Context-Aware Cloud Topology
Optimisation and Simulation

CACTOS Toolkit Version 2

Accompanying Document for Prototype deliverable D5.2.2

Henning Groenda, **Christian Stier (FZI)**,

P-O Östberg, Jakub Krzywda, Ahmed Ali-Eldin (UMU),

James Byrne, Sergej Svorobej, Gabriel Gonzales (DCU),

Zafeirios Papazachos (THE QUEEN'S UNIVERSI),

Craig Sheridan, Darren Whigham (Flexiant Limited),

Christopher Hauser, Athanasios Tsitsipas, Jörg Domaschka (UULM)

Due date: 31/03/2016
Delivery date: 31/03/2016

This project is funded by the European Union under grant agreement no. 610711

Dissemination Level

X	PU	Public
	PP	Restricted to other programme participants (including the Commission Services)
	RE	Restricted to a group specified by the consortium (including the Commission Services)
	CO	Confidential, only for members of the consortium (including the Commission Services)

Version History

Version	Date	Change	Author
0.1	27.10.2015	Creation of document based on D5.2.1 and feedback of review	Henning Groenda (FZI)
0.2	27.10.2015	Information on Service Interfaces	Henning Groenda (FZI)
0.2.5	18.12.2015	White-Box Application Support	Henning Groenda (FZI)
0.2.8	20.01.2016	Storage Modeling	Henning Groenda (FZI), Christian Stier (FZI)
0.2.9	10.02.2016	Flavour Support	Henning Groenda (FZI), Christian Stier (FZI)
0.3	15.02.2016	White-Box Application Support Update	Henning Groenda (FZI)
0.3.5	12.02.2016	Runtime Management and Infrastructure Integration	Henning Groenda (FZI), Christian Stier (FZI)
0.4	18.02.2016	Auto Scaling	Henning Groenda (FZI)
0.5	29.02.2016	Completion of document and preparation for review	Henning Groenda (FZI)
0.5.5	02.03.2016	Updates after internal review comments	Henning Groenda (FZI), Christian Stier (FZI)
0.6	05.03.2016	First review	Ahmed Ali-Eldin (UmU)
0.7	07.03.2016	Addressed comments from first review	Henning Groenda (FZI)
0.8	22.03.2016	Second review	Jörg Domaschka (UULM)
0.9	24.03.2016	Addressed second review	Christian Stier (FZI)
0.9.1	28.03.2016	Addressed PaaSage/CloudSocket integration	Jörg Domaschka, Christopher Hauser (UULM)
1.0	31.03.2016	Final document preparation	Henning Groenda (FZI)

EXECUTIVE SUMMARY

This document is accompanying material for the prototype deliverable D5.2.2. It describes the changes for the second version of the CACTOS Toolkit and provides details on the integration between the tools and toolkits. A main focus is on showing updated models, as this is how information is passed between the tools. Identical models are used during Runtime and Prediction time. Please refer to accompanying material for the prototype deliverable (D5.2.1 CACTOS Toolkit Version 1) for an overview on the CACTOS toolkits and an exemplary use case.

Note that there are two CACTOS toolkits: The CACTOS Runtime Toolkit (label before year 1: CACTOS Toolkit) and the CACTOS Prediction Toolkit. The CACTOS Runtime Toolkit contains the tools CactoScale and CactoOpt and is described in this deliverable. The CACTOS Prediction Toolkit is described in (D6.4 CactoSim Simulation Framework Final Prototype).

The major architectural additions to the CACTOS Runtime Toolkit since year 1 are added support for monitoring and scaling of White-Box Applications. White-Box Applications allow for monitoring of application internals on top of the VM-level metrics that CACTOS collects for all VMs. White-Box Applications such as PlayGen's DataPlay can use CACTOS AutoScaling services to let the CACTOS Runtime Toolkit adapt the degree of horizontal scaling based on the current load. This document describes both the additions to the models and CACTOS Runtime Toolkit that have been made to support monitoring and scaling of White-Box Applications.

Finally, the document provides detailed insight into the architecture and service structure of the CACTOS Runtime Toolkit. This includes a detailed description of the Virtualisation Middleware Integration (VMI) and VMI Controller that form the Cloud-specific connector CACTOS uses to translate its optimisation decisions to a running Cloud environment's API. Additionally, an overview over the Extensible Services Infrastructure architecture style is given. This architecture style allows for a dynamic reconfiguration of used optimisation algorithms and policies. The style also eases the coupling and analysis of optimisation algorithms in the CACTOS Prediction Toolkit.

