An Approach to Enable Cloud-Computing by the Abstraction of Event-Processing Classes

Jonathan Eccles and George Loizou

Department of Computer Science and Information Systems, Birkbeck,
University of London, London, WC1E 7HX, UK
jonathan.eccles@hp.com

***Abstract*—Following our introduction of the concept of Abstraction Classes, we present herein their realisation within a cloud environment. This is achieved using a combination of integrated service-location models, including Knowledge-Based Systems, and distributed metadata using XML. This is complemented by service control software invoked at the level of Abstraction Classes.**

***Keywords*—Cloud Architecture, Virtualisation, Abstraction Classes, Knowledge-Based Systems, Service Control**

1. INTRODUCTION

Recently we introduced the concept of an Abstraction Class [1] for the purpose of enabling the processing of a transaction within a cloud environment in a standard manner using the dynamic nature and virtualised properties of a cloud to good effect. The manner in which an Abstraction Class is invoked by an event and the subsequent protocol required for the processing of that event are presented. This leads to two areas of applied research, so that the processing patterns of such classes could be applied to a cloud architecture in general.

The first area is the construction of event classes incorporating those events employed in the processing of Abstraction Classes, so that their properties may induce a degree of standardisation through modelling. This enables the design and implementation of event classes to be simplified. The key point is that events that are produced from the receipt of external data are internal to the cloud, and herein we describe how they are derived; these become abstracted entities within a standard dataflow model. This will reduce the coding required by Abstraction Classes and will thereby reduce the overall level of processing required by the virtualised systems that host such operational classes, e.g. Initiator Process in Figure 1, within a cloud.

The second area of applied research is to distribute the method of forward lookup of the target goal of an invocation of a (set of) service(s) through an Abstraction Class. This is done through the application of research that was formerly employed in the use of predicate logic applied through the use of Knowledge-Based Systems in the determination of goals, where the target data may be distributed across a networked environment. One of the key problems that is addressed by our work in the area of Abstraction Classes concerns the fact that the data that is pertinent to cloud-based services and transactions is distributed over what may possibly be a wide area network, and the host cloud may be subject to dynamic changes in its access structure. Therefore, it is important that the design of a system – activating an event-based protocol enabling a cloud-based process – should be integrated with a system that communicates with the same state map of the cloud at the point of the invocation of that process.

This paper extends upon our previous work by describing how the said design can be modelled and outlines the initial construction of a demonstration system at a proof-of-concept level. This proof-of-concept is realised using Inter-Process Message (IPM) modules within a virtualised environment employing an HP c7000 chassis and HP blade technology as the host base.

2. EVENT ABSTRACTION

Adaptive computing [2] focuses on the methodology and implementation of systems that adjust to different situations. An adaptive system may change its own behaviour, so as to fulfil the goals, tasks, interests, and other features of individual users and the environment. Adaptivity is important for pervasive and ubiquitous computing. The terms *pervasive* and *ubiquitous computing* are used to describe a smart space, in our case occupied by a cloud which is populated by hundreds of intelligent devices that are embedded in their surroundings [3]. In this paper such devices are represented as Abstraction Classes. These form an effective vehicle upon which to form the initial basis of cloud management. The said computing devices blend into the background, unobtrusively collaborating to provide services that add value to the cloud for users. Services are thus essential to the success of this technology and, as a result, both service discovery and service management will play essential roles in enabling their continuous deployment within clouds. These facilities should be able to add value through their potential collaboration by invoking relevant Abstraction Classes. Additionally, the increasing complexity of such invocations in such a large-scale, distributed and dynamic environment should be hidden from users.

