A file-system abstraction for virtualized infrastructure

Vitalian A. Danciu
Munich Network Management Team
Ludwig-Maximilians-Universität München
danciu(a)mnm-team.org

***Abstract*—The projection of the structure and operations of virtualized infrastructure onto a file-system structure yields a familiar interface for management and the opportunity to perform management operations with non-specialised tools for file and directory manipulation. The file-system paradigm offers a well-known information model onto which the infrastructure data model is mapped. We describe the mapping of the static and dynamic structure and discuss its potential and limitations. The concepts are illustrated to the reader by means of the Virtualized Infrastructure File-System prototype, that is discussed in terms of architecture and operation.**

Keywords: Virtualization, Management, File-system

I. Introduction

The proliferation of cloud services lead, in turn, to management and control of virtualized infrastructure being increasingly layered vertically, as well as distributed horizontally. Traditional management layers include the administration of single resources, or structures created from them (network management), the management of the resulting distributed system as a whole and the management of services [1]. Horizontal distribution originates from the co-operation between different organisations of operators and service providers.

Management of the virtualized infrastructure is performed in a layered architecture including (beside the hardware) the virtualization facility (hypervisors, network abstractions, etc), a management middleware (tool-stacks, integrated management middleware) and a user interface (UI) as a text or graphical interface. Heterogeneity of the interfaces is an issue in all layers, prompting solutions that consolidate the tool landscape.

Through these efforts, virtualization solutions encompass a large number of middleware products and tools, each using its own flavour of interface to express common management information and operations. Consequently, there is a lot for administrators and tool authors to learn, disregarding whether the tools expose graphical or textual interfaces or the programming language bindings offered by middleware packages. The management platforms employed differ per organisation (often per IT organisation domain), and this heterogeneous management system needs to be integrated with all management tools pertaining to virtualization management solutions within the organisation.

The dimensions sketched in Figure 1 illustrate the idea, that in every organisation there are preferences based both on existing management targets and the prevalent technology for

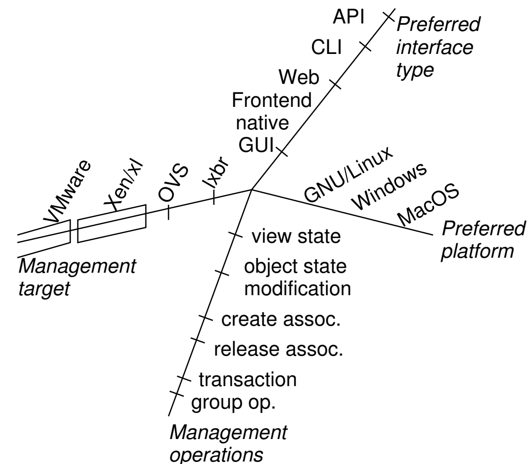

Figure 1. Admin view on the management interface

the work platform, in addition to the preference of the staff for certain types of interfaces.

Existing management solutions differ in at least:

- the user interface of the provided management tools
- the properties of the infrastructure, that are exposed
- the operations provided to administrators
- their model of virtual components and compounds
- the virtualization technologies supported

Their differences have several implications for operations and management, including:

- training administrators in all solutions/tools used in their organisation as well as co-operating, co-managed organisations.
- accepting in self-developed utilities a reliance on the models and interfaces of the tools used
- an impact in both training and re-factoring when introducing or migrating to different virtualization technologies or management tools

While standards have been developed early on and are evolving, they sensibly target the lower layer interfaces of virtualization products. But even at that level, the interfaces exposed by the different products differ in syntactic conventions and the semantics of their attributes and operations [2]. Thus, it seems interesting to experiment with a pre-existing,

978-3-901882-77-7 © 2015 IFIP

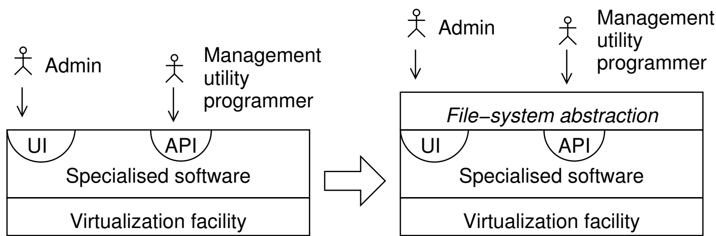

Figure 2. Management stack before and after introduction of FS abstraction

widely accepted interface as a "point of condensation" for normalisation efforts.

A. Approach

This paper explores the projection of virtual infrastructure and the management operations performed on it onto one of the most familiar metaphors — that of a file-system (FS). By viewing and manipulating objects in the FS, a manager (and even a user) is able to monitor and manage the virtualized infrastructure itself by using his favourite tools for file manipulation, as illustrated in Figure 2.