TABLE OF CONTENTS

EXECUTIVE SUMMARY I

TABLE OF CONTENTS II

LIST OF FIGURES III

LIST OF TABLES V

ABBREVIATIONS & TERMS VI

I. INTRODUCTION 1

II. ARCHITECTURE 2

1. DEPLOYMENT ARTEFACTS 4

A) AVAILABLE ARTEFACTS 4

B) EXAMPLE 5

2. INFRASTRUCTURE MODELS (UPDATE) 5

A) INFRASTRUCTURE MODEL 6

B) OPTIMIZATION PLAN MODEL 10

3. APPLICATION MODELLING 10

A) APPLICATION STRUCTURE 11

B) APPLICATION BEHAVIOUR 18

C) APPLICATION USER BEHAVIOUR (PREDICTION TOOLKIT ONLY) 23

4. INTEGRATION 25

A) STARTING INDIVIDUAL VIRTUAL MACHINES 25

B) STARTING WHITE-BOX APPLICATIONS 27

C) RUNTIME MANAGEMENT 28

5. EXTENSIBLE SERVICES INFRASTRUCTURE 28

B) AUTO SCALING 33

III. PROVISIONING OF THE CACTOS TOOLKIT 35

IV. EXAMPLE USE CASE 36

REFERENCES 42

LIST OF FIGURES

FIGURE 1: CACTOS TOOLKITS ARCHITECTURE 2
FIGURE 2: AVAILABLE ARTEFACTS FOR THE CACTOS TOOLKITS 4
FIGURE 3: ARTEFACTS DEPLOYED IN THE UULM TESTBED ON OPENSTACK 5
FIGURE 4: LOGICAL DATA CENTRE MODEL 6
FIGURE 5: UPDATED STORAGE MODELLING - LIVE DATA CENTRE EXAMPLE 7
FIGURE 6: UPDATED STORAGE MODELLING - DISK MODEL EXAMPLE 8
FIGURE 7: LOGICAL LOAD MODEL 9
FIGURE 8: OPTIMIZATION PLAN ACTION STEPS 10
FIGURE 9: APPLICATION TYPES OVERVIEW 11
FIGURE 10: MODELLING INFRASTRUCTURE FOR PROVIDING AND REQUIRING INTERFACES 12
FIGURE 11: MODELLING INFRASTRUCTURE APPLICATION INTERFACES 12
FIGURE 12: APPLICATION DESCRIPTION TEMPLATES (ASSEMBLY / CONFIGURATION TIME) 14
FIGURE 13: APPLICATION INSTANCES (INSTANCES OF EXISTING TEMPLATES) 17
FIGURE 14: MODELLING INFRASTRUCTURE FOR APPLICATION BEHAVIOUR ON THE TEMPLATE LEVEL 18
FIGURE 15: MODELLING INFRASTRUCTURE FOR APPLICATION BEHAVIOUR ON THE INSTANCE LEVEL 18
FIGURE 16: BEHAVIOUR SPECIFICATION FOR BLACK-BOX APPLICATIONS 19
FIGURE 17: BEHAVIOUR SPECIFICATION FOR GREY-BOX APPLICATIONS 20
FIGURE 18: BEHAVIOUR SPECIFICATION FOR WHITE-BOX APPLICATIONS 21
FIGURE 19: APPLICATION USER BEHAVIOUR SPECIFICATION 23
FIGURE 20: EXEMPLARY SEQUENCE FOR PROCESSING INCOMING VM REQUESTS 26
FIGURE 21: EXEMPLARY SEQUENCE FOR PROCESSING INCOMING APPLICATION REQUESTS 27
FIGURE 22: RUNTIME MANAGEMENT INTERFACE 28
FIGURE 23: OPTIMISATION SERVICE TEMPLATE 29
FIGURE 24: OPTIMISATION SERVICE IMPLEMENTATION EXAMPLE 30
FIGURE 25: OPTIMISATION SERVICE REGISTRY 30
FIGURE 26: EXAMPLE OF COMPLEXITY FOR OPTIMISATION SERVICE 31
FIGURE 27: PLACEMENT SERVICE ARCHITECTURE 31
FIGURE 28: EXAMPLE OF COMPLEXITY FOR PLACEMENT SERVICE 32
FIGURE 29: BEHAVIOUR INFERENCE SERVICE ARCHITECTURE 32
FIGURE 30: EXAMPLE OF COMPLEXITY OF BEHAVIOUR INFERENCE SERVICE 33
FIGURE 31: INTERFACE FOR AUTO SCALER INTEGRATION IMPLEMENTATIONS 33
FIGURE 32: INTERFACE OF THE AUTO SCALER 34
FIGURE 33: DATAPLAY MODELLING REFERENCE 36
FIGURE 34: DATAPLAY RESOURCE REQUIREMENTS (ALL HAVE 20 GB STORAGE AND ARE BASED ON UBUNTU 15.10 IMAGES 37
FIGURE 35: DATAPLAY FLAVOURS IN THE MODEL 37
FIGURE 36: DATAPLAY STRUCTURAL ARCHITECTURE VIEW 38
FIGURE 37: DATAPLAY INTERFACES 38
FIGURE 38: DATAPLAY APPLICATION USAGE SCENARIO 38
FIGURE 39: DATAPLAY BEHAVIOUR TEMPLATE 39
FIGURE 40: DATAPLAY BEHAVIOUR INSTANCE AND TEMPLATE LEVEL 40
FIGURE 41: DATAPLAY SEQUENCE EXAMPLE 40

