Toward an Architectural Model for Highly-Dynamic Multi-tenant Multi-service Cloud-Oriented Platforms

Adel Titous(✉), Mohamed Cheriet, and Abdelouahed Gherbi

Ecole de Technologie Superieur,
1100, rue Notre-Dame Ouest, Montreal, QC H3C 1K3, Canada
adel.titous.1@ens.etsmtl.ca

Abstract. The characteristics of the Cloud Computing paradigm make it attractive to be used along with other paradigms like mobile and pervasive computing, smart cities, etc. There is a need to develop new platforms in order to take advantage of those converged infrastructures, and to abstract its high heterogeneities and complexities. The design and development of such robust and efficient platforms is challenging, because of the high heterogeneities, complexities and the wide range of features they are supposed to offer.

In this paper, we define some fundamental requirements related to those converged paradigms and infrastructures, and by consequence the requirements for cloud-oriented platforms. We also develop an architectural model, based on a new concept, the semantically defined resource, which is the result of the need to simplify the definition of lightweight services and to introduce more semantics. We also present an example to illustrate how to design architectures basing on the new concept, and how the architectural model can be used concretely.

Keywords: SOA · EDA · SDR · Cloud computing · Semantic web · Converged infrastructures

1 Introduction

The wide adoption of the Cloud Computing (CC) model, and the convergence with other computing paradigms, like mobile computing, introduced some architectural issues. Considering CC as *"a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"* [1], many aspects must be taken into account when designing and developing cloud oriented platforms for different domains of application. For instance, in [2], Sanai et al. present some important challenges and problems related to the heterogeneity of mobile cloud computing, and present architectural issues as the first requirement, since it is crucial to develop a reference architecture.

A. Leon-Garcia et al. (Eds.): Smart City 2015, LNICST 166, pp. 203–214, 2016.
DOI: 10.1007/978-3-319-33681-7_17

By cloud-oriented platforms (see Fig. 1) we want to mention platforms to be developed for different domains, as MCC. While the boundaries between the cloud infrastructures and those platforms are not always clear, techniques and principles used to designe the coud-oriented platforms can be applied to design and develop a great part of the cloud infrastructure itself.

So cloud-oriented platforms should comply with different requirements with respect to different application domains, but in general we can distinguish some common fundamental requirements, like the high dynamicity and heterogeneity. Other caracteristics can be defined based on those fundamental requirements. For instance, we can not have efficient elasticity if we do not have adequate dynamicity.

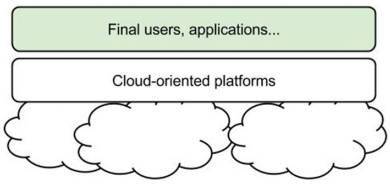

Fig. 1. Cloud oriented platforms

In a such context, the need for an architectural model facilitating the task of design and development of infrastructures is crucial. Several researches tried to take advantage of the Service Oriented Architecture (SOA) to design cloud systems, including [13–15], since SOA is technology-neutral and facilitates the design of large scale distributed systems based on loosely coupled services in highly heterogeneous environments [7,8]. SOA gives answers for some requirements but not for all of them, especially those related to the high dynamicity. Recently some efforts were spent in order to merge SOA with the Event Driven Architecture (EDA) [17,18]. The association between services and events can for example trigger services in response to some events, or produce events by services, which can improve the dynamicity of the classical SOA schema. However, the proposed solutions are restricted to a specific domain, because there is no real fusion between the concepts of services and events, and to our best knowledge, up to now there is no general architectural model, comparable to SOA for example, that can be efficiently applied to a large variety of problems.

On the other hand, the use of ontologies to categorize cloud services is an other track being investigated [3] in order to offer more possibilities to categorize, compare and choose best offered services.

In this paper, we merge all those three tracks into a unique and general architectural model, to be used in the design and development of cloud oriented platforms for different domains.

The remainder of the paper is organized as follows: the second section presents some related works, while the third section presents a brief background.

In the fourth section we define some fundamental requirements that any architectural model for cloud-oriented platforms must satisfy. The section five presents our introduced concept. The section six presents our architectural model. A use case is presented in the seventh section, followed by some discussions in the eight section. A conclusion conclude the paper.