DOI: 10.5176_2010-2283_1.1.23

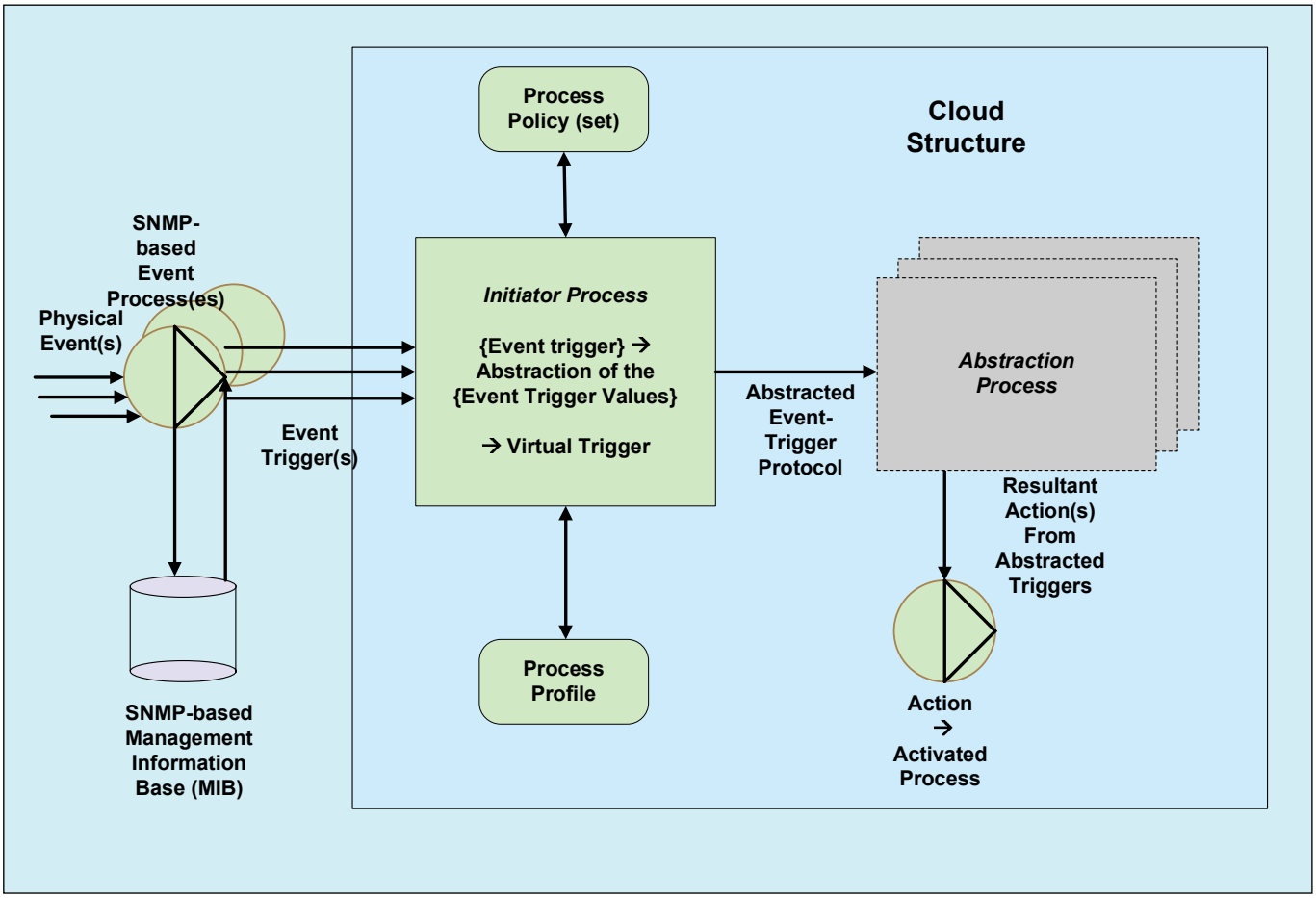

Fig. 1. The fundamental model of a simplified abstraction class for events external to a cloud leading to the corresponding virtualised event that will be used in the abstracted information flow within a cloud

In order for the expansion of such systems to be sustainable in the longer term within and between clouds, it is required that the mechanisms responsible for the location of the (sets of) required services should be enabled by integrating different methods which ultimately centralise on a model-based architecture rather than rely solely upon automatically generated search activities [4]. This approach increases the degree of control available within such a system.

