Enterprise Application Integration Based on Interactive Agent in Cloud Computing Environment

Djamel Benmerzoug
Department of Software Technologies and Information Systems
Faculty of New Technologies for Information and Communication
University Constantine 2, Algeria

Enterprise application integration (EAI) through integrated Cloud services is necessary to achieve agility in the current age of competition. The main challenge raised by EAI is in the number of autonomic entities involved and the complexity of the interactions within them. That is, the complexity that matters is not so much in the size of the code through which such entities are programmed but on the number, intricacy and dynamicity of the interactions in which they will be involved. This is why it is so important to put the notion of interaction at the center of research in EAI.

Multiagent systems (MAS) provide a promising paradigm for EAI development. In this paper, we propose an agent-based approach for EAI in Cloud environment. The approach introduces an Interactive agents-based architecture whose main goal is to address and tackle interoperability challenges at the Cloud application level. It enables the deployment of business applications at public, private or hybrid multi-Cloud environments.

Keywords: Agent Interaction Protocols, Enterprise application integration, Cloud Computing, Protocols Composition

1. INTRODUCTION

To thrive in the current competitive environment, businesses not only need to integrate their internal stovepipe applications, they also need to integrate their application systems with their supply chain partners' systems [Kishore et al. 2006]. Both the practitioner publications and academic literature have noted the significant benefits that information systems integration both within and across the enterprise can bring about for businesses in terms of improved planning, timely deliveries, reduced inventories, reduced costs, improved product line in tune with market needs, and responsive and improved customer service [Barjis et al. 2011][A. Artikis and Dignum 2010].

To address these issues, Cloud computing has been seen as a promising opportunity to improve EAI systems. Cloud computing is a rapidly growing IT paradigm, which transforms the Internet into a global market of on-demand resources [Zeginis et al. 2013]. Cloud services composition is the ability to integrate multiple services into higher-level applications. This integration necessitates a uniform description format that facilitates the design, customization, and composition. In this context, Maultiagent Systems (MAS) are a useful way for structuring communicative interaction among business partners, by organizing messages into relevant contexts and providing a common guide to the all parts.

It was noted in [Benmerzoug et al. 2007] that MAS are appropriate approaches to define and manage collaborative processes in B2B relationships where the autonomy of participants is preserved. Whereas in [Benmerzoug et al. 2008a][Benmerzoug et al. 2008b], we demonstrated the practicability of our approach by embedding it in a Web services language for specifying Agent Interaction Protocols (AiP), which conducive to reuse, refinement and aggregation of modular protocols. We also elaborated translation rules from interaction protocols notations used in our approach into Colored Petri Nets (CPN). These rules are implemented in AiP2CPN: the tool

we developed to automatically generate Petri nets from protocols specifications. Resulting Petri nets can be analyzed by dedicated tools to detect errors as early as possible.

To address the collaboration and interaction issues in modern enterprises, it is normal to turn our attention to Cloud Computing as it aims to provide both the economies of scale of a shared infrastructure and a flexible delivery model. In [Benmerzoug et al. 2013], we presented the idea of Cloud Business Protocol, which is a useful way for structuring interaction among Cloud consumers and service providers. In [Benmerzoug 2013b], we proposed a set of operators that allows the creation of new value-added protocols using existing ones as building blocks. This research is among the earliest efforts, to the best of the authors' knowledge, in adopting an AiP-based approach for supporting Cloud services composition.

In this present research, we describe the AiP4CSC: an agent-based approach that supports flexible scaling of enterprise application in a virtualized Cloud computing environment The approach introduces a Broker-based architecture whose main goal is to address and tackle interoperability challenges at the Cloud application level. It solves the interoperability issues between heterogeneous Cloud services environments by offering a harmonized API. Also, it enables the deployment of business applications at public, private or hybrid multi-Cloud environments.

The remainder of the paper is organized as follows: Section 2 introduces some key concepts and terminology. In Section 3, we present an agent-based architecture in a nutshell introducing the main modules and the core functionality. Section 4 discusses some aspects of the implementation. In Section 5, we present the experimental results on the proposed architecture. Section 6 overviews some related work. Finally, Section 7 concludes this paper and presents future directions.

2. THE PROPOSED APPROACH : CONCEPTS AND TERMINOLOGY

With the emergence of Cloud computing, applications are moving away from PC based or ownership-based programs to Web delivered hosted services [Tobaly 2010]. The software services are provisioned on a pay-as-you-go basis to overcome the limitation of the traditional software sales model.

2.1 Cloud Computing Models

According to the intended access methods and availability of Cloud computing environments, four major types of Cloud deployments are known: public Clouds, private Clouds, community Clouds, and hybrid Clouds [Mell and Grance 2009].

Private Cloud:. In this model, the Cloud infrastructure is exclusively used by a specific organization. The Cloud may be local or remote, and managed by the enterprise itself or by a third party.

Public Cloud:. Infrastructure is made available to the public at large and can be accessed by any user that knows the service location.

Community Cloud:. Several organizations may share the Cloud services. These services are supported by a specific community with similar interests such as mission, security requirements and policies, or considerations about flexibility.

Hybrid Cloud:. Involves the composition of two or more Clouds. These can be private, community or public Clouds which are linked by a proprietary technology that provides portability of data and applications among the composing Clouds.

According to the Forrester Research market [market:], many businesses want interoperability between their internal infrastructure combined with public Cloud resources. They might want to use private application to process data in the Cloud, they might want to use a Cloud-based application to process private data, or they might want to use applications or tools that will run both private and on the public Cloud. Consequently, we believe that a hybrid approach makes more sense for enterprises. In such approach, there is a need for complex developed business

applications on the Clouds to be interoperable. Cloud adoption will be hampered if there is not a good way of integrating data and applications across Clouds.

