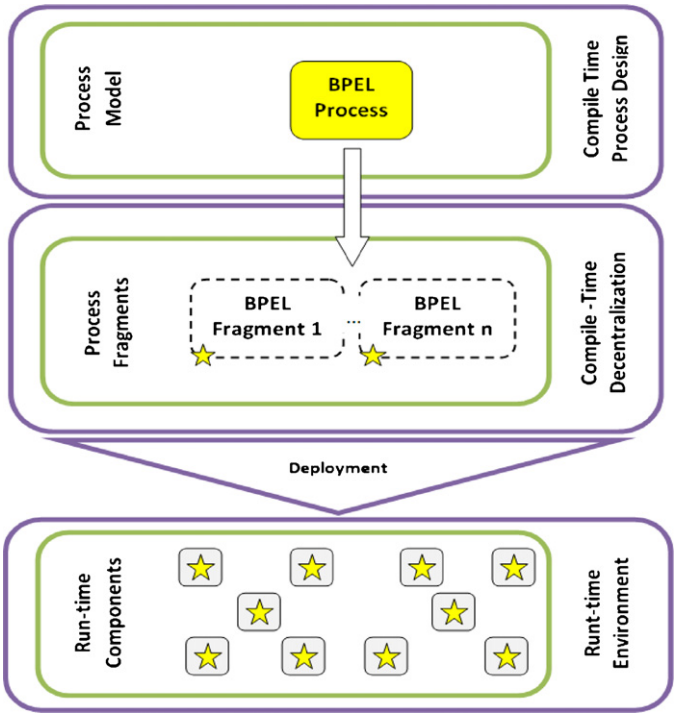

Fig. 3. BPEL view of loan application process.

right decentralization decisions at design time. As runtime circumstances change, a workflow engine has to adapt itself with this ever-changing environment. Currently, decentralization methods suffer from a lack of runtime based workflow decentralization. Indeed, design-time and run-time process decentralization have to be isolated to extenuate designers' role in compile time.

The process of decentralization may be configured so that produced fragments become adaptable with different aspects of runtime environment. For example, when a number of business process activities are closely interrelated, they can be fragmented together. It reduces the amount of message passing among the fragments at runtime. Alternatively, the fragments can be provided based on the number of available machines allocated for a workflow engine. For instance, in the case of one machine, creating one fragment suffices instead of locating a number of fragments on the same machine. It omits internal message passing among the fragments on each machine. More number of fragments can be generated in the case of more machines. The fragments can be shrunk or expanded based on environment conditions to realize adaptability. While a version of a process lives in a single machine, another version of the process may be fragmented and scattered on a number of nodes in a network. Based on environment conditions, they can even toggle their shape from the scattered to the centralized version or from the centralized to the scattered version.

Fig. 5 shows the recommended runtime based decentralization, in which each *process fragment* is converted to several workflow fragments based on runtime circumstances. The *workflow fragment* is different from the *process fragment* in that the former is created based on runtime environment feedbacks, while the latter is decentralized based on designers' opinion. Workflow fragments

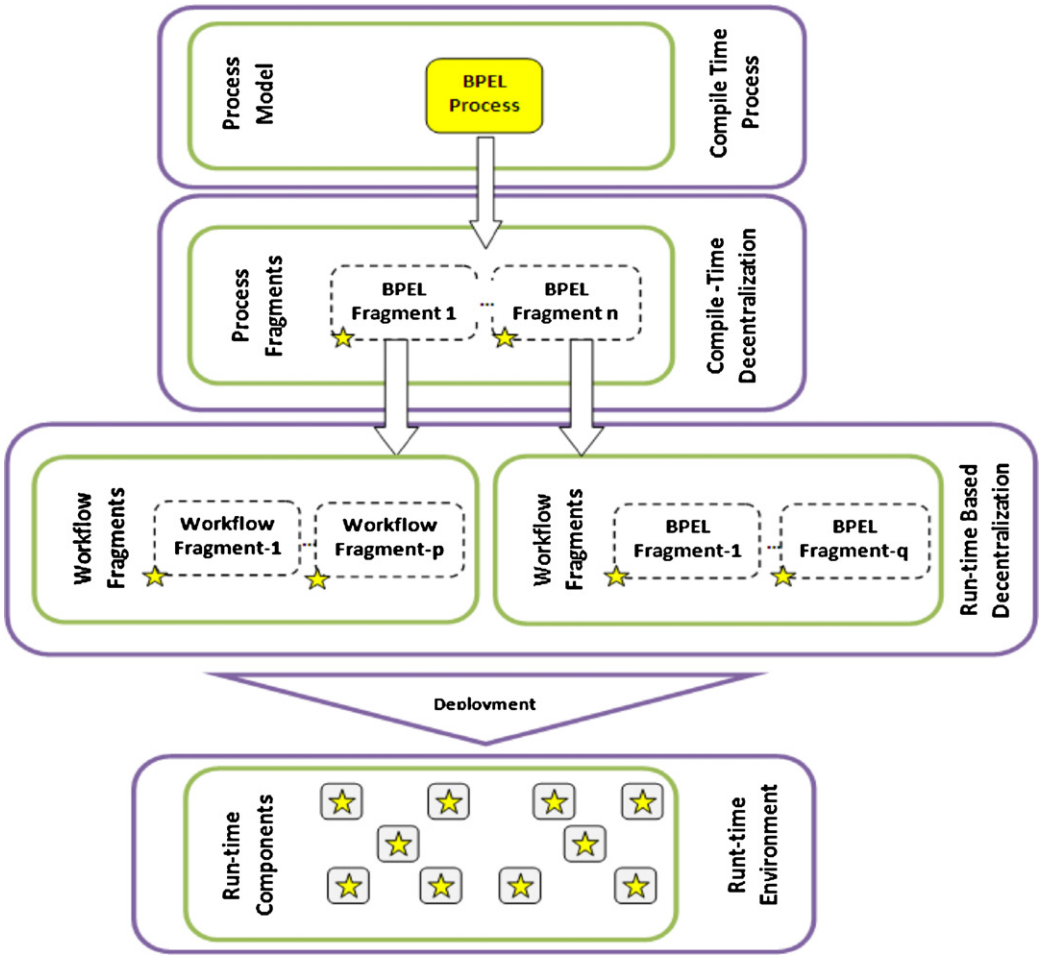

Fig. 4. Traditional decentralization.

will eventually be encapsulated in runtime components such as web services, agents, etc. and will be deployed in runtime environment.

The independency of compile-time and run-time decentralization can be realized through a three-layered architecture recommended for the orchestration layer of Service Oriented Architecture as shown in Fig. 2. The top layer is the specification of a BPEL process, which is specified by a process designer. The middle layer is a compile-time fragmentation, which is performed by process designers or tools. The last layer that is the originality of this study is introduced as an abstract layer that isolates compile-time and run-time decentralization of workflows. This layer can also be equipped with a number of intelligent techniques to make more effective decisions based on current circumstances of runtime environment, which is analogous to the adaptability.

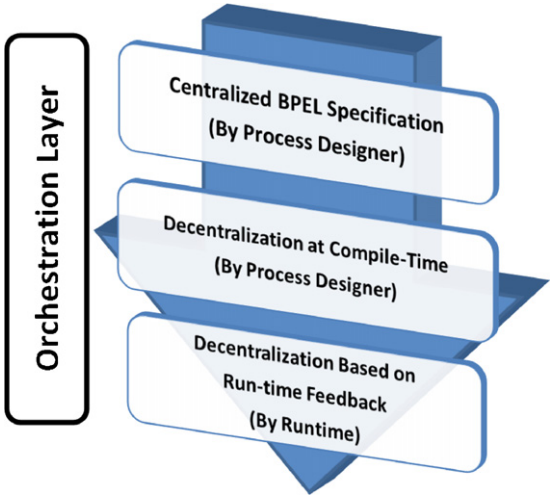

Fig. 5. Runtime based decentralization.

In this paper, the runtime adaptability of fragments is studied from two aspects. The first one is the frequent-path adaptability, which is equal to finding closely interrelated activities and encapsulating them in the same fragment to omit the communication cost of the activities. Another aspect is the proportional-fragment adaptability, which is analogous to the proportionality of produced fragments with number of machines. It extenuates the internal communication among the fragments on the same machine.

3.3. Formalization and realization of ADWEF architecture

In this section, the ADWEF is formalized and then realized by two types of architecture to reinforce the implementation of the adaptable and dynamic fragmentation of business processes at runtime. The ADWEF can be specified as shown in Eq. (1).

$$ADWEF = (P, F, WFC, M, \Phi, \Sigma, \Delta) \quad (1)$$

According to the ADWEF specification, $P = \{p_1, \ldots, p_i, \ldots, p_n\}$ is a set of business processes $p_i \in P$; $F = \{f_1, \ldots, f_i, \ldots, f_n\}$ is also a set of fragments $f_i \in F$ made from processes $p_i \in P$, so that each fragment

$f_i = \{pf_1, \ldots, pf_i, \ldots, pf_m\}$ is a set of process fragments $pf_i \in f_i$. Workflow components (also called performer/micro-workflow engine) are small size workflow engines, which are able to execute process fragments as workflow fragments at runtime. *WFC* is also a set of workflow components to run workflow fragments. *M* is a set of machines dedicated to an orchestration engine as well. The workflow components are also able to communicate with each other through a middleware. However, distinctions among workflow execution platforms are determined by Φ, Σ and Δ sets. To be more specific, $\Phi = \{\varphi_{HPD}, \varphi_{HIPD}, \varphi_{Centralized}, \varphi_{FPD}, \varphi_{IPD}\}$ is a set of fragment functions φ_i to convert a process to a set of fragments based on a specific policy and is defined as $\varphi_i : P \rightarrow F$. Functions φ_{HPD} and φ_{HIPD}, which will be presented in the following sections, fragment business processes applying a hierarchical and a mining approach, respectively. Both functions $\varphi_{Centralized}$ and φ_{FPD} are considered to model the previous works; the former translates the whole business process to one fragment and the latter translates it to a set of fragments so that each fragment includes just one process activity. Function φ_{IPD}, which will be discussed in more detail in the following sections, is actually the intersection of the block and graph-structured business processes from mining based decentralization view.

Set Σ includes encapsulate functions, which assign a workflow fragment to a workflow component. In fact, a workflow component can be an agent, a web service, etc. which is intrinsically implemented as a thread in runtime environment. Similarly, workflow fragments are implemented as runtime threads, which are assigned to runtime components. Consequently, creating workflow components and encapsulating workflow fragments in them result in creating a number of threads in runtime environment.

Moreover, assigning the fragments to workflow components is based on the type of the selected architecture. In addition, Δ is a deployment/distributor function, which maps workflow components to network machines and is defined as $\Delta : WFC \rightarrow M$, which is also dependent on the type of architecture. The implementation of Δ methods is a subject of load balancing as well.

In order to realize the ADWEF, two types of architecture called Type-1 and Type-2 are presented in Figs. 6 and 7 to support dynamic decentralization and execution of fragments. They include a ***runtime environment*** that is a set of ***machines*** each of which is responsible for running ***workflow components***. Workflow components are usually implemented by agents, web services, etc. through which ***workflow fragments*** are executed. The activities of workflows are logged by the runtime environment for different purposes such as security, management, etc. and the log data are recorded in a ***log repository***. The log repository can intrinsically be a shared file located on a rapid disk; although, a dozen of mining methods have discussed on managing distributed log files (Bhaduri et al., 2011; Hillol et al., 2004). ***Dynamic process decentralization component*** (***DPDC***) is a software component that decentralizes business processes based on feedback data from runtime environment and/or stakeholders. After making decision on the type of decentralization, fragments are built. Workflow fragments might be made on-the-fly at runtime or can be taken from a ***fragment repository***, which contains pre-compiled workflow fragments. As soon as the fragments become available, they are encapsulated in workflow components and deployed to the runtime environment afterwards. The whole components in both architectures are able to communicate through a ***middleware***.

The architecture Type-1 aims at providing a general container on each machine for each workflow. For instance in Fig. 6, workflow components $P_0 - WFC_0$ and $P_0 - WFC_1$ are created on both machines M_0 and M_1 for process P_0. Encapsulation functions Σ, assign fragments to workflow components applying random, round-robin, etc. policies. After fragmenting the process P_0, the fragments including $P_0 - f_0$, $P_0 - f_1$ and $P_0 - f_2$ are encapsulated

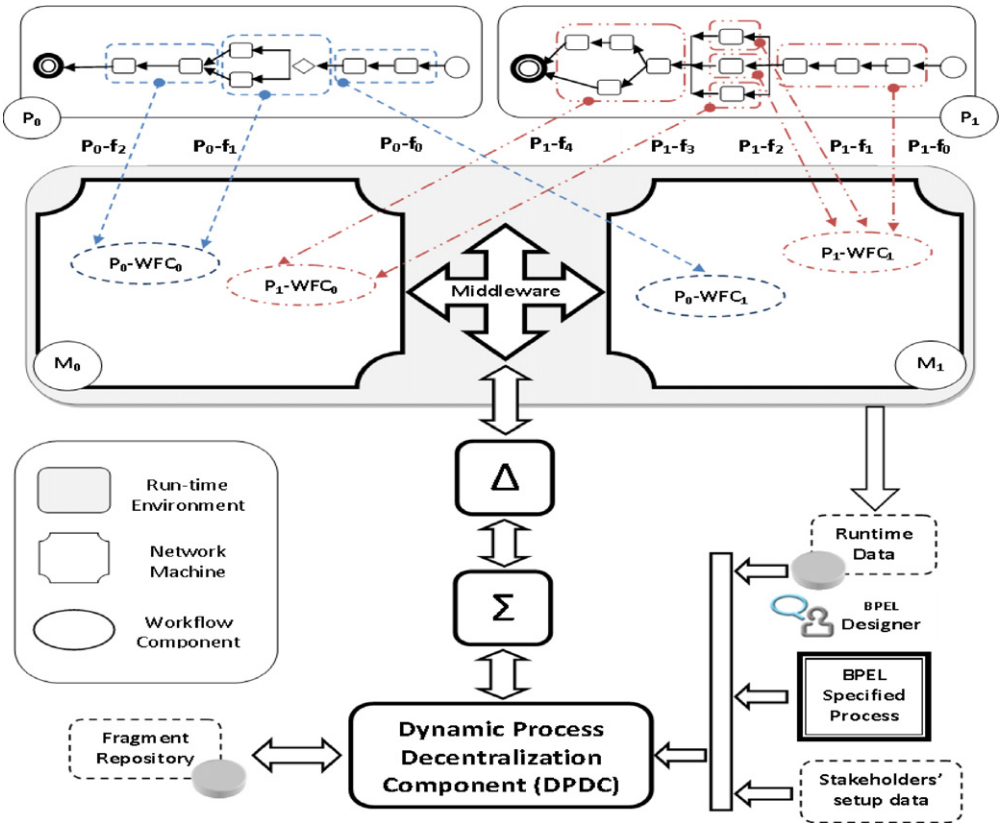

Fig. 6. ADWEF architecture Type-1.

