Scientia Iranica D (2016) 23(3), 1195–1216

SCIENTIA IRANICA

Sharif University of Technology

Scientia Iranica

Transactions D: Computer Science & Engineering and Electrical Engineering

www.scientiairanica.com

SCW+: A service-oriented framework for cloud workflow systems

M. Torkashvan and H. Haghighi*

Department of Computer Science and Engineering, Shahid Beheshti University G. C., Tehran, P.O. Box 1983963113, Iran.

Received 22 October 2014; received in revised form 16 August 2015; accepted 14 December 2015

KEYWORDS

Cloud workflow;
Cloud computing;
Cloud framework.

Abstract. Cloud workflow is a special type of cloud computing systems which mainly concentrates on workflow management. One of the major issues with cloud workflow systems is automatic multi-cloud workflow management. This paper proposes a service oriented framework for cloud workflow management which integrates heterogeneous multi-cloud platforms to provide integrated applications for users by minimizing the human intervention as far as possible. The proposed framework involves a language to define some basic entities for environments and uses these definitions to integrate applications and services in a cloud workflow. This framework has already been implemented. In addition, its main operations have been evaluated by a case study and the results show that the framework works properly as a cloud integrator and main activities of the framework are done automatically with a reasonable performance.

1. Introduction

There are some types of service platforms which facilitate the execution of distributed applications like workflow management systems. A workflow management system defines, manages, and executes workflows on computing resources [1]. According to this definition, there can be many platforms as computing resources, such as grid computing [2] and cloud computing, to run workflow management systems. Cloud workflow systems are those workflow management systems which are executed based on cloud computing as their underlying infrastructure. When the underlying layer consists of more than one cloud provider, the notion of multi-cloud workflow management emerges.

Although some cloud workflow platforms have been introduced, one of the main issues in this field is still heterogeneous multi-cloud workflow management. Heterogeneity in this context refers to different cloud platforms serving as a unified platform from different cloud providers, that is, each of these cloud platforms can advertise different services, and thus, a particular user may need a workflow whose atomic services are spread across various cloud platforms. Another issue in workflow management systems is to reduce the human intervention. While some existing systems do not need human intervention in their managerial operations, they are not practicable enough to be used by end users (i.e., they need skilled users to define ontologies, entities, and some other parts related to the workflow management). In order to conquer the aforementioned issues and, in other words, to provide an easy-working and heterogeneous multi-cloud workflow management with the least possible human intervention, this paper proposes a service-oriented framework for cloud workflows. This framework is named SCW+ that stands for Service-oriented Cloud Workflow + Ontology and Semantic web.

SCW+ is a framework with five essential layers each having access to the provided services from its

*. *Corresponding author. Tel.: +98 21 29904190;*
E-mail addresses: Milad.Torkashvan@gmail.com (M. Torkashvan); h_haghighi@sbu.ac.ir (H. Haghighi)

lower layer. To achieve the goals of SCW+, this paper defines one environment for each customer of SCW+ (although some customers may share their environments). Each environment has four major entities: Event, Task, Abstract service, and Concrete service operation. SCW+ utilizes an ontology language, which is called ETAS (Event, Task, and Abstract Service), for defining these entities and their relationships. ETAS provides a unified language and ontology for SCW+ which helps SCW+ to step toward unifying heterogeneous service provider as its underlying layer.

The rest of this paper is organized as follows: Section 2 discusses the related work; Section 3 shows an abstract view of SCW+; Section 4 gets deep into the most significant layer of SCW+; Section 5 provides some descriptions about the implementation of SCW+; Section 6 evaluates SCW+; Section 7 gives a comparison of SCW+ with other similar platforms; and finally Section 8 concludes the paper.

2. Related work

A lot of works have been done in the context of workflow management and its related areas, such as service composition and semantic web services. On the other hand, there are some cloud platforms which support workflow management. In this section, some backgrounds on service composition and workflow management are introduced, and then some cloud platforms, which support workflow management, are reviewed.

With the emergence of SOA (Service Oriented Architecture) [3], many workflow applications have been implemented using web services. One of the main benefits of SOA is reusing services; semantic web [4,5] has improved reusability by accessing the content of web services. Some languages, such as WSDL-S [6] and SA-WSDL [7], have enriched WSDL through semantic concepts; also, some descriptive languages, such as OWL-S [8], WSMO [9], and SWSO [10], have provided a semantic description based on IOPE (Input, Output, Precondition, and Effect). Since web service composition is needed to execute business processes and workflows, various languages have been designed to support service composition, such as BPMN [11], WFF [12,13], BPEL [14], XLANG [15], WSFC [12,13], and YAWL [16]; they are used by various workflow management systems and service composition frameworks.

SODIUM [17] and OPUCE [18] introduce two frameworks including a set of languages and tools needed for creating and executing business processes based on heterogeneous services. Unfortunately, using these frameworks, business process modeling is done manually. There are some other frameworks, such as SOA4ALL [19], MashArt [20], CRUISe [21], and MacroFlow [22], for service orchestration and workflow execution. The main problem with SOA4ALL is that input/output is defined manually through data flow models; MashArt, CRUISe, and MacroFlow all support dynamic service composition, but none of them support multi-provider service composition.

In this section, we have compared SCW+ with five other cloud platforms: SwinDew-C [23], WTE+ [24], Everything-as-a-service platform [25], Aneka [26-28], and SOCCA [29]. There are other known cloud platforms which support workflow management, such as Microsoft Azure and WSO2 Stratos. Tables 1 and 2 show the main strengths and weaknesses of the mentioned platforms, respectively. The meanings behind the given items or criteria (from both tables) are explained here:

- **Market orientation and providing services according to cost and time:** Cost and time are the most important parameters for selecting services which make a platform market-oriented. For example, according to the user request, a platform may select a composition of services with the lowest possible price;
- **Considering both types of workflow and non-workflow applications:** Some applications have one or more specified workflows which can be performed by the composition of web services; on the other hand, some other applications do not have specific service-based workflows and they perform some duties in a functional manner;
- **Utilizing more than one algorithm for scheduling:** There are many options for selection of the underlying web services so as to compose and execute the final workflow. This item refers to the platforms which do not rely on only one algorithm for service selections;
- **Supporting run-time software production and configuration:** This item addresses the platforms which give users the ability to define workflows dynamically and add new functionality (or configure an existing functionality) in their systems during runtime;
- **Possibility for manually manipulating the created mashups:** Since a mashup consists of some services in a proper manner to perform a specific job, in some platforms there is a facility for users to change these mashed-up services or the way the services are related to each other;
- **Proposing a programming model:** Does the platform prefer a new programming model or it just needs the developers use the platform APIs and deploy their applications?
- **Providing a common environment for heterogeneous service providers:** This item specifies if

Table 1. Cloud platform strengths.

Criterion	SwinDew-C	WTE+	Everything-as-a-service	Aneka	SOCCA	Azure	WSO_2
Market orientation and providing services according to cost and time	✓		✓				
Considering both types of workflow and non-workflow applications	✓						
Utilizing more than one algorithm for scheduling	✓						
Supporting run-time software production and configuration		✓					
Using semantic web and ontology for facilitating service composition		✓					
Possibility for manually manipulating created mashups		✓					
Proposing a programming model		✓		✓		✓	✓
Providing a common environment for heterogeneous service providers			Weakly				
Supporting multi-service provider in the infrastructure layer				✓	✓		
Supporting software life-cycle				✓	✓		
Supporting MapReduce model				✓			✓
Supporting automatic service composition				✓	✓		
Supporting ontology based service description					Weakly		
Supporting multi-tenancy					✓		✓
Simplicity of producing applications						✓	
Supporting complex event processing							✓
Supporting enterprise policies			✓				✓

a platform provides a common (shared) environment for some service providers so as to integrate them;

- **Supporting multi-service provider in the infrastructure layer:** It specifies whether a cloud framework uses other clouds to provide the service composition or not. Then, it shows that the cloud framework can act as a cloud integrator;
- **Supporting software lifecycle:** This parameter specifies if a cloud framework can support the entire software development lifecycle (analysis, architecture, design, implementation, test, and deployment);
- **Supporting map-reduce model:** This item refers to a platform that provides users with a built-in map-reduce infrastructure (such as Hadoop) such that the users can develop applications interacting with this infrastructure;
- **Supporting automatic service composition:** Does the platform compose services automatically or it needs the user to select them?
- **Supporting ontology-based service description:** This parameter usually appears in cloud frameworks, which provide service composition automatically, and specifies whether the cloud framework regards some semantic data in web service description. This data, which can appear as ontologies, facilitates the process of service composition;
- **Supporting multi-tenancy:** Refers to using com-

Table 2. Cloud platform weaknesses.

Criterion	SwinDew-C	WTE+	Everything-as-a-service	Aneka	SOCCA	Azure	WSO_2
No consideration on service selection algorithm and its effects on task scheduling	✓						
Depending on manually workflow definition in design time	✓		✓			✓	✓
No support for run-time configuration	✓		✓	✓	✓	✓	✓
Service description inflexibility		✓					
Cumbersome request creation		✓					
Depending on specialist users		✓					
Not using ontology for describing heterogeneous resources and services			✓	✓			
Single cloud provider						✓	✓

mon resources by different users and providing isolation for these users;

- **Supporting complex event processing:** Event processing is the analysis of information streams [30] and concluding from them. CEP (Complex Event Processing) is an event processing that combines data from multiple sources [31] to infer patterns that suggest more complicated situations to identify meaningful events [32] and respond to them as quickly as possible. This characteristic specifies whether the cloud framework supports CEP or not;
- **Supporting enterprise policies:** Does the platform consider (and have the ability to define) user (or enterprise) policies so as to affect the service selection process?
- **No consideration for service selection algorithm and its effects on task scheduling:** This item refers to the platforms that do not have any specific strategy for service selection and the user marshals services, together, manually;
- **Dependence on manual workflow definition in design time:** In service-oriented software development processes (like SOMA), service composition is usually addressed in design time and, in most of the cases, this is done manually. Many platforms strongly depend on this type of service composition;
- **No support for run-time configuration:** This item addresses the platforms which do not give users the ability to add new functionality (or configure an existing functionality) in their systems during runtime (in opposite to the "Supporting run-time software production and configuration" item);
- **Service description inflexibility:** Many of the platforms rely only on one or some specific service description languages (like WSDL); this item refers to platforms that are not flexible in service descriptions and can just accept some specific languages;
- **Cumbersome request creation:** In some platforms, the user (or the client-side application) needs to generate a request in a complex manner (like SOAP protocol) so as to trigger a workflow, whereas there are other platforms by which the user can trigger a workflow only through a simple message;
- **Dependence on specialist users:** Normally, service composition is not a common job for anyone to do and it should be performed by a software engineer with a great background and experience; after that, the process of system maintenance (like service/workflow configuration) must be done by these engineers. In this way, many platforms need system engineers to interact with each other, which are addressed by this item. It is not a weakness per se; yet, in comparison to the platforms that attempt to automatize most parts of the service composition (in design-time and even run-time), such as service selection and work flow generation, it can be referred to as a weakness;
- **Not using ontology for describing heterogeneous resources and services:** This item is in opposition to the mentioned "Supporting ontology-based service description" item in the strength criteria;
- **Single cloud provider:** In opposite to the mentioned "Supporting multi-service provider in the infrastructure layer" item in the strength criteria;

- **Simplicity of producing applications:** This item refers to platforms that provide a simple programming model for the developers. In these platforms, developers can focus on the main concerns and they are not supposed to be worried about messaging, communicating with services, triggering the workflows, and so forth.

