RESOURCES AND SERVICES VIRTUALIZATION WITHOUT BARRIERS

Deliverable White paper
RESERVOIR VMI White Paper
Release 1.0

The research leading to these results is partially supported by the European Community's Seventh Framework Programme ([FP7/2001-2013]) under grant agreement n° 215605.

Project Number	:	215605
Project Title	:	RESERVOIR : Resources and Services Virtualization without Barriers
Deliverable Type	:	White Paper

Deliverable Number	:	White paper
Title of Deliverable	:	RESERVOIR VMI White Paper
Nature of Deliverable	:	White Paper
Dissemination Level	:	Public
Internal Document Number	:	N/A
Contractual Delivery Date	:	May 12, 2009
Actual Delivery Date	:	May 12, 2009
Contributing WPs	:	WP3
Editor(s)	:	Lars Larsson

Abstract

This document is shared with the Open Cloud Computing Interface (OCCI) working group at the Open Grid Forum (OGF) as a white paper on the Virtual Execution Environment Manager Interfaces (VMI) as part of the ongoing collaboration between the OCCI and the RESERVOIR project.

Keyword List

virtualization, management interfaces, cloud computing

Contributors

Name	Organization	Sections
Elmroth, Erik	Umeå University	2
Henriksson, Daniel	Umeå University	2
Larsson, Lars	Umeå University	1, 2
Llorente, Ignacio M.	Universidad Complutense de Madrid	2
Montero, Ruben S.	Universidad Complutense de Madrid	2
Rochwerger, Benny	IBM	2
Rodero, Luis	Telefónica Investigación y Desarrollo	2
Sampaio, Americo	SAP	2
Tordsson, Johan	Umeå University	2
Vaquero, Luis M.	Telefónica Investigación y Desarrollo	2

Document History

Version	Date	Comment
1.0	12/5/2009	Made changes in accordance with comments and published on OCCI wiki page.
0.9	28/4/2009	Added Introduction chapter and modified the technical description contents from SVN slightly.

1 Introduction

A RESERVOIR cloud is an elastic infrastructure for Internet-scale datacenters, and offers Infrastructure as a Service (IaaS) to service providers. The long-term goal of the project is to enable federations of such clouds to be formed, and thus offering a market place for Infrastructure as a Service (IaaS) among cloud infrastructure providers. The VEE Manager Interface (VMI) is the interface used for the communication between the Service Manager (SM) and the Virtual Execution Environment Manager (VEEM), as well as between VEEMs at different sites within the RESERVOIR architecture. Such cross-site communication and collaboration is used for creating a federation of cloud sites.

The RESERVOIR architecture, as defined in the RESERVOIR High Level Architectural Specification (Internal document D1.1.1) and described in [2], states that management of services and the Virtual Execution Environments (VEEs) that implement them is divided among two components: the Service Manager (SM) and the VEE Manager (VEEM). A service is comprised of any number of service components, and each service component is implemented by VEEs. The SM is contacted by Service Providers (SPs) and offers the external interface for management of services, as well as functionality such as accounting and management of service elasticity (dynamic rescaling of service capacity allocation according to demand). The SM is, however, unaware of the actual implementation details, such as the exact location of the VEEs that comprise the service. Such lower-level details, e.g. placement, are the responsibility of the VEEM. Figure 1.1 shows the major components and the interfaces between them in the RESERVOIR architecture.