This is a type of program [4] that issues queries in order to automatically interact with web search engines. As a consequence, such a program can consume a large portion of the overall traffic within a system, in this case a cloud. First, program generated traffic can usually peak to a proportionately large amount of the total traffic volume in a relatively short period of time, causing an increase in the search engine response time. This is compounded by the increase in % CPU processing, which accompanies the increase in traffic and the increase in updating to search logs which are often retained for later analysis.

In order to achieve a degree of control in combination with a degree of management of the location and trace processes of required (sets of) services and applications within a cloud, it is required to minimise the number of events that traverse a cloud and also to standardise such events. This is achieved by, in effect, producing a virtual event to represent a particular class of abstracted information flow (Figure 1).

In order for this method of transmitting and controlling event processing to attain the performance level required, there must be a standard design behind the event protocol processing model. This is initially at the level of virtualisation of external events within the initiator class, and subsequently within the abstracted networks within a cloud. This is defined in an earlier paper [1] as follows and is illustrated in Figure 3 (cf. Figure 4):

Abstraction Classes linked through

1. Initiator: Initial Event Request to Initiator Process – Abstract Call via Command Language

2. Initiator Process uses Policy derived/obtained from Policy Database – Access Path obtained via local control file

3. In the context of the local Functional Domain (FD), relevant event Policy attributes are modified according to the result of *fi*(*fj*(*func.domain, event profile*), *local policy*)

4. Abstraction Class requests MetaData for information pertaining to required function/service

5. Parallel access process involving updating the MetaData database due to the dynamic nature of the cloud structure

6. The local control policy for the network node/ Virtual Machine hosting the Abstraction Class is accessed

7. Direct Access to the set of Abstraction Classes on Accessed Node

8. Target Evaluation Process by the invoked Abstraction Class(es) for the required service(s)

As shown in Figure 2, it is envisaged that fundamental large-scale entities, such as a cloud, should be described in terms of fundamental layers of abstraction as applied to Cloud Design. Currently, servers within a virtualised environment are described as Virtual Machines (VM's) whilst those within a conventional physical network are described as Physical Servers, both being members of the "Compute_Server" class as addressed herein. Within a cloud either type of server may exist and therefore classes of operation that are enacted upon them must either be able to deal with both types or exist as separate classes. Additionally, such operational classes must be able to deal with the fact that such servers possessing the same state may not mean that such operational classes react in the same manner. Consequently, we propose an "Operational_ Server" class which has a state-based attribute, and in turn a parent "Conceptual_Server" class to which many operational and system processes may be applied. The respective classes of events are derived summarily from their respective server classes in this context.

This event-processing model must be complemented by having the event protocol design integrated with a management layer that can be part of the overall service location system. The beginning of this integration layer is shown in Figure 1 in which there is an interaction between the Profile generation process utilised for the virtualised event produced in the Initiator process and the Policy accessed by the latter process. The Profile and Policy data and the information pertaining to their integration are contained within the MetaData/XML (see Figure 3) database information. This is located on nodes based on VM's containing XML-based systems within the cloud. The location of the VM nodes is maintained within the replicated central SQL database within the cloud. At this point such events have been produced from either Physical Servers or VM's and are referred to as the set of "Compute_Events" (Figure 2). In order that they can be transmitted within a cloud, these are transformed to a consolidated virtualised or abstracted event, referred to in Figure 2 as a "Conceptual_ Event". Specifically,