Remarks:

1. "Providing a common environment for heterogeneous service providers" is considered "weak" for "Everything-as-a-service" because it has not mentioned any details about service level and task level scheduling algorithm;
2. "Supporting ontology-based service description" is considered "weak" for "SOCCA" because it has not mentioned its utilized ontology language;
3. "Service description inflexibility" is considered for "WTE+" because it merely depends on OWL-S. In addition, "Depending on specialist users" is considered for "WTE+" because describing services through OWL-S needs special skills;
4. "Cumbersome request creation" is considered for "WTE+" because its messages should be in the XML format.

Considering strengths and weaknesses of the related work, the main goal of SCW+ is to integrate heterogeneous cloud platforms while diminishing the need for skilled users and human agent involvement (through providing automatic workflow management). So, SCW+ is aimed as a framework for cloud workflow with the following strength points:

- Multi-cloud support;
- Automatic service composition (with considering QoS) through a special service-level scheduling algorithm;
- Automatic task workflow design;
- Supporting enterprise policies;
- Complex event processing support;
- Ontology-based entity description;
- No mere dependency on a particular service description;
- Ease of work in user-side.

SCW+ has also some weaknesses which are as follows (we try to overcome them in our future work):

- No support for map-reduce programming;
- Proposing a special task-level scheduling algorithm;
- No support for multi-tenancy in the resource level.

3. SCW+

Since SOA is an appropriate approach to develop cloud-based software applications [30], this paper proposes SCW+ as a service-oriented framework for cloud workflow management systems. Inputs for SCW+ are defined as events: Every occurrence in the environment will be sent to SCW+ as an event and will be processed according to event processing definitions [31,32]. SCW+ proposes four essential entities (Figure 1) for each enterprise or environment on which the cloud is deployed: Event, Task, Abstract service, and Concrete service operation. These entities are defined for the system through ontology; they help the system to find what tasks can meet the goals of an event and what abstract services can meet the goals of a task. Finally, a concrete service composition will be generated to provide actual results. An abstract service is the container of concrete service operations which realize the functionality of that abstract service. We propose an ontology and XML-based language to describe all of the mentioned entities. This language is named ETAS (stands for Event, Task, and Abstract Service) and is described in [33].

3.1. SCW+ layers

This section discusses different layers of SCW+. In addition to three conventional layers, i.e. Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS), SCW+ proposes two novel layers, named INtelligence-as-a-Service (INaaS) layer and Environment layer.

3.1.1. Environment layer

This is the topmost layer in SCW+ which provides an abstract environment for the users of its underlying layers. Each customer has at least one environment, but more than one customer may have a common environment. An environment may be an enterprise, a cloud, or any external application. The only matter is that the administrator must define connection channels between environments. The only communication path among different clouds and costumers is via environments; each environment can access the entities of another cloud (such as services, tasks, etc.) through the environment of that cloud if permitted.

The data which can be interchanged between two environments depends on the type of environments. For instance, if both environments are clouds which have been built based on SCW+, all entities can be interchanged between environments (if permitted); however, if one of them is based on some other cloud platform (such as Azure), or it is an external application or enterprise, it is possible to access its service repository and extract its web services (service extraction is the responsibility of the Service Monitor component that will be described later).

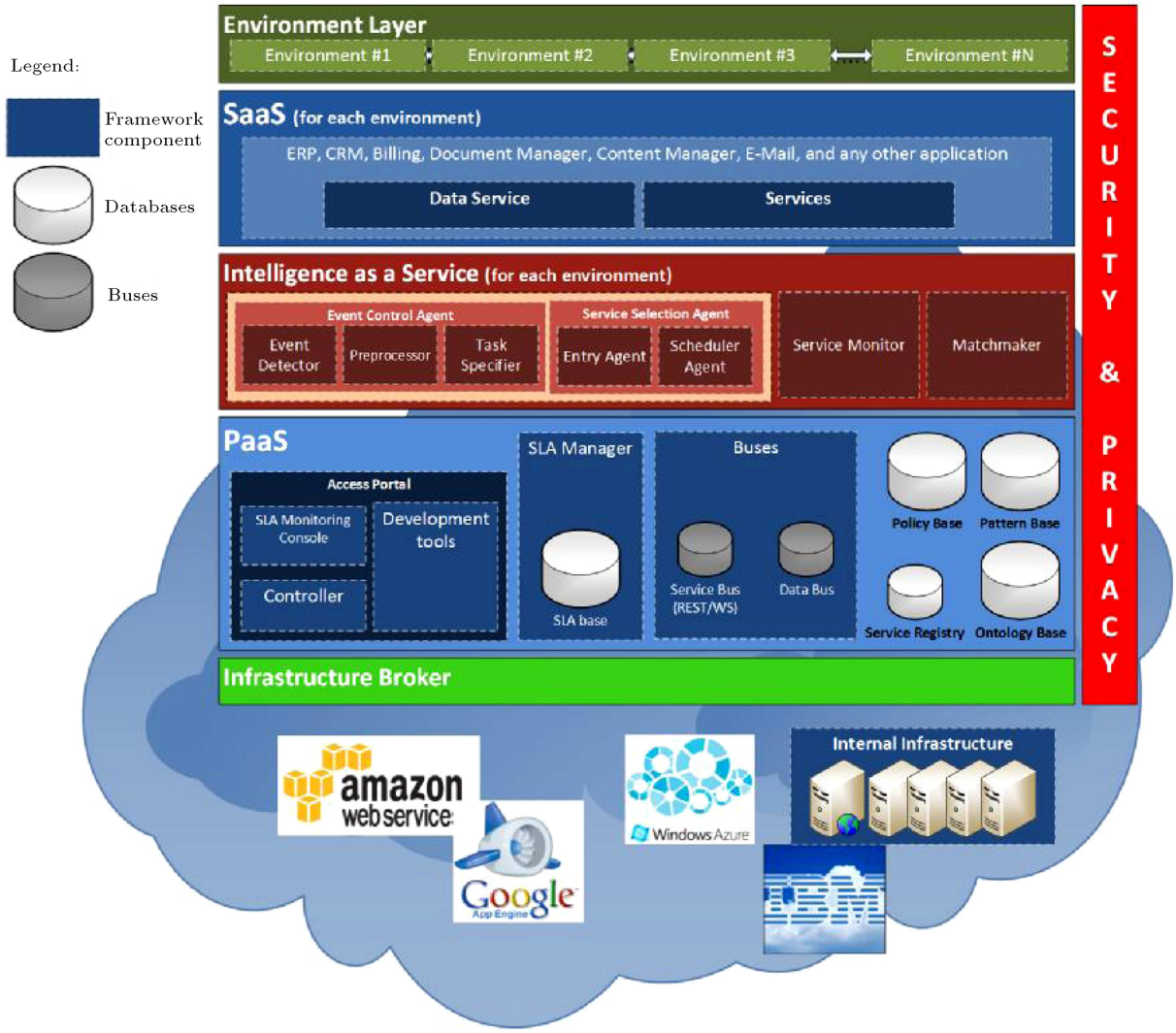

Figure 1. SCW+ architecture.

3.1.2. SaaS layer

In this layer, environments can access web services and data services. Every application such as ERP, CRM, and SCM can be utilized by the user through this layer. The abstraction level which is provided through the environment layer along with this layer realizes multi-tenancy characteristic for SCW+ in the application (service) level (since more than one costumer can have common environments and utilize the software and resources in those environments), but not in the resource level (because we do not focus on resource sharing and isolation). Software systems in this layer are web-based and can be implemented by any programming language. The only requirement is that the developer must use a user development kit which proposes an event-driven programming model to connect to the cloud and invoke events. Events occur through this layer and are sent to the lower layer (INaaS) to be processed.

3.1.3. INaaS layer

This layer is the heart of SCW+ and performs the main functionality of SCW+ described before. Therefore, INaaS is in charge of automatizing the procedure of mapping goals to the composed services. The main portion of this paper is to describe how INaaS does this job.

3.1.4. PaaS layer

This layer provides services and tools which are required to develop software systems. PaaS provides users with a graphical interface to connect to their environments and define events, event relationships and transformations, tasks, goals, policies, SLAs, and so forth. Significant components in PaaS are as follows: To refuse repetitive operations, Pattern Base is a repository which helps Event Control and Service Selection Agents (in INaaS) to remember the last task or service composition which has been selected for a particular event. Service base stores a list

of existing concrete web services and their related properties supposed to be extracted by Service Monitor (in INaaS). Ontology base is the place where goals, categories, events, tasks, abstract services, and their relationships are maintained. SLA Manager works based on CSLA (Cloud Service Level Agreement) framework described in [34]; SLA base stores the agreed documents between consumers and providers about service level agreements. There are also two types of buses: service bus and data bus. The data bus is used by internal components when they need to access some data from the existing bases; the service bus can be used by both internal components and external users to utilize the existing services. Access Portal interacts with privileged users directly; it provides abilities to develop services, configure SLAs, define events and tasks, and provide monitoring information and SLA definitions.