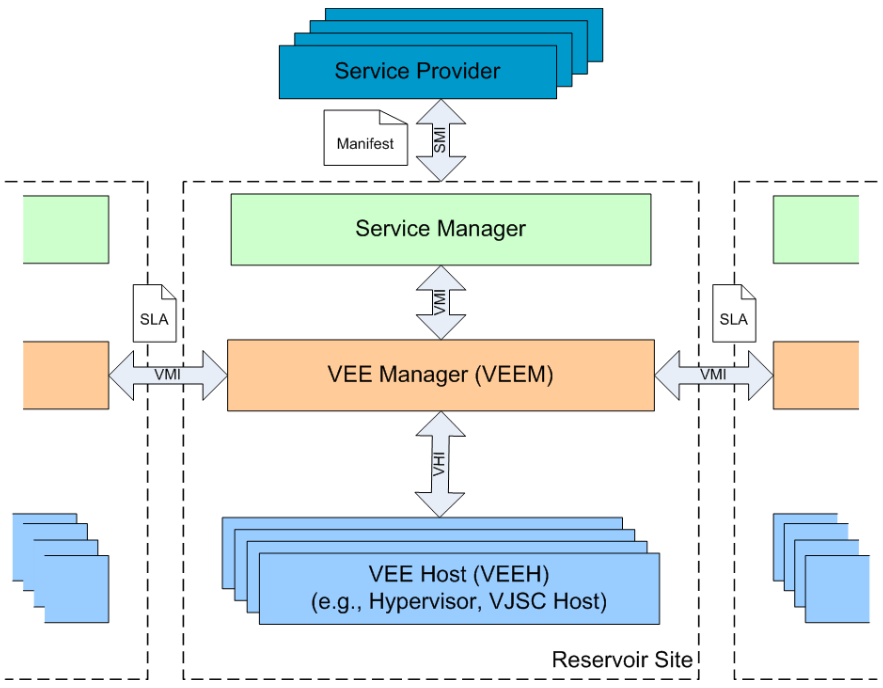

Figure 1.1: The major components of the RESERVOIR architecture and the interfaces that connect them [2].

The VMI is comprised of a number of interfaces: (a) submission; (b) control; (c) accounting; and (d) monitoring. The interfaces are developed to support communication in a RESTful [1] fashion. These interfaces expose the functionality needed to dynamically deploy, control, and provide accounting information on Virtual Execution Environments (VEEs). These interfaces will also in a future version provide the functionality needed to perform operations on VEEs across RESERVOIR sites, transparently to the Service Manager (SM) [3].

As stated previously, one of the long-term goals of RESERVOIR is to provide infrastructure solutions for federated cloud environments. The VMI will then be used also across sites, where the VEEM at one site acts as an SM for another site for the management of the VEEs that will be placed at the remote site [3]. However, for the 2008 end-of-year demo (henceforth, the term *first year* is used to describe the work carried out within the project in 2008), only a single site is supported and therefore there is neither inter-site communication nor inter-site placement of VEEs. The first year prototype handles only a single site with regard to communication (from SM to VEEM) and intra-site placement and migration.

In summary, the VMI provides a simple and minimalistic interface for management of the VEEs that comprise a service within a RESERVOIR site. Further development of the interface will add support for federations of RESERVOIR clouds.

2 Technical Description

This chapter describes the collection of interfaces for the Service Manager (SM) to interact with the Virtual Execution Environment Manager (VEEM). These interfaces expose the functionality needed to dynamically deploy, control, and provide accounting information on Virtual Execution Environments (VEEs). In a future iteration, these interfaces will also provide the functionality needed to perform operations on VEEs across RESERVOIR sites, transparently to the SM.

As stated in the RESERVOIR High Level Architectural Specification (Internal document D1.1.1), the VEEM is responsible for efficient and reliable management of VEEs (Section 6.1.2.2). Thus, the separation of concerns is clear: the SM manages the service as a whole with regard to elasticity of capacity allocation, accounting, etc, whereas the VEEM manages the service components on the VEE level including such issues as where VEEs are deployed in accordance with the site policy for VEE placement. This offers a layer of abstraction to the SM as it does not need to be concerned with low-level issues such as VEE placement. Issues of this nature are the sole responsibility of the VEEM.

