Virtual machine (VM) technology allows multiple operating systems each deploying multiple applications to run on a single host. This invention presents an effective method and system for virtual machine migration from a source host to a target host. The method and system concern the migration of both the service VM and the element managing it. State of the migrating VM is preserved so that it can resume its execution on the target host.
FIG. 1 (prior art)
FIG. 2 (prior art)
FIG. 3
FIG. 4
FIG. 5
Migration Policy

Notifies policies of change

VM Managed Element

Policy₁
Policy₂
...  
Policyₙ

FIG. 6
FIG. 7
Dynamically Loaded Application Service

Embott Application Framework

JNI

JVM (J9)

Embedded Operating System

**FIG. 8**
1. Send StartMigration Message to the target and wait for response

2. Timeout?  
   - NO  
   - Migration_Denied response received?  
     - NO  
     - Migration_Permitted response received?  
       - YES  
         - Generate exception and abort migration  
       - NO  
         - Proceed with migration  
   - YES  
     - Stop

**FIG. 9**
Queue events for the VM managed element and the dependent elements

Serialize state for the VM managed element and dependent elements

Send Management State Message to target for migration and wait for response

Timeout?

Is Managed_object_Instantiation_Error response received?

Is Managed_State_Failed response received?

Process management event state

Cleanup, generate exception and abort migration

Stop
Serialize queued events

Send Management_Event_State message to target and wait for response

Timeout?

Is Managed_Object_Instantiation_Error response received?

Is Management_Event_State Failed response received?

Cleanup, generate Migrate VM exception and abort migration

Stop

FIG. 106
1. Send Migration Complete Message and wait for response

2. Is Migration Complete Ack response received?
   - Yes: Destroy local VM managed element
   - No: Generate exception

3. Timeout?
   - Yes: Stop
   - No: Continue

**FIG. 12**
METHOD AND SYSTEM FOR VIRTUAL MACHINE MIGRATION

RELATED APPLICATIONS

[0001] The present patent application claims priority from the Canadian patent application serial number 2,547,047 to Anthony WHITE entitled “MANAGEMENT OF VIRTUAL MACHINES USING MOBILE AUTONOMIC ELEMENTS” filed on May 15, 2006, which is incorporated herein by reference.

FIELD OF INVENTION

[0002] The present invention relates to the management of virtual machines, and particularly to the management of virtual machine migration from one host system to another by using mobile autonomic elements.

BACKGROUND OF THE INVENTION

[0003] The drive to make more effective use of physical resources within an enterprise information technology (IT) infrastructure has led to the introduction of virtual machine technology. Virtual machine (VM) technology allows one or more guest operating systems to run concurrently on one physical device. There are several approaches to providing virtualization technology, the most recent being para-virtualization and native central processing unit (CPU) with basic input/output system (BIOS) or Extensible Firmware Interface (EFI) support. Concurrent with these approaches, the emergence of the management plane has occurred as the means by which hardware, operating system and applications are managed within the service plane.

[0004] One or more virtual machines may be operational on a single host computing system that will be referred to simply as a host system. A VM that may include an operating system with its concurrent applications is often separated from the elements that manage the VMs on the host system. The separation of management and service functionality has a number of distinct advantages that include separation of concerns, management of change and security improvements.

[0005] Finally, delegated management through the paradigm of Autonomic Computing has emerged. Autonomic Computing is a relatively recent field of study that focuses on the ability of computers to self-manage. Autonomic Computing is promoted as the means by which greater independence will be achieved in systems. This incorporates self-diagnosis, self-healing, self-configuration and other independent behaviors, both reactive and proactive. Such system will adapt and learn normal levels of resource usage and predict likely points of failure in the system. Certain benefits of computers that are capable of adapting to their usage environments and recovering from failures without human interaction have been known, including reducing the total cost of ownership of a device and increasing levels of system availability. Repetitive work performed by human administrators is reduced, knowledge of the system’s performance over time is retained, assuming that the machine records or publishes information about the problems it detects and the solutions it applies, and events of significance are detected and handled with more consistency and speed than a human could likely provide. Such autonomic elements are used in the context of this invention for virtual machine management.