2 Related Works

This section exposes some works in relation with clouds' architectures. Sanaei et al. [2] proclaim powerful dynamic representation and monitoring techniques, especially for heterogeneous wireless environments for cloud-mobile users.

In [3], Di Marino et al. present a semantic representation of services of Openstack platform, using OWL-S.

Celesti et al. [12] give a classification of CC market evolution. In [13] Tsai et al. present the Service Oriented Cloud Computing Architecture (SOCCA). In [15] Zhang and Zhou present the Cloud Computing Open Architecture (CCOA). In [16] Zou et al. assume that the internal structures of private and public clouds are consistent with each other, and present an architecture based on an added layer called cloud bus as an inter-cloud communication middleware.

Zhang et al. [17] develop an event driven SOA for internet of things based on a layered architecture. In [18] Laliwala and Chaudhary develop an Event Driven Service Oriented Architecture (EDSOA). Kim et al. presented ECO, a middleware for cloud of things [19].

Virtualization is another important concern, since it is being used at different levels: servers, storage, networking, etc. In [14] Duan et al. present service-orientation based efforts for networks virtualizations, and show that their are huge ambiguities such as different definitions of QoS, resulting in the absence of a consensus about definition of a network infrastructure description language to describe different networking resources.

3 Background

Some concepts are necessary to understand the work, they are summarized in the following subsections.

3.1 Service Oriented Architecture

SOA is an architectural style that promote service orientation. Thus large softwares can be organized into a collection of interacting services [6], which are autonomuous software packages (similar to classical APIs), and they are described and published in repositories. Tow main actors in SOA: the service provider and the service consumer (see Fig. 2).

Even Web services are the predominant technology used in it's implementation, SOA is technology-neutral, which is one of it's most important features.

Another feature is the abstraction, in a sense where the service consumer has no need to know how the service is implemented, all what he need is the description of the service. Those two features facilitate the design of distributed systems, especially those caracterized by high heterogeneities.

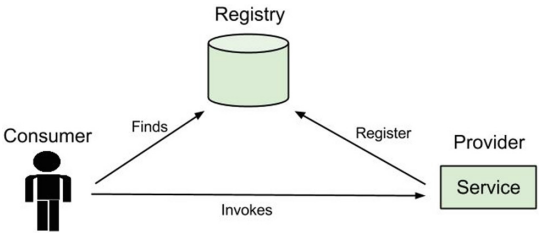

Fig. 2. Service oriented architecture

3.2 Event Driven Architecture

The Event Driven Architecture (EDA) is based on the concept of Events. An event can be any significant change in a given situation, relating to different contexts: overpassing a threshold value, overpassing a certain number of requests, a new situation in the business process, etc. i.e. any significant new situation that may be taken in account through triggering other events or process as a response. for instance, if the quantity of a product in the stock is under the permitted value, this event will not be just written in a file or a database, but will also trigger an ordering process. There are generally two types of events: ordinary and notables, where the last ones are used for signalling more important situations. An EDA is generally built using four layers:

- Event generator: which is the source that has generated the event. If a standard event format is used, the source has to transform the event in the right format before sending it on a channel.
- Event channel: transports events from the source to the event processor.
- Event processor: which process events, sometimes using some established rules. Two types of event processors can be distinguished: the simple ones process each event independently while the complex ones process an event accordingly to the context of prior and future events, possibly using patterns. Some companies like IBM provide complex event processing engines[1].
- Downstream Event-Driven Activity: downstream activities triggered after the processing of the event(s).

[1] As WebSpher Event Processing Software.

3.3 Ontologies and Semantic Web

There are many definitions of ontologies in artificial intelligence related literature. In [4] an ontology is defined as *"a formal explicit description of concepts in a domain of discourse (classes (sometimes called concepts)), properties of each concept describing various features and attributes of the concept (slots (sometimes called roles or properties)), and restrictions on slots (facets (sometimes called role restrictions)). An ontology together with a set of individual instances of classes constitutes a knowledge base"*. Ontologies can be machine readable, in order to infer new knowledge or, more important in our case, to verify consistencies of informations with the ontology. In the case of Web Services we can find OWLS, which is based on Web Ontology Language (OWL), a standard of the W3C [5].