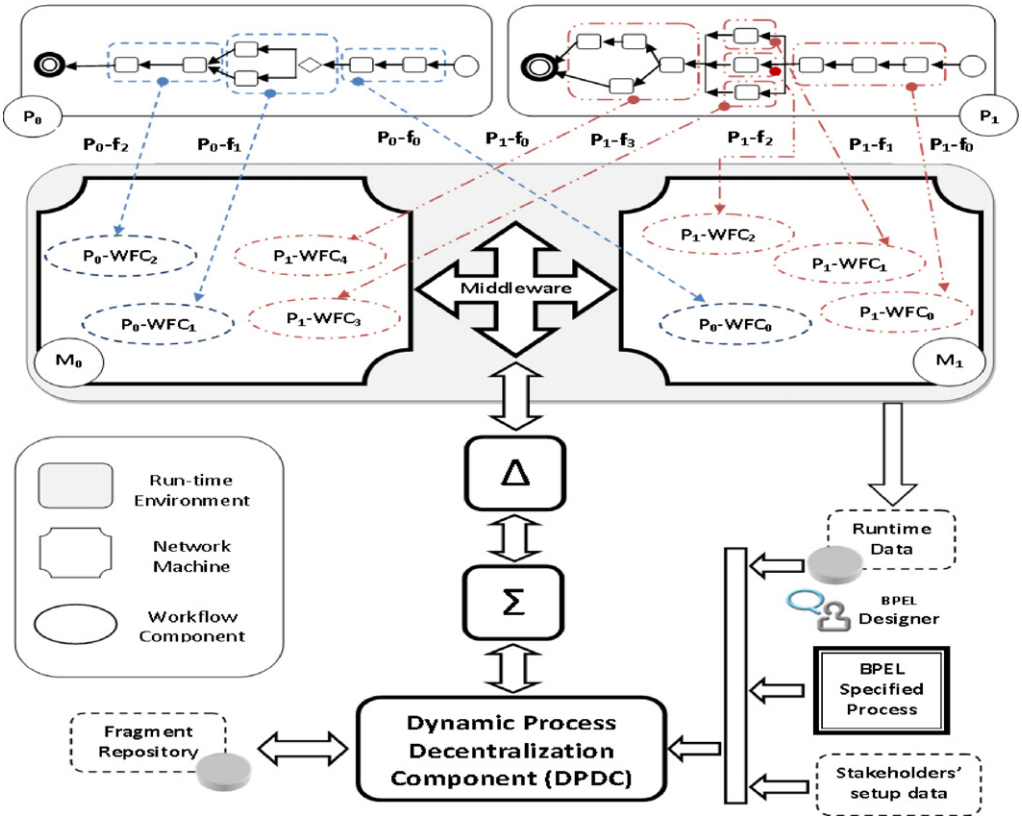

Fig. 7. ADWEF architecture Type-2.

into the dedicated workflow components located on each machine. Deployment functions Δ are quite simple in the architecture Type-1 because each process has already been assigned a workflow component on each machine. It also makes the management of workflow components more convenient. On the other hand, implementing such a general workflow component might consume a bulk of system resources. From runtime view, each workflow component is a single thread and a new workflow fragment is run as a new thread.

The architecture Type-2 as illustrated in Fig. 7, allocates a dedicated workflow component to each fragment. In this architecture, each fragment is encapsulated in its own workflow component, while in Type-1 each process is assigned a workflow component on each machine. For example process P_0 has three fragments; therefore, three workflow components $P_0 - WFC_0$, $P_0 - WFC_1$ and $P_0 - WFC_2$ are created for the fragments $P_0 - f_0$, $P_0 - f_1$, $P_0 - f_2$. Deployment functions Δ make decision on sending workflow components to network machines considering load balancing policies or some simple policies such as random, round robin, etc. distributions. In this architecture, workflow components are simpler compared to a general workflow component in Type-1 because each workflow component executes only one workflow fragment. It consequently results in consuming fewer system resources. Normally, for each workflow component a new thread is created and for each fragment running on each workflow component an extra thread is created. Due to the fact that dynamic fragmentation creates a number of fragments from one business process; therefore, a number of threads will be created that results in creating a number of workflow components. Fig. 8 determines the setup of the ADWEF architecture using its basic elements. Accordingly, based on the ADWEF architecture, one or more workflow components are created to run workflow fragments. Then, each process p_i has to be fragmented and the produced fragments are stored in the respective fragment set f_i. Workflow fragments must be encapsulated in workflow components applying different policies. Finally, various deployment policies can be applied to send all $wfc_i \in WFC$ to the dedicated machines M. Fig. 8 formalizes the setup of both architectures, which can be realized using WADE/JADE facilities as shown in Figs. 9 and 10.

3.3.1. *Miscellaneous aspects of ADWEF architecture*

There are several aspects of the ADWEF architecture, which this paper will not focus on them. Nonetheless, it is worth giving a brief explanation about how these issues can be handled. **Governance of**

1: **name:** ADWEF
2: **input:** $P, F, WFC, M, \Phi, \Sigma, \Delta$
3: **output:-;**
4: begin
5: $0 \leq |p|;\ \forall\, wfc_i \in WFC; wfc_i \leftarrow wfc_i \cup createWorkflowComponent();$
6: $\exists p_i \in P;\ \exists f_i \in F;\ f_i \leftarrow f_i \cup \Phi(p_i);$
7: $\forall wf_i \in f_i; \forall\, wfc_j \in WFC; wfc_i \leftarrow encapsulate(wf_i); \mid encapsulate \in \sum;$
8: $deploy(WFC, M); \mid deploy \in \Delta;$
9: end;

Fig. 8. ADWEF architecture setup.

1: For each machine create a performer agent with a unique name;
2: Compile each workflow fragment to its equivalent subflow;
3: ∑: each subflow must be assigned to a performer agent (Randomly, Round Robin);
4: Δ: distribute the performer agents on network machines (for each machine a performer agent);
5: Setup input parameters of the subflow;
6: Invoke the subflow;

Fig. 9. Realizing ADWEF architecture Type-1 using WADE/JADE facilities.

1: For each workflow fragment create a new performer agent with a unique name;
2: Compile each workflow fragment to its equivalent subflow;
3: ∑: each subflow must be assigned to the dedicated performer agent created in Step1;
4: Δ: distribute the performer agents on network machines (Randomly, Round Robin);
5: Setup input parameters of the subflow;
6: Invoke the subflow;

Fig. 10. Realizing ADWEF architecture Type-2 using WADE/JADE facilities.

workflows determines that workflow fragments must be governed by which organizational unit. These decisions are usually made by designers at design time so that the designers determine an activity or a set of activities as a fragment and assign them to a specific unit. There should be some facilities in a workflow engine to tag activities to determine which of them must not be fragmented or can be fragmented and run by specific machines inside/outside an organizational unit. **Managing state of workflows, transactions, and exception handling** are common problems of decentralized workflows, which are usually studied together. Although a database can be used to manage the state of workflows, transaction, and exception handling (Alonso et al., 1995; Muthusamy et al., 2009), research attempts such as Viroli et al. (2007) and Daniel Wutke and Leymann (2008) used to implement these aspects through the tuplespace concept. There are even some commercial implementations such as JavaSpaces (Philip and Nigel, 2003; Freeman et al., 1999) and Camel (2004) to handle state, transaction, and exception handling in distributed systems. A dynamic fragmentation mechanism needs to adopt a unique ID for each workflow activity, which is usually the case, and make it independent from the fragment in which it is encapsulated. In other words, if a workflow requires re-fragmentation/re-configuration then it must be able to retrieve its last state in a new fragment after being re-fragmented. A fragmentation/re-fragmentation algorithm can be activated when (1) a new workflow is requested; (2) a suspended long-running process comes alive; and (3) system thresholds are violated. Each of these reasons requires individual/batch re-fragmentation of workflows based on current runtime circumstances.

4. DPDC implementation policies

In this section, two implementation policies applied in the DPDC component are introduced for Hierarchical Process Decentralization (φ_{HPD}) and Hierarchical Intelligent Process Decentralization (φ_{HIPD}) methods. The HPD has a hierarchical view to business process decentralization, while the HIPD is an intelligent method based on a mining approach. It provides more customized fragments along with the HPD fragments.

4.1. Hierarchical Process Decentralization (HPD)

A block-based BPEL specified business process is a hierarchy of structured and unstructured activities nested in logical blocks, which can be modeled as a tree (Ouyang et al., 2006). A BPEL tree can be decentralized into its activities based on the level of decentralization, which is called Hierarchical Process Decentralization (φ_{HPD}). A fragmentation algorithm for a HPD-based DPDC component can be started from any level in a workflow tree applying a Breadth First Search/Traverse algorithm (Knuth, 1997).

Fig. 11 shows the HPD decentralization of a process based on the levels of a sample BPEL tree. A fragmentation which starts from the *level-0* of the process tree results in producing a fragment that includes the whole tree. A fragmentation started from the *level-1* results in three fragments including root node, left sub-tree, and right sub-tree. Similarly, the *level-2* of fragmentation provides six fragments among which four fragments include only one activity. Obviously, the *level-3* of fragmentation provides nine fragments with the smallest granularity, while the *level-0* of fragmentation provides the coarsest one. The coarser the granularity levels, the less the number of fragments. The softer the granularity levels, the more the number of fragments. The activities of each fragment are logically related with each other according to the process specification.

In the HPD decentralization, leaves are simple activities while structured activities are in the middle and they intrinsi-

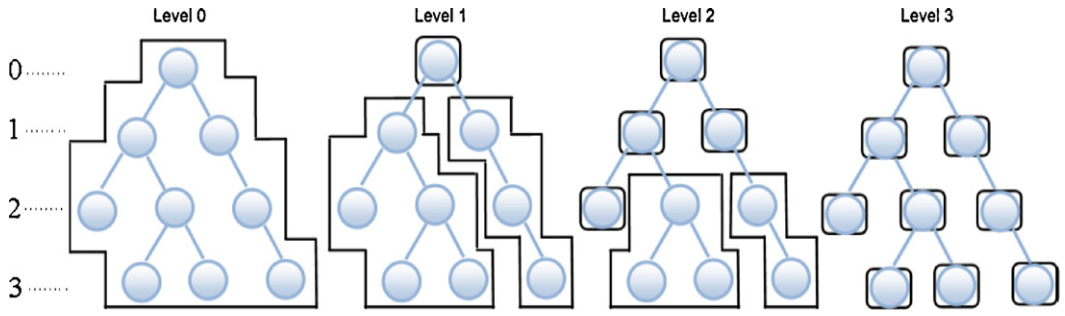

Fig. 11. Hierarchical Process Decentralization (HPD).

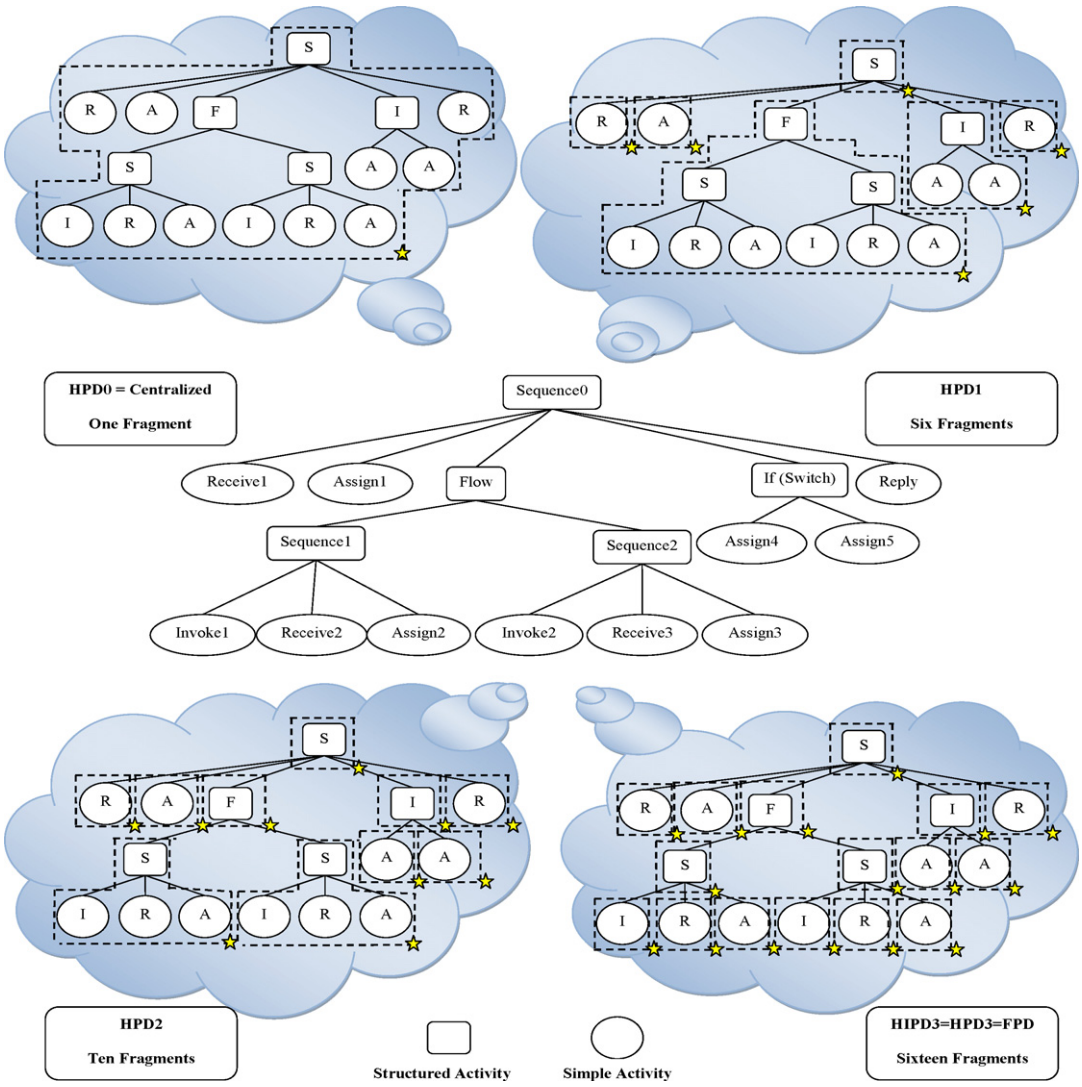

Fig. 12. HPD decentralization of loan application business process.

cally keep the closely interrelated activities together in a single fragment. The coarsest fragment is equal to the traditional Centralized method and the softest granularity is equal to the FPD method.

HPD Decentralization of Loan Application can be performed in four levels according to Fig. 12 where each fragment is marked by an asterisk icon. The HPD0, which is fragmentation in the *level-0*, wraps the whole process tree in one fragment as it is the case for the Centralized method. The HPD1, HPD2, and HPD3 are also subsequent levels of fragmentation; however, the HPD3 is decentralization in the last level and is equal to the FPD.

4.2. Hierarchical Intelligent Process Decentralization (HIPD)

This section introduces an approach for business process decentralization called Hierarchical Intelligent Process Decentralization (HIPD). As a matter of fact, the aforementioned HPD is a general method used when mining data is not available and has a top down view to business process decentralization. The HIPD applies the HPD and a frequent path mining algorithm together when mining patterns are available.

Fig. 13 is based on the assumption that dark nodes of a process tree are detected as frequent nodes. Basically, only frequent nodes are considered in frequent paths and infrequent ones are excluded.