[0006] The introduction of virtualization along with management and service plane separation has produced a new important problem. A VM may be required to migrate from one host system to another. Such a migration may be necessary in various situations. These include the increase in load of the system currently hosting the VM, the occurrence of a fault in the host system, and the temporary unavailability of the system for hosting a VM due to routine maintenance. Specifically, if a virtual machine migrates, the associated units of manageability need to move as well, where the problem extends to more than simply moving code.

[0007] The general area of code mobility is well researched. Various environments for the general mobility of software and state have been built. However, there has been no such infrastructure for an autonomic element, which applies specifically to the system management domain where virtual machines are under management. In particular there is no effective mechanism for transferring a VM from one host to another on which it can resume operation seamlessly. Thus there is a need in the industry for an effective method and system for virtual machine migration by using mobile autonomic elements.

SUMMARY OF THE INVENTION

[0008] Therefore there is an object of the present invention to provide a method and system for the management of virtual machine migration from one host system to another by using mobile autonomic elements.

[0009] According to one aspect of the invention there is provided a method for migrating a service Virtual Machine (VM), comprising a VM managed element including components providing a service, and a VM managing element managing the service VM, from a source host to a target host, the method comprising the steps of:

[0010] (a) queuing events to be processed by the service VM at the source host;

[0011] (b) sending information regarding a current state of the VM managed element from the source host to the target host;

[0012] (c) sending information regarding a state of the queued events from the source host to the target host;

[0013] (d) sending components of the VM managed element that have changed during the execution of step (a) - step (c) from the source host to the target host;

[0014] (e) terminating the service VM and the VM managing element on the source host; and

[0015] (f) resuming execution of the service VM by using the information sent in step (b) - step (d) and resuming the VM managing element on the target host.

[0016] According to another aspect of the invention there is provided a system for migrating a service Virtual Machine (VM), comprising a VM managed element including components providing a service, and a VM managing element managing the service VM, from a source host to a target host, the system comprising:

[0017] (a) means for queuing events to be processed by the service VM at the source host;

[0018] (b) means for sending information regarding a current state of the VM managed element from the source host to the target host;

[0019] (c) means for sending information regarding a state of the queued events from the source host to the target host;
(d) means for sending components of the VM managed element that have changed during the queuing of events to be processed, the sending of information regarding the current state of the VM and the sending of information regarding the state of the queued events;

(e) means for terminating the service VM and the VM managing element on the source host; and

(f) means for resuming execution of the service VM by using the information regarding the current state of the VM managed element, the state of the queued events and the components of the VM managed element that have changed sent from the source host to the destination host and resuming the VM managing element on the target host.

BRIEF DESCRIPTION OF THE DRAWINGS

Further features and advantages of the invention will be apparent from the following description of the embodiment, which is described by way of example only and with reference to the accompanying drawings in which:

FIG. 1 shows an example of an autonomic element to which virtualization infrastructure in accordance with an embodiment of the present invention is suitably applied;

FIG. 2 shows the autonomic element of FIG. 1 that is achieved by separating the autonomic manager from the managed element;

FIG. 3 presents a single management plane containing a single autonomic manager for each service plane under management;

FIG. 4 shows the movement of a service plane from one host to another;

FIG. 5 shows the movement of the autonomic manager from the original host to the host where the service plane now resides;

FIG. 6 shows the interaction between a policy that is involved in the migration of a virtual machine and the policies that manage that virtual machine;

FIG. 7 shows the one-to-many relationship that exists between an embot and the policies that effect autonomic management;

FIG. 8 shows the pluggable service architecture used to support migration where a migration service is shown as a plug-in;

FIG. 9 presents the flowchart that illustrates the steps of the method for virtual machine migration;

FIG. 10a presents the flowchart that illustrates the steps of the method for the procedure “Proceed with Migration” used in the flowchart of FIG. 9;

FIG. 10b presents the flowchart that illustrates the steps of the method for the procedure “Process Management Event State” used in the in the flowchart of FIG. 10a;

FIG. 11 presents the flowchart that illustrates the steps of the method for the procedure “Migrate VM” used in the in the flowchart of FIG. 10b; and

FIG. 12 presents the flowchart that illustrates the steps of the method for the procedure “Complete VM Migration” used in the in the flowchart of FIG. 11.