3.1.5. Interface broker

This layer stands on virtual machines and infrastructure resources to provide a virtual access to resources. Requests to access low-level resources, such as web services, files, and databases, are sent to this layer. A request will be compiled according to the target infrastructure type (e.g., IBM cloud and Amazon EC2) and the result will be sent to the requester (i.e., the layer component which has triggered the request). Requests are sent to the broker from PaaS which itself receives requests from INaaS; INaaS requests are triggered by events from SaaS. There is not an IaaS layer in SCW+, because instead of dedicating a particular layer SCW+, along with its infrastructure broker, is designed in a way that it can use any other clouds or infrastructures as the physical layer.

3.2. XML-based languages

Two XML-based languages, i.e. ETAS and BEF, used in INaaS layer, are introduced. The main purpose of ETAS is to define business entities from the highest level to the lowest one; and BEF (Business Entity Flow) is an XML-based language whose main purpose is to describe a composition of business entities (generated automatically). The entire INaaS layer acts as a knowledge management system (including an ontological model and an inference mechanism) in a way that ETAS and BEF are the ontological descriptions of the environment, and the main operational components of INaaS (Section 4) form the inference part of the system. We have used an ontological model because the main job of the system is to find out which services can make a "good" fit for the intended goal of the user. The "goodness" can be decided based on the price, time, or any other user's objective of the selected services; ETAS and BEF both provide the INaaS layer components with the required data so as to make a proper decision. In [33] we have elaborated on ETAS and BEF in more depth.

3.3. Dynamic view on SCW+

SCW+ has a general view on everything that can provide services, such as other cloud services, applications, enterprises, and so forth. Therefore, Infrastructure Broker provides the communication with these different environments. There are many managerial operations such as workflow definition, service definition, SLA definition, event definition, monitoring the operations, and so forth which are provided for cloud users with different access limitations in the PaaS layer. SCW+ is event-driven, that is events which are defined in the PaaS layer may occur in the user-side, and served by SCW+ in the server-side. Responding to events starts from the INaaS layer to construct proper workflows for the occurred events by Event Control Agent; then, SCW+ selects appropriate services to execute the constructed workflows by Service Selection Agent. INaaS continuously searches for new services in the defined environments by Service Monitor component and attempts to convert descriptions of the found services into the unified service description languages of SCW+ by matchmaker component. These unified languages provide the ability for SCW+ to manage services from different cloud providers. SaaS layer is the most concrete layer in SCW+ which contains the applications that are using services from SCW+ provider. These applications are typically workflow-based and trigger events which are served by the INaaS layer. The topmost layer is called Environment layer which provides a logical view for each different defined environment to have its own view on SCW+ and its applications. Figure 2 shows the main activities in SCW+ such as event definition, service monitoring, matchmaking, event detection, service selection, and execution.

Note that in Figure 2, AS-flow refers to abstract service flow and CS-flow refers to concrete service flow. Figure 2 depicts three main scenarios in SCW+, which are numbered from 1 to 3, on the point where they start (The numbered circles refer to the number of different scenarios at the start point.) Scenario 1 refers to entity definition by the user (mainly, administrator users); to do so, the user works with utilities provided in PaaS layer to define the entities and PaaS stores them on ontology base. Scenario 2 refers to environment definition and what happens after that. It shows that SCW+ connects to the defined environments through its infrastructure broker, then the service monitor component tries to find new service descriptions from those environments to store them in service repository (in PaaS) and convert them into ETAS description and store them on ontology base. Scenario 3 shows what happens when an event occurs in SaaS layer from behalf

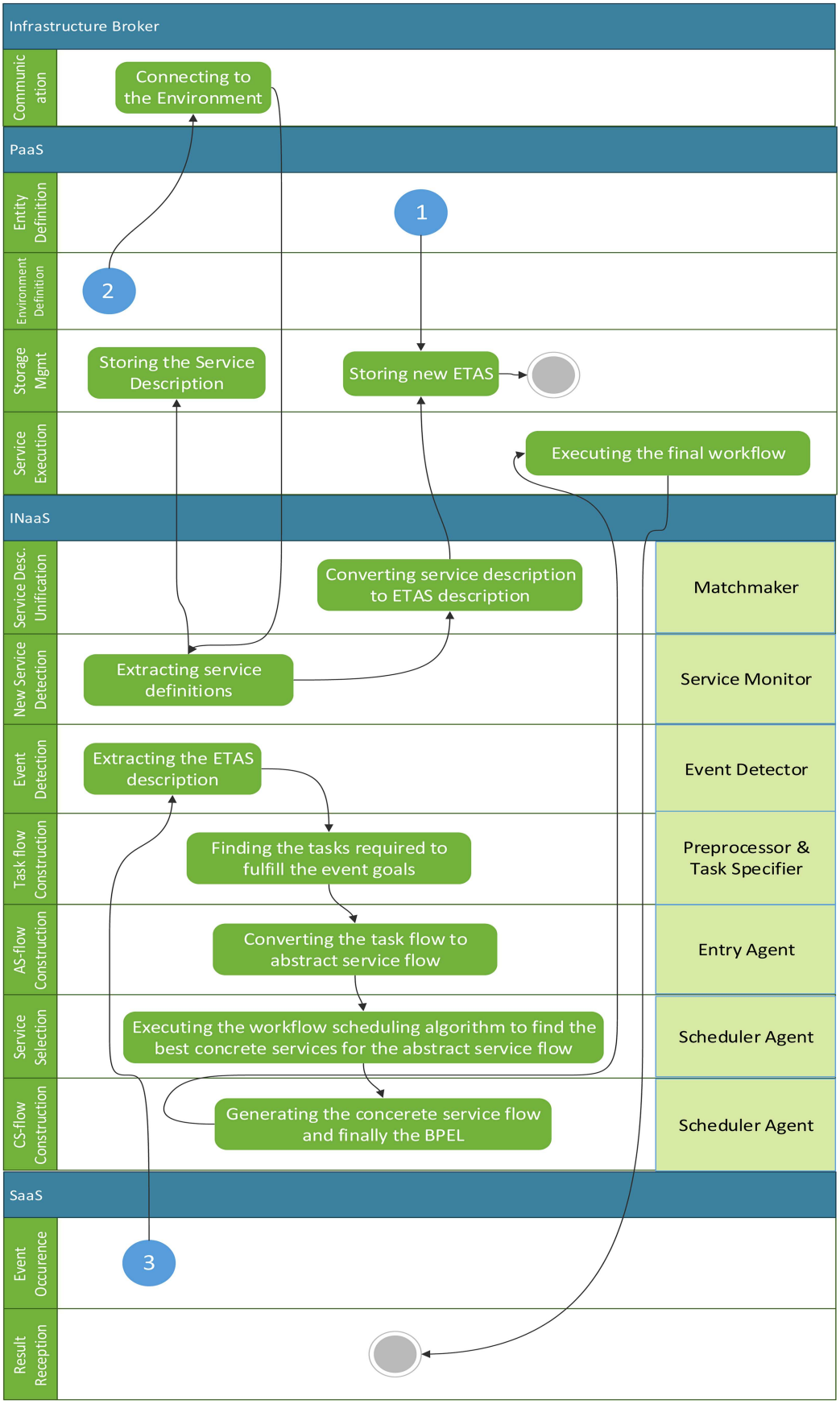

Figure 2. SCW+ activity diagram.

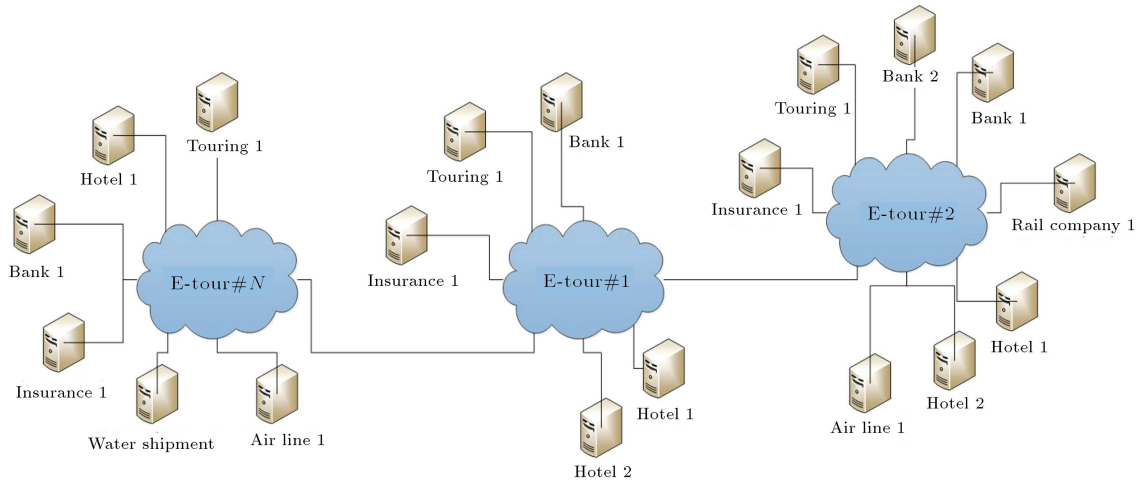

Figure 3. A sample of e-touring company.

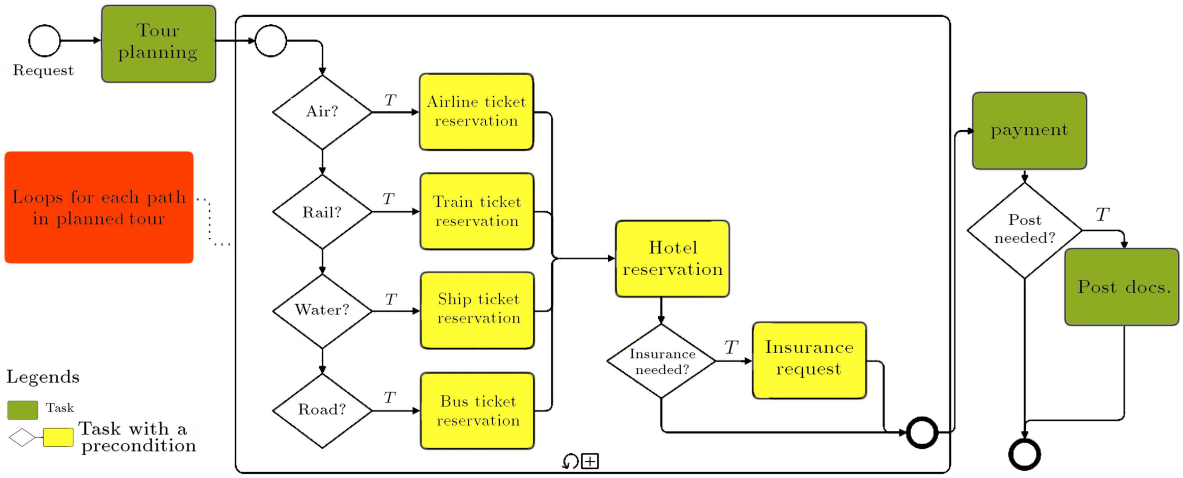

Figure 4. E-tour company workflow.

of an application. It shows that when INaaS detects an event, it tries to extract ETAS description of the event and, according to its goals, SCW+ finds the required tasks, abstract service flow, concrete service flow, and so forth to provide the appropriate results for the occurred event.