2.1 Relationship with other Components

The SM and the VEEM cooperate, but have different concerns. In the context of VMI, we note the following:

- Only the SM has the concept of services. This includes services, service components, and service component groups as defined in Section 2.1 in the RESERVOIR High Level Architectural Specification (Internal document D1.1.1). The VEEM is only aware of the VEEs, it is responsible for (local and remote placement), as well as their state and placement.
- The mapping of service components (and groups thereof) to VEEs is the sole responsibility of the SM.
- The SM is responsible for translating the Service Manifest, which contains the description and defintion of the service and its components, into VMI primitives to be communicated to the VEEM.
- For this first year prototype, there will be no concept, on the VEEM level, of a VEE group. The SM maintains mappings of service components to VEEs, and is responsible for issuing the VMI method call to affect the appropriate set of VEEs. Collections of VEEs do not imply "grouping" in any sense more permanent than for the context and duration of the method call.
- VEEMs may communicate to support federation and migration of VEEs. They do so using the VMI, where the primary VEEM issues VMI method calls in a SM-like fashion.

2.2 Use Cases and Scenarios

The following basic use cases have been devised to aid the development and guide the thoughts as the interface is evaluated before implementation. These use cases are aligned with the use cases that have been defined for the RESERVOIR project (Internal document D1.1.1). Due to the year 1 restrictions of supporting only a single RESERVOIR site, the use cases are all assumed to execute on a single site. It can also be assumed that the SM has parsed the Service Manifest correctly, created a refined data model that can be mapped to a Service Deployment Descriptor (SDD) and will issue the appropriate VMI commands to the VEEM. For each use case or scenario, it should be discussed what the correct VMI commands are, and in what order they should be issued.

2.2.1 Simple Service Deployment

Once the SM has created an absolutely refined data model, it states that a single service component should be started (very “small” service, runs in a single VEE).

2.2.2 Service Redeployment

The service described in the previous section is running at the site. The service has gained in popularity and the authors have prepared two new images that they want to replace the first one with (now, the database is to run in a separate service component). This should be performed with as little downtime as possible, and once the service goes offline, it MUST be replaced within few seconds (according to the business agreement). The components should run in a virtual local area network, making the database server accessible only from the other service component, not the Internet.

2.2.3 Service Component Replication

The service from the previous section has grown even more popular, and the database service component must be replicated (load balancing). The software running in the actual VEE is assumed to handle transferring the data to the replica once it is online. The replica must run the same master image as the original but may not run on the same physical host.

2.2.4 Service Shutdown

The service from the previous section has been compromised, and the customer has required it to be shutdown immediately. The (virtual) hard drive contents must be saved so it can be sent to the owners for analysis and damage estimation.

2.2.5 Service Restart

The contents of the hard drive from the previous section have been analyzed, the security holes have been patched, and a new version shall be uploaded and then be put into use immediately.

2.2.6 Update VEEs

The allotted RAM for the database service component VEEs from the previous use case is too low. The maximum size must be set to 8GB for both the original and the replica (preferably) without restarting the component and disrupting the service.

2.2.7 Stopping a Replicated VEE

Due to a drop in demand, the database service component no longer needs an active replica. The customers require that it be shut down to minimize costs, but in such a way that it can easily be restarted should the need arise.

2.2.8 Restarting a Stopped VEE

The need for restarting the service component from the previous section arose. Assume that once it goes online again, it will automatically synchronize with the original server. Again, it must run on a different physical host than the original.

VMI VEE State	VEEM VEE State
INITIAL	PENDING,HOLD
DEFINED	STOPPED
ACTIVE	ACTIVE
PAUSED	HOLD
SUSPENDED	SUSPENDED
DESTROYED	DONE,FAILED

Table 2.1: The VEE states from the VMI and VEEM point of view.

2.2.9 Removing All Data Related to a Service

The service is no longer required, and the service provider is no longer a customer. All data related to the service must be removed to reclaim resources (e.g. storage).

2.3 Related Interfaces and Protocols

VEEM operations will typically involve the interaction of entities at the Internet level. In order to perform its normal operation in this case VEEM assumes a set of protocols, namely:

- File transfer protocols to perform VEE image files operations, mainly remote copy.
- Security protocols to guarantee client authentication and secure communication messages.
- Dynamic Management of the VEE networks that may include network tunnels, dynamic DNS, etc.