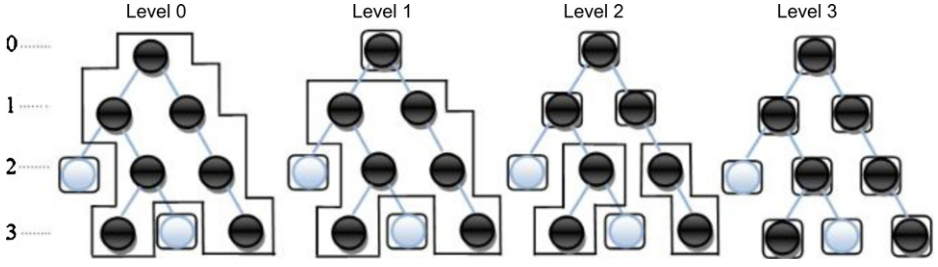

Fig. 13. Hierarchical intelligent process decentralization (HIPD).

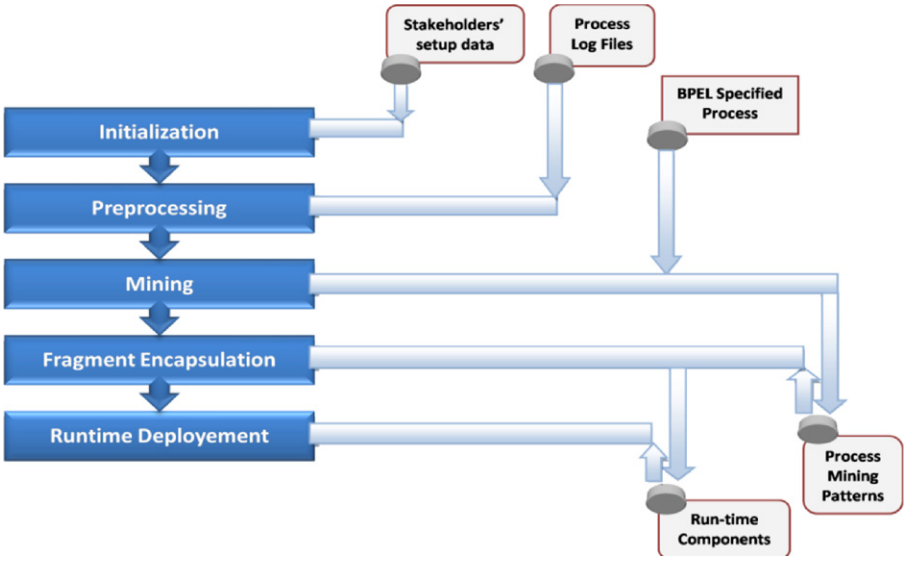

Fig. 14. DPDC structure for HIPD.

Thus, there are three fragments in the *level-0* where two fragments include only one activity and one fragment includes a frequent path. Accordingly, similar policies can be applied for subsequent levels.

The main purpose of the HIPD is to decentralize BPEL processes intelligently by applying a mining approach. In fact, a HIPD-based DPDC is able to apply any frequent path mining algorithm such as Dubouloz and Toklu (2005) to decentralize a block/graph-structured workflow; nonetheless, block-structured workflows are stressed in this work. Particularly, this method is expected to work well for those kinds of workflows in which there is a high frequent mainstream to distribute tasks among web services and sub-workflows. The mining approach can be used for both kinds of block-structured and graph-structured processes. The coarsest level of granularity in the HIPD method namely HIPD0 is distinguished and called IPD because it is a common point, where block and graph structured path mining methods join together.

4.2.1. HIPD-based DPDC

According to Fig. 14, five phases are considered to make a mining-based DPDC as follows:

Phase1 (Initialization): This phase initializes the frequent path mining based on the information provided by system stakeholders such as: (1) the minimum frequency or minimum support of all activities; and (2) the level of granularity for each workflow. **Phase2 (Preprocessing)**: This phase removes all noise data from the log repository. **Phase3 (Mining)**: In the beginning, no mining information is available. The DPDC component commences with the HPD method. When the system is warmed up, a process compiler uses up-to-date mining patterns to decentralize processes. Workflow Miner, which is the kernel part of the system, mines the process log files and maintains the extracted decentralization patterns in a process mining pattern repository. The key steps of mining process can be classified as follows: (1) a BPEL *process tree* is constructed according to BPEL specification. Leaf nodes are naturally basic BPEL activities, while non-leaf nodes are structured ones. Indeed, the children of non-leaf nodes are in their immediate sub-level. In addition, the ordering of each node's children from left to right shows the sequence of activities. (2) The *workflow tree*, which is by definition, all executed paths in a process tree must be synchronized with the process tree. It is performed by marking the executed activities according to workflow log file information. Each node in the process tree is assigned a visited counter that is incremented after each visit. (3) The frequent and infrequent paths of a workflow are detected in this step. It is worth mentioning that there is no stress on any frequent path detection and/or mining algorithm in this work. All previously presented frequent path detection algorithms can be applied in block/graph-structured workflows. Nonetheless, a granular frequent path detection algorithm is implemented in this section to granulate a detected frequent path. **Phase4 (Fragment Encapsulation)**: A BPEL compiler applying frequent and infrequent paths translates a process to workflow fragments and stores them in a repository. The frequent paths are then encapsulated into runtime components and wired so that they can communicate through a middleware. **Phase5 (Deployment)**: At deployment time the runtime components are taken from component repository, instantiated, and sent to the runtime environment.

4.2.2. Basic definitions

In this section the idea of the HIPD approach is formulated. Set $A = \{a\}$ is introduced as a set of activities and set E as a set of edges that are used to make a tree of activities. Set $BT = \langle A, E\rangle$ stands for a block-structured process tree, which is described by the BPEL language. Furthermore, set ET is considered 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 started from a process root to a leaf. A path is called a frequent path when all of its activities are frequent. An activity is also called a frequent activity when its frequency value exceeds a *min_sup* value (Han and Kamber, 2001). The *min_sup* is the minimum frequency value that a frequent activity must gain to be known as a frequent activity. In order to categorize activities and paths in a process tree several sets are required including *Frequent Activity (FA)*, *Frequent Path (FP)*, *Infrequent Activity (IA)* and *Infrequent Path (IP)* as follows.

The *FA* is a set of frequent activities, which their frequency is bigger than *min_sup* according to Eq. (2). The frequency of each activity is also calculated based on the type of the activity as illustrated in Eq. (3). If an activity is leaf; its frequency is equal to its iteration number. Otherwise, the frequency of the activity is equal to the frequency of the child with a maximum frequency number.

$$FA = \{a \in A \wedge frequency(a) \geq min\_sup\} \tag{2}$$

$$frequency(a) = \begin{cases} if \quad chilren(a) = \Phi, \quad return \quad activity \quad iteration \quad number \\ else, \quad return \quad max(frequency(children(a))) \end{cases} \tag{3}$$

There exist only one frequent path from root to leaves of a business process tree and the set *FP* includes activities of this frequent

path. Granulating this frequent path is important due to decentralization of a business process and providing a variety of process fragments. It comes about by considering levels of a process tree or a granularity degree G, which is defined in Eq. (4).

$$G = \{G_i | G_{depth-1} > \cdots > G_i > \cdots > G_{root} = G_0\} \tag{4}$$

Accordingly, set FP_G illustrated in Eq. (5) is defined as a set of granular frequent paths fp_G started from *level-G* of the tree to level *TreeDepth-1*. In order to formalize the granular frequent paths, function *level*(G) is applied, which returns the nodes in *level-G* of the tree. Should a path include only one activity, it is not considered as a frequent path.

$$FP_G = \{fp_G\} = \{fp(a_i) | 0 \le G \le depth - 1, \forall a_i \in level(G), |fp(a_i)| > 1\} \tag{5}$$

Function $fp(a_i)$ in Eq. (6) searches for subsequent frequent activities of the activity a_i and insert them into set fp_{ai}. If the activity a_i is a leaf node, then it is right added to the frequent path fp_{ai}. Otherwise, its subsequent children are checked for the same condition.

$$fp(a_i) = \begin{cases} if(children(a_i) = \phi), fp_{ai} = fp_{ai} \cup a_i \\ else, fp_{ai} = fp_{ai} \cup fp(children(a_i)) \end{cases} \tag{6}$$

Infrequent activities are defined in set *IA* shown in Eq. (7) and includes those activities whose frequency is less than *min_sup*.

$$IA = \{a_i | a_i \in A, frequency(a_i) < min\_sup\} \tag{7}$$

In addition, the set *IP* is the union of infrequent activities as well as frequent paths; with the cardinality value equal to one as shown in Eq. (8). Those frequent paths with only one activity are considered as infrequent path as well. It is worth mentioning that all paths in the sets *IP* and *FP* are encapsulated in runtime components afterwards.

$$IP = IA \cup \{fp(a_i) | \forall a_i \in A, |fp(a_i)| = 1\} \tag{8}$$

4.2.3. HIPD-based DPDC algorithms

In this section, the core algorithms of a HIPD-based DPDC are explained. The main algorithm applies three routines *calculateNodeFrequency*, *determineSequentialPaths*, and *frequentPathFragmentation*.

HIPD Fragmentation: The HIPD fragmentation algorithm illustrated in Fig. 15 aims at marking fragments in a process tree based on the information obtained by mining the log files. Three parameters are applied here including (1) the process tree; (2) the degree of granularity (G); and (3) the minimum support (*min_sup*) of the activities that shows the minimum frequency required for an activity to be included in a frequent path. In fact, the HIPD fragmentation algorithm can be specified in three phases. First, the frequency of each node in the process tree is calculated; second, sequential paths in the process tree are determined; and third, by applying

```
1: name: HIPD_Fragmentation;
2: input: root, min_sup, granularity;
3: output: root;
4: begin
5: | calculateNodeFrequency(root);
6: | determineSequentialPaths(root);
7: | frequentPathFragmentation (root, min_sup, granularity);
8: end;
```

Fig. 15. HIPD fragmentation algorithm.

```
1:  name: calculateNodeFrequency;
2:  input: node;
3:  output: node frequency;
4:  begin
5:  | if (node.childrenNumber > 0)
6:  | begin
7:  | | maxFrequency = 0;
8:  | for each child ε node.children()
9:  | begin
10: | | | tempFrequency=calculateNodeFrequency(child);
11: | | | if (tempFrequency > maxFrequency)
12: | | | maxFrequency = tempFrequency;
13: | | end;
14: | | node.frequency = maxFrequency;
15: | end;
16: end;
```

Fig. 16. Node frequency calculation algorithm.

```
1: name: levelNth;
2: input: node, required level number;
3: output: A list containing nodes of Nth level;
4: begin
5: | traverse the tree using Breadth First algorithm;
6: | return a list containing the nodes of level-n;
7: end;
```

Fig. 17. List of nodes in *Level-n*.

frequent and infrequent paths, each activity of the business process is marked by the fragment that it belongs to.

Calculating Frequency of Activities: In order to detect frequent paths of a business process, execution frequency of each activity is required. Thus, a workflow log file is searched to calculate how many times each node has been visited. According to Fig. 16, the frequent path detection is based on the presumption that if one

```
1:  name: determineSequentialPaths;
2:  input: node;
3:  output: -;
4:  begin
5:  | for (counter = 0; counter < depth-1; counter ++ )
6:  | for each node ε levelNth(root, counter)
7:  | if ((node.frequency > min_sup) and (node. type = 'Sequence'))
8:  | for each stNode ε sub_tree(node)
9:  | stNode.frequency = min_sup;//is flagged as frequent
10: end;
```

Fig. 18. Sequential path detection algorithm.

```
1:  name: frequentPathFragmentation;
2:  input: root, min_sup, granularity;
3:  output:-;
4:  begin
5:    f = 0; //fragment counter
6:    depth = 0; //tree depth counter
7:    while (0≤ depth ≤ granularity)
8:    begin
9:      for each node ε levelNth(root, depth)
10:     begin
11:       node.fragment = fragment_f;
12:       f++;
13:       if (depth = granularity)
14:        for each stNode ε (node ∪ sub_tree(node))
15:       begin
16:         if (stNode.frequency >= min_sup)
17:           stNode.fragment = node.fragment;
18:         else begin
19:           stNode.fragment = fragment_f;
20:           f++;
21:         end;
22:       end;
23:     end;
24:     depth++
25:   end;
26: end;
```

Fig. 19. Determining fragments based on frequent/infrequent paths.

simple activity is visited *n* times, then its frequency is *n*. A complex activity contains a number of simple and complex activities; therefore, its frequency is equal to the frequency of the most frequent child.

Sequential Path Detection: Simply, a sequential path is a sequence of activities, which run sequentially and are determined by a *Sequence* activity in the BPEL language. In order to access the nodes in the *level-n* of the process tree, the method *levelNth* is applied according to Fig. 17. Detecting sequential paths starts from the root node (*level-0*) of the tree to level *depth-1* because the *Sequence* activity is a structured activity and cannot be a leaf node. If a *Sequence* node is a frequent node then all of its children are considered as frequent nodes. Fig. 18 shows the sequential path detection algorithm as well.

Fragmentation Applying Frequent/Infrequent Paths: An algorithm is shown in Fig. 19 that marks the activities of a process tree based on the fragment they belong to. The algorithm requires the process tree, the minimum support (*min_sup*), and the granularity level (*G*). Briefly, all the process activities, which are not included in granularity levels upper than *G*, are marked as distinct fragments. Those activities in the *level-G* with a frequency parameter bigger than or equal to *min_sup* are marked in the same fragment. Other activities are marked in separate fragments as well.

HIPD Decentralization of Loan Application Process: The above algorithm starts from the *level-0*. The HIPD0, which is also called IPD, is the first level of decentralization based on detecting a frequent path. In this experiment, it is assumed that loan requests are mostly accepted and the path *[Sequence0 (Receive1, Assign1, Flow (Sequence1 (Invoke1, Receive2, Assign2) || Sequence2 (Invoke2, Receive3, Assign3)), If (Assign4), Reply1)]* is a frequent path and *[Assign5]* is infrequent; therefore, the infrequent activity is excluded from the path.

In order to perform the HIPD experiments, a workflow execution dataset (log file) must be provided. Precisely, to prepare the dataset, the *ActiveBPEL* open-source workflow engine version 2.1 (Active-Endpoints, 2008) and Tomcat servlet container version 5.0.28 (Tomcat, 2009) were used to run the loan application process. In order to make the above frequent path, the client was set to invoke the loan process 1000 times so that 70% of invocations resulted in the approbation and 30% in the rejection of the loan applications. The minimum support was also considered equal to 3%.