3.4. Running example

In the remaining of the paper, we get deep into INaaS layer and follow its different components and concepts along with a real world case to elucidate them. The case is an e-touring company with so many customers. This company is connected to many other touring companies to coordinate tours in different places; it, also, is in connection with many hotel and restaurant services. This company uses various banking services to do financial operations. In addition, it is in relation with many road traveling companies, airlines, shipment companies, and rail road companies to transfer passengers. If it is required to post any document, the company communicates with different post companies. Finally, the company may need some insurance services. Obviously, there are many connections and services from many service providers to plan a tour for customers (Figure 3). The tour planning workflow is depicted in Figure 4.

4. Elaborating on INaaS layer

Figure 5 shows the components of INaaS and the relationships among them. INaaS consists of four main components: Event Control Agent, Service Selection Agent, Service Monitor, and matchmaker. Table 3 represents the operations of INaaS.

The input for INaaS is provided through Event Detector (from Event Control Agent). Steps 1 and 2 are done by Task Specifier. Step 3 is done by Entry Agent (from Service Selection Agent) and Scheduler Agent. Entry Agent performs Step 4. In the proceeding, Service Monitor will be discussed first. Then, matchmaker, Event Control Agent, and Service Selection Agent will be elaborated; however, an overall

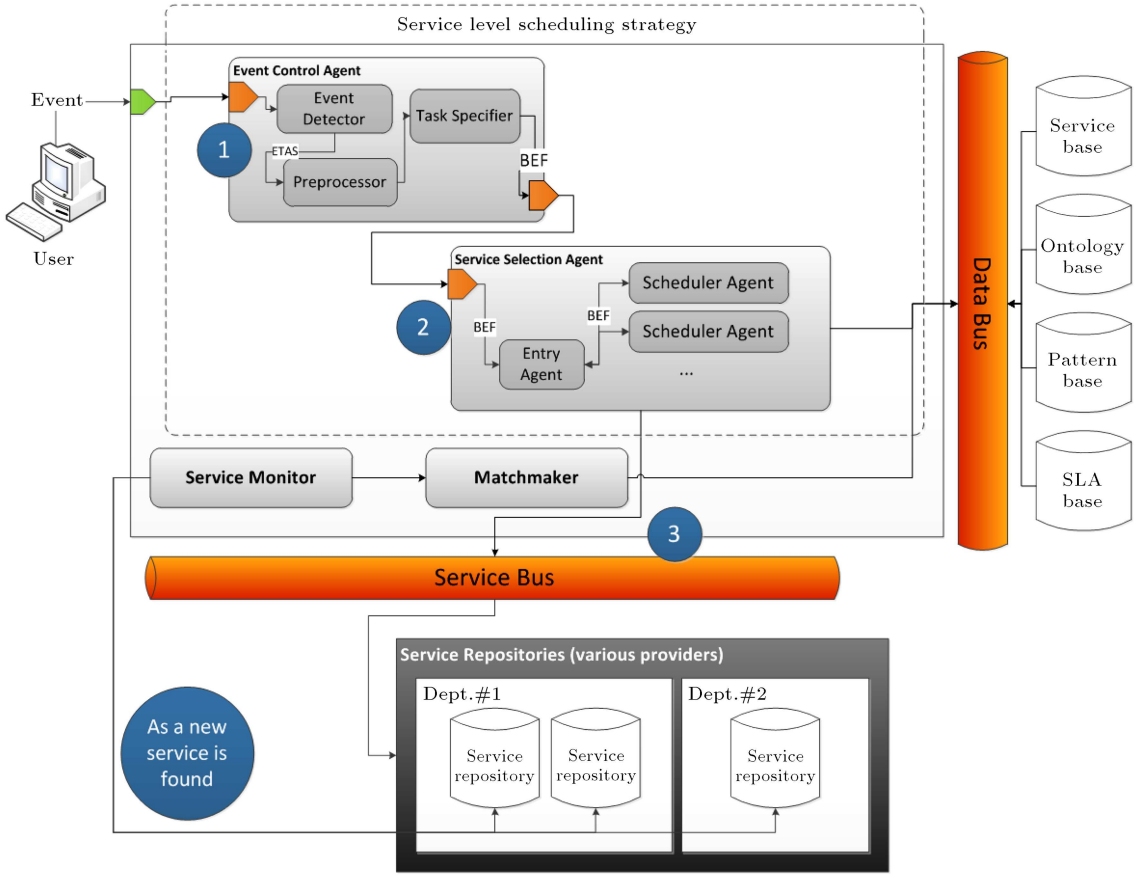

Figure 5. INaaS architecture.

Table 3. INaaS operations.

INaaS operation		
Input: Event occurrence from the user side, ontology base		
Output: BPEL file with the most suitable composition of services to gain the final result		
INaaS operations	**Step 1:**	A list of tasks is gathered that have the same goals and categories in comparison to the occurred event (the ontology base is searched for).
	Step 2:	A BEF file is made for each of the above tasks according to its preconditions and effects.
	Step 3:	Regarding all tasks, the most suitable abstract service composition is selected and described by a BEF.
	Step 4:	The BEF description of the selected abstract service composition is converted to a BPEL file.

overview of INaaS components can be found in the following:

- **INaaS:** Generating a BPEL with the most suitable composition of services for an occurred event;
  - **Service monitor:** Extracting web services (and their operations) from connected environments;
  - **Matchmaker:** Finding or creating an abstract service for each extracted web service operation;
  - **Event Control Agent:** Generating task level BEFs which fulfill an occurred event;
    - **Event detector:** Receiving an event, searching in pattern base if there is a BEF for that

event or not, and executing event preconditions;
  - **Task specifier:** Finding tasks that fulfill an occurred event and generating task level BEFs for them;
  - **Preprocessor:** Executing preprocesses of an occurred event.
- **Service Selection Agent:** Mapping the found task level BEFs into abstract service compositions (described via abstract service level BEFs), selecting the most suitable service composition, and generating its corresponding BPEL file;
  - **Entry agent:** Selecting the most suitable abstract service composition and mapping it into a BPEL file;
  - **Scheduler agent:** Mapping a task level BEF into an abstract service composition.

4.1. Service monitor

The Service Monitor and matchmaker components facilitate the integration of heterogeneous service providers. The main purpose of Service Monitor is to extract web services which are defined in other connected environments. These services along with their needed information (input, output, service goal, service category, service precondition, service effect, QoS information, and price) will be sent to matchmaker to be assigned to an appropriate abstract service. Operations of the Service Monitor component are shown in Table 4.

There is no constraint for the service description language (it can be WSDL, WSDL-S, SA-WSDL, OWL-S, and WSMO) and Service Monitor extracts the needed information according to the type of description. Table 5 shows what information can be extracted from a service description.

After extracting the required information, it will be sent to the matchmaker component which is described in the next subsection.

4.2. matchmaker component

This component benefits from an algorithm that is used for generating abstract services. As mentioned before, an abstract service contains a group of web service operations with the same functionality; finding or creating suitable abstract services for concrete web service operations is the main purpose of the matchmaker algorithm. matchmaker does its duty by

Table 4. Service monitor operations.

Service monitor operations		
Input: Web service description **Output:** Set of (input, output, goal, category, precondition, effect, QoS, price)		
As a service is found (from the connected environments) by service monitor	**Step 1:**	Required information {input, output, goal, category, precondition, effect, QoS, price} is extracted according to the type of service description.
	Step 2:	The extracted information is sent to matchmaker

Table 5. Extractable information from web service descriptions.

Required information	**Web service description parts**
Input	Web service data contract
Output	Web service data contract
Goal	When there is not enough annotation (such as WSDL): name of the service operation in the web service operation contact It is particularly specified when there is enough annotation (such as OWL-S)
Category	When there is not enough annotation (such as WSDL): name of the web service It is particularly specified when there is enough annotation (such as OWL-S)
Precondition	It is specified if the language is IOPE-based (such as OWL-S and WSDL-S)
Effect	It is specified if the language is IOPE-based (such as OWL-S and WSDL-S)
QoS	QoS data fields are addressed either directly in the description or its annotation
Price	Price is addressed either in the description or in the annotation

Table 6. Matchmaking algorithm operations.

matchmaker algorithm		
Input: Service operation (input, output, goal, category, precondition, effect, QoS, price)		
Output: An abstract service		
As the input is received	**Step 1:**	A list of existing abstract services is gathered.
	Step 2:	An abstract service with the same or similar goal, category, precondition and effect as input values is found; if not found, a new abstract service is created.
	Step 3:	Service operation name is appended to the set of concrete services of the found/created abstract service.
	Step 4:	Service operation input parameters are added to the abstract service input set, and service operation output parameters are added to the abstract service output set.
	Step 5:	The Min SLA parameter field of the abstract service is modified (if needed).

receiving input, output, goal, category, precondition, effect, QoS, and price information related to a service operation. Table 6 shows the main operation of the matchmaker algorithm.