The interfaces to provide such functionality are not part of the VMI. Defining and handling virtual area networks are part of the VMI, but any higher-level management of these networks must be conducted at the application level.

2.4 VEE States

From the VMI perspective, a VEE can be in one of the states presented in Table 2.1. The states are mapped to the ones in DMTF DSP2013 [1]

It should be noted that the states in Table 2.1 are only from the VMI perspective. The full set of states that a VEE may be in on the VEEH level is larger. These states are sufficient for expressing the required level of detail with regard to operations of the SM and VEEM.

The transitions are presented on page 25 of DMTF DSP2013. In VMI terms, these transitions are called ControlActions. Section 2.5.4 contains the specification of the ControlAction data type.

2.5 Data Types

2.5.1 VEE Accounting Data Type

All VEE Accounting messages are passed using the Accounting interface methods and are represented by messages conforming to some specification that will be developed by work packages dealing with accounting and measurements. Accordance with this specification will ensure that the measurement data contains all the

[1] www.dmtf.org/standards/published_documents/DSP2013_1.0.0.pdf

relevant information, and will be easy to parse. To ensure location unawareness in the SM, and to separate concerns, some accounting and monitoring information is passed using the methods of the VMI.

In the first year prototype, the measurements will be aggregated at (customizable) specified intervals and sent via the VMI to the interested listeners: the primary VEEM in the case of measurements coming from a remote VEEM and the local SM in the case of the local VEEM reporting to the SM.

2.5.2 VEE Descriptor Data Type

The VEE Descriptor data type (see Table 2.2 for more details) includes all information required by the VEEM, and remote VEEMs, to allocate resources and perform placement decisions. This information includes the specification of the virtual hardware (e.g. CPU and memory) as well as information on the networks the VEE should be part of.

A VEE component may use configuration scripts that require configuration information from the Service Manager (such as the IP from some other VEE, which it is not known before deployment), this is given through the ovf-env.xml file. This file is built by the Service Manager as defined in the Service Manifest, and included in one image prepared by the Service Manager to be mounted on one of the disks of the VEE replica. The parameters to the call to the VMI will be adapted accordingly so that disk is included along with the ones specified by the Manifest. The boot process of the VEE replica will mount that disk on a path that is known by the configuration scripts.

To support "marking" VEEs in a general and semantic-free way, VEE instances may be given a set of tags. These can be used by the SM to mark the type of the VEE (tagging it and others as "database") as well as provide hints for placement. Tags have been added to the VEE Descriptor data type primarily because they may be used as an extensible way of expressing meta-information on VEEs. Tags also provide designers with hints on future additions to the VEE Descriptor data type. If a certain tag would be required for all VEEs, then that kind of information may be turned into an attribute in a future design of the VEE Descriptor data type.

Description	**VEE Description Data**	**OVF Data**
Name	Name	ovf:id `<VirtualSystem>` attribute suffixed by a string that identifies univocally the replica (e.g. a numerical ID)
Amount of requested	CPU	CPU item in `<VirtualHardwareSection>`
Number of requested CPUs	CPU	`<rasd:VirtualQuantity>` in CPU item in `<VirtualHardwareSection>`

Amount of requested memory	memory	memory item in `<rasd:AllocationUnits>` plus `<rasd:VirtualQuantity>` combination in memory item in `<VirtualHardwareSection>`
Disk source	source	ovf:href `<File>` attribute
Disk size in GB	size	ovf:capacity `<Disk>` attribute
Disk target (device to map disk and used as root)	target	`<rasd:HostResource>` in the disk `<Item>` in `<VirtualHardwareSection>`
Disk clone	clone	None.
Device name to map interface	NIC target	`<rasd:Connection>` in the NIC `<Item>` in `<VirtualHardwareSection>`
Network name	Network	ovf:name `<Network>` attribute
Hypervisor type	RAW Type	`<vssd:VirtualSystemType>`

Semantic-free Tag	Tag	-
Raw data	Raw data	Raw Data

Table 2.2: VEE Description Data Type and Correspondence with OVF Data.