DETAILED DESCRIPTION OF THE EMBODIMENTS OF THE INVENTION

To facilitate the understanding of the present invention, a reference is made herein to the previously filed applications of Embotics Corporation, all of which are incorporated herein by reference:

Canadian patent applications serial numbers 2,435,655 and 2,475,387 to Shannon et al, both entitled “Embedded System Administration”;

Canadian patent applications serial numbers 2,504,333 and 2,543,938 to White et al, both entitled “Programming and Development Infrastructure For An Autonomic Element”.

The present invention focuses on systems that use autonomic computing principles to manage virtual machines in a scenario where management and service are separated in distinct execution environments, also referred to as management and service planes respectively. A single management plane may provide manageability for one or more service planes. The invention provides a method and system for VM migration, and the infrastructure to support the mobility of manageability components that provide autonomic management for a migrating virtual machine, or more generally an execution container, which constitutes a service plane.

FIG. 1 illustrates an example of an embot, an autonomic management elements developed by Embotics Corporation, to which virtualization infrastructure is applied. In FIG. 1, an autonomic element separates management from a managed element function, providing standard sensing (S) and effecter (E) interfaces for management. It should minimally impact the functions of the managed element. The managed element does not dominate, override, or impede management activity. For example, if the managed element and the autonomic manager share the same processor or memory address space this cannot be guaranteed owing to the management of these shared resources by a shared operating system. True autonomy requires a control plane, which has long been the view in the telecommunication domain.

The notion of the co-existence of a management and a service plane is explained with the help of FIG. 2. The management plane shown in FIG. 2 runs an application framework that provides a set of management services. One service is the management module runtime, which provides an execution environment for embots. All embots execute within this environment, which provides significant abstractions with respect to the service plane being managed. Embots are the smallest runtime units of manageability as provided by this invention. Embots are autonomic elements and created when a management module is deployed to the management plane and loaded. A management module is the smallest unit of deployable system administration. The nature of the management module is the subject of separate previous patent applications of Embotics Corporation cited above.

FIG. 2 shows that the embots running in the embot execution environment interact through the embot application framework with the service plane through sensing and effectors running on the service plane. While FIG. 2 shows a single service plane, a one-to-many management to service plane interaction is supported as would be typical in the scenario where the management plane is instantiated in a privileged virtual machine and the service planes are guest operating systems running within individual unprivileged virtual machines.

In some systems, referring to FIG. 1 and FIG. 2, an embot may represent the monitor, analyze, plan, execution
and knowledge parts of an autonomic manager. On other systems, several embots communicating through the channels shown by arrows connecting them in FIG. 2 could collectively constitute the same functionality.

FIG. 3, FIG. 4, and FIG. 5 demonstrate an example scenario in which a single management plane manages two service planes in a virtualized environment. In FIG. 3, FIG. 4, and FIG. 5, a Virtual Machine Manager (VMM) manages several VMs. Management of a VM is accomplished through an Autonomic Controller Engine (ACE), which is a software component running in the management plane and forms the autonomic element. Two types of VMs: a management VM and a service VM exist on the system (see FIG. 3 for example). The service VM includes a VM managed element with its components and its dependant elements that provide service to the user. The management VM is concerned with the management of one or more service VMs. A management VM is privileged. Privilege implies that the management VM is able to exert control of the resources made available to and consumed by a service VM. An example of a resource is network access and an example of management control could be denying access to the network. An example of a physical instantiation of a privileged virtual machine is Xen’s domain 0. Each service VM is managed by a VM managing element. Several policies execute within the management plane, the policies being implemented within one or more embots. In FIG. 3, policy \( p_{a1} \) is related to the management of virtual machine VM\(_{a1}\); policy \( p_{a2} \) is related to the management of virtual machine VM\(_{a2}\). FIG. 4 shows that VM\(_{a2}\) has migrated to a new host, host B. In order for VM\(_{a2}\) to continue to be managed autonomously the policies used to manage it must be migrated too. FIG. 5 captures the changed system state and implies a requirement for code mobility. Individuals knowledgeable in the art of mobile agents will realize that many instantiations of mobile code (or agents, the words are used interchangeably in this document) are possible.