In the next section, we introduce the Cloud Business Protocols, which are a useful way for structuring interaction among Cloud resources, by organising activities into relevant contexts and providing a common guide to the all parts.

2.2 The Cloud Business Protocol

As stated in [Nguyen et al. 2012], to ensure a meaningful composition of two or more Cloud services, there is a clear need for placing emphasis on how to develop enhanced composite service offerings at the application-level and assign or reassign different virtual and physical resources dynamically and elastically. In fact, combining different independent Cloud services necessitates a uniform description format that facilitates the design, customization, and composition.

Business protocol is a specification of the allowed interactions between two or more participant business partners. Applied to Cloud computing, we propose the following variation: *A Cloud Business Protocol is two or more business parties linked by the provision of Cloud services and related information.*

In fact, business parties in the Cloud computing area are interconnected by the Cloud business protocol. These parties are involved in the end-to-end provision of products and services from the Cloud service provider for end Cloud customers. Because protocols address different business goals, they often need to be composed to be put to good use. For example, a process for purchasing goods may involve protocols for ordering, shipping, and paying for goods.

Driven by the motivation of reuse, we would like to treat protocols as modular components, potentially composed into additional protocols, and applied in a variety of business processes. By maintaining repositories of commonly used, generic, and modular protocols, we can facilitate the reuse of a variety of well-defined, well-understood, and validated protocols.

In the next section, we propose a basis for a theoretical approach for aggregating protocols to create a new desired business application.

2.3 AiP for Cloud Services

Because protocols address different business goals, they often need to be composed to be put to good use. For example, an enterprise that is interested in selling books could focus on this protocol while outsourcing other protocols such as payment and shipment.

The composition of two or more AiP generates a new protocol providing both the original individual behavioral logic and a new collaborative behavior for carrying out a new composite task. This means that existing protocols are able to cooperate although the cooperation was not designed in advance.

Definition 1. (Composite Protocol) A Composite Protocol (CP) is a tuple $CP = (P, Op, Input, Output, P_{init}, P_{fin})$ where:

- P is a non empty set of *basic protocols*,
- Op is a non empty set of *operators*, $Op \subseteq (P \times P) \cup (P \times CP) \cup (CP \times P) \cup (CP \times CP)$,
- $Input, Output$ are a set of the elements required (produced) by the composite protocol CP,
- P_{init} is non empty set of *initial* protocols, $P_{init} \in P$ and
- P_{fin} is non empty set of *final* protocols, $P_{fin} \in P$.

2.3.1 *Protocol Contract.* The proposed approach provides the underpinnings of aggregation abstractions for protocols. To achieve this goal, we require an agreement between AiP in the form of a shared contract. A contract describes the details of a protocol (participants in the protocol, produced elements, required elements, constraints,...) in a way that meets the mutual understandings and expectations of two or more protocols. Introducing the contract notion gives

us a mechanism that can be used to achieve a meaningful composition.

Definition 2. (Protocol Contract) A contract C, is a collection of elements that are common across two protocols. It represents the mutually agreed upon protocol schema elements that are expected and offered by the two protocols.

(1) Let us denote by P_k^{in}, P_k^{out} the set of elements required (produced) by the protocol P_k where $P_k^{in} = \{x_i, i \geq 1\}$ and $P_k^{out} = \{y_j, j \geq 1\}$
(2) Let us define a function θ, called a *contract-mapping*, that maps a set of P_k^{out} elements (produced by the protocol P_k) and a set of P_r^{in} (consumed by the protocol P_r), $\theta = \{\exists\ y_i \in P_k^{out} \wedge \exists\ x_j \in P_r^{in} \mid (x_i, y_j)\}$, which means that the protocol P_r consumes the element y_j provided by the protocol P_k, and $C = (P_k^{out})_\theta(C) \cup (P_r^{in})_\theta(C)$.

2.3.2 *Protocols Composability relationship.* The Composability relationship means that AiP can be joined together to create a new protocol that performs more sophisticated applications. That composition can then served as a protocol itself.

We can classify the nature of protocols composition on the dependance degree between them. We may thus distinguish between two kinds of Composability relationship:

Definition 3. (Partial Composability)
Two protocols P_k and P_r meet the Partial Composability relationship *iff* $\exists\, x_i \in (P_r^{in})_\theta(C), \exists\, y_j \in (P_k^{out})_\theta(C) \mid\ (x_i, y_j)$, which means that the protocol P_r can be executed after the protocol P_k but it must wait until all its "required elements" will be provided by others protocols.

Definition 4. (Full Composability)
Two protocols P_k and P_r are called Full Composability *iff* $\forall x_i \in (P_r^{in})_\theta(C), \exists\, y_j \in (P_k^{out})_\theta(C) \mid (x_i, y_j)$, which means that the protocol P_r must be (immediately) executed after the protocol P_k because **all** its "required elements" are offered by the protocol P_k.

We note here that the partial (full) Composability relationship is not commutative. So, if a protocol P_k has a partial (full) Composability relationship with a protocol P_r, it does not means that P_r has a partial (full) Composability relationship with P_k .

Proposition Let P $=\{P_1, P_2, \ldots, P_m\}$ a set of AiP. The set P constitutes a meaningful composition if:

$\forall P_k \in (P - P_{init}, \forall x_i \in (P_k^{in})_\theta(C), \exists\ y_j \in (P_r^{out})_\theta(C) \mid (x_i, y_j)$, where $r, k \in [1, m]$ and $r \neq k$.

$P_{init} \in P$ and represent the initial protocols, which their "required elements" are provided by external events.

This proposition states that, if we have a set of protocols P $=\{P_1, P_2, \ldots, P_m\}$ where all their "required elements" are offered (by other protocols), this means that all the protocols of P can be executed.