2.5.3 Initialization Type Enumeration

Initialization of service component groups may be conducted either according to the *best effort* or the *two-phase* submission procedure. The definitions, as per the architecture document (Internal document D1.1.1), are (slightly paraphrased and updated to match the DTMF-compatible set of states for VEEs):

- TWOPHASE, initialization is performed in two phases. First, the VEEM initializes internal data structures. Then, in the second phase, the VEEM looks for a feasible deployment for the new service. Both phases must be successfully completed for all service components, otherwise an error is returned.
- BESTEFFORT, only the service's internal data structures are initialized and the corresponding result is returned. All the associated VEEs will remain in the INITIAL state until there are enough resources to deploy the VEEs.

The difference between two-phase and best effort initialization is that the calling component may not expect the VEEs to be placed and run neither immediately nor at all. Such VEEs may be run if the VEEM can create a beneficial placement strategy for the VEEs.

The VEEM placement engine runs continuously in the background, and may optimize placement during such background runs. If the placement engine detects that it could run a VEE submitted via best effort submission, it will attempt to do so.

When the best effort initialization returns successfully, the caller is guaranteed that the VEEs that were passed as parameters are in the INITIAL state. The monitoring framework should deliver information on when the VEE has been placed so the SM can activate it.

For the first year prototype, only the *best effort* initialization model will be implemented. This will be sufficient, because we are making the assumption that there will always be enough capacity at the VEEM.

2.5.4 ControlAction Enumeration

The VEEControlAction enumeration specifies the type of operation to be performed on a given VEE, and it is used as an input parameter to the control methods provided by the VMI interface:

- ACTIVATE, which puts a VEE into the ACTIVE state;
- PAUSE, which puts a VEE in the ACTIVE state into the PAUSED state;
- SUSPEND, which puts a VEE in either the ACTIVE or PAUSED state into the SUSPENDED state;
- SHUTDOWN, which puts a VEE in either the ACTIVE, PAUSED, or SUSPENDED state into the DEFINED state; and
- DESTROY, which puts a VEE in any state into the DESTROYED state.

See Section 2.4 for further information on the states of VEEs and the transitions between these states.

2.5.5 Monitoring Event Type Enumeration

The monitoring Events that can flow up to the SM from the VEEM are listed below:

- Hardware VEE Data
- Probe Data
- Agent Data

2.6 Semantic-Free Collections of VEEs

In order to separate concerns between the SM and the VEEM and keep the VEEM unaware of the concept of services as well as service component groups, the VMI methods shall for the Y1 prototype accept a semantic-free collection of VEE identifiers as a parameter. This design must be evaluated for the future prototypes.

Collections of VEEs are free from semantics, meaning that membership of a collection for a method call does not indicate any type of relationship between the referenced VEEs. They may even belong to different unrelated services.

The only semantics provided by this type of method call is with regard to order. Order is determined by the type of the collection: a list (ordered) or a set (unordered). The list type implies that the VEEs must be handled in sequential order (e.g. for issuing a command to start up the virtual machines in a certain order as specified in the Service Manifest), whereas VEEs in a set may be dealt with in arbitrary order.

The collection-based methods do not guarantee atomicity: the system permits that one or more of the referenced VEEs cannot be manipulated in the way that the method specifies. Atomicity, should it be required in some service, must be handled by the service provider and is outside the scope of RESERVOIR.

2.7 Interfaces

According to all the previous sections, the proposed interfaces for this first year are as follows. For all the interfaces, it is assumed that the involved parties (SM and VEEM or a pair of VEEMs) have authenticated each other in some secure fashion, and their identity is known and can be implied (no need to send identifiers for each call).