After preprocessing the log repository, the mining algorithm goes into action. According to the *calculateNodeFrequency* algorithm

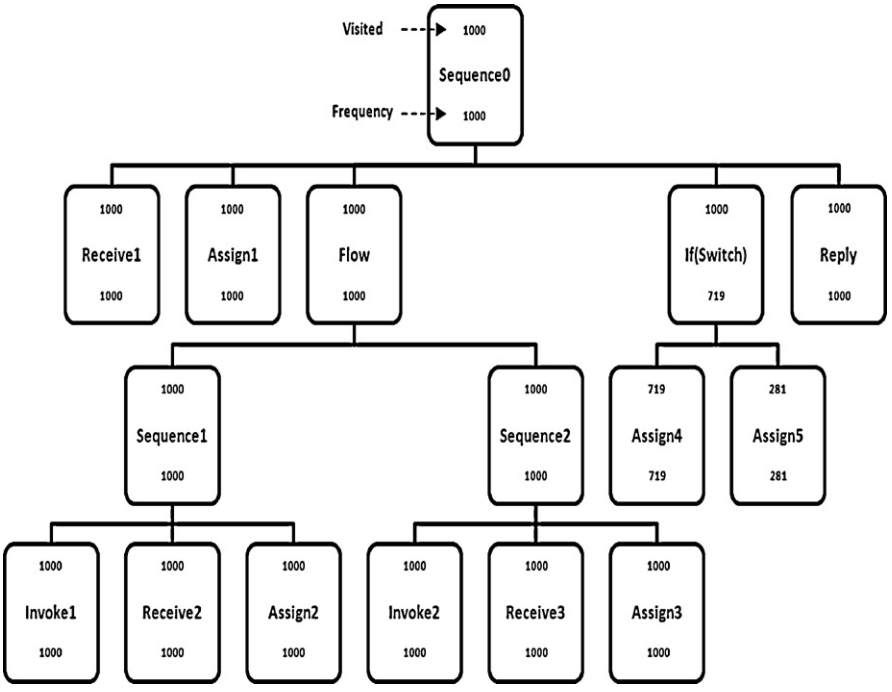

Fig. 20. Tree view of loan application process.

in Fig. 16, for each node there exist *visited* and *frequency* variables and their values are respectively shown on the top and bottom of the nodes in Fig. 20. The *visited* and *frequency* counter of non-leaf nodes is equal to the maximum frequency number of their children. After detecting the sequential paths of the process tree by *determineSequentialPaths* algorithm specified in Fig. 18, the tree is sent to the *frequentPathFragmentation* algorithm in Fig. 19 The output is a fragmented business process so that each process activity is categorized in a specific fragment. The execution result of this algorithm is illustrated in Fig. 21 where each fragment is marked by an asterisk icon.

4.3. Relations of workflow decentralization methods

There exist relations among different decentralization functions as shown in Fig. 22. In some cases, fragment sets produced by decentralization methods may have some intersections. As mentioned before, $\varphi_{Centralized}$ translates a business process to only one fragment so that $|\varphi_{Centralized}| = 1$, which contains the whole process and runs as a multi-threaded object by one machine. In addition, φ_{FPD} fragments a business process to the finest level of granularity so that each fragment contains one process activity. Indeed, the FPD method for an n-activity process results in n fragments, which means $|\varphi_{FPD}| = n$. Furthermore, φ_{IPD} fragments a business process to one frequent path as well as several infrequent paths. These functions work well for both block/graph-structured business processes. The HIPD and HPD methods are based on the hierarchy of activities in a business process tree; therefore, the assumption here is that business processes are block-structured. In order to show a set of fragments in the granularity g, the name of decentralization method comes along with the granularity level g. For instance HPD_g, where $0 \leq g < tree\text{-}depth$ is a set of fragments pro-

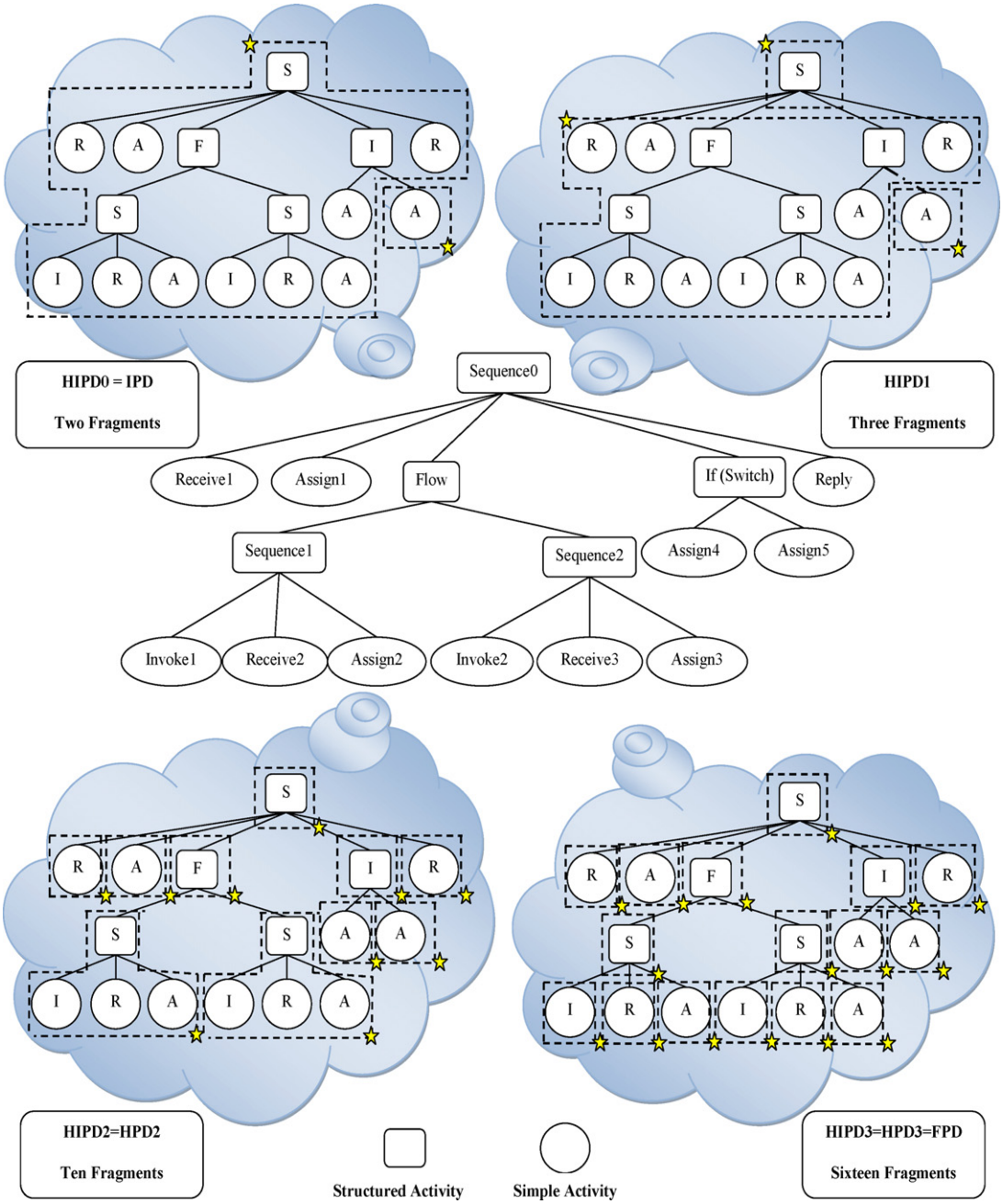

Fig. 21. HIPD decentralization of loan application business process.

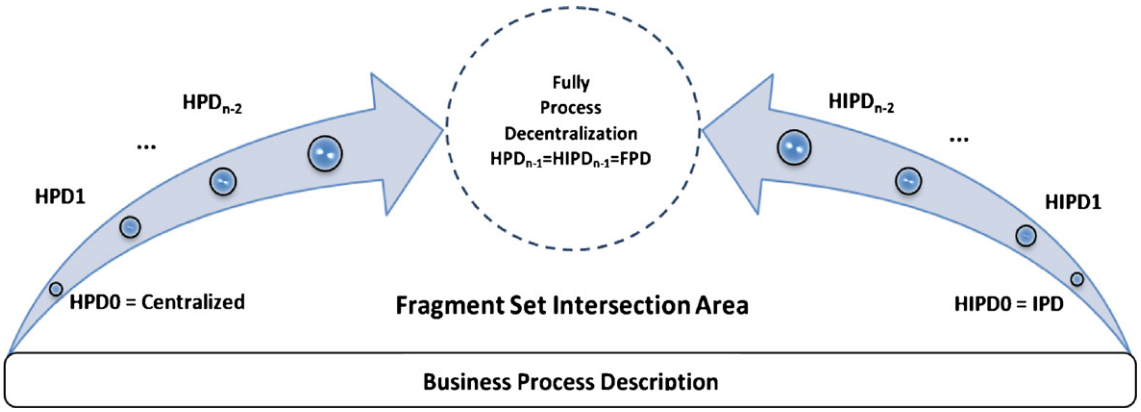

Fig. 22. Decentralization policies relations.

duced by φ_{HPD} for level g of the process tree. According to Fig. 22, for a process $p_i \in P$ both HPD and HIPD methods have at least one similar fragment, $|\phi_{HPD}(p_i) \cap \phi_{HIPD}(p_i)| \geq 1$ which is φ_{FPD}. In addition, both IPD and HIPD methods have exactly one similar fragment, $|\phi_{IPD}(p_i) \cap \phi_{HIPD}(p_i)| = 1$ which is the $HIPD_0$. In other words, the IPD for block-structured BPEL processes considers a frequent path from the process root to process leaves. In the case of the running example, there exists two overlaps among the HPD and HIPD fragments, which are HPD2 = HIPD2 and HPD3 = HIPD3 = FPD as shown in Fig. 21.

4.4. Decentralizing block-structured vs. graph-structured BPEL

The dependency of process activities is determined by a workflow metaphor, which can be block or graph structured. According to the block/graph-structured part of the literature, the block-

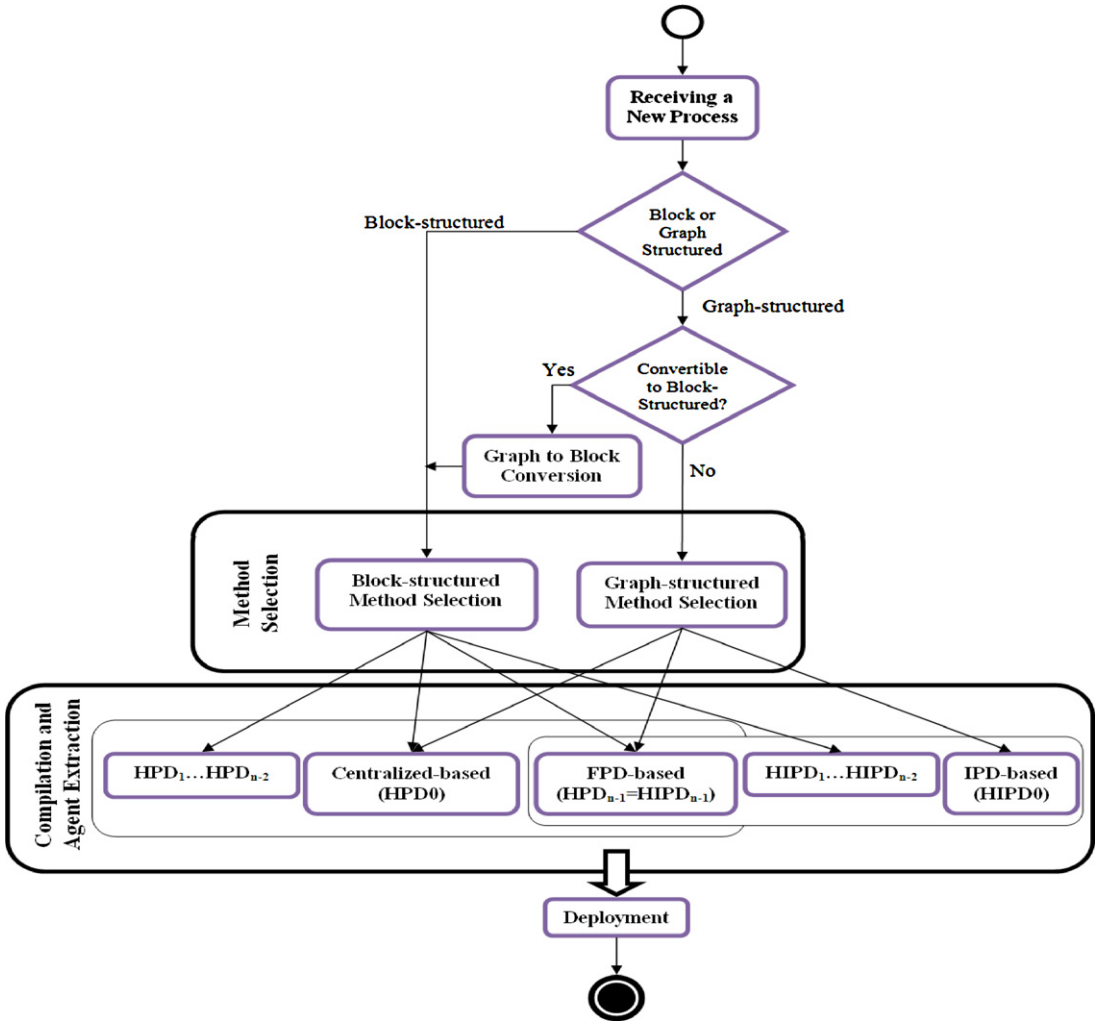

Fig. 23. Choosing a suitable decentralization method based on BPEL type.

structured BPEL is more popular compared to the graph-structured and the focus of this study is on the block-structured BPEL. Nonetheless, a decentralized workflow engine should be able to run different types of business processes. Based on the aforementioned methods of executing a business process, different methods can be taken into account to execute block/graph-structured business processes. According to Fig. 23, a decentralized workflow engine is able to execute graph-structured processes by compiling them to the Centralized, IPD or FPD models. In contrast, block-structured processes can be executed by compiling them to the HIPD, HPD, IPD, FPD and Centralized methods. It is worth noting that the IPD is equal to the HIPD0 for block-structured business processes and it decomposes block/graph-structured business processes to frequent and infrequent paths.

5. Evaluation of proposed model

The main purpose of this section is to evaluate the proportional-fragment and frequent-path adaptability aspects along with the HPD and HIPD decentralization policies in the architecture Type-1. It is achieved by studying the behavior of response-time, throughput, and bandwidth-usage. The experiments were run on a variety number of machines in the absence/presence of request-rate/message-size. Appendix C presents the same experiments for the architecture Type-2.

5.1. Runtime evaluation metrics of workflows

The salient metrics that are usually used to evaluate workflow systems are introduced in this section. The first one is *Response-time* that is considered as the time difference between invoking a workflow until receiving a reply from the workflow. The next metric is *Throughput* that is the number of completed workflows to the total number of requested workflows per time unit. Another metric known as *Bandwidth-usage* is the number of exchanged messages that is counted. *Adaptability* is a dependant metric that can be measured by the improvement of bandwidth-usage, response-time, and throughput. The *frequent-path adaptability* is evaluated by comparing bandwidth-usage, response-time, and throughput of the HIPD fragments with the fragments of other methods. The *proportional-fragment adaptability* is evaluated by the comparison of response-time and throughput of HPD and HIPD fragments running on different number of machines.

5.2. Experimental setup

In order to implement the evaluation experiments, WADE/JADE platform introduced in Appendix A, was applied. The configuration of the experiment environment based on this platform is also shown in Fig. 24. In the whole experiments, WADE was run on Sun's JDK 1.6.0 to manage the performer agents. On the client side, graphical WADE/JADE remote agent management console and the StartBootDaemon process were run. Moreover, the server sides executed workflow agents by means of the StartBootDaemon process. All machines were also of type Intel(R) Core(TM) 2 Duo CPU 2.40 GHz with 3.23 GB RAM memory.