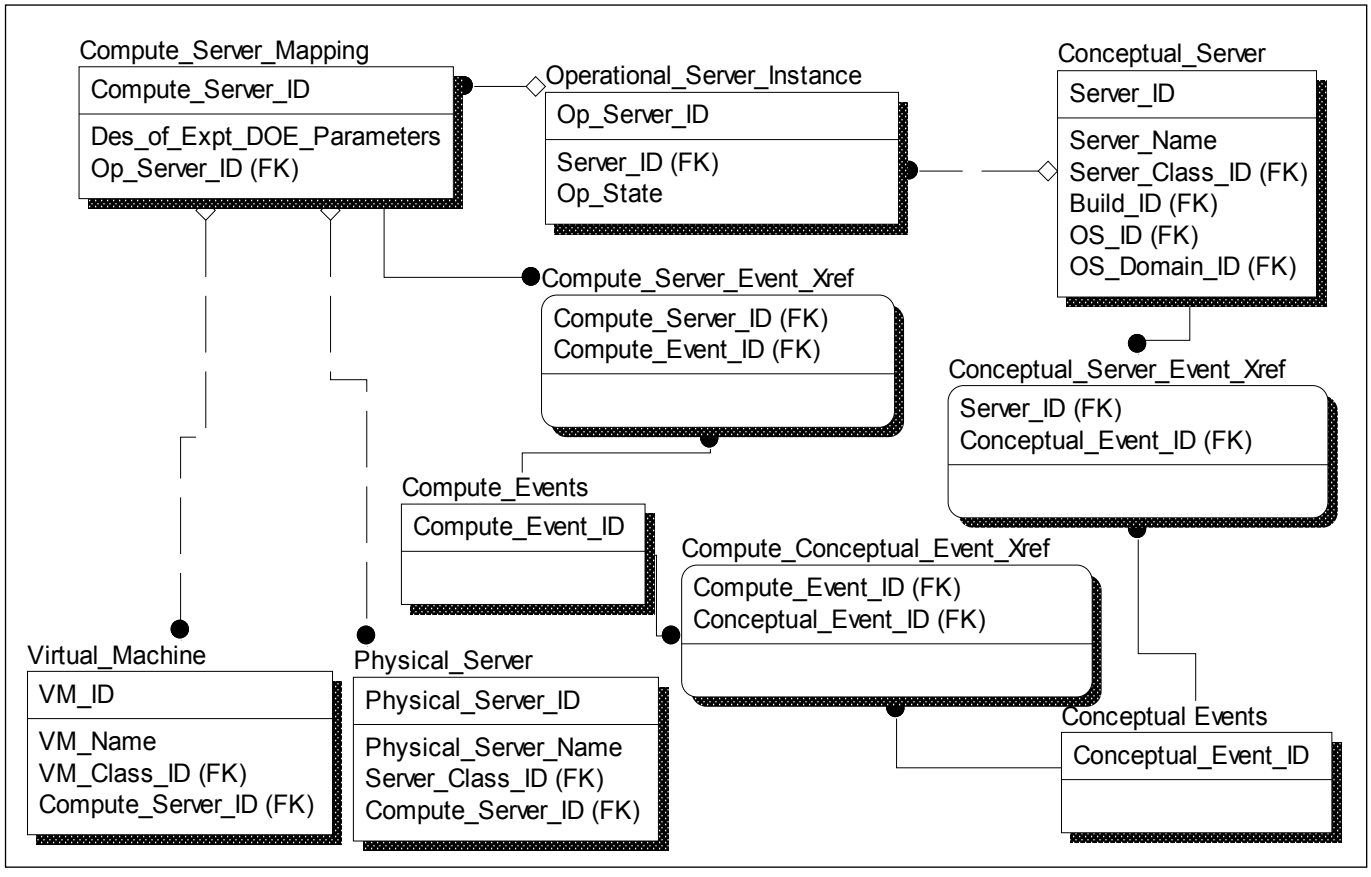

Fig. 2. An ER subschema of the overall cloud model that describes the conceptual-operational relationship (OS and DOE stand for Operating System and Design of Experiment, respectively)

```
Conceptual_Events := {Cloud_Events | Conceptual_Events ☐ {Cloud_Events}
                                   ☐ ( Compute_Events ☐ {Cloud_Events}
                                   ☐ ( Compute_Events ☐ {Conceptual_Events}) ) }

        ⇨    Max( Sort({Conceptual_Events}) ) → {Event_Seq[ Index ]} | Index = 1
```