2.7.1 VEE Submission Interface

2.7.1.1 InitializeVEEs

The InitializeVEEs method is used to define VEEs and in particular make them known to the VEEM so that they can be referred to by a VEE identifier in future calls.

Parameters:

- VEE description as specified by a VEE Descriptor data type (c.f. Section 2.5.2)
- Initialization model, either TWOPHASE or BESTEFFORT, as described in Section 2.5.3. *Note that for the first year, this value may only be* BESTEFFORT.

Return values:

- Set of VEE identifiers. The items in the set are guaranteed to be unique identifiers that have never been used before.

- A set of VEE initialization values. This value is a VEE Descriptor data type that contains any values that may have been generated in the process by the VEEM. For instance, the public IP address of the VEE (should one be required).
- Success or failure. Success indicates that the VEE identifiers have been defined to be instances of the VEE as described by the VEE Descriptor data type. Success guarantees that the VEEs are in the DEFINED state (as in Section 2.4), and can be referenced using the VEE identifiers returned. Failure indicates that the VEEs could not be defined as specified in the VEE description. When a two-phase initialization is specified a failure could mean that the VEEM does not have enough resources to satisfied the initialization request.

Errors:

- An error is raised if the VEE description is malformed.
- Not enough resources for the initialization request (only for the TWOPHASE initialization model)
- Access denied, not enough privileges to perform the action

2.7.1.2 SubmitVEEs

This method is used to submit a collection of VEEs. The VEEM will deploy and activate the VEEs.

Parameters:

- Set of VEE identifiers. The items in the set must be VEE identifiers that have been returned by a previous call to the InitializeVEEs method.

Return values:

- Success or failure. Success indicates that the VEE is being deployed on a VEEH (if a TWOPAHSE initialization was used) or will be deployed in the future (if a BESTEFFORT initialization was used).

Errors:

- Wrong state, if the VEE is not in the DEFINED state.
- Access denied, not enough privileges to perform the action (e.g. trying to submit a VEE initialized by other user).

2.7.2 VEE Control Interface

2.7.2.1 UpdateState

This method updates the state of a collection of VEEs.

Parameters:

- Collection of VEE IDs. The IDs of the VEEs that should be affected by the update. If the collection is a list, the VEEM guarantees that the update to state is applied in the list order. If the collection is a set, the VEEM may issue the state updates in arbitrary order.
- VEE Control Action. As specified in Section 2.5.4

Return values:

- Set of VEE IDs, where membership indicates successful application of the state update (state updates may fail due to some transitions being undefined, e.g. it is impossible to go from the DEFINED state to the SUSPENDED state without having entered the ACTIVE state first).

Errors:

- None.

2.7.3 VEE Accounting Interface

Using the producer/consumer pattern, the VEE Accounting interface is used by a VEEM to send information back to the primary VEEM or SM about the state of VEEs. Whether Monitoring information should also be sent this way is still (at time of writing) a topic of discussion in the mailing lists. Should this information pass through the VEEM, we can consider both to be conceptually equal — the only difference being in the exact information that is transmitted (CPU/Memory usage vs number of used CPU/memory units).

2.7.3.1 ReceiveAccountingInformation

This method is required by an interface that all listeners for accounting information have to implement.
Parameters:

- Accounting information

Return values:

- None.

Errors:

- None.

2.7.3.2 RegisterAccountingInformationListener

The method is used to register as a listener for accounting information. A component may only receive information about a VEE at one specified interval: repeated calls to this method for a given VEE will change the interval rather than add a new one (i.e. if the interval for VEE A was 300000ms, a new call to the method with a parameter of 600000ms will update the interval to 600000ms).
Parameters:

- A set of VEE identifiers. The caller will be sent accounting information concerning the identified VEEs.
- An interval specification in milliseconds. The caller will be notified by the receiver at the specified intervals. This monitoring information is sent at exactly these intervals (may not be received at these intervals due to network latency, the system is still asynchronous), even if there is no accounting information available.

Return values:

- None.