Modelling AiP composition requires control structures involving loops, choice and parallelism. This will enable complex behaviour of AiP, such as concurrent execution, or iteration while a certain condition holds. Consequently, in [Benmerzoug 2013b], we proposed a set of operators that allows the creation of new value-added AiP using existing ones as building blocks. The formal semantics of the composition operators is expressed in terms of Petri nets by providing a direct mapping from each operator to a Petri net construction. In addition, the use of a formal model allows the verification of properties and the detection of inconsistencies both within and between AiP.

In the next section we present an agent-based architecture that supports the work previously developed. The proposed architecture has been designed to enable interoperability and cross-Cloud application management.

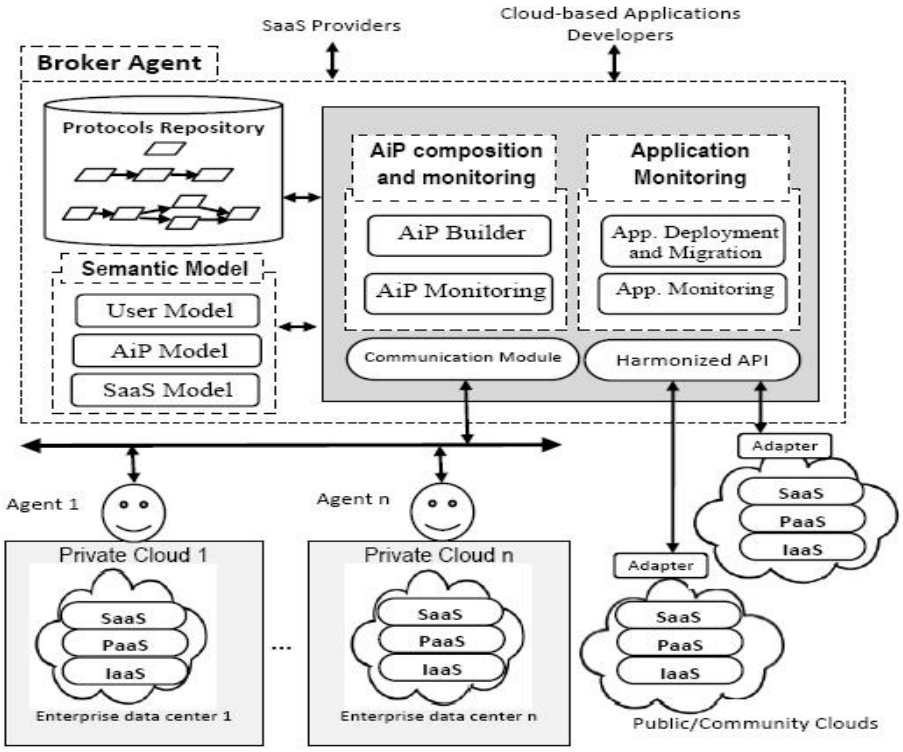

Figure 1: AiP4CSC: The Proposed Agent-Based Architecture

3. AIP4CSC: AN AGENT BASED ARCHITECTURE FOR CLOUD SERVICES COMPOSITION

Service composition in multi-Cloud environments must coordinate self-interested participants, automate service selection, (re)configure distributed services [Sim 2012], store and share the composite services to allow other software artifacts to (re)use them.

The new challenges that Cloud computing poses to service composition, emphasize the need for the agent paradigm [Benmerzoug 2013b][Wang et al. 2006][Sim 2012]. Multi-agent systems represent a distributed computing paradigm based on multiple interacting agents that are capable of intelligent behavior.

In order for enterprises collaborate to fulfill their requirements, it is important to consider that more than one type of Cloud can be used. However, enterprises are driven by different reasons to maintain their own data center, such as legislation of storing data in-house, investments in the current infrastructure, or the extra latency and performance requirements. This drive is supported by the fact that enterprises have already invested heavily in their own private server equipment and software [Hoecke et al. 2011].

Consequently, we defined two types of agent, namely, the Enterprise Agent representing an individual enterprise, and the Broker Agent, which facilitates the Cloud based application developers in searching for, deploying and governing their business applications on the SaaS offerings that best match their needs (see Fig. 1).

3.1 Description of the Broker Agent

The main roles of the Broker agent, which implements the core functionalities offered by the architecture, are the creation, monitoring, and control of AiP life cycle. Its architecture features the following modules:

—**Application Monitoring:** supports the efficient deployment and governance of applications.

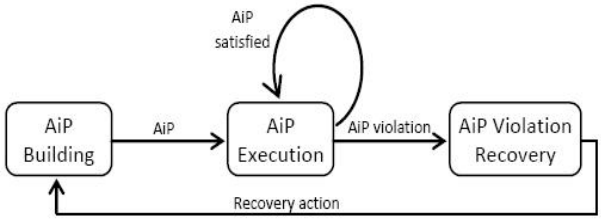

Figure 2: AiP Building, Execution, and Recovery Process

The developers can manage the life-cycle of their applications as a homogenized way independently of the specific platform offering the application is deployed.

—**AiP Composition and Monitoring:** orchestrate AiP and control the access to them. It receives requests to resolve requirements from applications developers. Then, it handles the requests via their associated AiP. It also provides operations for monitoring interaction (i.e., creating and deleting instances).

—**Communication Module:** contains all the processes required to handle agent to agent communication, such as: reception, filtering, and translation of incoming messages, and formulation and sending of the outgoing messages. Agent to agent communication occurs via FIPA Agent Communication Language [FIPA-ACL 2001], where XML will be used for the description of the content of the message.

—**Harmonized API:** provides the necessary tools, which enable the management of applications across different Cloud offerings.