4 The Requirements of Converged Infrastructures

The architecture of a system is the fundamental organization of it's components and the relations between them, and is developed in response to some constraints and requirements. In the context of converged infrastructures presented in the introduction, cloud-oriented platforms must satisfy high requirements such as mobility and dynamicity, where the needs of users change dynamically, and moreover capabilities of resources also change dynamically.

So the first question to be asked is what are the requirements that the architectural model must satisfy?

First of all, a huge heterogeneity caracterize the converged infrastructures: different devices from different constructors, networks of sensors, different systems and applications with different requirements, architectures, etc. Thus, an adequate abstraction is required to overcome this huge heterogeneity and inherent complexity.

The second requirement is the multi-granularity, as the ability to deal efficiently with different levels of granularities, from the finest one to the coarsest one, since in a such context we can find resources like virtualized devices and hardware, as well as business related services, usually with coarse granularity.

Finally, we find in [20] a description of volatile systems, characterized by:

- Failure of devices and communication links,
- Changes in the characteristics of communications such as bandwidth,
- Creation and destruction of associations[2] between software components resident on different devices.

We can add two characteristics:

- Spontaneous appearance and disappearance of resources, devices and services,
- Dynamic changes in the characteristics and capabilities of resources.

[2] Logical communication relationships.

In the remainder of this paper, we use the term of dynamicity to summarize all of those characteristics, which represent our third requirement.

Even SOA provides adequate abstraction power, we think that it has some lacks, especially dynamicity. SOA was developed in the context of companies' information systems, totally different from our context. Also fine-granularity services are not supported as coarse-granularity services. We think that there is a need to make as lightweight as possible the definition and the description of fine-granularity services.

On the other hand, merging SOA and EDA has improved the dynamciity, but do not provide better support for fine-granularity services.

We have developed our model in a fundamentally different way, basing on a new concept, which is in the intersection between SOA, EDA, and Semantics.

5 The Concept of SDR

In order to comply with the context described in section four, we define a new concept based on SOA, EDA and semantics. The idea is to take advantage of SOA, but with improvement of the support for the very-fine granularity services, making their definitions as lightweight as possible by using a formal semantic technique rather than heavy description documents published in a registry. The interaction between providers and consumers is insured by using event processing engines.

Thus, we introduce the Semantically Defined Resource (SDR). An SDR is the abstraction of any resource that architects consider useful in a given context: a distributed object or component, a buffering space, a stack, a processor... etc. Table 1 gives a comparison between SDRs and Services.

Table 1. Services versus SDRs.

Criteria	Services	SDRs
Nature	software	software and hardware
Granularity	varied, but mostly coarse	finest as possible
Description	in repositories	semantic

Maybe in a given context, a resource can be abstracted as an SDR, but in another context the same resource may be integrated with other resources into a more wide scope SDR, in a similar way classes are designed in an object oriented approach.

SDRs are very loose coupled entities, and it is recommended that they have the finest possible granularity, since this will improve modularity and facilitate the maintenance, and augment sharing possibilities, but more important, to fit with the nature of devices available in the converged infrastructures context. Figure 3 present a conceptual diagram of the model: on the right side, there is a

layered structure. The first layer is the SDR layer, abstracting different resources, which are defined semantically. Since SDRs may have dependencies in a given situation, the ability to gather them in structures like containers for example is useful, and this is the role of the second layer: the SDR-Containers (SDR-C). Of course, an SDR may be included into more than one SDR-C. It is important to note here that we assume that isolation and security concerns are provided by virtualization technology, or synchronization mechanisms, SDR-Cs deal with slices of resources. Like SDRs, SDR-Cs are defined semantically. The last layer is the Services layer, representing the known SOA services, with one or many related description documents in the registry. Instead of being published in the registry as services, SDRs are defined semantically and managed by the Event Processing Engine.