Errors:

- The caller is not authorized to receive accounting information for the specified VEEs.
- The VEEs are not known to the system.

2.7.3.3 DeregisterAccountingInformationListener

The method is used to de-register as a listener for accounting information.

Parameters:

- Collection of VEEs identifiers. The caller will no longer receive information concerning the identified VEEs. If the caller is not a listener to the specified VEE, this call has no effect.

Return values:

- None.

Errors:

- Raise an error if at least one of the VEE identifiers is invalid.

2.7.4 VEE Monitoring Interface

Using the push pattern, the VEE Monitoring interface is used by a VEEM to send information back to the primary VEEM or SM about the state of VEEs as it is produced. The SM monitoring components will be in charge of aggregation or processing the information as needed upon arrival. The information is based on events generated by the VEEs and the VEEHs in which these VEEs run, and is used by the SM to (e.g.) determine SLA violations. Also, information flowing up to the SM from the probes and the Software Agents within the Service Provider images are said to flow through the VMI.

2.7.4.1 MonitoringInformation

Monitoring information is passed through the VMI from one VEEM to another until it can reach the SM. Monitoring information contained as a message conforming to some specified schema. There are three data items all events have (header):

- *Event Type*. Type of event. See Section 2.5.5 for further details.
- t_0. Instant the event happened at (milliseconds UTC format).
- Δt. For monitoring events which carry aggregated data from measurements taken during a certain time interval, this item represents the length of the interval (milliseconds) [2].

Also, depending on the event type, the events will have the following data associated (body of the message):

- **VEE HW Measurements**.

  FQN of the measurement. For example:

  tid.customers.EasyJet.services.Web.serviceComponents.DBServer.replicas.1.mem.fre e

  Measurement value.

- **Agent Monitoring**.

  FQN of the KPI. For example:
  tid.customers.Sub.services.SGE.serviceComponents.master.kpis.queueSize

  Measurement value.

[2]Please note that a mechanism to synchronize all the events across RESERVOIR must be enabled for the second year when an actually federated infrastructure will be available.

- **Probe Monitoring**. Same as agent monitoring.

The body of the message can contain any type of information, time averaged data, single point data or aggregated registers. The SM components accessing this information are assumed to have enough intelligence so as to deal with the information contained in the body.

2.7.4.2 PutMonitoringInformation

This method is required by an interface that the Receiving VMI handler must implement so as to enable all the data sources to put the data as they are produced.

Parameters:

- Monitoring information, see Section 2.7.4.1.

Return values:

- None.

Errors:

- None.

Bibliography

[1] R. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[2] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galán. The RESERVOIR Model and Architecture for Open Federated Cloud Computing. *IBM Journal of Research and Development (accepted)*, 10 2008. [online] `http://www.reservoir-fp7.eu/twiki/pub/Reservoir/PublicationsPage/RESERVOIR_for_ISJ_-_Final.pdf`.

[3] M. B. Yehuda, O. Biran, D. Breitgand, K. Meth, B. Rochwerger, E. Salant, E. Silvera, S. Tal, Y. Wolfsthal, J. Cáceres, J. Hierro, W. Emmerich, A. Galis, L. Edblom, E. Elmroth, D. Henriksson, F. Hernández, J. Tordsson, A. Hohl, E. Levy, A. Sampaio, B. Scheuermann, M. Wusthoff, J. Latanicki, G. Lopez, J. Marin-Frisonroche, A. Dörr, F. Ferstl, S. Beco, F. Pacini, I. Llorente, R. Montero, E. Huedo, P. Massonet, S. Naqvi, G. Dallons, M. Pezzé, A. Puliato, C. Ragusa, M. Scarpa, and S. Muscella. RESERVOIR - an ICT infrastructure for reliable and effective delivery of services as utilities. Technical report, IBM Haifa Research Laboratory, 2 2008. [online] `http://domino.watson.ibm.com/library/CyberDig.nsf/papers/A44F6256BB697FCE852574E10052DDEE`.