The obvious benefit of the approach is the immediate capability to view the state of and perform management operations on the virtualized infrastructure without the need of specialised tools: the user is allowed to employ their favourite tools (graphical file-managers, shell commands, . . .), that they use to manipulate files and directories. In the same manner, the FS offers an equally well-known and entrenched programming interface for which practically any programmer has received training.

Associated concepts such as access control and ownership can be specified in a well-known manner, documentation can be integrated into the tree describing the infrastructure and a view on collaborating organisations can be integrated, as well. Another important benefit is the choice of arbitrary platform (all OSs have FSs) and topological location (an FS may be remote) of the FS user in relation to the management station.

B. Synopsis

Section II summarises related work and similar approaches in other domains. The following Section III develops the basic model underlying the projection of the virtualized infrastructure onto the structure and operation primitives of a FS. Section IV details the mapping of infrastructure elements to FS objects, while Section V shows the mapping of operations on the infrastructure onto operations performed on the FS. Section VI sketches the architecture of a prototype FS as a proof-of-concept. We proceed with a discussion in Section VII before concluding in Section VIII.

II. Related work

The representation of infrastructure by means of an FS might be viewed as a metaphor of a second order, the FS itself being a metaphor for an office archive. Thus, we create *"an abstract conceptual space from embodied experiences, i.e. interactions with the real world"*[3] as described by Guhe, Smaill and Pease.

Kuhn and Frank have proposed a formalism for (user interface) metaphors, that describes their essence in terms of the *objects, operators and axioms* involved, thus enabling comparison between the original (source) domain objects and the target objects [4]. In our case, such a comparison would be between the management view of infrastructure and its view in the FS paradigm. While we forego the formal notation due to the significant space it requires, the principle of projection is similar: we project managed objects, operations and axioms, i.e. the invariants and conventions pertaining to a certain class of objects.

The idea of applying the FS metaphor to virtualized infrastructure seems new, though it has been leveraged in other relevant areas. On some operating systems, *proc* [5] and *sysfs* [6] are host-local pseudo-FSs, that provide some management data and some opportunity to set system parameters.

The XSEDE project proposes to employ the familiar concept of an FS to manage distributed data but also computational resources in high-performance computing settings [7].

The containment model of the infrastructure developed in Section III is projected onto the Portable Operating System Interface (POSIX) [8], that is a standard developed jointly by the IEEE and The Open Group. It specifies, as its name implies, a common interface for operating systems. A sub-set of it defines a model of FS and a number of basic operations on FS objects and offers, due to its age and widespread support a suitable projection surface.

III. Location and containment model

As noted in the introduction, a proper representation of virtualized infrastructure must be able to represent physical infrastructure as well as the virtual components provided by it. Further, it must be able to represent different organisations, that may be in co-operation.

A suitable representation is that of a containment tree, that represents the location of any organisational or infrastructure entity within a containing one. The root of the tree denotes the "infrastructure as a whole", its children are provider organisations (management domains). Both administrative and technical containment needs to be modeled. From the point of view of a provider organisation, relevant infrastructure is within the own management domain and within associated domains.

A. Hierarchy

We differentiate between physical locations, logical infrastructure partitions, physical components and structures, virtual components and structure, and services. Vertical scoping may be employed to define the zone of authority of administrative roles. Figure 3 illustrates these concepts.

Physical locations include data centres, points-of-presence or other deployment sites tied to an organisation. They may be structured into logical infrastructure partitions arbitrarily by the organisation they pertain to. Such partitions may be created in order to structure the organisation itself into organisational units responsible for a certain part of the infrastructure or they may be employed to differentiate between different infrastructure partitions according to some other, again arbitrary,

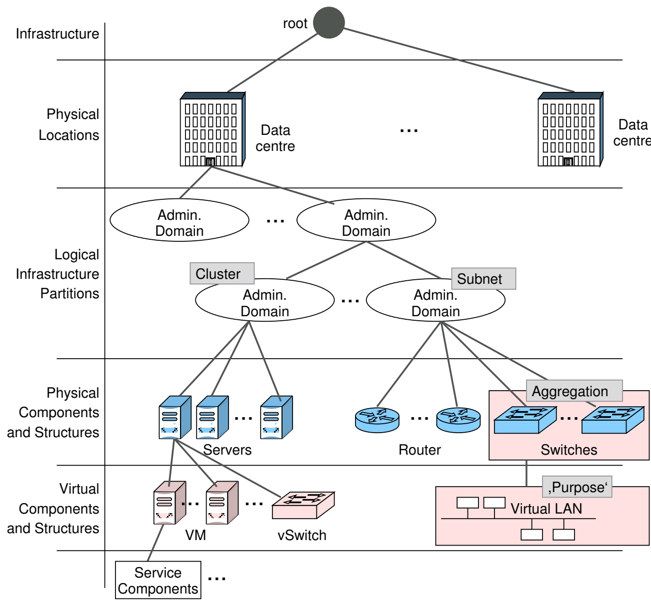

Figure 3. Containment hierarchy

criterion of the organisation (e.g., by service, by customer, by technology).