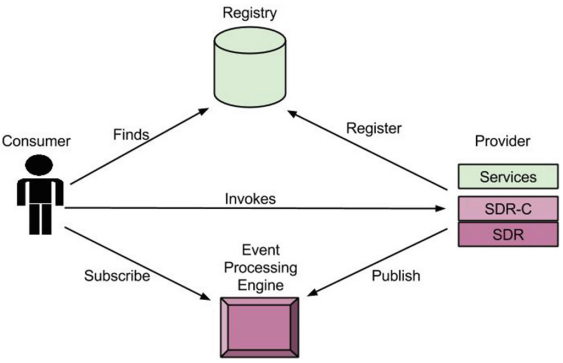

Fig. 3. The complementarity of services and SDRs

6 The Architectural Model

Let introduce some definitions to help the understanding of the model. Let assume that we have a resource with some related characteristics or properties, if the resource is a buffering space for eg. the available capacity is an important related characteristic. $\mathcal{D}$ is a semantic definition of the resource, and $\mathcal{P} = \{p_1, p_2, ...p_n\} \subseteq \mathcal{D}$ the set of its properties.

Definition 1. *(SDR) An SDR is a tuple* $< \mathcal{D}, \mathcal{S}, \mathcal{A}, D_{in}, D_{out} >$*, where:*

- $\mathcal{S}$ *is the set of states related to the resource.*
- $\mathcal{A}$ *is a set of activities that can change the state of the SDR.*
- D_{in} *are inputs sent by the consumer.*
- D_{out} *are outputs received by the consumer.*

Two states $s_i, s_j \in \mathcal{S}$ are different, if and only if $\exists!$ p $\in \mathcal{P}$, where the value of p is different in s_i and s_j. Figure 4 shows the states of a given resource from the perspective of middleware and users, where a resource can be available, not available or available with different states, for eg. the available capacity of the buffer can change dynamically.

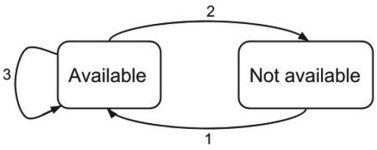

Fig. 4. States of resources

Definition 2. *(Event) An event* $\mathcal{E} \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$.

Basing on event definition, an event can be produced each time the state of SDR change, i.e. in the Fig. 4 each time a transition is made, i.e. through arc 1, 2 or 3.

SDR-Cs are containers, or virtual execution environments for SDRs, and they are also defined semantically, for instance, in the same ontology than SDRs.

Definition 3. *(SDR-C) An SDR-C is a tuple* $< \mathcal{D}, \mathcal{S}, \mathcal{A}, D_{in}, D_{out}, \mathcal{L}_{SDR} >$, *where* $\mathcal{L}_{SDR}$ *is the list of SDRs involved in the SDR-C.*

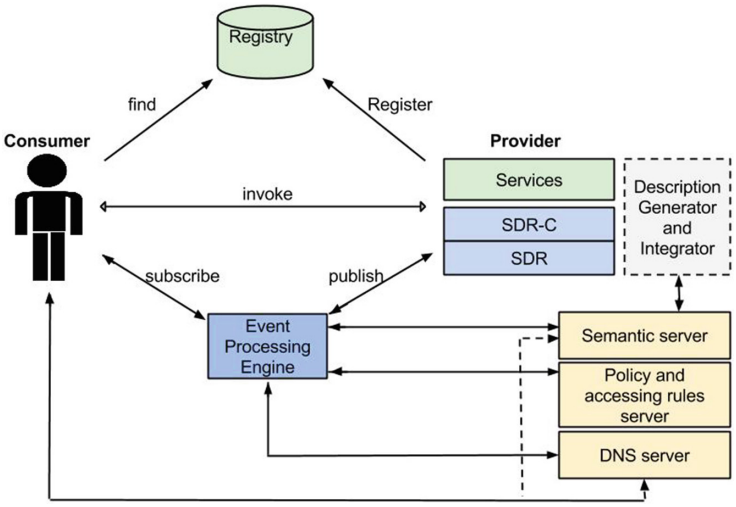

Fig. 5. The architectural model

An SDR-C may contain other SDR-Cs, some added code playing the role of interfaces between SDRs, or to have some special SDRs configurations.

SDR-Cs can be defined as services and published in service registry. It is up to the architect to design his solution basing on needs, and already available SDRs, SDR-Cs and services.