A summary of the matchmaker algorithm is depicted in Figure 6. Suppose that AS is an Abstract Service and I is the received concrete service operation. AS will be selected as a matched abstract service if:

$$SimilarityOf(I.Category, AS.Category) = 1.0 \wedge SimilarityOf(ToString(I.Effect), ToString(AS.Effect)) = 1.0 \wedge SimilarityOf(ToString(I.PreCondition), ToString(AS.PreCondition)) = 1.0 \wedge SimilarityOf(I.Goal, AS.Goal) = 1.0.$$

SimilarityOf() is a function which has two input arguments and checks whether these two arguments are lexically similar or not (in our implementation, we have used WordNet.Net project which measures the similarity according to the WordNet lexical database [35]). The result of this function is an integer number showing how similar are two inputs (in the range of 0.0 to 1.0 where 1.0 means two inputs are the same). *ToString()* is used to convert *PreCondition* and *Effect* into a text string (Step 1 and 2).

If the above condition is not met, a new abstract service must be generated and then the following steps will go on. In Step 3, $I.OperationName$ is appended

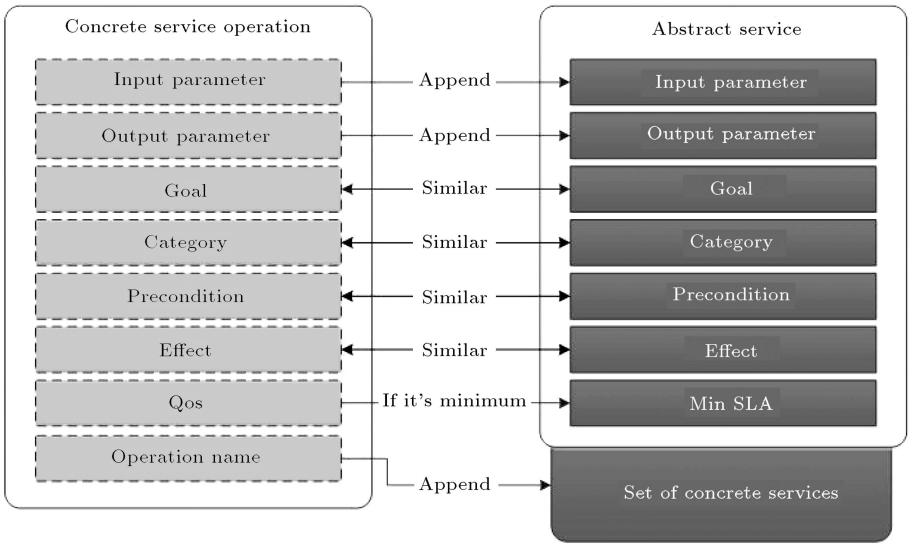

Figure 6. Matchmaking a concrete service operation with a typical abstract service.

Table 7. BEF files which meet event goal, category, input, and output are generated.

Event control agent		
Input: Event, ontology base, pattern base		
Output: A set of task level BEF files		
As an event is received by the agent	**Step 1:**	Event ETAS description is extracted.
	Step 2:	Event precondition is executed: going to the next step if the precondition is met; otherwise, replying an error message.
	Step 3:	Pattern base is investigated to find a pattern for the event; jumping to the Service Selection Agent, if the pattern exists (pattern base keeps found task level BEFs for events in past).
	Step 4:	Event preprocesses are executed if exists.
	Step 5:	Existing tasks are extracted from ontology base.
	Step 6:	BEF files which meet event goal, category, input and output are generated.

Table 8. Task specifier operations.

Task specifier		
Input: Event input, event output, event goal, event category, and ontology base.		
Output: A set of task level BEFs.		
As an input is received from event detector	**Step 1:**	A list of existing tasks is gathered from ontology base.
	Step 2:	Tasks that have the same or similar goal and category in comparison to the input event, and their inputs cover the event inputs are selected.
	Step 3:	A flow of tasks is constructed for each of the selected tasks from Step 2, according to their preconditions and effects.
	Step 4:	A BEF file is generated for each task selected in Step 2 according to its flow constructed in Step 3.

to *AS.ConcreteServiceOperations*. In Step 4, *I.Input* parameters are added to *AS.Input* parameter. *I.Output* parameters are also added to *AS.Output* parameters.

$$AS.Input = AS.Input \cup I.Input,$$

$$AS.Output = AS.Output \cup I.Output.$$

In Step 5, *AS.MinSLA* is modified (if needed); first, *I.QoS* is normalized; then, the normalized value is compared with normalized *AS.MinSLA*. A conventional normalization function is the following one:

$$Normalized\ value = \sum_{(1 \le i \le m)} Q_i * W_i.$$

Q_i indicates the value of the ith quality parameter in the SLA, W_i corresponds to the weight of this parameter, and m is the number of these parameters.

4.3. Event control agent

The main purpose of this agent is to detect events and find appropriate tasks to handle the detected events.

Event Control Agent contains three components: Event Detector, Preprocessor, and Task Specifier. Event Detector detects events occurring through SDK described later. In addition, Steps 1, 2, and 3 in Table 7 are performed by this component; the Preprocessor component is in charge of Step 4; finally, Steps 5 and 6 are done by Task Specifier which constructs task flows for the detected events. At the end, Event Control Agent prepares one or more than one task-level BEF and sends them to Service Selection Agent to find the most suitable service composition. Table 8 shows the main operations of Task Specifier.

In Step 2 of Table 8, we select those tasks that are consistent with the input event. Suppose (I, O, G, C, L) is an input for Task Specifier in which I, O, G,

C, L are event input, event output, event goal, event category, and ontology base, respectively. Task T can be selected in Step 2 under the following condition.

$$SimilarityOf(T.Goal, G)$$
$$= 1.0 \wedge SimilarityOf(T.Category, C)$$
$$= 1.0 \wedge InputSetOf(T) supports I,$$

$$P_1 \ Supports \ P_2 \Leftrightarrow \forall p_2 : P_2 . \exists p_1 : P_1 . (TypeOf(p_1)$$
$$= TypeOf(p_2)) \wedge (SimilarityOf(NameOf(p_1),$$
$$NameOf(p_2)) = 1.0).$$

For two sets P_1 and P_2, P_1 Supports P_2 if for each parameter in P_2 there is at least one parameter in P_1 with similar name and type.

Step 3 must construct the flow structure for the task found in Step 2 considering both task precondition and task effect sections. These two sections are investigated in two opposite directions and according to the needed invocations in them, the flow structure is constructed; this bidirectional investigation is shown in Figure 7.

As shown in Figure 7, there is a set of tasks that should be connected to the found task so as to construct the task flow. The following relation is needed to be met so that two tasks t_1 and t_2 get connected to each other:

$$(t_1 \rightarrow t_2)$$
$$\Leftrightarrow (t_1 \ isinvokeddirectlyinPreconditionOf(t_2)$$
$$\vee \exists i : OutputSetOf(t_1).$$
$$i \ iscalledinPreconditionOf(t_2))$$
$$\vee (t_2 \ isinvokeddirectlyinEffectOf(t_1)$$
$$\vee \exists i : InputSetOf(t_2).$$
$$i \ iscalledinEffectOf(t_1)).$$

At first, this relation must be checked for the initially found task t. When some tasks are determined to be connected to t, the above relation will be applied to these tasks as well; this process will keep on iteratively until every remaining task has no tasks to be connected to. Now, the issue is how to generate the workflow plan. There are some main flow structures in a workflow plan, such as Sequence, Parallel, Join, Switch Case, and Loop. Also it is possible to invoke another task or assign a value to a variable. The proper workflow structure is generated according to the connected tasks. A parallel structure is made when an OR operator exists in the effect section of the considered task. A Loop structure is required when the precondition of task t invokes another task whose effect section invokes t, too. Switch Case structure will be required when one or more comparisons are needed in the precondition and/or effect of a task. If more than one task invoke a particular task in their effect section, Join structure is considered. At last, based on the selected flow structures, the workflow plan will be generated. Since it is possible that more than one plan are generated for each event, after plans are ready, we have proposed an algorithm which finds the most suitable service composition and will make it executable as a BPEL file. Due to the space limitation, we have not discussed the algorithm in this paper and the algorithm can be found in [36].

4.4. Service Selection Agent

This agent constructs an abstract service composition for each input task-level BEF received from the Event Control Agent and then selects the most suitable composition among them. Finally, the service-level BEF of the selected composite service is converted into BPEL (Table 9).

Service Selection Agent consists of two compo-

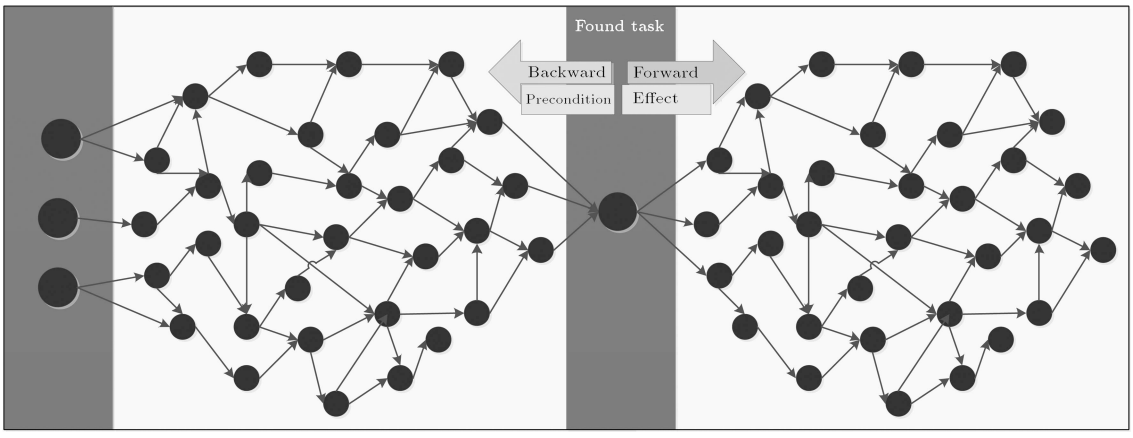

Figure 7. Bidirectional investigation.

Table 9. Service Selection Agent operations.

Service Selection Agent		
Input: A set of task level BEFs, ontology base, pattern base, and service base **Output:** A BPEL file		
As BEF files are received from the event control agent	**Step 1:**	An abstract service composition is found for each BEF.
	Step 2:	The most suitable composite abstract service is selected.
	Step 3:	The service level BEF of the selected composite service is converted into BPEL.

nents: Entry Agent and Scheduler. Entry Agent is in charge of Steps 2 and 3; scheduler performs Step 1. Figure 8 shows a hierarchical view for the operations of Service Selection Agent. Entry Agent disperses task-level BEFs among Scheduler instances to find an abstract service composition for each BEF. After that, the most suitable abstract service composition will be selected according to the total weight of compositions. Selecting the most suitable service composition is done by the service-level scheduling algorithm whose detailed description can be found in [36].