—**Protocols Repository:** maintains repository of commonly used and generic protocols. It facilitates the reuse of a variety of well-defined, well-understood and validated protocols. It provides the AiP specification that describes the functionality, input and output of protocols. An AiP is described by the requirement it resolves, and the parameters of the requirement correspond to the input of the AiP. The AiP output is a set of parameters that results from resolving the requirement.

—**Semantic Model:** is the backbone of the architecture and spans the entire architecture, resolving Interaction conflicts by providing a common basis for publishing and searching different SaaS offerings.

3.2 Reusing Historical Composition Experiences

Using a number of various components such as services or types of Clouds can cause the AiP to be complex. Depending on enterprise requirements, one Cloud may not be able to offer the complete service they have requested. Current techniques suggest that enterprises will acquire related tools and perform integration activities locally. The knowledge and expertise to perform composition activities is hard to attain and equally difficult to maintain. Thus, the knowledge obtained during Cloud services composition is not stored, reused, or shared. To be competitive, enterprises must be able to transfer and reuse knowledge attained after each composition scenario.

Consequently, we have proposed the concept of *Agent Interaction Protocols as a Service* (AiPaaS), which aims at reducing the need for a composition infrastructure and allows to compose and deploy existing AiP.

An AiPaaS capability could acquire the previously mentioned protocols as input and suggest a specific protocol to achieve a new specific need. Enterprises can maintain a protocols repository to facilitate reusing previously used protocols and composition routines in the future. Furthermore, the AiPaaS approach can learn from historical composition information to augment future recommendations.

To deal with a AiP composition in an automatic way (i.e. to have mechanisms that automate the AiP building, monitoring and execution), the *AiP Composition and Monitoring* module (Fig.

1) offers three main functionalities that enable building and execution of AiP, as well as recover from an AiP violations (Fig. 2):

—**AiP Building.** Allows the automatic creation of AiP, based on previous AiP (from the protocols repository) and the semantic description of requirements specified by the application developer. In fact, the AiP descriptions contain semantic input and output parameters. Afterwards, a semantic matching algorithm links semantically similar outputs to corresponding inputs, obtaining a chain of basic AiP that results in the service composition.

—**AiP Execution.** Maintains a global monitor of the AiP execution. We note here that the Broker agent expects each EA member to communicate its communication state in regular intervals, or at least whenever the agent changes its state. This provides the Broker agent with accurate information on the state of the team. To ensure the successful completion of the interaction, the Broker agent must know how many responses should be expect from the EA. The analysis of AiP rules defined in the protocols repository and the semantics of the ACL allow when if others messages may be received or not.

—**AiP Violation Recovery.** When the AiP Composition and Monitoring module detects an error of the AiP execution (the execution of the business application does not satisfy the AiP), the protocol execution is stopped, and the error is replicated to the application developer.

4. IMPLEMENTATION OF THE AIP4CSC ARCHITECTURE

We developed a first prototype of the AiP4CSC architecture utilizing different advanced tools like: Java agent development framework (Jade) [C. Trappey and Ku 2009], WSBPEL[IBM et al. 2003], Windows Azure Cloud platform [Neil 2011], and Windows Communication Foundation [Pablo et al. 2010].

Agents of the proposed architecture are developed with JADE, which complies with the FIPA standards. JADE is completely written in Java and includes two main components: a FIPA compliant agent platform and a framework to develop Java agents.

Creating a JADE agent is as simple as defining a class extending the *core.Agent* class and implementing the *setup()* method. The *setup()* method is intended to include agent initializations. The actual job an agent has to do is presented as JADE behaviors.

WSBPEL represents the merger of two process description language, IBM's Web Services Flow Language (WSFL) and Microsoft's XLANG. It provides both graph-based and block-based control structures, making it capable of representing a wide range of control flows. In our case, we have used WSBPEL to describe the AiP by stating whom the participants are, what services they must implement in order to belong to the services composition. for example, the $< partners >$ section defines the different parts that participate in the AiP. Each partner is given a service link type and the role it will perform as part of the service link. The $< variables >$ section defines the variables used by the AiP. The protocol definition occurs after the fault handlers section and before the close process tag. The AiP is defined using WSBPEL activity constructs (sequence, flow, while, switch, etc).

Windows Azure Platform is a Cloud platform which provides a wide range of Internet Services. It is a Windows based Cloud services operating system providing users with on-demand compute service for running applications, and storage services for storing data in Microsoft data centers. In our case, we can deploy the Broker agent container as instances of Windows Azure Worker role which gets access to protocols repository on the Windows Azure environment via the Windows Azure Managed Library.

The AiP4CSC architecture uses Web service technology like Windows Communication Foundation (WCF) in order to implement basic services required for composition. The decision to use WCF has primarily been made because it already offers established and standardized ways of communication and allows easy integration into existing manufacturing infrastructures. In fact, all services are considered to be WCF-based services. WCF provides the platform for developing service-oriented solutions with .NET. One advantage of utilizing WCF is that it provides features