Figure 5 shows the architectural model based on SDRs. The central component is the Event Processing Engine (EPE), which manage and process different events emitted using publish/subscribe communication pattern. Publishers are providers, consumers are subscribers. EPE manage subscribers lists by related event servers, in other words, the event servers life cycle management is performed basing on informations in the semantic server. The model contains also a semantic server containing different semantic definitions (SDRs and SDR-Cs) available for all tenants: the provider, the consumer and EPE. This is because all tenants have to share the same formal definitions of SDRs and SDR-Cs. Because SDRs and SDR-Cs are published on server events, managed EPE, accessing rules must be setted for consumers and providers for accessing different event servers, which motivate the policy and accessing rules server. Finally a DNS server will help to locate different event servers and parts of the middleware.

7 Use Case

The Fig. 6 shows the role of the ontology and event processing engine. Classes in the ontology are used by the event processing engine to create event-servers, in order to manage SDRs. Thus, it may be an event-server for each SDR. Imagine that in a smart house, we need to copy a buffer `b1` in another one `b2`, while `b2` is momentarily full, but another buffer `b3` is available. We have defined a

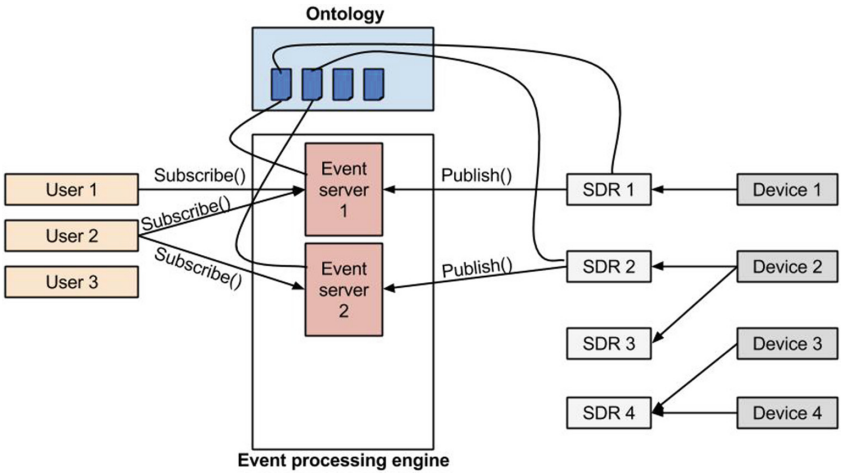

Fig. 6. The roles of the ontology and event processing engine

simple ontology partially represented in Fig. 7 using UML class diagram. The class `DecodingNode` is composed of a `Buffer` and a `Processor`.

The publisher/subscriber communication paradigm is used: the first event-server is `BufferServer`, on which `b3` is published, the subscriber `b1` is notified about the availability of `b3` by `BufferServer` with indication of the size of `b3`, `b1` will be copied in `b3` and unsubscribed from BufferServer. Since `b3` is greater than `b1`, it will stay published on `BufferServer`, but with an updated size.

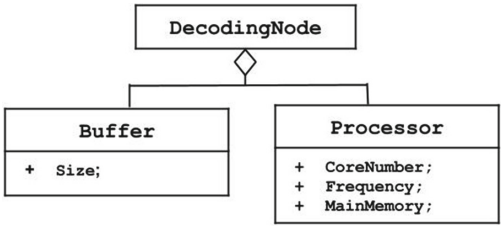

Fig. 7. A part of the ontology

8 Discussion and Future Works

By merging the two architectural paradigms SOA and EDA, and basing on SDRs, we try to make definition of resources as lightweight as possible, and at the same time to comply with the fundamental requirements defined in section four. The abstraction is ensured by using a common semantic definition for the domain in case of SDRs, and the definition of services in the registry. The multi-granularity is ensured by the fact that SDRs are very-fine granularity entities. The dynamicity is ensured by the EPE.

We think that in many cases, problems of inter-operability related to converged infrastructures will be reduced into engineering problems about accessing different event servers, rather than dealing with the static nature of Service Level Agreements (SLAs) of services. Providers will have possibility to define different policies for different SDRs, and moreover changing them on the fly for performances reasons, prices, energy consumption concerns, etc. The use of publish/subscribe model within Event Processing Engine will diminish the network load.

As future works, we aim to develop an Event Processing Engine and tools for providers and consumers. Also to develop projects basing on architectures instantiated from the model, as more complete and concrete use cases.