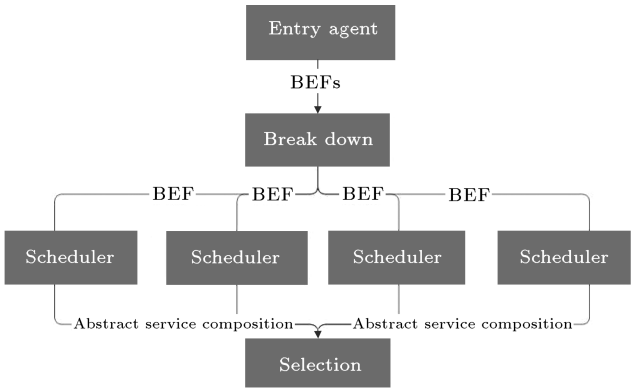

Figure 8. Hierarchical view for operations of Service Selection Agent.

5. SCW+ implementation

SCW+ has been implemented and deployed in two sides, server and client sides that are discussed in the following subsections, separately.

5.1. Server-side deployment

SCW+ has been implemented as a back-end server. A portal application has been developed for permitted users to administer and configure SCW+; see Figure 9. This application provides permitted users with some facilities to connect to SCW+ and define events and tasks

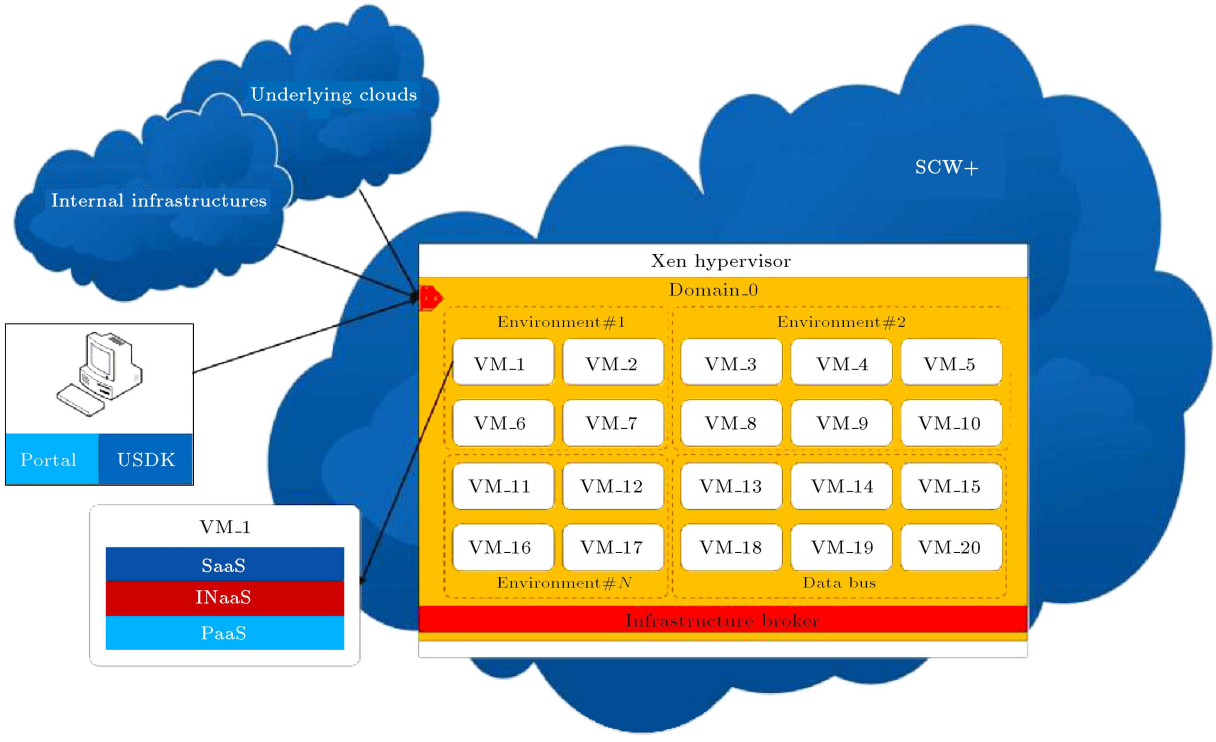

Figure 9. Server-side deployment view.

along with required relationships. Every VM in the server, which is allocated to an environment, deploys an instance of SCW+ core which itself contains PaaS, INaaS, and SaaS. Every environment can include one or more than one VM, and more than one environment can have common VMs.

5.2. User-side deployment

SCW+ has three types of roles: system administrators, service developers, and consumers. System administrators work with the portal application and are able to use every capability which is provided by the portal. Service developer is an independent role who develops web services and deploys them on any type of infrastructures (an internal infrastructure or other clouds). Consumer is an application that invokes events or can be informed about the occurrence of events. Consumers utilize an SDK (Software Development Kit) which has been provided for the proposed framework; see Figure 10.

This SDK provides an event-driven programming model for users to develop software applications such as ERP and CRM. The proposed model is a two-tiered model which allows the developer to concentrate only on his/her user interface tier without any concerns about data and control tiers. An under-developing software just needs to use this SDK to subscribe available events for firing them and retrieving their results. For the first step, the software must be authenticated for the cloud; once the authentication is completed, and the user is known as a permitted one, SDK acts as a middleware to retrieve a list of accessible events for this user and sets them forth. These events are then accessible to be used same as the other events in. Net framework. Then, the user subscribes an event if he/she wants to get its responses; the user can also call the event to fire it.

6. Evaluation

To evaluate SCW+, we first investigate how the targeted goals and sub-goals for SCW+ are obtained in SCW+. Then, we evaluate the most important components of the framework using the e-tour case; this evaluation shows that not only SCW+ is applicable, but also it provides acceptable performance.

6.1. Goals review

As stated before, SCW+ attempts to meet three main goals: supporting heterogeneous multi-cloud, minimizing human involvement, and omitting the need for skilled users. To meet these goals, some sub-goals are required to be met which are listed and described as follows:

- **Heterogeneous multi-cloud support:** This goal is provided through Service Monitor, matchmaker, and infrastructure broker. Aside from the type of the environment, every existing service from every cloud is extracted by Service Monitor; extracted services are classified in abstract services by matchmaker. This abstraction provides SCW+ with the possibility of mapping workflow plans to abstract service compositions. Hence, this abstraction is the key point of supporting heterogeneous multi-cloud services;
- **Minimum human involvement:** Omitting or minimizing human involvement requires some important factors, such as automatizing the process of workflow planning through web service composition, which are met by the following sub-goals:
  - **Automatic task workflow design:** According to the system design and what happens in the INaaS layer, as an event occurs, one or more appropriate tasks will be found and according to their preconditions and effects, some workflow

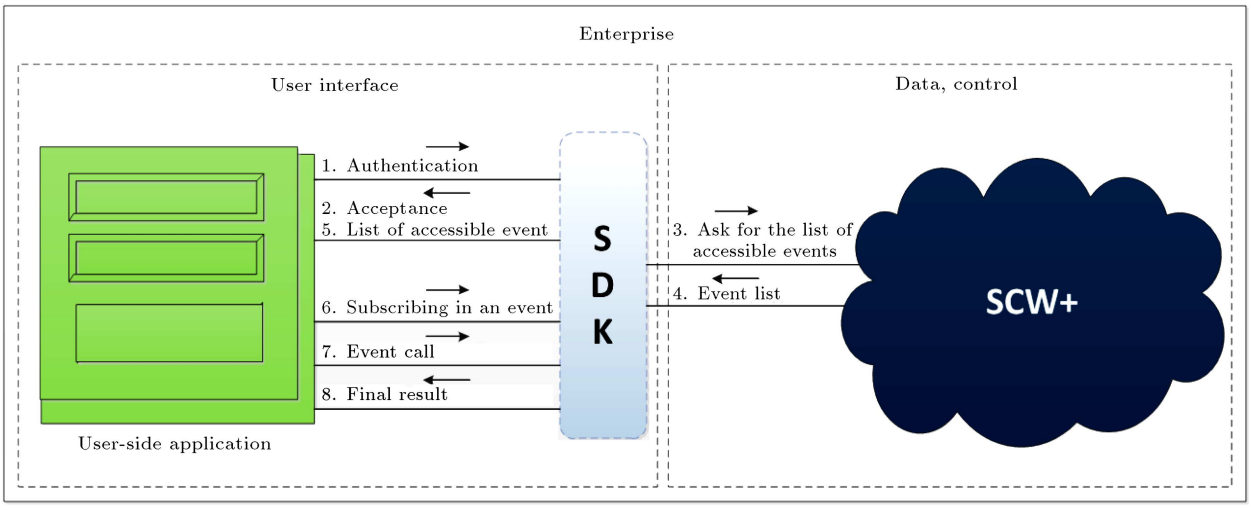

Figure 10. User-side deployment view.

plans will be defined; all of these stages are done automatically by Event Detector and Task Specifier components;

– **Automatic service composition:** Our service-level scheduling maps workflow plans to service compositions automatically.

- **No need for skilled user:** Working with SCW+ in both server and client sides is not a cumbersome task. Some other sub-goals that realize this goal are mentioned below:
  – **Supporting enterprise policies:** Enterprise policies are defined as preprocesses of events and preconditions and are automatically handled by Event Control Agent;
  – **No mere dependency on a particular service description:** The system does not rely on a particular web service description language and Service Monitor translates every description to ETAS to provide a unified language. This unification helps the system for further automaticity;
  – **Ease of work from user-side:** SDK enables users to connect to SCW+ easily, invoke events, and receive results.

6.2. SCW+ components evaluation

To evaluate applicability and performance of SCW+, we applied the e-touring case to the essential components of this framework. To do our experiments, we produced different sample web services with various functionalities (banking, insurance, hotel, etc.) and with different dependency lengths (based on preconditions and effects) from 0 to 4 (each web service could be sequentially dependent on 0 to 4 other services). Each of the web services had two quality parameters: time and cost. Time values were between 1 and 7 and cost values were between 10 and 35. We did random service generation tests for 10 times with different numbers of services.