The next fixed layer is that of *physical components*, including computing elements, network elements, storage elements. It is conceivable to model physical components as a containment hierarchy themselves, e.g. to describe an I/O component as being contained within a computer, however, for the purpose of this paper we will not explore that avenue further.

Physical components may support *virtual components and structures*. As with physical components, a containment hierarchy is possible within this layer. It is worth to note, that the provisioning of virtual components nested in virtual components (e.g., by the execution of a hypervisor within a virtual machine) is a much more compelling argument to develop the containment model within this layer.

Service instances provided by *single* virtual components are said to be contained in the virtual component.

Compound entities include clusters of computing elements and storage elements, both physical and virtual.

B. Horizontal aggregation

Containment may represent single entities, but not horizontal structures. Horizontal aggregation, i.e., *grouping by purpose* is employed for networks or clusters spanning several tree branches. The purpose may be the delivery of a service or the creation of a structure. The aggregation of switch ports to a virtual LAN (VLAN) shown on the right side in Figure 3 illustrates a horizontal aggregation. Its purpose is the creation of the VLAN, a virtual structure.

C. Goals and principles

Having concluded the generic containment model, we proceed by projecting it onto the features offered by common FSs, after discussing the goals and principles of the projection.

Replacing a specialised interface with a metaphor—and one intended for another use, no less—requires careful balancing between retaining the intuitive use of the resulting system and replicating the function of the specialised interface. We require at least the following properties:

1) The resulting FS should be capable to represent:
   a) objects within the infrastructure
   b) binary associations between objects
   c) native attributes of technical objects
   d) translated (harmonised) attributes and meta-attributes of managed objects
2) The resulting FS should conform sufficiently in order to be usable with existing, common file-management tools, graphical or otherwise.
3) A minimal set of additional conventions to the FS metaphor should be introduced.

To ensure the independence of the approach from specific implementation, we rely on POSIX [8] as an accepted and well-implemented standard for operating systems. Thus, the approach is viable for implementation on any platform, that complies to POSIX. In the following sections, we will first project the model structure onto FS objects (Section IV; then we will proceed by tying semantics to operations on those FS objects, that represent elements of the virtualized infrastructure.

IV. Projection of structure

File-systems contain three types of objects, that are of interest: directories, regular files, and symbolic links. We will represent all aspects of the virtualized infrastructure by using these object types.

The general idea is simple: Managed objects are represented by directories. Their attributes are represented by (hidden) files within those directories. Relationships between MOs are represented by symbolic links, e.g., the connection of the virtual network interface of a VM to a port of a virtual switch. Entities are arranged within the location/containment tree.

A. Entity addresses

Without loss of generality, we will in the following assume a UNIX-like syntax with a slash delimiting path elements. In addition, we follow the convention that file names beginning with a period denote hidden files.

In consequence, every entity within the infrastructure, including organisational constructs, can be addressed by path. For example, the leftmost VM in Figure 3 is described by: `/Data Centre 1/Admin. Domain 2/Cluster 1/Server 2/VM1/`

It is important to note, that this containment hierarchy is distinct from the DNS (Domain Name System) domain and zone hierarchies or other directory hierarchies commonly employed to maintain models relevant to management. Thus, while the use of Uniform Resource Identifiers (URI) might seem to lend itself to addressing entities, its use of domain names depicts a different hierarchy than that of containment.

Prefix	Name	Semantics
.	period	component attribute
_	underscore	special file (meta-data)
+	plus sign	push-button file

Table I. FILE PREFIXES

B. Attributes

An entity's attributes are represented as text files within the directory representing the entity itself. The file name is patterned after the attribute name, and the file contains the attribute's value as a text string. For example, the attribute of a Xen VM ("domain"), that describes the maximum memory for the domain in KiB will be stored in: `/Data Centre 1/Admin. Domain 2/Cluster 1/Server 2/VM1/max_memkb`

Its content is a text string representing the value, that can thus be viewed with the basic tools provided by the OS.

We use prefix-based meta-typing of attributes to differentiate between those originating in the entity itself, those describing the entity from a management system perspective and other files within an entity's directory. Prefixes are proposed in Table I.