As shown in FIG. 6, the notified policies contact their embot containers indicating that migration should occur. One embot can contain one or more policies as shown by the one-to-many mapping on FIG. 7 along with managed elements representing the resources being managed in the service plane(s). Embot containers include behavior that support movement of manageability from one management plane to another, including the ability to move both code and state. A service for moving code and data provided as part of a mobile code infrastructure may be used by the affected embots to schedule themselves for migration. FIG. 8 provides a view of the plug-in nature of the infrastructure that can be used to support migration.

A typical scenario that provides an example of the utility of VM migration for achieving load distribution is provided next.

In this scenario there are two service planes running on domain 1 and domain 2 of a system as virtual machines.

A virtual machine managed element (VMManagedElement) has been created for each virtual machine. VMManagedElements are instantiated within the management plane, i.e. domain.

A migration policy that has a sensor which monitors the overall CPU utilization for the host is loaded.

4. The migration policy polls the CPU utilization sensor for percentage load data.

5. The migration policy consolidates the data into a moving average over a user-defined window, e.g. 15 minutes.

6. The average value is tested and if found to exceed a user-defined threshold, e.g., 80%, the migrate eVM Application Programming Interface (API) on the VMManagedElementHome object is invoked.

7. The VMManagedElementHost object is responsible for managing the lifecycle of all VMManagedElement objects. In this scenario two objects exist. The first VMManagedElement object found has its migrate API invoked. The migrate API executes the method that is presented in FIG. 9 and is described in the next paragraph. Should the migrate API throw an exception it is handled within the VMManagedElementHost object. In one embodiment a log is generated. Once the exception is handled, it is thrown again and handled within the migration policy.

The method for the VM migration is explained with the help of flowcharts 900-1200 that are captured in FIGS. 9 to 12. The service VM including the VM managed element as well as the VM managing element for this service VM are migrated from a source host to a target host. Note that both the managed element as well as its manageability units are objects of migration. The steps of the methods illustrated in FIGS. 9 to 12 are executed on the source host.

FIG. 9 is explained in detail below. Upon start (box 902) the source host on which the VM to be migrated is currently deployed sends a Start Migration message to the target host where the VM is to be migrated (box 904). After sending the message the source host waits for a response from the target host (box 906). If the response is not received before the occurrence of a timeout, the procedure exits ‘YES’ from box 906, generates an exception, aborts the migration (box 912) and exits (box 916). Note that when a migration is aborted the source host sends an Abort_Migration message to the destination host. If the response arrives before the occurrence of the timeout, the procedure exits ‘NO’ from box 906 and checks whether the Migration_Denied response that signifies the inability of the target to accept the migrating VM is received (box 908). If such a response is received the procedure exits ‘YES’ from box 908, generates an exception, aborts the migration (box 912) and exits (box 916). If the response is not Migration_Denied, the procedure checks whether a Migration_Permitted response is received (box 910). If such a response that signifies the ability of the target host to accept the migrating VM is not received, the procedure exits ‘NO’ from box 910, generates an exception, aborts the migration (box 912) and exits (box 916). On the other hand if a Migration_permitted response is received the procedure proceeds with the migration (box 914) and exits (box 916).