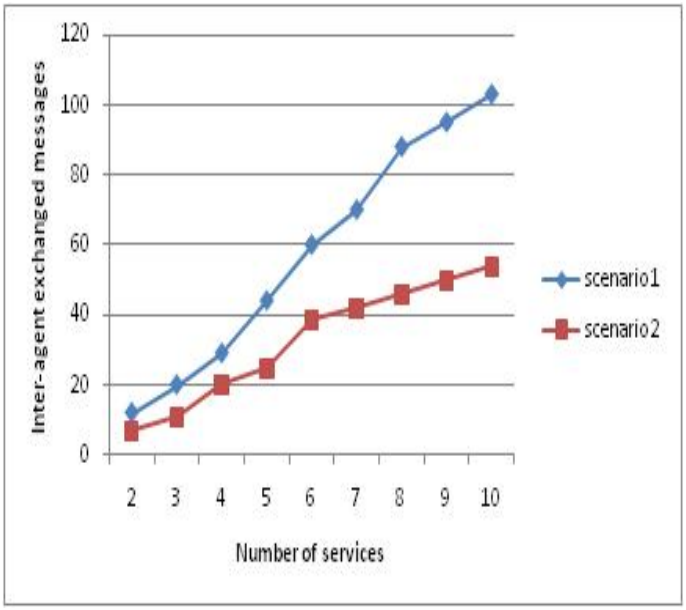

Figure 3: Number of Inter-agent Exchanged Messages

that directly support the application of service-orientation principles. The SOA-friendly qualities such as loose coupling, autonomy, statelessness, composability, discoverability, and reusability can be achieved through WCF.

5. EVALUATION OF THE AIP4CSC ARCHITECTURE

To evaluate the efficiency of the proposed architecture, we have identified two scenarios. In the first scenario, only the private Clouds are used, which means that all the basic services are managed by the *Enterprise Agents* (see Fig. 1). In the second scenario, enterprise provides and manages some internal resources and the others provided externally.

The Broker agent was assumed to submit a task that has n basic services required to complete it (where $n \in [2, 10]$). Following the first scenario, all these services are provides by private Clouds, whereas in the second scenario, 50% of these services are provides by public Clouds. We assume that each basic service is provided by one and only one Cloud.

In this first experiment (scenario 1), the following agent were involved: 10 EA and the Broker agent.

As shown in Fig. 3, the number of exchanged messages increased at a constant rate of the basic services involving in the composition. This shows that services composition was achieved with a linear messages exchanged complexity. In the case of scenario 1, the agents normally exchange approximately 100 messages during services composition (where basic services = 10). However, and since there no Agent-to-Agent communication between the AiPCSC architecture and other public Cloud, in scenario 2, agents will need to communicate only 50 messages.

In other hand, Fig. 4 shows that private Cloud services composition adopting EA as interfaces took substantially shorter time. In fact, the Broker agent using protocols repository distributes the AiP description over all agents, achieving service compositions in shorter times. On the contrary, in case of public Cloud services, the broker agent centralized the control under a single component, and this caused some deterioration in performance. The values shown are the average of the results obtained from repeating each composition 10 times. From these results we see that there is an overhead in time imposed by the system, but, in general, it is acceptable taking into account the added-value provided.

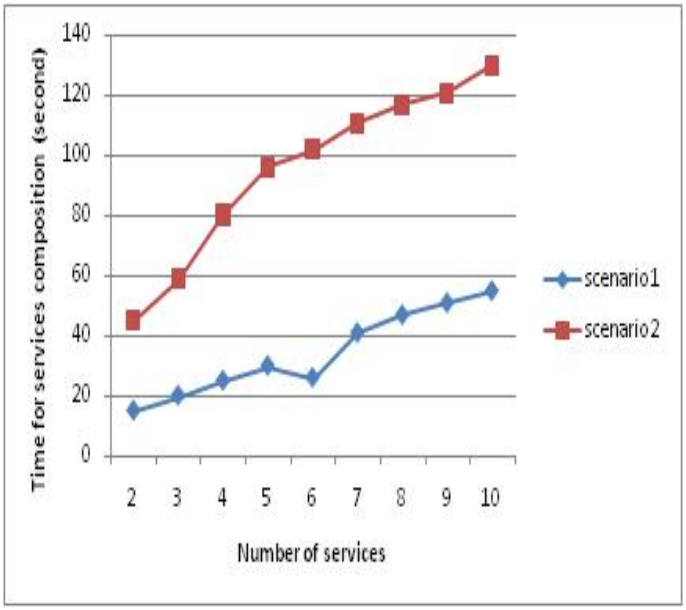

Figure 4: Comparison of Performance between scenario 1 and scenario 2

Also, the test shows that AiP4CSC can provide benefits to developers willing to adopt a hybrid approach, since it allows deploying, governing and monitoring both parts of a hybrid Cloud from the same single place. It provides useful information about the performance of the applications and the fulfillment of AiP and allows bursting an application in case of AiP violation.

Further, the implementation makes us realize that the basic components needed to address service composition in the Cloud are similar to the components in conventional service composition. However, AiP4CSC architecture gives the Cloud-based application developers more chances to get computational services and provide on-demand dynamic service composition. Also, services composition is specified as modular AiP, which conducive to reuse, refinement and aggregation of protocols. Consequently, the AiP4CSC acquires the previously mentioned protocols as input and suggest a specific protocol to achieve a new specific need. It maintains protocols repository to facilitate reusing of previously used protocols and composition routines in the future.

6. RELATED WORK

Since this work focuses on agent-based applications integration in modern enterprises, the related areas are: (i) Cloud computing based approaches, (ii) Web services based approaches, and (iii) agent based approaches.

6.1 Cloud Computing Based Approaches

Today, large technology vendors as well as open-source software projects both address the hybrid Cloud market and are developing virtual infrastructure management tools to set-up and manage hybrid Clouds [Hoecke et al. 2011].

The Reservoir architecture [Rochwerger et al. 2009] aims to satisfy the vision of service oriented computing by distinguishing and addressing the needs of service providers and infrastructure providers. Service providers interact with the end-users, understand and address their needs. They do not own the computational resources; instead, they lease them from infrastructure providers which interoperate with each other creating a seamlessly infinitive pool of IT resources.

The VMware workstation [Bugnion et al. 2012] offers live migration of virtual appliances and machines between data centers and allows service providers to offer IaaS while maintaining com-

patibility with internal VMware deployments.