As the first criterion, we evaluated matchmaker performance. As the second evaluation criterion, we tried to compare our service composition approach, which is the main part of Service Selection Agent, with an ant-colony-based approach to show its performance and its result optimality. Finally, we evaluated the crucial part of SCW+ (i.e., the INaaS layer) which includes event detection, task specification, workflow (BEF) generation, and finally finding the best service composition to be executed. Table 10 shows the configuration of 10 different stages of the experiment. In this random experiment, in addition to web services, tasks and events were also generated automatically with different characteristics in different stages. Every task and event had from 1 to 6 input parameters and from 1 to 4 output parameters. Similar to web services, preconditions and effects of tasks were defined automatically and randomly in such a way that made dependency length from 1 to 4 tasks.

Table 10. Automatically generated events, tasks, and web services.

	Events #	Tasks #	Concrete services #
Exp_1	8	18	50
Exp_2	16	36	100
Exp_3	24	54	150
Exp_4	32	72	200
Exp_5	40	90	250
Exp_6	48	108	300
Exp_7	56	126	350
Exp_8	64	144	400
Exp_9	72	162	450
Exp_10	80	180	500

Table 11. The number of abstract services in different experiments.

	Abstract service #
Exp_1	6
Exp_2	36
Exp_3	40
Exp_4	74
Exp_5	79
Exp_6	112
Exp_7	113
Exp_8	148
Exp_9	146
Exp_10	185

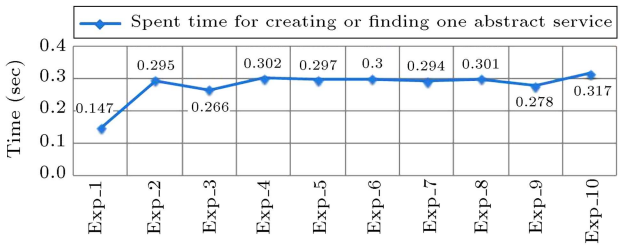

Figure 11. Spent time for creating or finding one abstract service (in sec).

6.2.1. Evaluating the matchmaker component

We executed the matchmaker component for 10 times, and it generated a different number of abstract services in each of the 10 stages (Table 11).

Figure 11 depicts how much time is needed to find or create one abstract service in different experiments. Figure 12 shows the total time to find or create all web services.

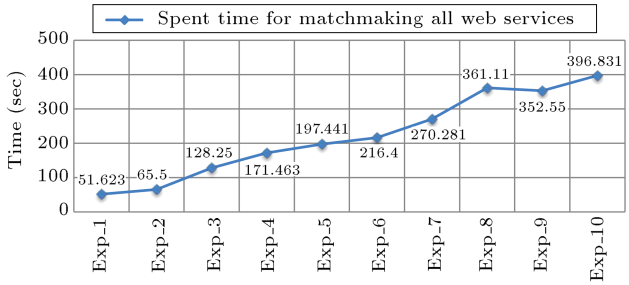

Figure 12. Spent time for matchmaking all web services (in sec).

Figures 11 and 12 show that creating and finding abstract services are done in a reasonable time manner. For example, in experiment #10, with 500 web services, the matchmaking time is 396.831 seconds, and this time will be 0.793 for each web service; therefore, matchmaking time consumption is also small. Since Monitoring and Matchmaking components are proposed by SCW+, and there is not any equivalent component for them in other platforms, we cannot provide a comparison of these two components with other works.

6.2.2. Evaluating the service composition algorithm

ACO (Ant Colony Optimization) is one of the most common approaches to target multi-objective optimization [37]; hence, in order to evaluate our approach to service composition, we compared this approach with an ant-colony-based approach described in [37]; formulation of the main variables was in the way mentioned in [37].

Figure 13 represents the spent time for the execution of two under-comparison service composition approaches. As Figure 13 shows, our approach has much better time consumption; this is because it finds the composition on the first iteration and does not need any more iteration. More details are available in [36].

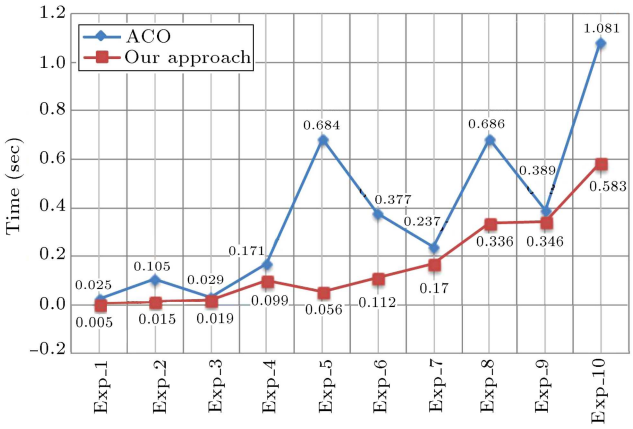

Figure 13. Spent time for service composition (in sec) for ACO approach and ours, when both are working based on abstract services.

6.2.3. Performance evaluation of INaaS

This section evaluates the performance of INaaS, which includes event detection, task specification, workflow (BEF) generation, and finally finding the best service composition to be executed. The service composition part was evaluated in the prior section; instead, Figure 14 covers both Event Control and Service Selection (i.e., from event detection to the final service composition) Agents and presents the average spent time according to ten experiment runs. As this figure shows, the range of spent time from 50 concrete services to 500 concrete services is approximately between 2 and 5 seconds (in addition, this time will be cut back to some extent using the pattern base, because the pattern base refuses repetitive operations); it enables SCW+ to make fast decisions, for example, in environments, such as mobile computing, that decision-making speed is critical.

Like Figures 11 and 12, these results are not compared with any other existing method, because INaaS, as the heart of SCW+, has no equivalent in other works so as to compare its performance with.

7. Discussion

Now, we intend to present a brief comparison between some cloud computing frameworks which can support both workflow management and user-side programming with ours based on some qualitative characteristics acquired from [38,39]. Most of these characteristics were described in Section 2 and the others were described as follows:

- **Workflow support:** This characteristic specifies whether the cloud framework supports business process design and execution or not. The platform must support a workflow designing language (such as BPMN) and execution language (such as BPEL);
- **Policy registry:** Policy registry stores business policies and helps the cloud framework to compose

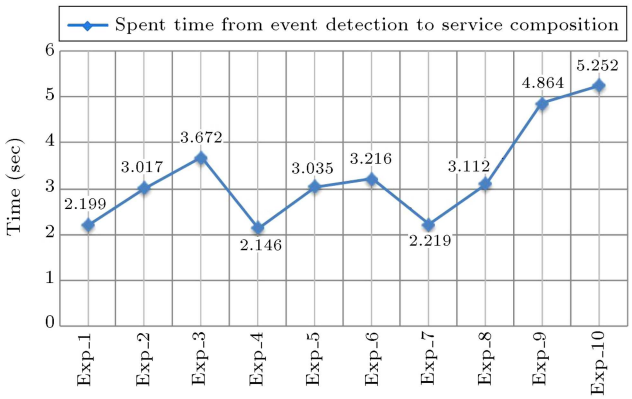

Figure 14. Average time for INaaS main process execution (in sec).

those services that are under the defined policies. This parameter specifies if a cloud framework has a policy registry;

- **Integration with on-premise software development tools:** This characteristic refers to the issue concerning how much the cloud framework supports working with different software development tools, such as analysis tools, UML design tools, IDEs, and so forth. Basically, this characteristic specifies the variety of tools.

Table 12 shows that SCW+ has many positive features against other similar frameworks. Some comparison results are discussed as follows:

- **Automatic service composition:** SCW+ composes services, automatically, by using ETAS and our service composition approach. SOCCA has a SOA layer on the top which is in charge of service-oriented operations, such as service composition, that are done automatically in this framework;
- **Multi-cloud service composition:** As stated before, SCW+ provides this facility by using environments, Service Monitor, and infrastructure broker. Moreover, SCW+ uses ETAS to unify different services from heterogeneous providers. SOCCA uses ontology to unify underlying resources and then treats these resources similarly;
- **Policy registry:** SCW+ and WSO_2 have a policy registry involved in service selection. SCW+ defines these policies as events preprocesses and preconditions which are handled automatically by Event Control Agent;

Table 12. Framework components.

	SCW+	**SOCCA**	**Aneka**	**Azure**	**WSO_2 Stratos**
Service type	**Iaas, Paas**	**Paas**	**Paas**	**Paas**	**Paas**
Deployment Model	Public, private	Public, private	Public	Public	Public, private
Workflow support	Yes	Yes	Yes	Yes	Yes
Automatic service composition	Yes	Yes	No	No	No
Multi-cloud service composition (Multi-infrastructure support)	Yes	Yes	No	No	No
Ontology-Based service provision	Yes	No	No	No	No
Policy registry	Yes	No	No	No	Yes
Automatic task template generation	Yes	No	No	No	No
Integration with on-premise software development tools	Medium	-	Low	Medium	Low
Programming model	Event-driven/ web service	Web service	Task/ thread model	Web service	Message passing/ web service
Programming language support (user-side)	.Net framework (till now)	-	.Net Framework	.Net framework	Java
Multi-tenancy	Yes (mediocre)	Yes (poor)	No	Yes (poor)	Yes (poor)
Complex event processing	Yes	No	No	No	Yes
MapReduce	No	No	Yes	No	Yes (poor)

- **Automatic task template generation:** SCW+ generates workflow templates according to task preconditions and effects, automatically; but other frameworks require manually designed workflows;
- **Integration with on-premise software development tools:** Since no document exists about SOCCA implementation, we cannot say anything about this framework. Aneka has a particular web service type, so integration with other software development tools in Aneka is cumbersome. WSO_2 has its special development tools as well. Azure is fully consistent with Visual Studio development environment, but it is limited to. Net tools. SCW+ development is based on both event management and web services; but, at the moment, SCW+ is based on. Net; therefore, it supports Azure development tools at least;
- **Multi-tenancy:** As mentioned in [40], there are two types of multi-tenancy patterns in the application level: Multiple application instance and single application instance. SOCCA tries to combine these two patterns to support multi-tenancy in a balanced way. Azure and WSO_2 also provide an ability by which multiple users can utilize common resources and run multiple application instances; then, they support the multiple-application instance pattern. In SCW+, more than one environment can contain common VMs, so it supports multiple application instance patterns. SCW+ also provides a new type of multi-tenancy because it considers events, tasks, and abstract services as resources, and different environments can share these resources among one another;
- **Complex event processing:** In essence, complex event processing is implemented through event process networks [31] defined by event-level ETAS descriptions and provided through Event Control Agent in SCW+;
- **Map-Reduce:** Aneka and WSO2 (limitedly) support this type of programming.