1) Special attribute files: The interaction with the FS requires contextual information. In particular, users need to be able to differentiate between the types of entities being represented by a directory, and to be able to determine the state of that entity, the technology that it uses and the source of its configuration. To accommodate this requirement, each directory representing an infrastructure entity contains special attribute files with reserved names, that contain meta-data. As with files representing an entity's attributes, these files contain the value of the meta-data attribute. For example, `_type` is the type of the managed object represented by the directory. An alternative representation mechanism using the *extended attributes* supported by some FSs might seems attractive; we discuss it in Section VII-C.

2) Meta-data: Some information can be projected onto FS standard built-in features.

a) Time stamps: Three different time stamps are specified for POSIX FS objects: last access time, last modification time and last status change time. In a simplified view, the "access" means the file was read, "modification" means the file was written and "change" refers to changes to attributes such as ownership or access mode.

The time stamps can be leveraged to provide meta-data about the managed objects or attributes represented by directories and files in the same manner. In particular, the modification time of an attribute is the time when the attribute value was procured from the source.

b) Ownership and access control: File-system objects are owned by a user and a group. Both are identified by integer values for user-ID (UID) and group-ID (GID), respectively. The conversion to corresponding textual values (login name of a user, name of a group) is provided by the OS, which may employ different means to perform the mapping. The most common include reading from a file (`/etc/passwd` on UNIX-like systems) or retrieving the value from a directory service such as an LDAP directory service. Therefore, the management of users and groups are not an issue of the FS itself (which deals only with the numerical values). To control access to the objects within the FS, an implementation needs to introduce administrator identities (user-IDs) and roles (group-IDs) into whatever directory or other source is used on the machine where the FS is executed. Once implemented, this mechanism allows fairly fine-grained control of the objects represented in the FS. Alternatives and its scalability are discussed further in Section VII-C.

3) Push-button files: So-called *push-button files* offer the possibility to execute management operations on an MO by simply updating the modification time of the file. This can be realised for example with the POSIX command `touch(1)`, which opens the file and sets the modification time to the current time. The content of a push-button file documents the effect of "pushing the button". For example, the file `+shutdown` within a directory representing a virtual machine can be modified in order to shutdown that virtual machine.

The use of push-button files is discussed in Section V-B. Although they introduced in this section (for closeness to other structural conventions), push-buttons are actuators, not attributes. However, their presence in an entity directory signifies the availability of the operations they represent.

C. Volatile and persistent data

Dynamically created FS objects are *volatile*, as their lifetime is tied to the existence of some managed object. If that managed object is destroyed, the corresponding FS object is meaningless.

Other objects within the FS are persistent. These include manually created objects such as the directory tree describing the organisation's structure or documentation files added to directories corresponding to MOs. In this context, "persistent" means that the object persists across the termination of the FS program; in contrast, a volatile object will be destroyed when the FS is unmounted.

In summary, the FS contains a directory hierarchy, where each directory represents an organisational or technical element of the infrastructure. Sub-directories of a directory represent contained elements. All files within a directory represent attributes or meta-attributes of the element represented by the directory. The content of these files is the textual representation of the attribute's value. Any symbolic link to a directory or file represents an association between source and target of the link.

V. PROJECTION OF OPERATIONS

To allow effective management by means of manipulating FS objects, management operations must be mapped onto the set of operations supported by the FS. In the following, we will first assess the operations exposed by the FS, then discuss their mapping onto a set of desired management actions.

A. File-system operations

We differentiate roughly between three classes of operations. The first two are read-only operations, where the content of the FS is queried and modifying operations, the execution of which modifies the FS. They correspond to monitoring and

Class	Function	Description (informal)
Read-only	stat	read the attributes of a file
	access	check active user's permissions for a file
	readlink	read the target of a link
	read	read from an open file
	readdir	read the contents of directory
Modifying	mount	mount a FS
	umount	unmount a FS
	creat	create a file and open it
	link	create a (hard) link
	mkfifo	create a uni-directional pipe
	unlink	remove a file name, decrease link count
	symlink	create a symbolic link
	mknod	create a device file, pipe, or socket
	mkdir	create a directory
	rmdir	delete a empty directory
	rename	rename an FS object
	chmod	change access mode
	chown	change ownership and group ownership
	utime	change time stamps
	write	write to an open file
Auxiliary	open	open a file
	release	close a file
	fsync	synchronise memory contents with storage contents

Table II. FILE-SYSTEM OPERATIONS (READ-ONLY, MODIFYING, AUXILIARY)

control operations, in management terms. The third class of auxiliary operations includes opening/closing files. Table II summarises the relevant selection of POSIX FS operations according to this classification.