HP [Collins 2009] provides three offerings for hybrid Cloud computing: HP Operations Orchestration for provisioning, HP Cloud Assure for cost control, and HP Communications as a Service for service providers to offer small businesses on-demand solutions. The Cloud-based HP Aggregation Platform for SaaS streamlines operations for both the service provider and the businesses customer by automating processes such as provisioning, activation, mediation charging, revenue settlement and service assurance.

The Cafe project [Mietzner 2010] provides a relevant approach for Cloud-based SaaS development, which offers an ad-hoc composition technique for application components and Cloud resources following the service component architecture. However, this approach requires SaaS developers to possess deep technical knowledge of the application architecture and the physical Cloud deployment environment to select and compose the right application components and Cloud resources.

In [La and Kim 2009], a systematic process for developing high-quality Cloud SaaSs has been proposed, taking into considerations the key design criteria for SaaS and the essential commonality/variability analysis to maximize the reusability.

Model-driven approaches are also employed for the purpose of automating the deployment of complex IaaS services on Cloud infrastructure. For instance, in [Moscato et al. 2012], authors propose the mOSAIC Ontology and the MetaMORP(h)OSY methodology for enabling model driven engineering of Cloud services. The methodology uses model driven engineering and model transformation techniques to analyse services. Due to the complexity of the systems to analyse, the mOSAIC Ontology is used in order to build modelling profiles in MetaMORP(h)OSY able to address Cloud domain-related properties.

When examining the Cloud based approaches we observe a recurrent theme. They do not allow for easy extensibility or customization options. Better ways are necessary for Cloud service consumers to orchestrate a cohesive Cloud computing solution and provision Cloud stack services that range across networking, computing, storage resources, and applications from diverse Cloud providers.

The work presented in this paper is considered as a first step toward AIPaaS (Agent Interaction Protocols as a Service). The AIPaaS approach is based on the idea of reducing the complexity involved when developing a composite application. In our work, the knowledge obtained during Cloud services composition is stored, shared, and reused. In fact, we proposed a basis for a theoretical approach for reusing and aggregating existing protocols to create a new desired business application. The proposed approach provides the underpinnings of aggregation abstractions for protocols. Our approach provides a set of operators that allows the creation of new value-added protocols using existing ones as building blocks.

6.2 Web Services Based Approaches

With the growing trends of service oriented computing, composition of Web services has received much interest to support flexible inter-enterprise application integration. As stated in [Milanovic and Malek 2004] and [Sheng et al. 2009], current efforts in Web services composition can be generally grouped into three categories: manual, automatic, and semi-automatic composition.

By manual composition, we mean that the composite service is designed by a human designer (i.e., service provider) and the whole service composition takes place during the design time. This approach works fine as long as the service environment, business partners, and component services do not or rarely change. On the other hand, automatic service composition approaches typically exploit the Semantic Web and artificial intelligence planning techniques. By giving a set of component services and a specified requirement (e.g., user's request), a composite service specification can be generated automatically [Berardi et al. 2005].

However, realizing a fully automatic service composition is still very difficult and presents several open issues [Milanovic and Malek 2004],[Berardi et al. 2005],[Bronsted et al. 2010]. The basic weakness of most research efforts proposed so far is that Web services do not share a full

understanding of their semantics, which largely affects the automatic selection of services.

There exist some research efforts that encourage manual and automatic compositions [Papazoglou et al. 2010][Montali et al. 2010]. Instead of coupling component services tightly in the service model, such approaches feature a high-level abstraction (e.g., UML activity model, protocol specifications, and interface) of the process models at the design time, while the concrete composite services are either generated automatically using tools or decided dynamically at run time (e.g., BPEL4WS [IBM et al. 2003]).

Our proposition is similar to these approaches in the sense that we also adopt a semi-automatic approach for application integration. The collaboration scenario is specified as interaction protocols not high-level goals and the component applications are selected, at run time, based on the protocol specification specified at the design time.

Also, the Web services related standards for services composition and interoperability, such as the BPEL4WS are lower level abstractions than ours since they specify flows in terms of message sequences. Also, they mix interaction activities and business logic making them unsuitable for reuse. In contrast to our approach, the BPEL4WS elements are only used to specify messages exchanges between the different business partners. Afterwards, this specification is used by agents to enact the integration of business processes at run time. Agents have the capability to dynamically form social structures through which they share commitments to the common goal. The individual agents, through their coordinated interactions achieve globally coherent behavior; they act as a collective entity known as a multiagent system. In our previous work [Benmerzoug et al. 2007][Benmerzoug 2013a], we have explored the relationship between Web services, multiagent systems and enterprise application integration.

6.3 Agent Based Approaches

Multiagent systems are a very active area of research and development. In fact, several researchers are working at the intersection of agents and collaborative enterprise systems.

For example, Buhler et al. [Buhler and Vidal 2005] summarize the relationship between agents and Web services with the aphorism Adaptive Workflow Engines = Web Services + Agents: namely, Web services provide the computational resources and agents provide the coordination framework. They propose the use of the BPEL4WS language as a specification language for expressing the initial social order of the multi-agent system. [Buhler and Vidal 2005] does not provide any design issues to ensure the correctness of their interaction protocols.

In [Garcia and Sim 2012], J. Octavio et al. proposed an agent-based approach to compose services in multi-Cloud environments for different types of Cloud services. Agents are endowed with a semi-recursive contract net protocol and service capability tables (information catalogs about Cloud participants) to compose services based on consumer requirements. However, the agent collaboration is limited to that of the contract net protocol.