The step of Proceed Migration (box 916) is explained with the help of Flowchart 1000 shown in FIG. 10a. Upon start (box 1002), the procedure queues the events to be processed by the service VM. These include events for the VM managed element as well as for its dependant elements (box 1004). Instead of letting the events be processed the events are queued because the VM responsible for processing the events is being migrated to a different host. The procedure then serializes the state for the VM managed elements and its dependant elements for building a message
This message containing the management state that includes the current state of the VM managed element and the state of its dependant elements is sent to the target host and the source host waits for a response (box 1008). If a timeout occurs before the arrival of a response, the procedure exits ‘YES’ from box 1012, cleans up the system memory, generates an exception, aborts the migration (box 1020) and exits (box 1022). If the request arrives before the timeout occurs, the procedure exits ‘NO’ from box 1012, and checks whether a Managed_Object_Instatiation_Error response is received (box 1014). If such a response is received, it means that the target host was unable to instantiate the desired management objects and the procedure exits ‘YES’ from box 1014, cleans up the system memory, generates an exception, aborts the migration (box 1020) and exits (box 1022). If such a message is not received, the procedure exits ‘NO’ from box 1016 and proceeds to process the management event state (box 1018) and exits (box 1022).

The step of Process management event state is explained further with the help of Flowchart 1050 presented in FIG. 10b. The role of this procedure is to transfer the queued events from the source host for processing at the target host once the VM migration is completed. Upon start (box 1054), the procedure serializes the queued events for building a message (1056). This message containing the management event state that corresponds to the state of the queued events is then sent to the target host and the source host waits for a response (box 1058). If a timeout occurs before the arrival of a response, the procedure exits ‘YES’ from box 1062, cleans up the system memory, generates an exception, aborts the migration (box 1070) and exits (box 1072). If the response arrives before the timeout occurs, the procedure exits ‘NO’ from box 1062 and checks whether a Managed_Object_Instatiation_Error response is received (box 1064). If such a response is received, it means that the target host was unable to instantiate the desired object, and the procedure exits ‘YES’ from box 1064, cleans up the system memory, generates an exception, aborts the migration (box 1070) and exits (box 1072). If such a response is not received, the procedure exits ‘NO’ from box 1064 and checks whether a Managed_State_Failed response message is received (1066). If such a message is received, the procedure exits ‘YES’ from box 1066, cleans up the system memory, generates an exception, aborts the migration (box 1070) and exits (box 1072). If such a message is not received, the procedure proceeds to migrate the VM (box 1068) and exits (box 1072).

The step of Migrate VM (box 1068) in Flowchart 1050 is explained further with the help of Flowchart 1100 presented in FIG. 11. Upon start (box 1102), the procedure attempts to migrate the VM from the source host to the destination host. If the attempt is not successful, the procedure exits ‘NO’ from box 1106, cleans up the system memory, generates an exception, aborts the migration (box 1122) and exits (box 1124). If the migration attempt is successful, the procedure exits ‘YES’ from box 1106 and checks if there are dirty objects for the VM to be migrated (box 1108). Note that since the VM being migrated is still in operation on the source host, some of the objects may change (become dirty) after the migration attempt is started. These objects include the components of the VM managed element that have changed. These dirty objects thus need to be transferred to the target host where the VM is designated to execute. If there are no dirty objects the procedure exits ‘NO’ from box 1108, completes the VM migration (box 1118) and exits (box 1124). If dirty objects exist, the procedure exits ‘YES’ from box 1108 and serializes these dirty managed objects and prepares a message (box 1110). The message containing the serialized dirty managed objects are then sent to the target host and the procedure waits for a response (box 1112). If a timeout occurs before the arrival of a response, the procedure exits ‘YES’ from box 1114, logs the occurrence of the timeout (box 1120), cleans up the system memory, generates an exception, aborts the migration (box 1122) and exits (box 1124). If a response is received before the occurrence of the timeout, the procedure exits ‘NO’ from box 1114 and checks whether a Migration_State_Success response is received. If such a response is received, it means that the dirty managed objects sent are successfully deployed at the target host and the procedure exits ‘YES’ from box 1116 and loops back to the entry of box 1108 to check if new dirty objects have been created. If the response received is not Migration_State_Success, the procedure exits ‘NO’ from box 1116, cleans up the system memory, generates an exception, aborts the migration (box 1122) and exits (box 1124).