Read-only operations include reading the contents of a directory (listing the files and directories contained within it), reading the contents of a file, reading the target of a symbolic link, reading the attributes (ownership, time-stamps, access mode) of an FS object and reading the state of the FS itself.

Modifying operations include the creation of FS objects, the modification of their attributes, renaming and moving FS objects, changing the target of links and modifying the content of files.

Auxiliary operations bound the modification phase of an FS object. Thus, they signal the beginning and the end of a process that modifies the content or the attributes of an object.

B. Management operations

In this section, we discuss the realisation of management actions by means of the FS operations. The set of projected operations is not comprehensive: it is intended as a subset of possible actions for examination and discussion.

Operations on the FS can be viewed as management actions where the operation type corresponds to the method identity (the "function name") and the FS objects acted upon represent its parameters. The limitations of this model and the possibly constraints on the breadth of the management operation repertoire is discussed further in Section VII-B.

When the user (administrator) executes an operation on a part of the FS, that represents an infrastructure element or subtree, their operation is intercepted and triggers the execution of a management operation. The original operation on the FS succeeds if and only if the management operation succeeds.

We will constrain ourselves in the following to basic management operations from common management categories; their re-combination is up to the FS user. We also discuss the FS operations that are intercepted (hooks), in order to execute the management tasks they trigger.

1) Monitoring infrastructure state:

a) Navigating: The FS can be navigated with common tools in order to view its structure. No interception is necessary.

b) Reading attribute values: Dynamically changing attributes are re-read on-demand, when the user attempts to read the attribute value. Thus, the `read()` function is intercepted.

2) Configuration management:

a) Changing attribute values: Configuring an attribute is done by writing the desired value into the file that represents it. The intent of changing the attribute value is bound to the `release()` function (that closes the file). If the value in the file is valid and differs from that before all `write()` operations since the file was opened, the attribute is reconfigured. Internally, the `open()` function is intercepted in order to cache the old value of the attribute for comparison.

An attempt to open an immutable attribute for writing (such as the amount of RAM of a physical machine) fails. If the new attribute value is deemed to be invalid, or if its effective reconfiguration in the infrastructure fails, the value is read again from the infrastructure element it pertains to, and an error is logged.

b) Creating associations: A directed association between to elements is instantiated by creating a symbolic link from the directory of one of the elements to that of the other.

A prominent example is the link between a network interface of a virtual machine to a virtual switch. This case is realised by creating a symbolic link within the directory representing the virtual switch, pointing to the directory representing the virtual machine and being named after the interface of the virtual machine. Note that this manner of representing a binding between a network interface and a switch is a convention. It is conceivable to create a file within the VM's directory to point to, or even have the symbolic link point the other way, i.e. from such a file to the virtual switch.

For the creation of associations, the `symlink()` function is intercepted.

c) Destroying associations: For the destruction of associations, the symbolic link representing them is removed with the function `unlink()`. The function will fail if the operation is not permitted. This can be the case if the user has insufficient privileges (i.e., due to FS semantics) or if the association may not be removed due to management policy, or if the association is immutable (e.g., a physical link).

d) Changing an association: Changing an association can either reduced to destroying the old association and creating a new one, or it can be atomic. When the association link is simply modified, the function `rename` can be intercepted, and the change is atomic. Such an operation can either succeed or fail. The deletion of the link and subsequent creation of a new one cannot be intercepted, as there is no state in which the management intent is clear. The deletion and the creation

operations (i.e., calls to `unlink()` and `symlink()` can fail independently.

3) Access control:

a) Testing access control: The access control attributes of an FS object are not modified dynamically. Therefore, they can be tested (with the `access()` function, internally) as usual.

b) Changing ownership: User and group ownerships are changed as usual, with the `chown()` function. This function need not be intercepted, but it may be if a record of changes to privileges is desired, or if it is desired to enforce a management policy.

c) Changing access mode: Access mode bits may also be set as usual with the function `chmod`. The same rational as with changing of ownership applies.

4) Life-cycle: Life-cycle management operations include the creation and destruction of virtual components, as well as suspend/resume operations for most infrastructure components.

a) Initialisation: The FS is initialised with pre-configured administrative domains and the physical components assigned to them. Virtual components are queried dynamically. This initialisation can be realised by means of configuration files or by evaluating persistent state information from preceding executions of the FS. See Section VI-A2 for a description of how this is realised in the prototype presented in this text.

b) Activating and deactivating components: Virtual components can be configured but inactive. For example, an administrator might have prepared images and configuration files for the creation of a VM, but the VM is presently not running.

c) Adopting sub-trees for management: Introducing previously unknown components into the managed set requires the creation of a directory containing a `_type` file, that contains the correct type. The name of the component should reference it in its container. The component will be adopted into the tree when a special file with the name `_managed` is created in its directory, if the component's location and type are valid.