Driven by the motivation for reuse of interaction protocols, [Vitteau and Huget 2004] and [Desai and Singh 2007] consider protocols as a modular abstractions that capture patterns of interaction among agents. In these approaches, composite protocols can be specified with a Protocol Description Language (such as: CPDL or MAD-P). Although formal, [Vitteau and Huget 2004] and [Desai and Singh 2007] do not not provide any step for the verification of the composite protocols.

Agent-oriented software methodologies aim to apply software engineering principles in the agent context e.g. Tropos, AMCIS, Amoeba, and Gaia. Tropos [Penserini et al. 2010] and AMCIS [Benmerzoug et al. 2004] differ from these in that they include an early requirements stage in the process. Amoeba [Desai et al. 2009] is a methodology for business processes that is based on business protocols. Protocols capture the business meaning of interactions among autonomous parties via commitments. Gaia [Cernuzzi et al. 2011] differs from others in that it describes roles in the software system being developed and identifies processes in which they are involved as well as safety and liveness conditions for the processes. It incorporates protocols under the interaction model and can be used with commitment protocols.

Our methodology differs from these in that it is aimed at achieving protocol re-usability by separation of protocols and business rules. It advocates and enables reuse of protocols as building blocks of business processes. Protocols can be composed and refined to yield more robust protocols.

7. CONCLUSION AND FUTURE WORK

Enterprise integration through integrated Cloud services is one of the most important issues facing business enterprises in this age of competition. The present research highlights the synergies between Cloud computing and agent paradigm. In such environments, the complexity that matters is not so much in the size of the code through which such entities are programmed but on the number and dynamicity of the interactions in which they will be involved. In this context, Agent Interaction Protocols (AiP) are a useful way for structuring communicative interaction among business partners, by organizing messages into relevant contexts and providing a common guide to the all parts.

Our review above indicates that the multiagent systems (MAS) paradigm provides an excellent modeling approach, architecture, and technological platform for developing and implementing Cloud-based enterprise systems that are flexible and can easily and quickly grow and adapt to changing business environments. Our synthesis of the EAI modeling and MAS literatures has led us to propose a multi-agent-based integrative architecture. The proposed architecture is based on AiP that supports ad hoc composition and deployment of new messages. AiP are treated as modular components, potentially composed into additional protocols, and applied in a variety of business processes. By maintaining repositories of commonly used, generic, and modular protocols, we can facilitate the reuse of a variety of well-defined, well-understood and validated protocols. For example, a payment protocol can be used in a process for purchasing goods as well as in a process for registering for classes at a university. Further, the repository would expand as newly composed protocols are inserted into it.

In the immediate future, we plan to conduct experiments on a much larger scale to evaluate the scalability of the AiP4CSC in real world settings. Also, we will focus on deploying the proposed architecture in a semantic web service framework.

Acknowledgements

The author would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.

REFERENCES

A. Artikis, J. Bentahar, A. K. C. and Dignum, F. 2010. Protocol refinement: Formalization and verification. In *AAMAS Workshop on Agent Communication (AC)*. 19–36.

Barjis, J., Gupta, A., and Sharda, R. 2011. Knowledge work and communication challenges in networked enterprises. *Information Systems Frontiers 13,* 5, 615–619.

Benmerzoug, D. 2013a. Agent approach in support of enterprise application integration. *International Journal of Computer Science and Telecommunications 4,* 1, 47–53.

Benmerzoug, D. 2013b. An Agent-Based Approach for Hybrid Multi-Cloud Applications. *Scalable Computing: Practice and Experience 14,* 2, 95 – 109.

Benmerzoug, D., Boufaida, M., and Boufaida, Z. 2004. From the Analysis of Cooperation Within Organizational Environments to the Design of Cooperative Information Systems: An Agent-Based Approach. In *OTM Workshops*. LNCS, vol. 3292. Springer, Larnaca, Chypre, 495–506.

Benmerzoug, D., Boufaida, M., and Kordon, F. 2007. A Specification and Validation Approach for Business Process Integration based on Web Services and Agents. In *Proceedings of the 5th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems, MSVVEIS-2007, In conjunction with ICEIS 2007*. NSTIIC press, 163–168.

Benmerzoug, D., Gharzouli, M., and Zerari, M. 2013. Agent interaction protocols in support of cloud services composition. In *6th International Conference on Industrial Applications of Holonic and Multi-Agent Systems*. LNAI, vol. 8062. Springer, Prague, Czech Republic, 293 – 304.

Benmerzoug, D., Kordon, F., and Boufaida, M. 2008a. A Petri-Net based Formalisation of Interaction Protocols applied to Business Process Integration. In *Advances in Enterprise Engineering I, 4th International Workshop on Enterprise & Organizational Modeling and Simulation (EOMAS'08)*. LNBIP, vol. 10. Springer, Montpellier, France, 78–92.

Benmerzoug, D., Kordon, F., and Boufaida, M. 2008b. Formalisation and Verification of Interaction Protocols for Business Process Integration: a Petri net Approach. *International Journal of Simulation and Process Modelling 4,* 3 - 4, 195–204.

Berardi, D., Giacomo, G. D., Mecella, M., and Calvanese, D. 2005. Automatic composition of process-based web services: a challenge. In *Proc. 14th Int. World Wide Web Conf.*

Bronsted, J., Hansen, K. M., and Ingstrup, M. 2010. Service composition issues in pervasive computing. *IEEE Pervasive Computing 9,* 1, 62–70.

Bugnion, E., Devine, S., Rosenblum, M., Sugerman, J., and Wang, E. Y. 2012. Bringing virtualization to the x86 architecture with the original vmware workstation. *ACM Trans. Comput. Syst. 30,* 4 (Nov.), 12:1–12:51.

Buhler, P. A. and Vidal, J. M. 2005. Towards adaptive workflow enactment using multiagent systems. *International Journal On Information Technology and Management 6,* 1, 61–87.