In all the experiments, the client was a multi-thread WADE/JADE agent running on a Linux Suse Real-time version (Novell, 2010) evolved with Java Real-time version (Sun-Oracle, 2010). The client ran a number of client threads each of which called the server side workflows. Each client request resulted in creating a thread in the server side to execute a workflow fragment. Receiving replies from the fragments, the client measured response-time and throughput using gathered data.

In order to refer to the experiments conveniently, each experiment was named according to the following convention: decentralization method name + granularity level + _ + number of fragments + _ + number of server machines. For instance, HPD3_4_2 was equal to using the HPD policy to decentralize a process in the granularity level $G = 3$ and four obtained fragments were run by two server machines.

5.3. Evaluation experiments

The experiments are classified into three categories including boundary conditions, the architecture Type-1, and the architecture Type-2 experiments. The boundary-condition experiments introduced in Appendix B show the functionality of WADE/JADE elementary agents in boundary conditions. The next two categories of experiments are dependent on the type of architecture and apply

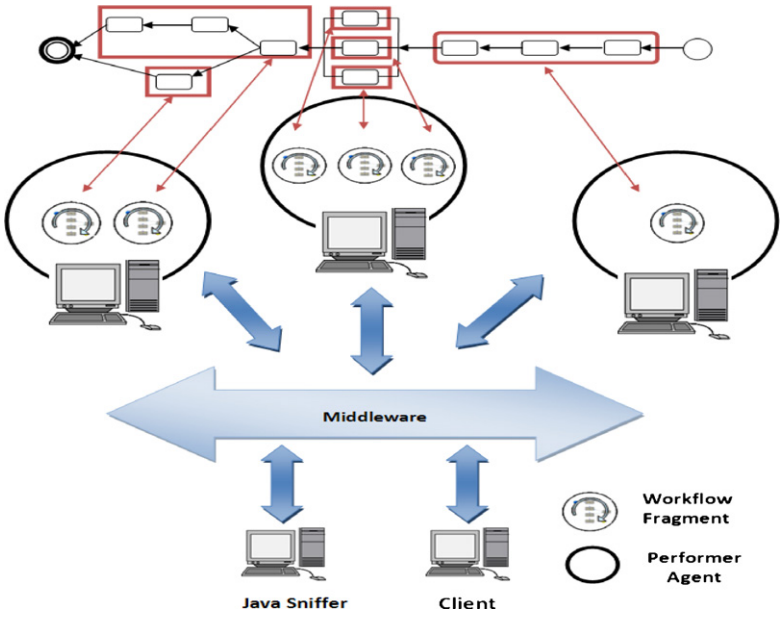

Fig. 24. Experiment environment configuration.

the loan application business process. There exist five experiments for the architecture Type-1 elaborated in the sequel that scrutinize the proposed model from different aspects. In addition, the same experiments as Type-1 are repeated for the architecture Type-2 in Appendix C.

Experiment-1 aims at evaluating the effect of decentralization level on bandwidth-usage in various request rates. Experiment-2 is for evaluating the effect of request rate on response-time and throughput when message size is constant. Experiment-3 evaluates the effect of message size on response-time and throughput when request rate is constant. Experiment-4 focuses on evaluating fragment proportionality with variable request rates when message size is constant applying a range of machines within one to ten. Experiment-5 evaluates fragment proportionality with variable message sizes when request rate is constant applying a range of machines within one to ten. Finally, the improvement percent of different fragmentation models are compared to the FPD based on Eqs. (9) and (10). It is worth noting that the response-time (RT) of the FPD was improved by other methods and this is the reason why the improvement percentage of RT was calculated as shown in Eq. (9). The throughput of each method (TP) is measured in terms of percentage; therefore, the improvement of throughput is calculated as depicted in Eq. (10).

$$improvement_{RT} = \frac{(RT_{fpd} - RT_{othermethod})}{RT_{fpd}} \times 100 \quad (9)$$

$$improvement_{TP} = TP_{othermethod} - TP_{fpd} \quad (10)$$

5.3.1. Running example experiments

This section focuses on the evaluation of the loan application process in the architecture Type-1. As the main goal is measuring the evaluation metrics of the decentralization methods on the process itself, no extra code is considered inside process activities and the time cost of calling external web services is considered equal to zero as well. The method of each experiment is explained separately, nonetheless, the following explanations are the same for Experiment-1 to Experiment-5. There exist two methods of dynamic decentralization implemented by φ_{HPD} and φ_{HIPD} functions. Encapsulation (Σ) is also assigning the fragments to performer agents. In the architecture Type-1, a Round Robin algorithm is selected to encapsulate fragments into performer agents in order to balance the load of servers. The next step is deploying performer agents to server machines (Δ) so that there is a dedicated performer agent on each machine for each workflow in the architecture Type-1.

Experiment-1 evaluates the effect of decentralization level on bandwidth in variable request rates. After calling the workflow, a sniffer counts the number of exchanged messages among the process activities. Experiment-2 and Experiment-3 are implemented in both minimum and maximum of fragment distribution (Min-Max distribution) on network machines. In the architecture Type-1, the minimum distribution applies only one machine, which means that several fragments are simultaneously executed by one performer agent. The maximum distribution for this architecture also means to allocate exactly one machine to each performer agent. The performer agents are able to run various fragments at the same time.

Experiment-4 and Experiment-5 aim at finding the relation between the number of produced fragments and the number of machines allocated for running workflows. A decentralization method is more adaptable when it provides better response-time and throughput in presence of a specific number of machines within one to ten. Experiment-4 measures the adaptability with variable request rate and constant message size, while in Experiment-5 request rate is constant and message size is variable.

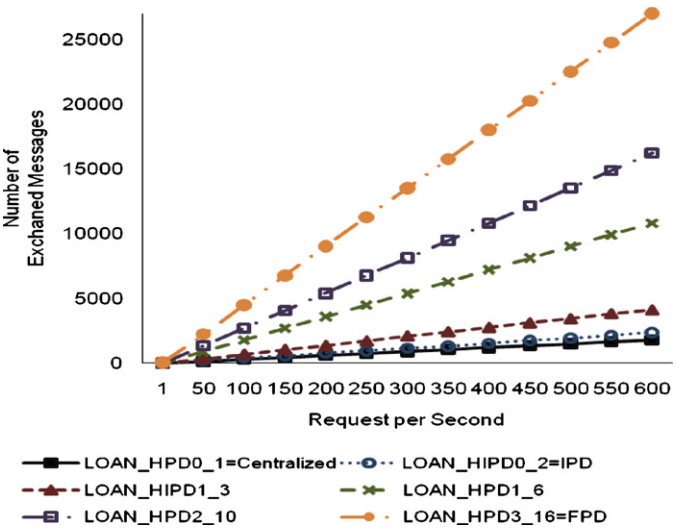

Fig. 25. Loan process message exchange comparison.

5.3.1.1. Experiment-1 (evaluating the effect of decentralization level on bandwidth-usage, variable-request-rate, constant-message-size). Fig. 25 shows the comparison of number of exchanged messages by the loan process fragments in different levels of decentralization. According to the figure, the Centralized and HIPD methods were considerably similar due to the less number of fragments. The FPD method showed the worst number of message passing, while the Centralized and HIPD were the best methods. The middle fragmentation levels of granularity between the *level-0* to the *level-3* exchanged different number of messages. The less number of fragments, the less amount of exchanged messages.

5.3.1.2. Experiment-2 (evaluating the effect of request rate on response-time and throughput, variable-request-rate, constant-message-size). Response-time and throughput can be affected by the request rate and the size of messages passed through process activities. In this experiment, the message size was considered equal to zero for omitting its effect. Requests with different speeds varied from 1 to 600 requests per second were later sent to the servers and the average response-time and throughput of each request were measured.

Figs. 26 and 27 show response-time and throughput behavior of the loan fragments in (Min-Max) distribution. Fig. 28 also shows the improvement trend of the FPD by other decentralization methods. The average improvement percentage of the FPD in terms of response-time and throughput is illustrated and it is clear that the Centralized and HIPD fragments improved the FPD from both response-time and throughput aspects. The FPD was improved around 99.96%, 99.95% and 99.90% by the Centralized, IPD and HIPD1_3_1. However, HPD1_6_1 and HPD2_10_1 improved the FPD by 92.13% and 71.53% due to more number of fragments and message passing. The improvement of throughput also showed similar results so that the Centralized, IPD and HIPD1_3_1 improved the FPD around 54.03%, 50.99% and 54.03%, while the throughput of the FPD was improved around 9.01% and 2.25% by the HPD1_6_1 and HPD2_10_1, respectively.

The Centralized, IPD, and HIPD1_3_1 improved the FPD due to less number of message passing, the encapsulation of closely interrelated activities, and less number of fragments. The HPD1_6_1 and HPD2_10_1 also improved the FPD less than the other fragmentation methods because they had more fragments and message passing.

This experiment showed that when a workflow is requested in different rates, in absence of message size, the FPD response-time and throughput were improved. In maximum distribution,

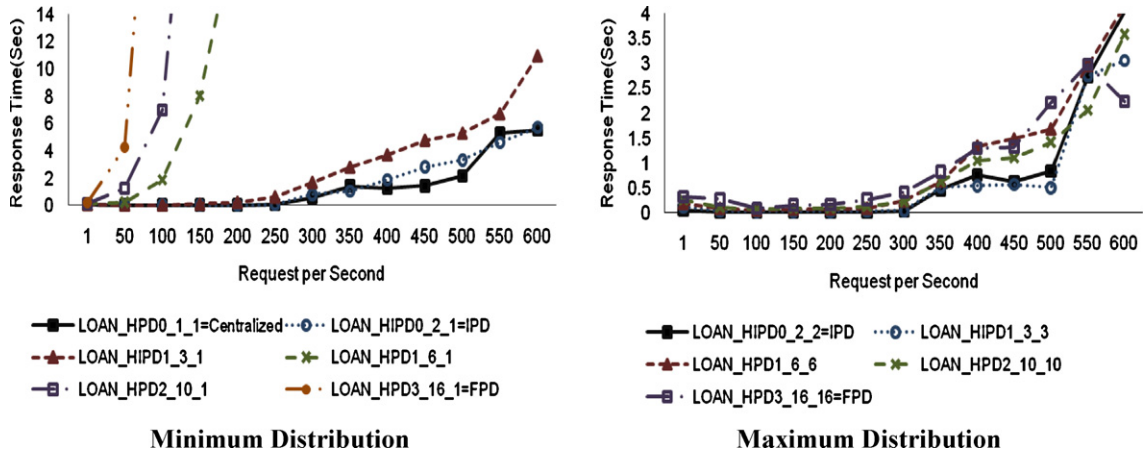

Fig. 26. Type-1, loan fragments response-time comparison, variable-request-rate/constant-message-size, min-max distribution.

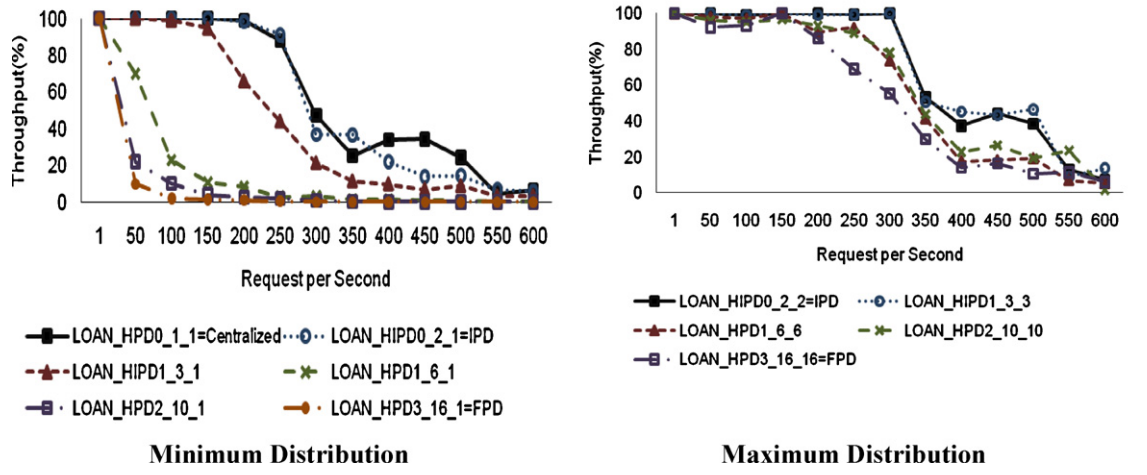

Fig. 27. Type-1, loan fragments throughput comparison, variable-request-rate/constant-message-size, min-max distribution.

both response-time and throughput of fragments tended towards the Centralized method. On the other hand, the comparison of decentralization methods in maximum distribution was not fair due to different number of machines dedicated to them. Thus, there will not be any discussion on maximum distribution and it will be turned to by Experiment-4 and -5 where the proportionality of fragments with number of machines is discussed.

5.3.1.3. Experiment-3 (evaluating the effect of message size on response-time and throughput, constant-request-rate, variable-message-size). Response-time and throughput can be affected by the request rate and size of messages exchanged among process activities. In this experiment varied messages sizes of 50Kb, 100Kb, 150Kb, 200Kb, and 250Kb were applied and in order to omit the effect of request rate, the experiment was repeated two times with constant request rates of 50 and 500 requests per minute.

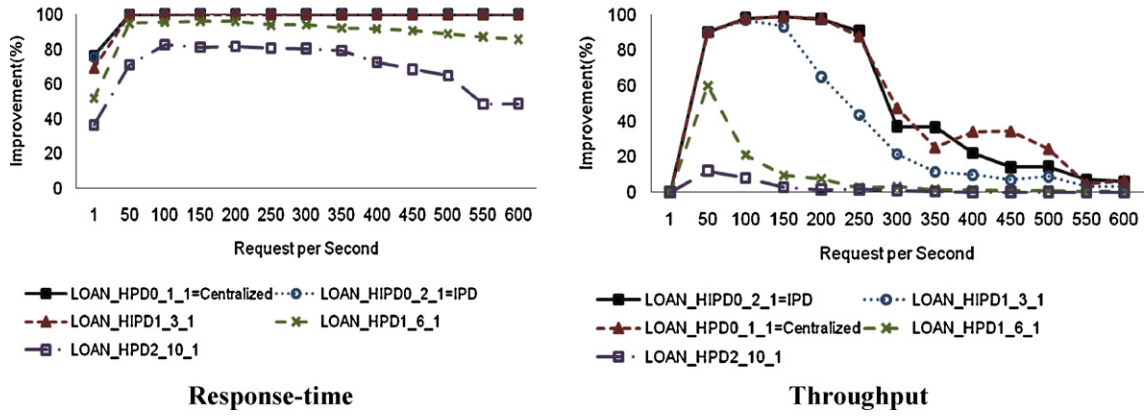

Fig. 28. Type-1, loan fragments response-time and throughput Improvement, variable-request-rate/constant-message-size.