8. Conclusion

In this paper, we proposed a service-oriented framework, named SCW+, for cloud workflow systems. The main components of SCW+ were evaluated by regarding a familiar case study. We discussed, deeply, the main components of the system and the underlying ontological languages for describing the environment for the system to work in. After all of these architectural and design concepts, we talked about the deployment view of the proposed framework and evaluated it with some different range of inputs. The evaluation implies that SCW+ is applicable and provides acceptable performance. In future, we are going to support Map-Reduce and extend SCW+ to support mobile cloud computing. Moreover, we are planning to implement infrastructure broker to support the integration of SCW+ with Amazon EC2, Azure, etc.

References

1. Van der Aalst, W.M.P. and van Hee, K.M., *Workflow Management: Models, Methods, and Systems*, MIT Press, Cambridge, MA, USA (2002).
2. Yu, J. and Buyya, R. "A taxonomy of workflow management systems for grid computing", *Journal of Grid Computing* (2006). Doi: 10.1007/s10723-005-9010-8
3. Beisiegel, M., Blohm, H. and Booz, D. "Service component architecture: Building systems using a service oriented architecture, SAP", http://www.sybase.com/sb_content/1038547/SCA_White_Paper1_09.pdf, Accessed February 2013 (2005).
4. Hawke, S., Herman, I. and Prud'hommeaux, E. "Semantic web, providing a common framework that allows data to be shared and reused across application, enterprise, and community boundaries", http://www.w3.org/2001/sw/, Accessed February 2013 (2001).
5. Berners-Lee, T. and Hendler, J. "The semantic web. A new form of web content that is meaningful to computers will unleash a revolution of new possibilities", *Scientific American Magazine*, **299**(4), pp. 34-43 (2001).
6. Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M., Sheth, A. and Verma, K. "WSDL-S, web service semantic", http://www.w3.org/Submission/WSDL-S/, Accessed February 2013 (2005).
7. Verma, K. and Sheth, A. "Semantically annotating a web service", *IEEE Internet Computing*, **11**(2), pp. 83-85 (2007). Doi: 10.1109/MIC.2007.48
8. Martin, D., et al "OWL-S, semantic markup for web services", http://www.w3.org/Submission/OWL-S/, Accessed February 2013 (2005).
9. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres Feier, C., Bussler, C. and Fensel, D. "Web service modeling ontology", *Applied Ontology*, **1**(1), pp. 77-106 (2005).
10. Battle, S., Bernstein, A. and Boley, H. "SWSO, semantic web services ontology", http://www.w3.org/Submission/SWSF-SWSO/, Accessed February 2013 (2005).
11. OMG "Business Process Model and Notation (BPMN), version 2.0", Object Management Group, Technical report (2011).
12. Peltz, C. "Web services orchestration and choreography", *Computer*, **36**(10), pp. 46-52 (2003). Doi: 10.1109/MC.2003.1236471

13. Aalst, W.V.D., Dumas, M. and Hofstede, A.T "Web service composition languages: old wine in new bottles?", *Proceeding of the 29th EUROMICRO Conference: New Waves in System Architecture, IEEE Computer Society*, Los Alamitos, pp. 298-305 (2003). Doi: 10.1109/EURMIC.2003.1231605

14. OASIS (WSBPEL) "Web services business process execution language version 2.0", Technical Report, Organization for the Advancement of Structured Information Standards (OASIS) (2007).

15. Thatte, S., *XLANG Web Services for Business Process Design*, Microsoft Corporation (2001).

16. Aalst, W. and Hofstede, A. "YAWL: Yet another Workflow Language", *Information Systems*, **30**(4), pp. 245-275 (2005).

17. Tsalgatidou, A., Athanasopoulos, G., Pantazoglou, M., Pautasso, C., Heinis, T., Grønmo, R., Hoff, H., Berre, A.-J., Glittum, M. and Topouzidou, S. "Developing scientific workflows from heterogeneous services", *SIGMOD Record*, **35**(2), pp. 22-28 (2006).

18. Yelmo, J.C., Trapero, R., del Álamo, J. "User-driven service lifecycle management -adopting internet paradigms in telecom services", *Fifth International Conference*, Vienna, Austria, pp. 342-352 (2007). Doi: 10.1007/978-3-540-74974-5_28

19. Lécué, F., Delteil, A. and Léger, A. "Towards a semantic state transition system for automated generation of data flow in web service composition", *Int. J. Semantic Computing*, **3**(4), pp. 499-526 (2009). Doi: 10.1142/S1793351X09000896

20. Daniel, F., Casati, F., Benatallah, B. and Shan, M.-C. *Hosted Universal Composition: Models, Languages and Infrastructure*, in mashArt, in Alberto, Springer, pp. 428-443 (2009). Doi: 10.1007/978-3-642-04840-1_32

21. Pietschmann, S. "A model-driven development process and runtime platform for adaptive composite web applications", *International Journal on Advances in Internet Technology*, **2**(4), pp. 277-288 (2009).

22. Daniel, F., Soi, S., Tranquillini, S., Casati, F., Heng, C. and Yan, L. "From people to services to UI: Distributed orchestration of user interfaces, in *Richard Hull Business Process Management*, Springer Berlin Heidelberg, pp. 310-326 (2010). Doi: 10.1007/978-3-642-15618-2_22

23. Yang, Y., Liu, K., Chen, J., Liu, X., Yuan, D. and Jin, H. "An algorithm in SwinDeW-C for scheduling transaction-intensive cost-constrained cloud workflows", *4th IEEE International Conference on E-Science (e-Science08)*, pp. 374-375 (2008). Doi: 10.1109/eScience.2008.93

24. Hristoskova, A., Volckaert, B. and Turck, F.D. "The WTE+ framework: automated construction and runtime adaptation of service mashups", *Automated Software Engineering* (2011). Doi: 10.1007/s10515-012-0105-8

25. Li, G. and Wei, M. "Everything-as-a-service platform for on-demand virtual enterprises", *Journal of Inf. Syst. Front* (2012). Doi: 10.1007/s10796-012-9351-3

26. Vecchiola, C., Chu, X. and Buyya, R. "Aneka: A software platform for .Net-based cloud computing", *High Speed and Large Scale Scientific Computing*, IOS Press (2009).

27. Sukumar, K., Vecchiola, C. and Buyya, R. "The Structure of the new it frontier: aneka platform for elastic cloud computing applications", *Strategic Facilities Magazine*, **25**(6), pp. 599-616 (2010).

28. Chu, X., Nadiminti, K., Jin, C., Venugopal, S. and Buyya, R. "Aneka: next-generation enterprise grid platform for e-science and e-business applications", *3rd IEEE International Conference on e-Science and Grid Computing*, pp. 10-13 (2007). Doi: 10.1109/E-SCIENCE.2007.12

29. Tsai, W.T., Sun, X. and Balasooriya, J. "Service-oriented cloud computing architecture", *Seventh International Conference on Information Technology: New Generations (ITNG)*, pp. 684-689 (2010). Doi:10.1109/ITNG.2010.214

30. Huang, Y., Kumaran, S. and Chung, J.Y. "A service management framework for service-oriented enterprises", *International Conference on E-Commerce Technology*, pp. 181-186 (2004). Doi: 10.1109/ICECT.2004.1319732

31. Etzion, O. and Niblett, P., *Event Processing in Action*, Manning, Stanford, US (2011).

32. Maréchaux, J. "Combining service-oriented architecture and event-driven architecture using an enterprise service bus", *IBM Journal*, **2**, pp. 66-69 (2006).

33. Appendix, Available at: http://ticksoft.sbu.ac.ir/?page id=5116 (2015).

34. Torkashvan, M. and Haghighi, H. "CSLAM: A framework for cloud service level agreement management based on WSLA", *Sixth International Symposium on Telecommunications (IST)*, pp. 577-585 (2012). Doi: 10.1109/ISTEL.2012.6483055

35. *Wordnet, Lexical Data Base*, Princeton University, http://wordnet.princeton.edu/, Accessed May 2013 (2013).

36. Torkashvan, M. and Haghighi, H. "A greedy approach for mapping workflows to service compositions in cloud workflows", *Sixth International Symposium on Telecommunications (IST)*, pp. 929-935 (2012). Doi: 10.1109/ISTEL.2012.6483119

37. Zhang, W., Chang, C. and Feng, T. "QoS-based dynamic web service composition with ant colony optimization", *IEEE 34th Annual Computer Software and Applications Conference*, pp. 493-502 (2010). Doi: 10.1109/COMPSAC.2010.76

38. Peng, J., Zhang, X., Lei, Z., Zhang, B., Zhang, W. and Li, Q. "Comparison of several cloud computing platforms", *Second International Symposium on Information Science and Engineering*, pp. 23-27 (2009). Doi: 10.1109/ISISE.2009.94

39. Rimal, B.P., Choi, E., Lumb, I. "A taxonomy and survey of cloud computing systems", *Fifth International Joint Conference on INC, IMS and IDC, IEEE Computer Society*, Washington, DC, USA, pp. 44-51 (2009). Doi: 10.1109/NCM.2009.218

40. Huang, Y., Su, H., Zhang, J.M., Guo, C.J., Xu, J.M., Jiang, Z.B., Yang, S.X. and Zhu, J. "Framework for building a low-cost, scalable, and secured platform for web-delivered business services", *IBM Journal of Research and Development*, **54**(6), pp. 1-14 (2010). Doi:10.1147/JRD.2010.2065891

Biographies

Milad Torkashvan is graduate of Information Technology from the University of Shahid Beheshti, Tehran, Iran. He also took his bachelor degree from University of Bahonar, Kerman. He has written some papers about service-level agreement in cloud computing, service composition, and mobile cloud computing. His main research interests are cloud computing and System security.

Hassan Haghighi is an Assistant Professor in Computer Science and Engineering Department, Shahid Beheshti University, Tehran, Iran. He received his PhD degree in Computer Engineering-Software from Sharif University of Technology, Iran, in 2009. His main research interests are formal methods, software testing, and service-oriented architecture.