The step of Complete VM migration (box 1118) in Flowchart 1100 is explained with the help of Flowchart 1200 presented in FIG. 12. Upon start (box 1202), the procedure sends a Migration_Complete Message that indicates the completion of the VM migration to the target host and waits for a response (box 1204). If a timeout occurs before the response is received, the procedure exits ‘YES’ from box 1206, generates an exception (box 1212) and exits (box 1214). If the response is received before the occurrence of the timeout, the procedure exits ‘NO’ from box 1206 and checks whether a Migration_Complete_Ack that indicates that the migration is successfully completed is received. If such a response is not received, the procedure exits ‘NO’ from box 1208, generates an exception (box 1212) and exits (box 1214). If the Migration_Complete_Ack response is received, the procedure exits ‘YES’ from box 1208, terminates the service VM and the VM managing element at the source host (box 1210) and exits (box 1214). The migration of the service VM is now complete and the execution of the service VM and the VM managing element are resumed on the target host.

A number of steps of the procedures described in the context of the flowcharts presented in the previous paragraph is further discussed. In response to the Start_Migration message sent from the source host in box 304 of FIG. 9, Procedure A is executed on the target host to allow the target management plane to decide whether to accept the migrating virtual machine. A migration policy associated with the migration service on the target management plane processes the message.

Procedure A (Start_Migration Message) Returns Message

1. The migration policy associated with the migration service is notified of the migration request.
This request includes the location of the source of the request (either IP address or full qualified domain name).

2. The policy checks to see if migration from the requesting source is allowed. If not, a Migration_Denied message is returned.

3. If allowed, the policy checks to see if sufficient resources are available to run the migrating virtual machine. If not, a Migration_Denied message is returned.

4. If sufficient resources are available, a Migration_Permitted message is returned.

 Executed on the source host, Procedure B used in box 1004 in FIG. 10, starts the process of queuing events for the managed elements that are being migrated to the target management plane.

Procedure B (ManagedElement)

1. The sensor service is located using the service registry provided by EAF.

2. The queueEvents API is invoked on the sensor service with the ManagedElement passed as context. This API causes a queue of events to be created within the sensor service such that if any other events will be forwarded to the ManagedElement; they will simply be queued pending reactivation of event forwarding.

3. For all dependent managed elements of this ManagedElement, the queueEventsRecursive API is invoked on the dependent managed element. This causes Procedure B to be invoked recursively.

 Executed on the source or the destination host, Procedure C is used when it is required to re-start the process of dispatching events to managed elements. The queue of events is destroyed once messages are processed by the managed elements.

Procedure C (ManagedElement)

1. The sensor service is located using the service registry provided by EAF.

2. The startEvents API is invoked on the sensor service with the ManagedElement passed as context. This API causes the queue of events created within the sensor service for the ManagedElement to be added to a time-ordered queue of events stored within the sensor service. The queue associated with the ManagedElement is destroyed.

3. For all dependent managed elements of this ManagedElement, the startEventsRecursive API is invoked on the dependent managed element. This causes Procedure C to be invoked recursively.

 Executed on the source host, Procedure D used in box 1210 of FIG. 12, removes the event queues associated with the managed elements and deregisters the (now migrated) managed elements.

Procedure D (ManagedElement)

1. The sensor service is located using the service registry provided by EAF.

2. The stopEvents API is invoked on the sensor service with the ManagedElement passed as context. This API causes events to cease being forwarded to the ManagedElement.

3. For all dependent managed elements of this ManagedElement, the destroyRecursive API is invoked on the dependent managed element. This causes Procedure D to be invoked recursively.
6. If no errors have been detected return a Management_Event_State_Success message is returned.