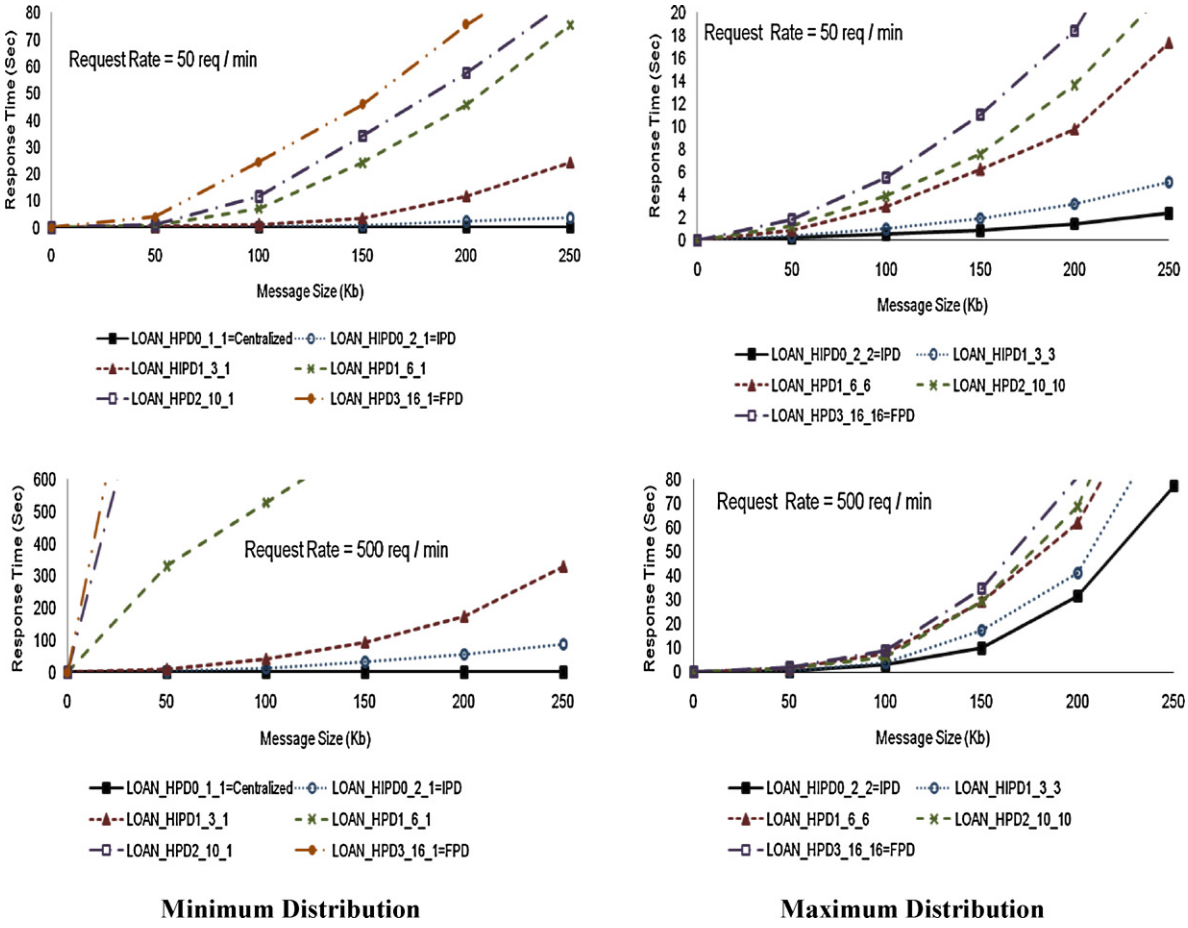

Fig. 29. Type-1, loan fragments response-time comparison, constant-request-rate/variable-message-size, min-max distribution.

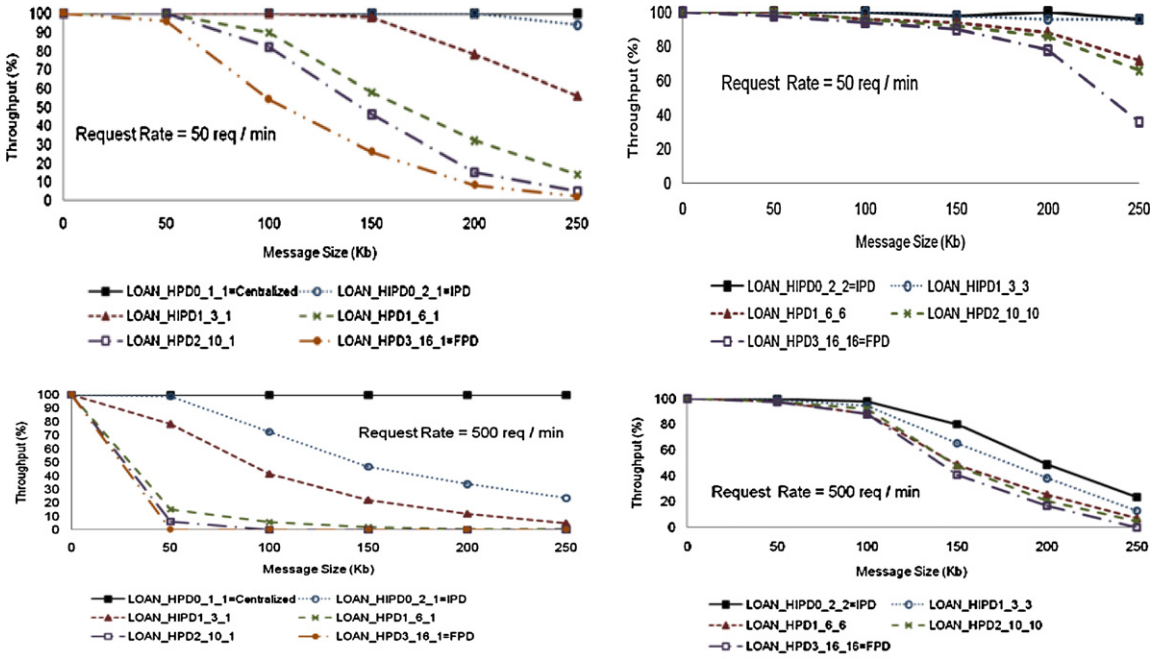

Fig. 30. Type-1, loan fragments throughput comparison, constant-request-rate/variable-message-size, min-max distribution.

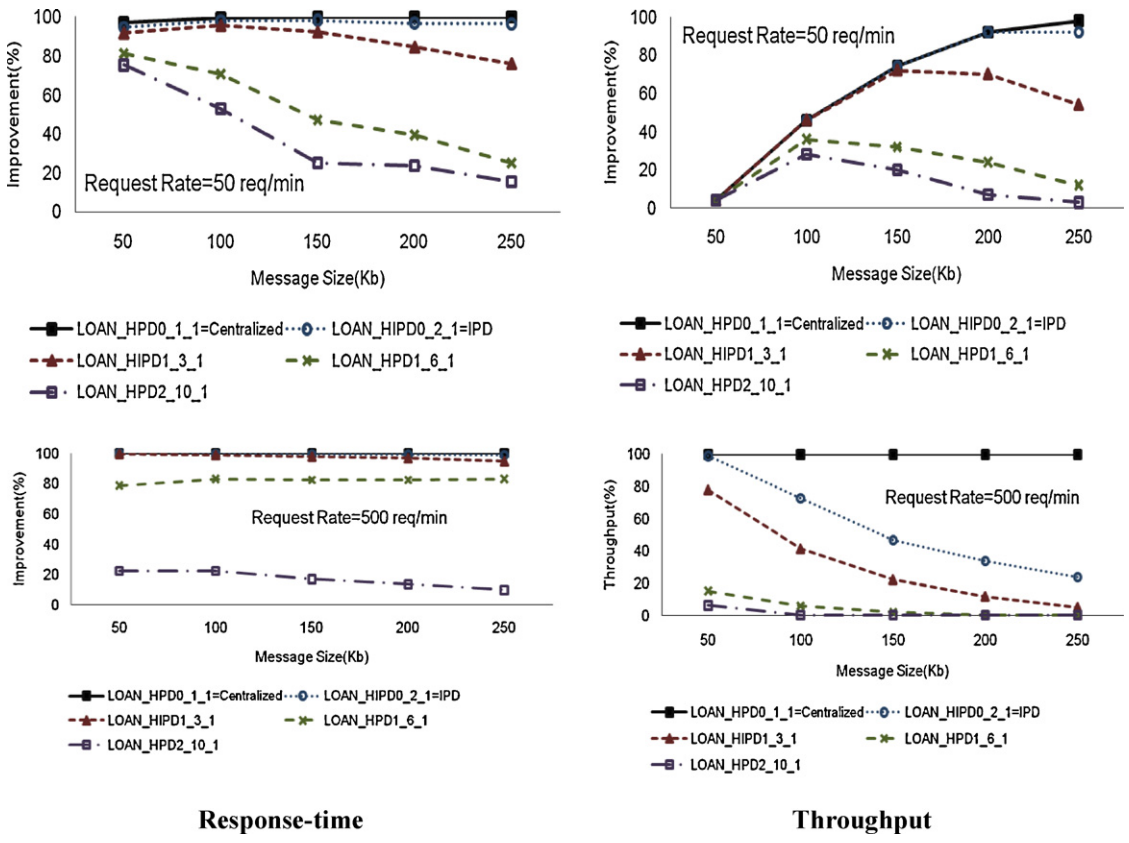

Fig. 31. Type-1, loan fragments response-time and throughput improvement, constant-request-rate/variable-message-size.

In minimum distribution, Figs. 29 and 30 show the effect of message size on response-time and throughput when request rates were constant values 50 and 500 requests per minute. Fig. 31 also depicts the response-time and throughput improvement of the FPD fragments by other loan fragments. In request rate 500, the average improvement of response-time by the Centralized, IPD and HIPD1_3_1 was 99.23%, 9.99% and 97.39%, respectively. In comparison, HPD1_6_1 and HPD2_10_1 improved the average response time about 81.86% and 16.85%. Similarly, the throughput of the FPD was outweighed by the other methods as well. In request rate 500, the average improvement of throughput by the Centralized, IPD and HIPD1_3_1 was 62.80%, 61.60% and 49.20%, respectively. In comparison, HPD1_6_1 and HPD2_10_1 improved the average throughput around 21.60% and 12.40%.

The number of fragments, number of exchanged messages, and size of messages exchanged among process activities were effective in this experiment. The FPD was improved; particularly, when the request rate and the size of messages were increased. The throughput was also improved due to considering the frequent-path and proportional-fragment adaptability aspects.

This experiment showed that even with huge-size messages, the FPD and other fragmentation methods tended towards the Centralized method response-time and throughput in maximum distribution of fragments. On the other hand, the comparison of decentralization methods in maximum distribution was not fair due to different number of machines dedicated to them. Thus, there will not be any discussion on maximum distribution and it will be turned to by Experiment-4 and -5 where the proportionality of fragments with number of machines is discussed.

5.3.1.4. Experiment-4 (evaluating fragment proportionality, variable-request-rate, constant-message-size). This experiment elaborated on the relation of number of produced fragments and number of machines allocated for a decentralized workflow engine in variable request rate when message size was around zero. The main goal was to find the number of fragments that must be provided when there exist a specific number of machines (within 1–10). The response-time and throughput of workflows were measured after running process fragments by different number of machines. For each experiment only those fragment sets were chosen that their number of fragments was equal to the number of dedicated machines.

Figs. 32 and 33 depict the improvement of the response-time and throughput. This experiment showed that the closeness of the number of fragments and number of machines resulted in better response-time and throughput; although, a huge number of requests sent to the servers created long message queues in memory.

Omitting internal messages among the fragments on the same machine extenuated the memory consumption on each machine. In other words, the proportionality of the number of servers and number of machines has resulted in this improvement. Indeed, according to the ADWEF definition (1) for each business process, those fragments that satisfied the condition $|F| \approx |M|$ were more fragment proportional than others.

5.3.1.5. Experiment-5 (evaluating fragment proportionality, constant-request-rate, variable-message-size). This experiment elaborated on the relation of number of produced fragments and number of machines allocated for a decentralized workflow engine. The request rate was constant values at 50 and 500 requests per minute, while the message size was varied values of 50Kb, 100Kb, 150Kb, 200Kb, and 250Kb. The main goal was studying what number of fragments must be provided when a

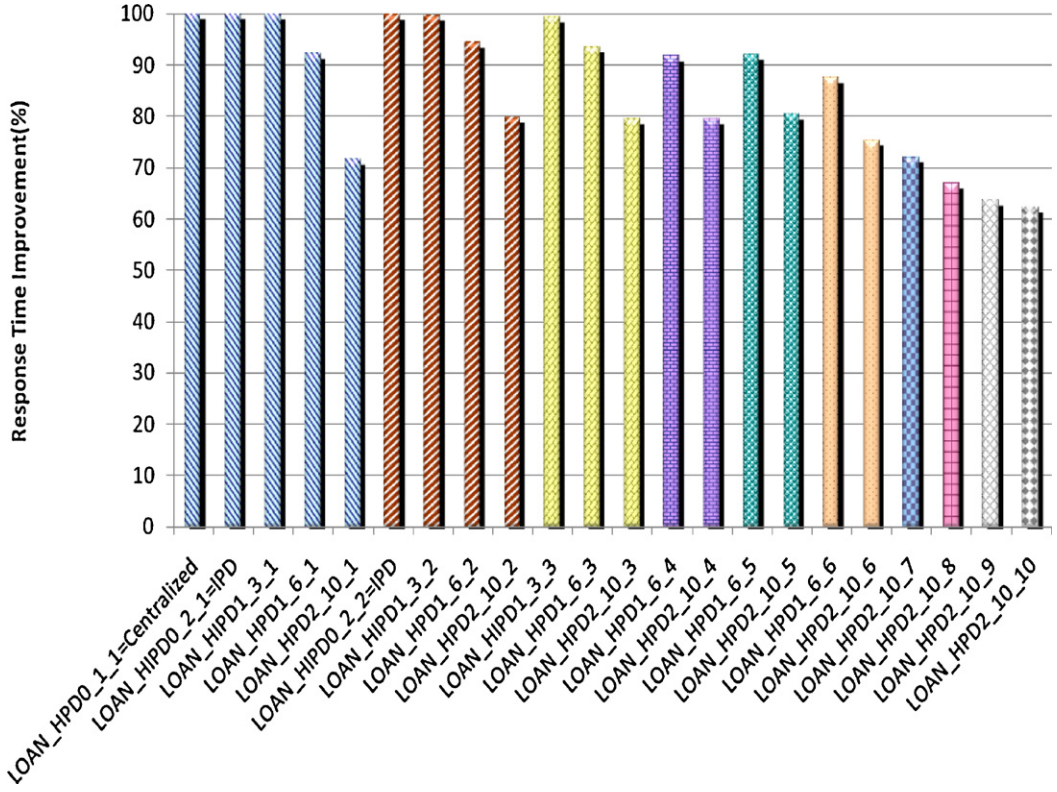

Fig. 32. Type-1, average response-time improvement of loan fragments, variable-request-rate/constant-message-size.

specific number of machines (within 1–10) were available and messages with different sizes were exchanged among the process fragments. Thus, the loan application process was fragmented and executed on various numbers of machines. For each experiment only those fragment sets were selected so that their number of fragments was equal to the number of dedicated machines.