9 Conclusion

In this paper, we have proposed an architectural model for cloud-oriented platforms, in order to comply with some fundamental requirements related to the

converged infrastructures. The model can be seen as a refinement of Service Oriented Architecture, merged with Event Driven Architecture, alongside with the use of semantics to define fine-granularity resources. The model is based on the Semantically Defined Resource, the concept introduced in this paper. We think that with an improved dynamicity and efficient mechanisms for very fine granularity resources, the model will offer possibilities for concurrent access to resources, and to define and share different building blocks of highly dynamic platforms, systems and applications through a converged infrastructures environment.

References

1. Mell, P., Grance, T.: The NIST definition of cloud computing. Commun. ACM **53**(6), 50 (2010)
2. Sanaei, Z., Abolfazli, A., Gani, A., Buyya, R.: Heterogeneity in mobile cloud computing: taxonomy and open challenges. IEEE Commun. Surv. Tutor. **16**(1), 369–392 (2014). First quarter
3. Di Martino, B., Cretella, G., Esposito, A., Carta, G.: Semantic representation of cloud services: a case study for openstack. In: Fortino, G., Di Fatta, G., Li, W., Ochoa, S., Cuzzocrea, A., Pathan, M. (eds.) IDCS 2014. LNCS, vol. 8729, pp. 39–50. Springer, Heidelberg (2014)
4. Noy, N.F., McGuinness, D.L.: Ontology development 101: a guide to creating your first ontology, Technical report KSL-01-05, Stanford Knowledge Systems Laboratory (2001)
5. http://www.w3.org/OWL/
6. Buyya, R., Vecchiola, C., Selvi, S.T.: Mastering Cloud Computing Foundations and Applications Programming. Morgan Kaufmann Editions, San Francisco (2013)
7. Erl, T.: SOA Principle of Services Design. Prentice Hall Publishing, Upper Saddle River (2008)
8. Josuttis, N.M.: SOA in Practice. The Art of Distributed Systems Design. O'Reilly, Sebastopol (2007)
9. Shroff, G.: Enterprise Cloud Computing: Technology, Architecture, Applications. Cambridge University Press, Cambridge (2010)
10. Toosi, A.N., Calheiros, R.N., Buyya, R.: Interconnected cloud computing environments: challenges, taxonomy, and survey. ACM Comput. Surv. **47**(1), Article 7 (2014)
11. Petcu, D.: Portability and interoperability between clouds: challenges and case study. In: Abramowicz, W., Llorente, I.M., Surridge, M., Zisman, A., Vayssière, J. (eds.) ServiceWave 2011. LNCS, vol. 6994, pp. 62–74. Springer, Heidelberg (2011)
12. Celesti, A., Tusa, F., Villari, M., Puliafito, A.: How to enhance cloud architectures to enable cross-federation. In: IEEE 3rd International Conference on Cloud Computing (2010)
13. Tsai, W., Sun, X., Balasooriya, J.: Service-oriented cloud computing architecture. In: IEEE Seventh International Conference on Information Technology (2010)
14. Duang, Q., Yan, Y., Vasilakos, A.V.: A survey on service-oriented network virtualization toward convergence of networking and cloud computing. IEEE Trans. Netw. Serv. Manage. **9**(4), 373–392 (2012)
15. Zhang, L.J., Zhou, Q.: CCOA: cloud computing open architecture. In: IEEE International Conference on Web Services (2009)

16. Zou, C., Deng, H., Qiu, Q.: Design and implementation of hybrid cloud computing architecture based on cloud bus. In: IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks (2013)
17. Zhang, Y., Duan, L., Chen, J.: Event-driven SOA For IoT Services. Int. J. Serv. Comput. **2**(2) (2014). (ISSN 2330–4472)
18. Laliwala, Z., Chaudhary, S.: Event-driven service-oriented architecture. In: 2008 International Conference on Service Systems and Service Management. IEEE (2008)
19. Kim, S.H., Kim, D.: Multi-tenancy support with organization management in the cloud of things. In: IEEE 10th International Conference on Services Computing (2013)
20. Coulouris, G., Dollimore, J., Kindberg, T., Blair, G.: Distributed Systems, Concepts and Design, 5th edn. Addison-Wesley, Boston (2012)
21. Ericsson Mobility Report, November 2014
22. A Conceptual Model for Event Processing Systems. IBM (2010)