Procedure H (Abort_Migration Message)

1. All deserialized managed objects are destroyed.

2. The sensor service is looked up.

3. The destroyEvents API is invoked on the sensor service. This API causes all events to be removed from the sensor service.

Procedure I (SerializationStream)

1. The sensor service is looked up.

2. The serializeQueue API on the sensor service is invoked and the resulting byte stream written to the SerializationStream.

Procedure J (Abort_Migration message)

1. All deserialized managed objects are destroyed.

2. The sensor service is looked up.

3. The destroyEvents API is invoked on the sensor service. This API causes all events to be removed from the sensor service.

Procedure K processes changes that occur to ManagedElements (as captured in box 1108 and box 1110 of FIG. 11) while the VM migration is in process.

Procedure K (ManagedElement, SerializationStream)

1. While migration is in progress there is a potential for state to change—affected managed elements are then marked as dirty in this case. Access to the ManagedElement is synchronized for the execution of Procedure K.

2. Therefore, if ManagedElement is dirty, the state of the ManagedElement is written to the SerializationStream.

3. The dirty bit for the ManagedElement is reset.

4. Procedure K is recursively invoked for dependent of the ManagedElement.

5. Return the result of the dirty bit of step 2 along with the result of the recursive call.

Procedure L is used to restart the migrated VM on the target host after the reception of the Migration_Complete Message sent from the source host in box 1204 of FIG. 12.

Procedure I (Migration_Complete message)

1. The migrated managed objects register with the sensor service for events.

2. The VMManagedElement is located within the set of migrated managed objects. Procedure C is then invoked with the VMManagedElement as context.

The embodiment of the present invention has the following features:

Migration of an autonomic manager and the VM it manages;

Management state preservation during migration;

Lifecycle maintenance of management software in a virtualized environment; and

Fault recovery of the management plane when migration of management components cannot be moved in conjunction with a migrated VM.

The embodiment of the invention has the following advantages:

Improved system management through effective delegation;

Results in reduced cost of ownership of system;

Higher system availability;

Management is delegated; management infrastructure responds dynamically to changes in service infrastructure;

Ability to dynamically react to changes in the applications deployed on a system, e.g., if a new application is deployed, the system can automatically acquire and configure management functionality for it; and

Provides a mechanism for coherent management of heterogeneous virtualized platforms, e.g., Windows and Linux operating systems.

The system used in the embodiment of this invention includes computing devices. A computing device has a memory for storing the program that performs the steps of the method for achieving VM migration.

Numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the given system characteristics, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. A method for migrating a service Virtual Machine (VM), comprising a VM managed element including components providing a service, and a VM managing element managing the service VM, from a source host to a target host, the method comprising the steps of:

(a) queuing events to be processed by the service VM at the source host;

(b) sending information regarding a current state of the VM managed element from the source host to the target host;

(c) sending information regarding a state of the queued events from the source host to the target host;

(d) sending components of the VM managed element that have changed during the execution of step (a)-step (c) from the source host to the target host;

(e) terminating the service VM and the VM managing element on the source host; and

(f) resuming execution of the service VM by using the information sent in step (b)-step (d) and resuming the VM managing element on the target host.

2. A system for migrating a service Virtual Machine (VM), comprising a VM managed element including components providing a service, and a VM managing element managing the service VM, from a source host to a target host, the system comprising:

(a) means for queuing events to be processed by the service VM at the source host;
(b) means for sending information regarding a current state of the VM managed element from the source host to the target host;
(c) means for sending information regarding a state of the queued events from the source host to the target host;
(d) means for sending components of the VM managed element that have changed during the queuing of events to be processed, the sending of information regarding the current state of the VM and the sending of information regarding the state of the queued events;

(e) means for terminating the service VM and the VM managing element on the source host; and
(f) means for resuming execution of the service VM by using the information regarding the current state of the VM managed element, the state of the queued events and the components of the VM managed element that have changed sent from the source host to the destination host and resuming the VM managing element on the target host.