In addition, Figs. 34 and 35 depict the improvement of response-time and throughput when request rate is 50 and 500 request per minute. The closeness of the number of fragments and number of machines resulted in better response-time and throughput; although, huge-size messages created long message queues in memory.

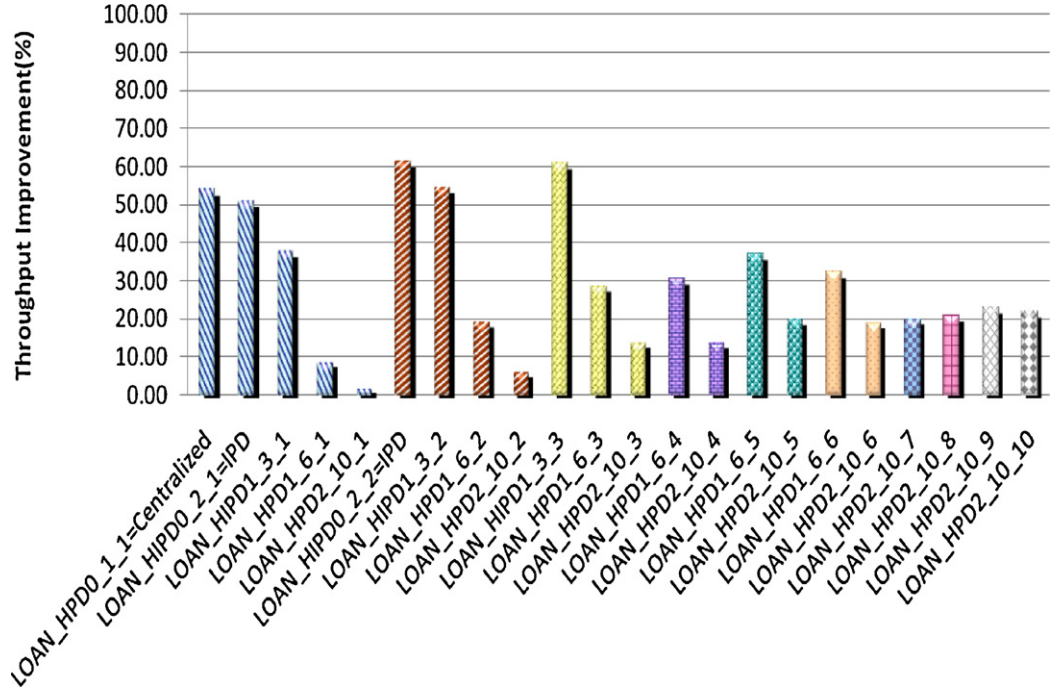

Fig. 33. Type-1, average throughput improvement of loan fragments, variable-request-rate/constant-message-size.

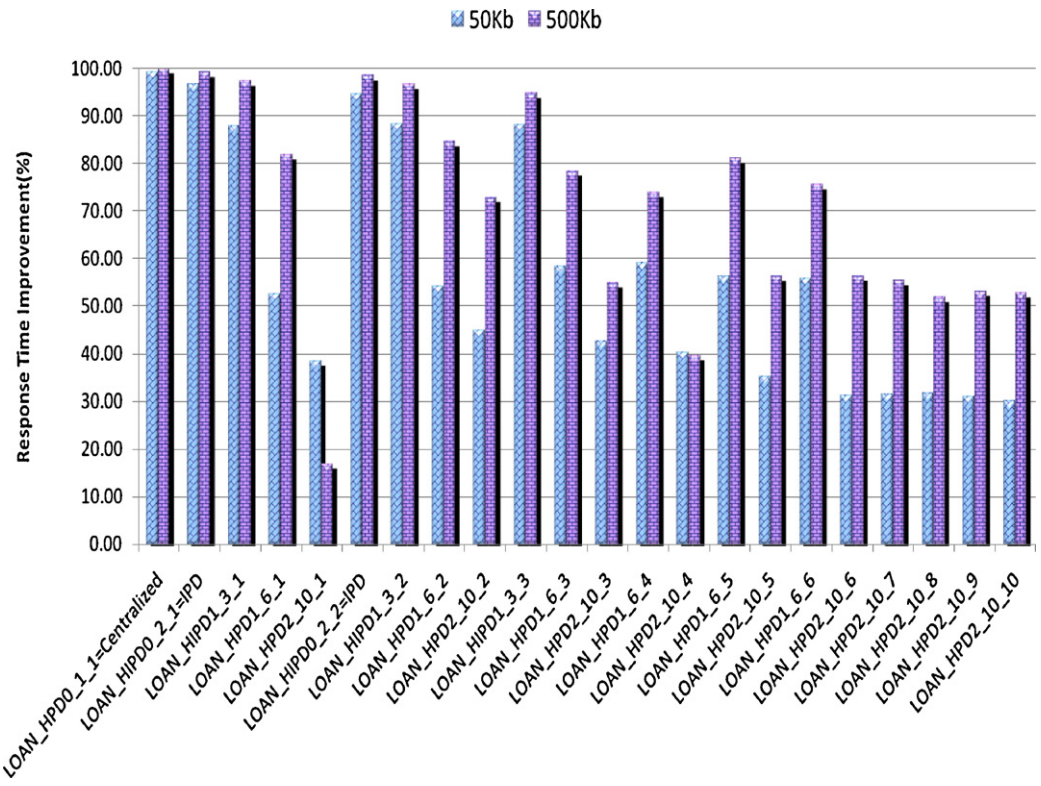

Fig. 34. Type-1, average percentage of FPD response-time improvement by HPD and HIPD methods, constant-request-rate/variable-message-size.

Omitting internal message passing among the fragments on the same machine mitigated the memory consumption. It came about when the number of fragments was proportional to the number of servers dedicated to a workflow engine. Indeed, this experiment again proved that according to the ADWEF definition (1) for each business process, those fragments that satisfied the condition $|F| \approx |M|$ were more fragment proportional than others.

5.4. Evaluations and discussions

The experiments shown in the prior sections were classified into three categories. The experiments in the first category were on examining several elementary WADE/JADE agents containing very basic fragments. The experiments showed: (1) the best functionality belonged to Configuration-1, which was based on the

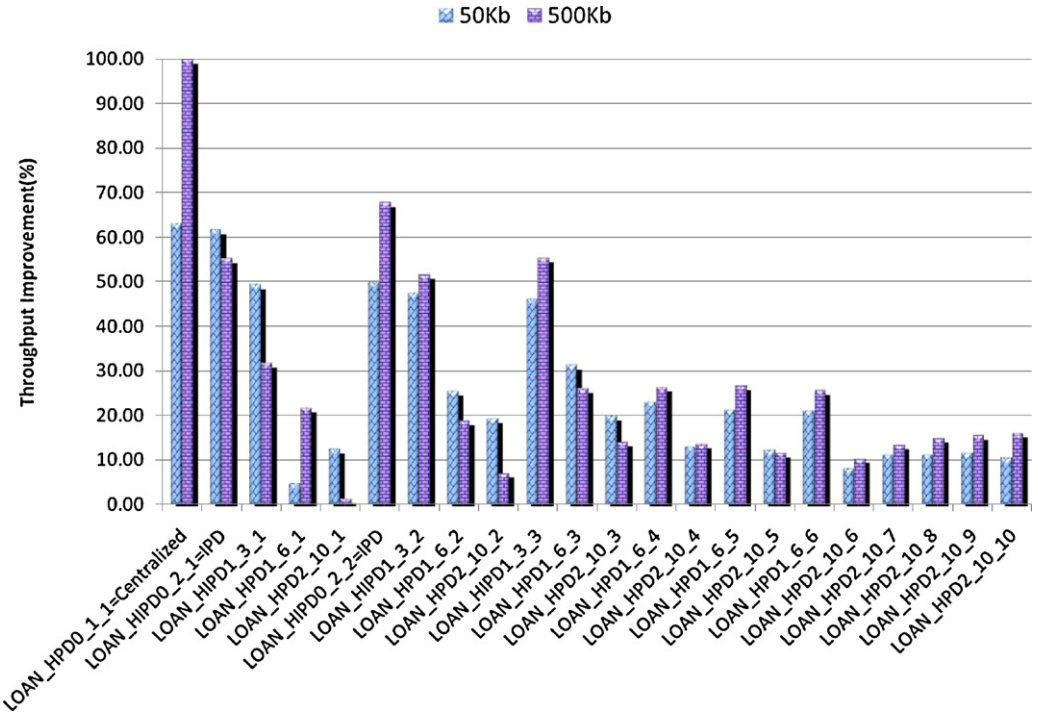

Fig. 35. Type-1, average percentage of FPD throughput improvement by HPD and HIPD methods, constant-request-rate/variable-message-size.

centralized implementation of activities and did not use message passing among the fragments. (2) Configuration-2 showed the worst functionality with two different fragments running inside one performer agent using internal message passing. It could be considered as a limitation of the WADE/JADE platform as well. (3) Configuration-4 was outweighed by Configuration-3; particularly, in the pick of load where there were a lot of message passing among fragments.

The experiments in the second and third categories executed in the architectures Type-1 and Type-2, applied the loan application business process as a more complex case. In order to compare two decentralization methods, the loan process was fragmented by the HPD and HIPD methods and they were later executed by both architectures. Response-time, throughput, and number of exchanged messages were measured to evaluate the methods. Experiment-1, -2 and -3 proved that the frequent-path adaptability dramatically improved the FPD method and considerably approached to the Centralized model. Experiment-2 also showed that when a workflow is requested in different rates, in absence of message size, the FPD response-time and throughput were improved in both architectures Type-1 and Type-2. Improvement in Type-1 was more than Type-2, which showed the resiliency of Type-2 compared to Type-1 in executing different fragment types. In addition, Experiment-3 showed that even with huge-size messages, the FPD method and other fragmentation methods tended towards the Centralized method response-time and throughput in maximum distribution of fragments. This experiment also illustrated that when a workflow is requested in different rates, in absence of message size, the response-time and throughput of the FPD were improved in both architectures. The improvement in the architecture Type-1 was more than Type-2, which showed the resiliency of the architecture Type-2 compared to Type-1 in executing different fragment types.

Experiment-4 and -5 substantiated that both frequent-path and proportional-fragment adaptability aspects improved the FPD method by applying different numbers of machines. They showed that the closeness of the number of fragments and number of machines resulted in better response-time and throughput; although, a huge number of requests with/without large-size messages were sent to the servers. Omitting internal messages among the fragments executed on the same machine extenuated the memory consumption on each machine. In other words, the proportionality of the number of servers and number of machines resulted in this improvement.

In addition, there were several common behaviors in the experiments, which can be mentioned as follows: (1) according to Figs. 25 and 28, the Centralized method in comparison to the minimum distribution of other methods in variable-request-rate/constant-message-size showed the best response-time, throughput, and bandwidth-usage in all cases. In addition, as shown in Fig. 31, the Centralized method was also the best in response-time and throughput with constant-request-rate/variable-message-size. The achieved results was due to the parallel execution of workflow fragments in runtime threads and albeit the least amount of message passing. (2) In addition, the HIPD method in different levels of fragmentation and in minimum and maximum distribution was considerably similar to the Centralized method; although, the HIPD was a decentralized method. It was due to the encapsulation of closely interrelated activities in the same fragment and the omission of communication among them. (3) The last level of fragmentation, which was equal to the FPD, showed the worst response-time, throughput, and message passing in maximum distribution due to the high amount of exchanged messages among fragments and the high number of created fragments. However, according to Figs. 26 and 27, in maximum distribution with constant-message-size/variable-request-rate, after scaling up the system, the FPD method gradually showed better results. Nonetheless, the FPD results were weaker than other methods either. The same result was observed in constant-request-rate/variable-message-size as shown in Figs. 29 and 30. (4) Furthermore, in minimum distribution, when all fragments were executed on the same machine, response-time and throughput were in the worst case. In contrast, in maximum distribution, response-time and throughput of all processes became considerably closer to each other and tended towards the Centralized model. However, in maximum distribution, when huge-size messages were flown among process activities more differences were revealed among the response-time and throughput of the Centralized, HPD and HIPD fragments. Huge-size messages and workflow decentralization were against each other so that by increasing the size of messages, the response-time and throughput of HIPD1_6_1, HPD2_10_1 and the FPD with more number of fragments were sharply rocketed and plummeted, respectively. (5) The fragment proportionality experiments showed that when the number of produced fragments was approximately equal to the number of dedicated machines, the response-time and throughput were considerably improved compared to the FPD method that did not consider fragment proportionality. It can be deduced from Figs. 32 and 33 for variable-request-rate/constant-message-size and from Figs. 34 and 35 for constant-request-rate/variable-message-size. (6) The experiments substantiated that frequent paths and fragment proportionality aspects of adaptability can be realized by applying the HPD and HIPD decentralization methods. (7) The architecture Type-1 was more sensitive to the number of fragments compared to the architecture Type-2.

By and large, the Centralized method suffers from scalability problems, while fully process decentralization methods are limited to message passing constraints. Decentralization is suitable when it is needed and it must be commensurate with runtime circumstances. For instance, the HPD and HIPD methods provided workflow fragments, which were neither centralized nor fully distributed; however, their response-time, throughput, and number of exchanged messages were considerably close to the Centralized method, while they were decentralized.

6. Conclusion and future work

For the time being, the Service Oriented Architecture (SOA) executes BPEL specified business processes applying centralized and non-scalable orchestration engines and clustering of the centralized orchestration engines is not a final solution for the scalability issues. Alternatively, several decentralized orchestration engines are emerging; their purpose is to decentralize a BPEL process to improve system non-functional quality factors. A number of decentralization methods have been presented; however, none of them consider runtime environment requirements in workflow decentralization.

In this paper, the runtime adaptability of fragments was studied from two aspects. The first one was the *frequent-path* adaptability, which was equal to finding closely interrelated activities and encapsulating them in the same fragment to omit the communication cost of activities. Another aspect was the *proportional-fragment* adaptability, which was analogous to the proportionality of produced fragments with number of machines. It extenuated the internal communication among fragments on the same machine. An ever-changing runtime environment along with the mentioned adaptability aspects might result in producing variety versions of a process at runtime. Thus, an Adaptable and Decentralized Workflow Execution Framework (ADWEF) was introduced in order to propose an abstraction of adaptable decentralization in the SOA orchestration layer. Furthermore, both architectures Type-1 and Type-2 were introduced to support the ADWEF along with two

decentralization methods HPD and HIPD to produce customized fragments. However, mapping the current system conditions to the decentralization methods was considered as future work. Evaluations of an implementation of the ADWEF decentralization methods demonstrated a range of improvements compared to the previous methods as follows: (A) the frequent-path adaptability along with variable-request-rate/constant-message-size improve response-time and throughput about [71–99%] and [9–54%] in Type-1, [44–87%] and [4–50%] in Type-2. (B) The frequent-path adaptability along with constant-request-rate/variable-message-size improve response-time and throughput about [16–99%] and [1–99%] in Type-1, [47–96%] and [16–66%] in Type-2. (C) Both adaptability aspects along with variable-request-rate/constant-message-size result in response-time and throughput improvement [62–99%] and [2–61%] in Type-1, [10–88%] and [5–50%] in Type-2. (D) Both adaptability aspects along with constant-request-rate/variable-message-size result in response-time and throughput improvement [16–99%] and [1–99%] in Type-1, [26–99%] and [5–93%] in Type-2.