For example, adding a previously unknown VM host machine into a cluster would require the following three steps:

1) Create a directory with the DNS name of the machine within the directory representing the cluster.
2) Within the VM host's directory, create a file named `_type` containing the type `vmhost`.
3) Within the VM host's directory, create a file named `_managed`.

The creation of the special file will be intercepted and the sub-tree representing the machine (attributes, other special files, pushbutton files) will be instantiated. In addition, as the machine in our example is a VM host, directories will be created and populated for every virtual component (VMs, virtual switches, etc.) hosted on the machine in question.

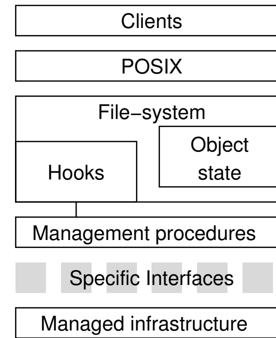

Figure 4. Architecture overview

d) Releasing sub-trees from management: Releasing a sub-tree from management can be performed by deleting the special file `_managed` from the top directory of the tree. Releasing the components within the sub-tree from management does not imply that they are shut down or destroyed. They simply cease to appear in the FS.

e) Creating virtual components: Creating components is a complex matter, due to the need for additional conventions: while components can be created from templates, the parametrisation of the templates and the actuation of the virtualization facility to create a component require recognisable structures that must be agreed upon instead of being intuitive to FS users. Further investigation is necessary to determine the minimum of conventions necessary for the extension of this part of the FS metaphor.

f) Destroying, suspending and resuming: Components, that support suspend and resume operations offer corresponding push-button files (see Section IV-B3 called `+suspend` and `+resume` within the directories, that represent them. The dynamic attributes of the component are removed while it is suspended and re-created when it is resumed. Creating files with the names of those attributes in the mean time will fail.

Destroying components implies the destruction of all contained components. The consistent manner of supporting this operation would be to employ push-button-files, as with suspend/resume. In order to prevent accidental destruction, we require in addition, that the keyword *destroy* be written into the push-button file.

g) Migration: VM migration is represented by the movement of the directory tree representing the VM from the containing directory representing its host to a directory representing the target host of the migration. This operation corresponds to the POSIX `rename()` system function.

VI. Architecture

The realisation of the projection can be accomplished in an architecture as the one depicted in Figure 4. The FS itself exposes the POSIX standard interface that can be accessed by clients. Calls to this interface may trigger hooks to management procedures, that (via an adaptation layer) perform the salient management operations on the managed infrastructure. The FS organises the state of the represented managed objects. The quality and timeliness of this state is a

matter of implementation: infrastructure state can be queried constantly or on demand, when an FS object is accessed.

A. Prototype

A FS called *VIFS* (Virtualized Infrastructure File-System) as described in the previous sections has been implemented by the author as a FUSE (Filesystem in USErspace) application. This prototype has bindings to the Xen hypervisor with the xenlight (xl) tools-stack and to the Open Virtual Switch (OVS). Bindings to other virtualization technology can be added easily by extending only the management hooks of the FS, without interfering with the actual FS code. A synopsis of features and functions can be found in [9]. In the following, we discuss selected implementation-specific topics with regard to this FS.

1) Notifications with management semantics: Notifications presented to the user as a consequence of manipulating the FS are in terms employed for FSs. This behaviour is intrinsic to the approach. It implies, that the management semantics of the notification must be interpreted by the user. However, the constrained set of notifications specified for POSIX operations is not sufficient to unambiguously interpret the meaning of the notification in management terms.

For example, if a user attempts to migrate a VM between two hosts (by moving one directory to another) and the operation fails, the FS will issue the error code `EFAULT`, which is specified to mean, that either the old location path or the new location path point outside the "accessible address space". This error code was chosen to avoid misinterpretation by client programs, but it gives no hint as to the reason of the failure in management terms, which would indicate the reason for the failure of the migration operation: the failed operation may have been an attempt to migrate a physical host or another meaningless operation in management terms. In addition, that error code may not even be presented to the user: the manual page of the common GNU implementation of the `mv (1)` command merely differentiates between zero and non-zero return codes; graphical clients typically present their own interpretation of the error in a dialogue window.

To address this problem, the prototype provides a special file within the FS that delivers textual status and error messages with management semantics. In this manner, the notifications are made available to the user even when the FS is used remotely, over a protocol such as NFS or CIFS/SMB (Network File System, Common Internet File System; specified in [10], [11], [12]).