Below follows a small sample of the coding pertinent to the class of Conceptual_Event :

```
        Event_Seq[ 1 ] = Compute_Corp_Event_In$
        Event_Seq[ 2 ] = Compute_Mgt_Event_In$
        Event_Seq[ 3 ] = Compute_System_Event_In$

Select
        Case MemberOf( Compute_Corp_Event_In$, "Service_A$")        ; Corporate users
                call "$syslib\gen_profile(Policy_A$, Service_A$)"
                call "$syslib\gen_policy(Profile_A$, Service_A$)"
                call "$syslib\gen_concept_event(Profile_A$, Policy_A$, Service_A$, Conceptual_CorpA$)"

        Case MemberOf( Compute_Mgt_Event_In$, "Service_B$")         ; Mgt users
                call "$syslib\gen_profile(Policy_B$, Service_B$)"
                call "$syslib\gen_policy(Profile_B$, Service_B$)"
                call "$syslib\gen_concept_event(Profile_B$, Policy_B$, Service_$, Conceptual_CorpB$)"

    Case MemberOf( Compute_System_Event$_In, "Service_C$")          ; System users
                call "$syslib\gen_profile(Policy_C$, Service_C$)"
                call "$syslib\gen_policy(Profile_C$, Service_C$)"
                call "$syslib\gen_concept_event(Profile_C$, Policy_C$, Service_C$, Conceptual_CorpC$)"
    Case 1                                                          ; catchall
        $msgstr = "No Services for Cloud Domain Group Membership."
        call "$syslib\msgdisp.scr"
EndSelect
```

The above script in Kix2010 [7] is an example of the class of a Conceptual_Event (e.g. Conceptual_CorpA$) being derived in response to the received Compute_Event class Compute_Corp_Event_In$. This is complemented by the evaluation of the Policy (e.g. Policy_A$) and Profile (e.g. Profile_A$) in response to the event class being a member of a specific Compute_Event subclass (e.g. Service_A$). The metadata is accessed so as to derive where classes of input Compute_Events occur most often, and therefore the scripts and KBS clauses [8] are dynamically reconstructed, so that the hierarchy will intercept the required goal (input event) in a minimum number of steps. This helps to reduce both processing and traffic in cloud-based Abstraction Classes.

Each input event trigger to a cloud has a unique reference associated with it in keeping with SNMP event traps [5][6]. This reference includes an Enterprise ID as well as an event code for the trap, specific to the external device which is the source of the event. More events of this type will be present on the cloud-based network as more devices or software respond to a particular state change. These events are referred to, in Figure 2, as "Compute_Events". Therefore one of the main functions of the Initiator Process Node event and target search protocol, as exhibited in Figure 3, is to present the idea of a derived "Conceptual_Event" (Figure 2). This is intended to reduce the overall number of events occurring within the cloud and therefore the amount of overall traffic. A full testing program for this is now being specified in a proof-of-concept environment.

A change of state is produced due to a change of the condition of an operation. Each external change of state can be equated with a specific class of "Compute_Event" (Figure 1). Each event class can be equated using a standard time base with an event instance object. Therefore, if a set of such events occurs within a certain time threshold, then these events may form a set of events associated with the same change of state. Therefore each "Conceptual_Event" indicates a set of "Compute_Events" that are likely to be associated with the same system problem.

Each event class can be associated with a unique numeric identifier. If these identifiers ({prime numbers} in event protocol) for events with such a common timestamp are added together then a unique value is derived. The total value derived should then be cross-referenced with the library of system events totals. Thus, by looking at the time base, this total gives the sequence of events that constitute the sequence of Compute_Event objects in the entire Conceptual Event instance. This helps the diagnostics with respect to complex service requests from Abstraction Classes.