C. Trappey, A. Trappey, C. H. and Ku, C. 2009. The design of a JADE-based autonomous workflow management system for collaborative SoC design. *Expert Syst. Appl. 36,* 2, 2659–2669.

Cernuzzi, L., Molesini, A., Omicini, A., and Zambonelli, F. 2011. Adaptable multi-agent systems: the case of the gaia methodology. *International Journal of Software Engineering and Knowledge Engineering 21,* 4, 491–521.

Collins, D. 2009. Communications as a service for midsize businesses.

Desai, N., Chopra, A. K., and Singh, M. P. 2009. Amoeba: A Methodology for Modeling and Evolving Cross-Organizational Business Processes. *Journal of ACM Trans. Softw. Eng. Methodol. 19,* 2.

Desai, N. and Singh, M. P. 2007. A modular action description language for protocol composition. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*. AAAI Press, 962–967.

FIPA-ACL. 2001. FIPA Communicative Act Library Specification. Tech. Rep. http://www.fipa.org/specs/XC00037/, FIPA - Foundation for Intelligent Physical Agents.

Garcia, J. O. and Sim, K. M. 2012. Agent-based cloud service composition. *Applied Intelligence, The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies 22,* 2 (September).

Hoecke, S. V., Waterbley, T., Devos, J., Deneut, T., and Gelas, J. D. 2011. Efficient management of hybrid clouds. In *The Second International Conference on Cloud Computing, GRIDs, and Virtualization*. Rome, Italy, 167–172.

IBM, Microsoft, SAP, and Siebel, S. 2003. Business process execution language for web services version 1.1. Tech. rep.

Kishore, R., Zhang, H., and Ramesh, R. 2006. Enterprise integration using the agent paradigm: foundations of multi-agent-based integrative business information systems. *Decision Support Systems 42,* 1, 48–78.

La, H. J. and Kim, S. D. 2009. A systematic process for developing high quality saas cloud services. In *Proceedings of the 1st CloudCom '09*. Springer-Verlag, 278289.

market:, T. F. R. http://www.forrester.com/.

Mell, P. and Grance, T. 2009. The NIST Definition of Cloud Computing.

Mietzner, R. 2010. *A method and implementation to define and provision variable composite applications, and its usage in cloud computing*. Dissertation, Universitat Stuttgart, Germany.

Milanovic, N. and Malek, M. 2004. Current solutions for web service composition. *IEEE Internet Computing 8,* 6, 51–59.

Montali, M., Pesic, M., van der Aalst, W. M. P., Chesani, F., Mello, P., and Storari, S. 2010. Declarative specification and verification of service choreographiess. *International journal of ACM Transactions on the Web 4,* 1.

Moscato, F., Martino, B. D., and Aversa, R. 2012. Enabling Model Driven Engineering of Cloud Services by using mOSAIC Ontology. *Scalable Computing: Practice and Experience 13,* 1, 29–44.

Neil, M. 2011. Microsoft windows azure development cookbook. Packt Publishing.

Nguyen, D. K., Lelli, F., Papazoglou, M., and van den Heuvel, W.-J. 2012. Blueprinting Approach in Support of Cloud Computing. *International Journal of Future Internet 4,* 1, 322–346.

Pablo, C., Kurt, C., Fabio, C., and Johann, G. 2010. Professional WCF 4: Windows Communication Foundation with .NET 4. Wrox Publishing.

Papazoglou, M., Pohl, K., Parkin, M., and Metzger, A., Eds. 2010. *Service Research Challenges and Solutions for the Future Internet - S-Cube - Towards Engineering, Managing and Adapting Service-Based Systems*. Lecture Notes in Computer Science, vol. 6500. Springer.

Penserini, L., Kuflik, T., Busetta, P., and Bresciani, P. 2010. Agent-based organizational structures for ambient intelligence scenarios. *Ambient Intelligence and Smart Environments 2,* 4, 409–433.

Rochwerger, B., Breitgand, D., Levy, E., Galis, A., Nagin, K., Llorente, I. M., Montero, R., Wolfsthal, Y., Elmroth, E., Cáceres, J., Ben-Yehuda, M., Emmerich, W., and Galán, F. 2009. The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development 53,* 4 (July), 535–545.

Sheng, Q. Z., Benatallah, B., Maamar, Z., and Ngu, A. H. H. 2009. Configurable composition and adaptive provisioning of web services. *IEEE T. Services Computing 2,* 1, 34–49.

Sim, K. M. 2012. Agent-based cloud computing. *IEEE Trans Serv Comput 5.*

Tobaly, G. 2010. Why traditional enterprise software sales fail. http://sandhill.com/article/why-traditional-enterprise-software-sales-fail/.

Vitteau, B. and Huget, M.-P. 2004. Modularity in interaction protocols. In *Advances in Agent Communication.* LNCS, vol. 2922. Springer, 291309.

Wang, S., Shen, W., and Hao, Q. 2006. An agent-based web service workflow model for inter-enterprise collaboration. *Expert Syst. Appl. 31,* 4, 787 – 799.

Zeginis, D., D'Andria, F., Bocconi, S., Cruz, J. G., Martin, O. C., Gouvas, P., Ledakis, G., and Tarabanis, K. A. 2013. A user-centric multi-PaaS application management solution for hybrid multi-Cloud scenarios. *Scalable Computing: Practice and Experience 14,* 1, 17–32.

Djamel Benmerzoug is currently an assistant professor in the department of Software Technologies and Information Systems at Constantine 2 University, Algeria. He holds a PhD in Computer science from Pierre & Marie Curie University (Paris - France) and UMC University (Constantine - Algeria), respectively. His current research interests include Cloud computing, multiagent systems, service oriented computing, and business processes modelling and verification.