2) Persistence: The non-volatile portions of the FS are written to a file when the FS is unmounted. They are read and interpreted when the FS is re-mounted. The differentiation between volatile and persistent files is made internally. All dynamically created attributes originating from queries to a component are deemed to be volatile. In addition, all virtual components are deemed to be volatile. Organisational structure and physical components are persistent.

VII. Discussion

The projection of structure and state (Section IV and V-B) suggests that a large number of management-relevant information items and operations can be mapped in a fairly intuitive manner to FS objects and operations. In this section we discuss possible extensions, after outlining the limitations inherent to the "abuse" of file-system semantics for management purposes.

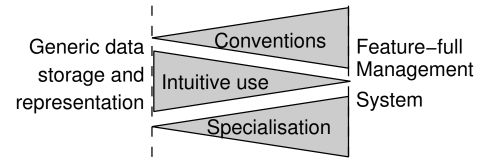

Figure 5. Trade-off between specialised and intuitive use

A. Double-edged conventions

Concepts and tools become specialised in order to more precisely suit a particular intent. In particular, the concept of a file-system has been devised for the organisation of data in files, and management systems have been devised for controlling a specific type of objects. If one is used for the purposes of the other, it is natural that a subset of features will not constitue an exact fit and require *conventions of use* to be introduced in order to replicate the feature set of the projected concept. For example, we have introduced name prefixes to distinguish file with special meaning: some life-cycle operations on managed objects have been realised as push-button files (see Section V), others represent attributes or meta-attributes. While similar conventions exist in normal use of FS (e.g. the use of name extensions to signal data format), but they are different from those introduced in this work.

All conventions must be conveyed to users before they are effective. Thus, at the same time the FS abstraction offers a beneficially familiar interface and poses the challenge of how to project specific management artefacts onto it: any additional convention constitutes a specialisation and requires effort. The trade-off is between intuitive use on one hand and powerful operations on the other, as illustrated in Figure 5. In essence, this is a challenge of human-computer-interface (HCI) design which may require the examination of convention candidates based on users' response to them; such an examination goes beyond the scope of this paper.

B. Operations

We have discussed how operations on FS objects are projected onto management operations in Section V-B. Although this projection is straight-forward on the surface, changing the purpose of the interface exposed towards clients raises the possibility that they may use it in a different manner than anticipated.

1) Unanticipated client behaviour: One such possibility is the use of an unanticipated sequence of operations, that would lead to the same (anticipated) result when operating on files, but would fail to trigger the corresponding management operations. For example, it is conceivable, that a rename operation be performed by copying every relevant file to the new location before deleting the original structure. A mapping expecting to detect a `rename(2)` function call would instead observe a sequence of `open(2), read(2), close(2)` and `unlink(2)`. Hitherto, I have not observed such behaviour in the FS clients on GNU/Linux, including Nautilus, the Rox filer,

GNU Bash, the standard file-utilities (`cat, mv, ln, ...`); they behave as expected, though this does not eliminate the issue in principle.

2) Group operations: A more acute issue involves *group operations*, i.e. operations on groups of managed objects. While they can exploit the support for regular expressions, that is widely implemented for files-system tasks, the knowledge about the set of objects being manipulated is retained by the FS tool, that performs the operation. To use the example above, moving a set of directories representing VMs might be actuated by a drag-and-drop gesture in a graphical file-manager, that will generate a sequence of `rename(2)` operations to the FS. Thus, instead of the single operation on a group of objects, as the gesture might lead an operator to believe, the FS will observe a sequence of single operations, instead. This behaviour obviates the opportunity to optimize the resulting set of migration operations performed on the VMs. Furthermore, moving the same set of "VM directories" to a directory representing a cluster would ideally trigger a placement algorithm to distribute the VMs on the target cluster. An approach to counter this issue is the *trap-correlate-execute* scheme: the FS might "pretend" to perform the first `rename` call by returning a success value and wait for a short time for the next function call that might form a group operation together with the first one. When a time-out is experienced, the list of operations accumulated until then is executed as a whole, enabling the introduction of optimization. This, approach, however, introduces other issues, such as concurrent access to the FS from different clients performing unrelated (group) operations, the possibility of failure of the operation on one of the objects (which had been returned as successful) and the choice of time-out parameter for differently loaded FSs. Some management group operations might require *transaction semantics*, and it is difficult to ascertain that a group of operations correlated into a single operation contains all members.

C. Expressive power

The meta-data items of FS objects were intended to describe simple files, directories and links — not to serve as a replacement for the more elaborate management information one would wish for to describe managed objects.