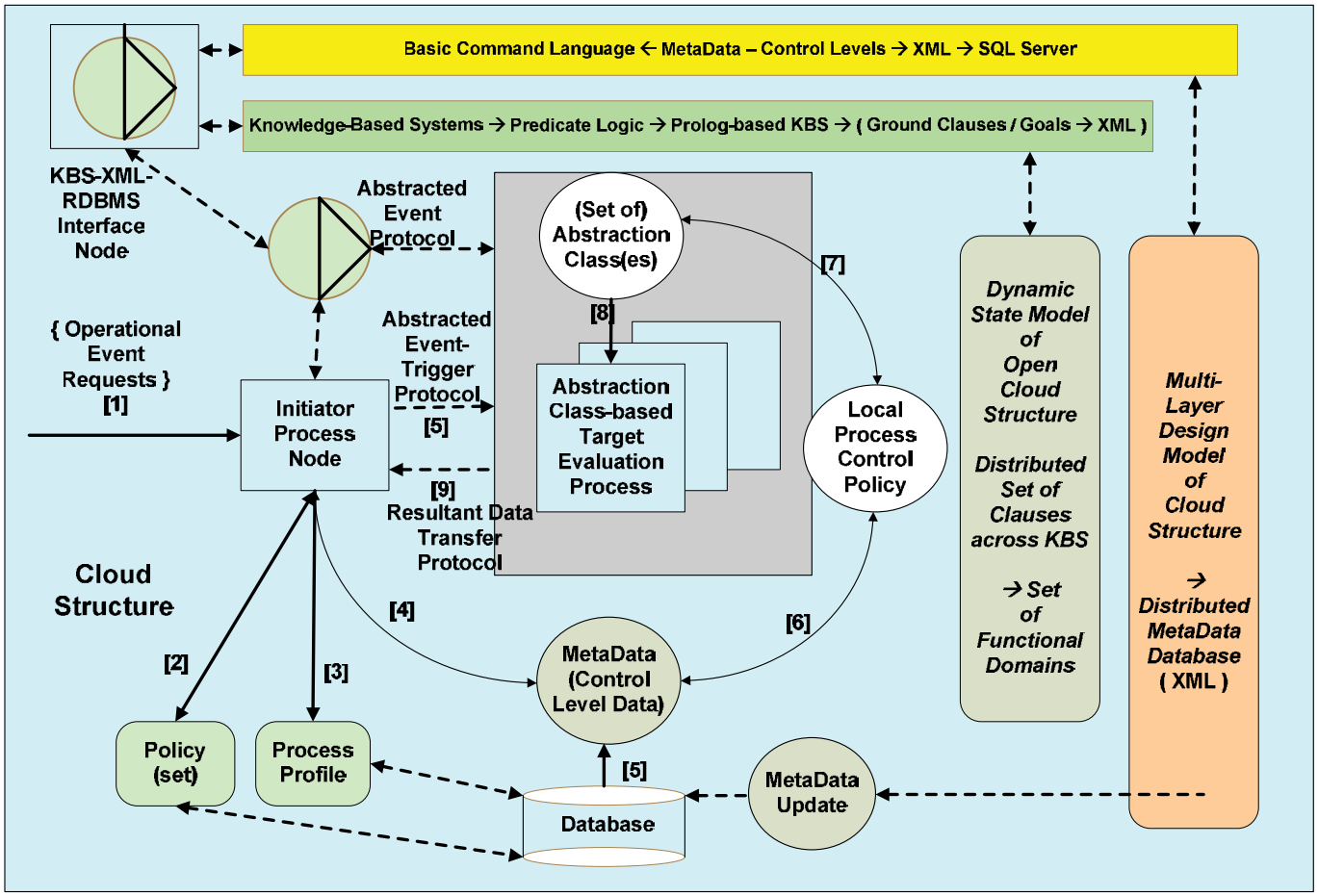

Fig. 3. The inclusion of the current event-processing protocol within the framework of the distributed KBS and a distributed metadata architecture, with XML data transfer [6] between them via sets of distributed IPM's on VM's

3. EXTENSION OF THE EVENT PROTOCOL SCOPE

Figure 4 shows how the event protocol described in this paper may fit into (the basis for) an Open Systems Framework [10], by presenting a standard model to enable the control of the operation of an abstraction class-based process depending on the state of the system (cloud). This is presented in Figure 4 as the "Process Controller". This utility is located in a cloud, and will be accessed through the use of Abstraction Classes. This "Process Controller" module will integrate with the event protocol used to enable and control the abstraction classes in relation to the state of the system (cloud). This may in turn dynamically control the state of an accessed process, following the principles of adaptive control referred to earlier. For example, concerning the state of an aircraft (primary cloud) with respect to a secondary cloud (on-board client/ client Abstraction Classes):