The idea of adaptable decentralization was introduced in this paper and the idea was substantiated by several experiments that applied the loan application business process; however, examining the idea on alternative processes can also be taken into account. The existence of a runtime feedback loop to choose a suitable decentralization method was anticipated; however, it was excluded from this paper. Our current focus is on designing an algorithm to map runtime feedbacks to a suitable method of decentralization. In addition, the current work focuses on block-structured processes due to their extensive usage; however, similar effort may consider graph-structured processes as well. Furthermore, when a workflow engine approaches system thresholds, effective reconfiguration algorithms should go into action. The algorithms must re-fragment the workflows based on the current situation. The adaptable decentralization of workflows may be applied by some new cloud computing ideas such as software/platform as service or in the case of this paper workflow engine can be regarded as a service. In the cloud platform, a runtime environment may receive thousands of requests from around the world. Thus, tasks in a cloud may be described by workflow metaphors and are then conveniently decentralized and adapted to the runtime environment. In addition, satisfying stakeholders' requirements is of high importance; particularly, when they are ensured by Service Level Agreements (SLA) (Davide Lamanna and Emmerich, 2003). The idea presented in this paper may also be integrated with the SLA concept to better ensure the SLA commitments. Finally, alternative architectures for the ADWEF can be taken into account to improve the presented architectures.

Appendices. Supplementary data

Supplementary data associated with this article can be found, in the online version, at doi:10.1016/j.jss.2011.03.031.

References

Aalst, W.v.d., 2000. Loosely coupled interorganizational workflows: modeling and analyzing workflows crossing organizational boundaries. Information Management 37, 67–75.

Active-Endpoints. ActiveVOS Training. Available at: http://www.activevos.com/community-educationcenter.php.

Ahson, S.A., Ilyas, M., 2011. Cloud Computing and Software Services: Theory and Techniques. CRC Press.

Alonso, G., Mohan, C., Gunthor, R., Agrawal, D., Abbadi, A.E., Kamath, M., 1995. Exotica/FMQM: a persistent message-based architecture for distributed workflow management. In: IFIP WG 8.1 Workgroup Conference on Information Systems Development for Decentralized Organizations (ISD095) , Trondheim, Norway.

Atluri, V., Chun, S.A., Mukkamala, R., Mazzoleni, P., 2007. A decentralized execution model for inter-organizational workflows. Distributed and Parallel Databases 22, 55–83.

Active-Endpoints, 2008. ActiveBPEL Engine – Open Source BPEL Server. Available at: http://www.activevos.com/cec/training/content/a_selfPacedTraining/.

Barbara, D., Mehrotra, S., Rusinkiewicz, M., 1996. NCAs: managing dynamic workflows in distributed environments. Journal of Database Management 7, 5–15.

Baresi, L., Maurino, A., Modafferi, S., 2005. Workflow partitioning in mobile information systems. Mobile Information Systems, 93–106.

Bellifemine, F., Caire, G., Poggi, A., Rimassa, G., 2008. JADE: a software framework for developing multi-agent applications. Lessons learned. Information and Software Technology 50, 10–21.

Benatallah, B., Sheng, Q.Z., Dumas, M., 2003. The self-serv environment for web services composition. IEEE Internet Computing 7, 40–48.

Bhaduri, K., Das, K., Liu, K., Kargupta, H., Ryan, J., 2011. Distributed Data Mining Bibliography. Available at: http://www.cs.umbc.edu/~hillol/DDMBIB/.

Bruni, P., Caurim, M.H.S., Koerner, A., Law, C., Liberman, M., Schuh, W., Aerschot, E.V., Wang, J., Wansch, P., 2006. Powering SOA with IBM Data Servers. IBM.

Cai, T., Gloor, P.A., Nog, S., 1996. DartFlow: A Workflow Management System on the Web Using Transportable Agents. Dartmouth College, Hanover, NH.

Camel, A., 2004–2010. The Apache Software Foundation – Apache Camel. Available at: http://camel.apache.org/index.html.

Carriero, N., Gelernter, D., 1989. Linda in context. Journal of Communications of the ACM 32, 444–458.

Chafle, G.B., Chandra, S., Mann, V., Nanda, M.G., 2004. Decentralized orchestration of composite web services. Presented at the Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers and Posters, New York, NY, USA.

Daniel Wutke, D.M.F.L., 2008. Model and infrastructure for decentralized workflow. In: ACM/SAC 2008 , Fortaleza Cerara, Brazil, pp. 90–94.

Daniel Wutke, D.M., Leymann, F., 2008. Model and infrastructure for decentralized workflow. In: ACM/SAC 2008 Fortaleza Cerara, Brazil, pp. 90–94.

Daniel Wutke, D.M., Leymann, F., 2009. A method for partitioning bpel processes for decentralized execution. In: Erster zentraleuropäischer Workshop, ZEUS 2009, Stuttgart, Germany, pp. 109–114.

Das, S., Kochut, K., Miller, J., Sheth, A., Worah, D., 1997. ORBWork: a reliable distributed CORBA-based workflow enactment system for METEOR2.

Davide Lamanna, J.S.D., Emmerich, W., 2003. SLAng: a language for defigning service level agreements. In: The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03).

Dimitris, N.C., 2011. Cloud Computing Strategies. CRC Press.

Discenza, F.C.a.A., 2001. Modeling and managing interactions among business processes. Journal of Systems Integration 10 (April (2)), 145–168.

Dubouloz, B., Toklu, C., 2005. Discovering the most frequent patterns of executions in business processes described in BPEL. Lecture Notes in Computer Science 3806, 701.

Fabio, B., Agostino, P., Giovanni, R., 2001. Developing multi-agent systems with a FIPA-compliant agent framework. Software Practice and Experience 31, 103–128.

Fdhila, W., Yildiz, U., Godart, C., 2009. A flexible approach for automatic process decentralization using dependency tables. Presented at the Proceedings of the 2009 IEEE International Conference on Web Services.

Fidler, E., Jacobsen, H.A., Li, G., Mankovski, S., 2005. The PADRES distributed publish/subscribe system. Feature Interactions in Telecommunications and Software Systems VIII (2005)12.

Fortino, G., Garro, A., Russo, W., 2006a. Distributed workflow enactment: an agent-based framework. In: WOA2006.

Fortino, G., Garro, A., Russo, W., 2006b. From modeling to enactment of distributed workflows: an agent-based approach. Presented at the Proceedings of the 2006 ACM Symposium on Applied Computing, Dijon, France.

Freeman, E., Hupfer, S., Arnold, K., 1999. JavaSpaces(TM) Principles, Patterns, and Practice. Sun Microsystems Inc.

Garca-Banuelos, L., 2008. Pattern Identification and Classification in the Translation from BPMN to BPEL, in *OTM2008,* pp. 436–444.

Guo, L., Robertson, D., Chen-Burger, Y., Cisa, I., 2005. A Novel Approach for Enacting the Distributed Business Workflows Using BPEL4WS on the Multi-Agent Platform, pp. 657–664.

Han, J., Kamber, M., 2001. Data Mining: Concepts and Techniques. Academic Press.

Hillol, K., Liu, K., Kargupta, H., Ryan, J., 2004. Distributed data mining bibliography. Available at: http://www.cs.umbc.edu/~hillol/DDMBIB/.

IBM, 2001. Web Services Flow Language (WSFL). Available at: http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

Khalaf, R., Leymann, F., 2006. E role-based decomposition of businesses using BPEL. In: IEEE International Conference on Web Services (ICWS'06).

Khalaf, O.K.R., Leymann, F., 2008. Maintaining data dependencies across bpel process fragments. International Journal of Cooperative Information Systems, 17 (2008) 259–282.

Knuth, D., 1997. The Art of Computer Programming (TAOCP), 3rd ed. vol. 1. Addison-Wesley, Boston.

Kopp, O., Martin, D., W.D., L.F., 2008. On the choice between graph-based and block-structured business process modeling languages. In: Modellierung betrieblicher Informations systeme (MobIS 2008), Saarbrücken, Germany.

Li, G., Muthusamy, V., Jacobsen, H.-A., 2010. A distributed service oriented architecture for business process execution. ACM Transactions on the Web (TWEB) 4 (1) (article 2).

Maurino, A., Modafferi, S., 2005. Partitioning rules for orchestrating mobile information systems. Personal and Ubiquitous Computing 9, 291–300.

Mendling, J., Lassen, K.B., Zdun, U., 2008. On the transformation of control flow between block-oriented and graph-oriented process modelling languages.

International Journal of Business Process Integration and Management 3, 96–108.
Microsoft, 2001. XLANG. Available at: http://www.gotdotnet.com/team/xml_wsspecs/xlang-c.
Miller, J.A., Sheth, A.P., Kochut, K.J., Wang, X., 1996a. CORBA-based run-time architectures for workflow management systems. Journal of Database Management 7, 16–116.
Miller, J., Sheth, A., Kochut, K., Wang, X., 1996b. CORBA-based run-time architectures for workflow management systems. Journal of Database Management 7, 16–116.
Muth, P., Wodtke, D., Weisenfels, J.Aa., Dittrich, G.W.A.K., 1998. From centralized workflow specification to distributed workflow execution. Journal of Intelligent Information Systems 10 (2), 159–184.
Muthusamy, V., Jacobsen, H.-A., Chau, T., Chan, A., Coulthard, P., 2009. SLA-driven business process management in SOA. Presented at the Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research, Ontario, Canada.
Nanda, M.G., Karnik, N., 2003. Synchronization analysis for decentralizing composite Web services. Presented at the Proceedings of the 2003 ACM Symposium on Applied Computing, Melbourne, Florida.
Nanda, M.G., Chandra, S., Sarkar, V., 2004. Decentralizing execution of composite web services. ACM SIGPLAN Notices 39, 170–187.
Novell, 2010. Suse Linux Enterprise Real Time Extension. Available at: http://www.novell.com/products/realtime/.
OASIS, 2007. Advancing Open Standards for the information society. Available at: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
OMG, updated August 31, 2010. BPMN Specification. Available at: http://www.bpmn.org/.
Ouyang, C., Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., 2006. From business process models to process-oriented software systems: the BPMN to BPEL way. BPM Center Report BPM-06-27, BPMcenter.org.
Philip, B., Nigel, W., 2003. JavaSpaces in Practice. Addison-Wesley.
Sadiq, W., Sadiq, S., Schulz, K., 2006. Model driven distribution of collaborative business processes, pp. 281–284.
Safi Esfahani, F., Azmi Murad, M.A., Sulaiman, Md.N., Udzir, N.I., 2008. An intelligent business process distribution approach. Journal of Theoretical and Applied Information Technology 4 (December), 1236–1245.
Safi Esfahani, F., Azmi Murad, M.A., Sulaiman, Md.N., Udzir, N.I., 2009a. Using process mining to business process distribution. In: SAC2009 , Hawaii/Honolulu, pp. 1876–1881.
Safi Esfahani, F., Azmi Murad, M.A., Sulaiman, Md.N., Udzir, N.I., 2009b. SLA-driven business process distribution. In: IARIA/eKnow2009 , Mexico.
Safi Esfahani, F., Azmi Murad, M.A., Sulaiman, Md.N., Udzir, N.I., 2009c. A frequent path detection method to intelligent business process decomposition. In: WORLDCOMP/SWWS'09 – The International Conference on Semantic Web and Web Services , Las Vegas, Nevada, USA.
Safi Esfahani, F., Azmi Murad, M.A., Sulaiman, Md.N., Udzir, N.I., 2009d. A case study of the intelligent process decentralization method. In: WCECS , San Francisco, USA, 20–22 October 2009.
Sheng, Q.Z., Benatallah, B., Dumas, M., Mak, E.O.Y., 2002. SELF-SERV: a platform for rapid composition of web services in a peer-to-peer environment, p. 1054.
Sheth, A.P., Kochut, K., Miller, J.A., Worah, D., Das, S., Lin, C., Palaniswami, D., Lynch, J., Shevchenko, I., 1996. Supporting state-wide immunisation tracking using multi-paradigm workflow technology. Presented at the Proceedings of the 22nd International Conference on Very Large Data Bases.
Silva Filho, R.S., Wainer, J., Madeira, E.R.M., 2003. A fully distributed architecture for large scale workflow enactment. International Journal of Cooperative Information Systems 12, 411–440.
SUN-ORACLE, 2010. Sun Java Real-Time System. Available at: http://java.sun.com/javase/technologies/realtime/index.jsp.
Tan, W., Fan, Y., 2007. Dynamic workflow model fragmentation for distributed execution. Computers in Industry 58, 381–391.
Tomcat, A., 2009. The Apache Software Foundation – Apache Tomcat. Available at: http://tomcat.apache.org/.
Van der Aalst, W.M.P., Van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A., 2003. Workflow mining: a survey of issues and approaches. Data & Knowledge Engineering 47, 237–267.
Vanhatalo, J., Völzer, H., Koehler, J., 2009. The refined process structure tree. Data & Knowledge Engineering 68, 793–818.
Viroli, M., Denti, E., Ricci, A., 2007. Engineering a BPEL orchestration engine as a multi-agent system. Journal of Science of Computer Programming (January).
Wodtke, J.W.D., et al., 1996. The Mentor project: steps toward enterprise-wide workflow management. ICDE, 556–565.
1998. Workflow Management Coalition: Process Definition Interchange, vol. 1.0.
Yildiz, C.G.U., 2007a. Centralized versus decentralized conversation-based orchestration. In: Technology, IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services (EEE 2007), CEC/EEE, pp. 289–296.
Yildiz, C.G.U., 2007b. Towards decentralized service orchestrations. Presented at the Proceedings of the 2007 ACM Symposium on Applied Computing, Seoul, Korea.
Zhai, Y., Su, H., Zhan, S., 2007. A Data Flow Optimization Based Approach for BPEL Processes Partition.

Faramarz Safi Esfahani is currently a PhD candidate in University of Putra Malaysia (UPM). He has also been on faculty at Department of Computer Science, Islamic Azad University, Najafabad branch since 2002. His research interests include distributed systems and architectures, software agents, and intelligent computing.

Masrah Azrifah Azmi Murad received her PhD from the University of Bristol, UK in 2005. She is currently a senior lecturer at the Department of Information Systems, Universiti Putra Malaysia. She is a member of IEEE and IEEE Computer Society. Her current research interests include text mining, applied informatics, and automated software engineering.

Md. Nasir bin Sulaiman is a lecturer in Computer Science in Faculty of Computer Science and Information Technology, UPM and as an Associate Professor since 2002. He obtained PhD in Neural Network Simulation from Loughborough University, U.K. in 1994. His research interests include intelligent computing, software agents and data mining. He is currently Head of Intelligent Computing Research Group, FSKTM, UPM.

Nur Izura Udzir is a Senior Lecturer at the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM). She received her Bachelor of Computer Science (1996) and Master of Science (1998) from UPM, and her PhD in Computer Science from the University of York, UK (2006). She is a member of IEEE Computer Society. Her areas of specialization are access control, secure operating systems, coordination models and languages, and distributed systems. She is a member of the Information Security Group at the faculty.