An administrator may be assigned multiple roles, which is reflected by users being members in multiple groups. *Access control* with the basic POSIX attributes alone can be cumbersome when many roles are employed, as FS objects can only pertain to a single group-ID. Access control lists (ACL) constitute a more powerful alternative to this traditional access control mechanism. Unfortunately, they are by far not as well standardised across platforms, even when they are available. Thus, while it might be possible to employ the ACLs specified in POSIX, their use may undermine the applicability of many standard file manipulation programs. Therefore, we prefer the more common, albeit simpler mechanism.

1) Time: Time attributes cover only the most basic temporal aspects of FS objects: access and modification times for the object itself and the change time of its meta-data. In contrast, effective management of IT components may require to track, for example, transitions between states and life-cycle phases. Accounting management may require to record access times and durations, service management processes might require information on incidents and time-to-repair. Due to the plethora of conceivable requirements on the time-related attributes of a managed object, it is perhaps cautious not to attempt a projection of some of these information items onto the core elements of the FS but instead favour the mechanism provided by "special files" as described in Section IV-B1.

2) Extended attributes: Modern FS support so-called "extended attributes" that enable the association of name-value pairs with any FS object. Extended attributes could be employed instead of files to store the values of the attributes of entities within the infrastructure. While this possibility is tempting, it appears that popular remote FS protocol implementations do not have practical support for extended attributes. In addition, it would require tools (e.g., file-manager applications) with good support for extended attributes. In the future, when extended attributes are more widely supported and entrenched, their use would facilitate a more elegant information model.

3) Management data sources: Component internals may be exposed as FS objects as well. Infrastructure components commonly offer consoles and logfiles, that can be mapped onto character device files or specially named logfiles, respectively, in order to integrate in-component data sources into the management FS.

VIII. Conclusion

We have introduced a mapping of virtualized infrastructure state and operations onto FS state and functions. The purpose is to explore the use of a familiar interface and tools familiar to any user in a context dominated by specialised management software. The research prototype constructed as a proof-of-concept shows as expected, that the basic operations can be mapped successfully for one of the major virtualization platforms.

However, experimentation also reveals a few intrinsic shortcomings of the use of the POSIX interface as a front for management operations. Experimentation in the domain of group operations and transactions could show if there is a chance to detect and group operations speculatively.

Finally, although the FS metaphor is well-known, its use for active management merits examination in usability testing: it remains to see if what has become intuitive for files remains just as intuitive for managed objects and their attributes.

References

[1] H.-G. Hegering, S. Abeck, and B. Neumair, *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application*. Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999.

[2] V. Danciu, N. gentschen Felde, M. Kasch, and M. Metzker, "Bottom–up harmonisation of management attributes describing hypervisors and virtual machines," in *Proc. 5th Int. DMTF Wsh. Systems and Virtualization Management: Standards and the Cloud (SVM 2011)*, Distributed Management Task Force (DMTF). IEEE Xplore, 2011.

[3] M. Guhe, A. Smaill, and A. Pease, "A formal cognitive model of mathematical metaphors," in *Proceedings of KI 2009*, ser. LNAI, B. Mertsching, M. Hund, , and Z. Aziz, Eds. Springer Verlag, 2009.

[4] W. Kuhn and A. U. Frank, "A formalisation of metaphors and image-schemas in user interfaces," in *Cognitive and Linguistic Aspects of Geographic Space*, ser. NATO ASI Series D, A. U. Frank and D. M. Mark, Eds. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1991, vol. 63.

[5] J. Birnbaum, "The linux /proc filesystem as a programmers' tool," *The Linux Journal*, Jun. 2005. [Online]. Available: http://www.linuxjournal.com/article/8381

[6] P. Mochel, "The sysfs filesystem," in *Proceedings of the 2005 Linux Symposium*, Ottawa, Canada, 2005.

[7] F. Bachmann, I. Foster, A. Grimshaw, D. Lifka, M. Riedel, and S. Tuecke, "XSEDE architecture level 3 decomposition," version 0.972, Jun. 2013. [Online]. Available: http://hdl.handle.net/2142/50274

[8] "POSIX.1-2008," IEEE and The Open Group, Specification IEEE 1003.1, The Open Group Technical Standard Base Specifications, Issue 7, 2013. [Online]. Available: http://pubs.opengroup.org/onlinepubs/9699919799/

[9] V. Danciu, "Presenting the cloud as files and directories," in *Proceedings of the 11th IEEE International Conference on eScience*. Munich, Germany: IEEE, Aug. 2015, (To appear.).

[10] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, "Network File System (NFS) version 4 Protocol," RFC 3530 (Proposed Standard), Internet Engineering Task Force, Apr. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3530.txt

[11] *Common Internet File System (CIFS) Protocol*, Microsoft Corporation, Oct. 2012.

[12] *Server Message Block (SMB) Protocol Versions 2 and 3*, Microsoft Corporation, Oct. 2012.