1. Parked : Initiator / Abs.Classes ON
2. Taxiing : Initiator / Abs.Classes OFF
3. Take-off : Initiator / Abs.Classes OFF
4. Flight : Initiator / Abs.Classes ON
5. Landing : Initiator / Abs.Classes OFF
4. Future Work

The most immediate area of future work is to focus in showing the complete distributed method and associated sets of algorithms outlined herein. These are involved in the linkage from the event protocol interpretation, the event package contents and the translation of the request to the required service call. This request is translated to a KBS goal and the result stored both as a set of distributed KBS clauses and as a set of equivalent distributed XML files. The solution to this stage of the problem is to integrate all stages of the solution in a seamless manner. The result of this will then be integrated in cloud management systems, taking into account the recent DMTF work [9].

5. CONCLUSION

This paper presents the basis for event-based interaction with Abstraction Classes with a view to this becoming an initial stage towards a command and control layer of an operational system for a cloud. It is shown that such an event-based protocol has the potential for extension towards integration with control systems from other functional domains or other clouds. The natural extension of this is towards management via techniques and designs utilising Open Systems Frameworks [9][10].

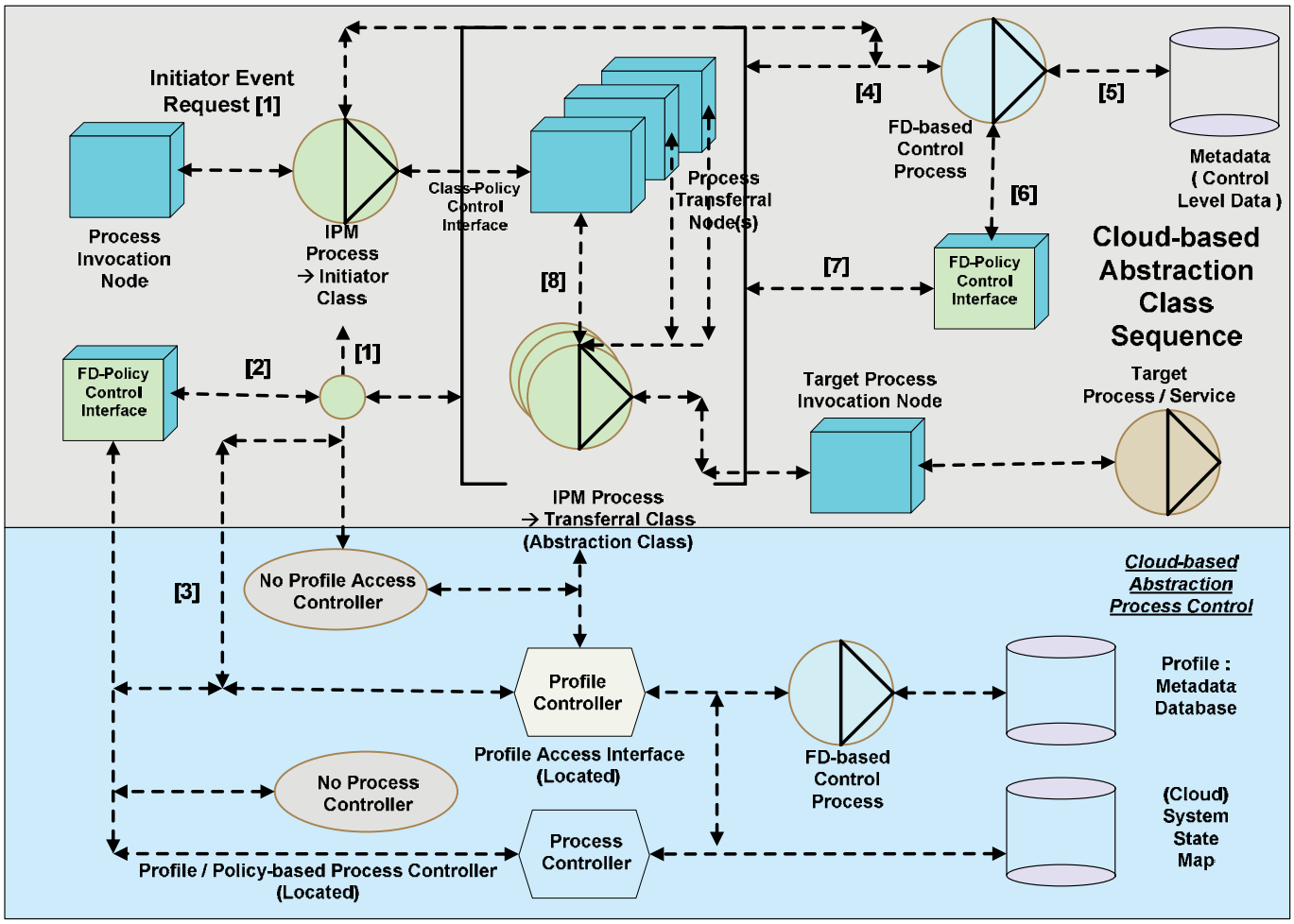

Fig. 4. To show how the event processing protocol which, by virtue of being a standard construct, may lead to extended degrees of functionality

REFERENCES

[1] J.Eccles, G.Loizou, 2010 A Cloud-Computing Environment Based on a Model of Integrated Abstraction Classes. Annual International Conference on Cloud Computing and Virtualization (CCV 2010), pp.153-162

[2] B.Solomon, D.Ionescu, M.Litoiu, M.Mihaescu, 2007 A Real-Time Adaptive Control of Autonomic Environments. IBM Centre for Advanced Studies, Toronto, pp.1-13

[3] R.Harbird, S.Hailes, C.Mascolo, 2004 Adaptive Resource Discovery for Ubiqiutous Computing. ACM 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing, Toronto, Canada, pp.155-160

[4] H.Kang, K.Wang, 2010 Large-Scale Bot Detection for Search Engines. ACM Proceedings of the 19th International Conference on World Wide Web (WWW 2010)

[5] J.Yoon, H.Ju, J.Hong, 2003 Development of SNMP-XML Translator and Gateway for XML-based Integrated Network Management. Int. J.of Network.Mgmt, 13, pp.259-276

[6] C.Tsai, R.Chang, 1998 SNMP through WWW. Int.J.of Network Mgmt, 8, pp.104-119

[7] R.Van Velsen, 2010 Kix2010 v4.50. Microsoft Netherlands, www.kixtart.org

[8] I.Bratko, 1986 Prolog Programming for Artificial Intelligence. Addison Wesley

[9] I.Loy, F.Galan, I.Loy, A.Sampaio, V.Gill, L.Rodero-Merino, 2009 Service Specification in Cloud Environments Based on Extensions to Open Standards. ACM Communication System Software and Middleware (COMSWARE 09), Dublin, Ireland

[10] I.Traore, D.B.Aredo, H.Ye, 2003 An Integrated Framework for Forma1 Deve1opment of Open Distributed Systems. ACM Symposium on Applied Computing (SAC2003), pp.1078-1085

Photo Not Available

George Loizou is currently Emeritus Professor of the Mathematics of Computation. He obtained a first class honours degree in Mathematics, a Postgraduate Diploma in Numerical Analysis and his PhD in Computation Methods. He serves on the editorial board of AMCT, and is the Editor-in-Chief of Section A of the International Journal of Computer Mathematics.

Jonathan Eccles has more than 20 years of experience in the integration and design of network operating systems and programming languages incorporating a variety of different technologies within EDS and lately HP. This is complemented by a wide interest in analysis and design methodologies in the area of systems engineering. This has been enhanced through academic research in order to produce new ideas in the field of systems engineering. He is currently a Technical Architect / Design Engineer applying Virtualisation to Enterprise Systems for HP. He is a member of the Institute of Electrical & Electronic Engineers (IEEE) and the Association of Computing Machinery (ACM). Additionally he has been named a Distinguished Systems Engineer (SE) at HP Enterprises.