LIST OF TABLES

ABBREVIATIONS & TERMS

Abbreviation	Description
CACTOS	Context-Aware Cloud Topology Optimisation and Simulation
CI	Continuous Integration
COTS	Commercial-Off-The-Shelf
EDP2	Experiment Data Persistency & Presentation
FCO	Flexiant Cloud Orchestrator
FDL	Flexiant Development Language
rpm	revolutions per minute
SLA	Service Level Agreement
SVN	Apache Subversion
OSP	OpenSourceProjects.eu
QoS	Quality of Service
VM	Virtual Machine
Flavour	Virtual hardware templates defining storage, memory, and number of virtual cores. Flavours can be tenant-specific.

I. INTRODUCTION

In *Infrastructure as a Service (IaaS*) cloud data centres, customers run their software on the virtualised infrastructure of a data centre. Optimization is possible as VMs often stay idle and rarely use all available resources. The efficient utilisation of the underlying physical infrastructure including management and topology optimisation determines the costs and ultimately the business success for data centre operators.

The CACTOS Toolkit Version 2 described in this document provides the implementation for such an automated infrastructure management in two Cloud middlewares – FCO and Open Stack.

This document is accompanying material for the prototype deliverable D5.2.2. Please refer to accompanying material for the prototype deliverable (D5.2.1 CACTOS Toolkit Version 1) for an overview on the CACTOS toolkits and an exemplary use case. There are two CACTOS toolkits: The CACTOS Runtime Toolkit (label before year 1: CACTOS Toolkit) and the CACTOS Prediction Toolkit. The CACTOS Runtime Toolkit contains the tools CactoScale and CactoOpt and is described in this deliverable.

The architecture is presented in section II with a focus on the updated parts of the models in section II.2, a focus on application modelling in section II.3, and technical details on the implementation and integration in section II.4. Section III references the guidelines on how to get the CACTOS tools provisioned in your tested. Section IV presents an exemplary use case on how a complex VM-spanning application looks like in the models.

Related deliverables are (D5.2.1 CACTOS Toolkit Version 1), (D5.1 Model Integration Method and Supporting Tooling), (D5.4 Evaluation Methodology for the CACTOS Runtime and Prediction Toolkit), and (D5.3 Operational Small Scale Cloud Testbed Managed by the CACTOS Toolkit). The accompanying document for the prototype deliverable (D5.2.1 CACTOS Toolkit Version 1) described the architecture of the CACTOS toolkit and is updated by this deliverable. (D5.1 Model Integration Method and Supporting Tooling) describes the tools and scope in detail, presents the data centre model, integration and interaction of tools, architecture, and development and build infrastructure. This deliverable extends (D5.1 Model Integration Method and Supporting Tooling) and provides an updated view on the models storing all information about a data centre and the architecture. (D5.4 Evaluation Methodology for the CACTOS Runtime and Prediction Toolkit) focussed on metrics and updated the overview and architecture from (D5.1 Model Integration Method and Supporting Tooling). This deliverable extends the description of (D5.4 Evaluation Methodology for the CACTOS Runtime and Prediction Toolkit). (D5.3 Operational Small Scale Cloud Testbed Managed by the CACTOS Toolkit) focussed on an operational version of the CACTOS toolkit in a small-scale testbed. (D5.3 Operational Small Scale Cloud Testbed Managed by the CACTOS Toolkit) described the provisioning and deployment in the testbeds FCO and Open Stack. This deliverable updated the description in (D5.3 Operational Small Scale Cloud Testbed Managed by the CACTOS Toolkit) with respect to the toolkit architecture and deployment artefacts.

This document is structured as follows. Section II shows the changes on the architectural level of CACTOS and presents the models, which are used by all tools in order to exchange information, as well as Integration and 3rd party extension points. Section III provides information on the provisioning of CACTOS. Section IV provides information on examples of using CACTOS.

II. ARCHITECTURE

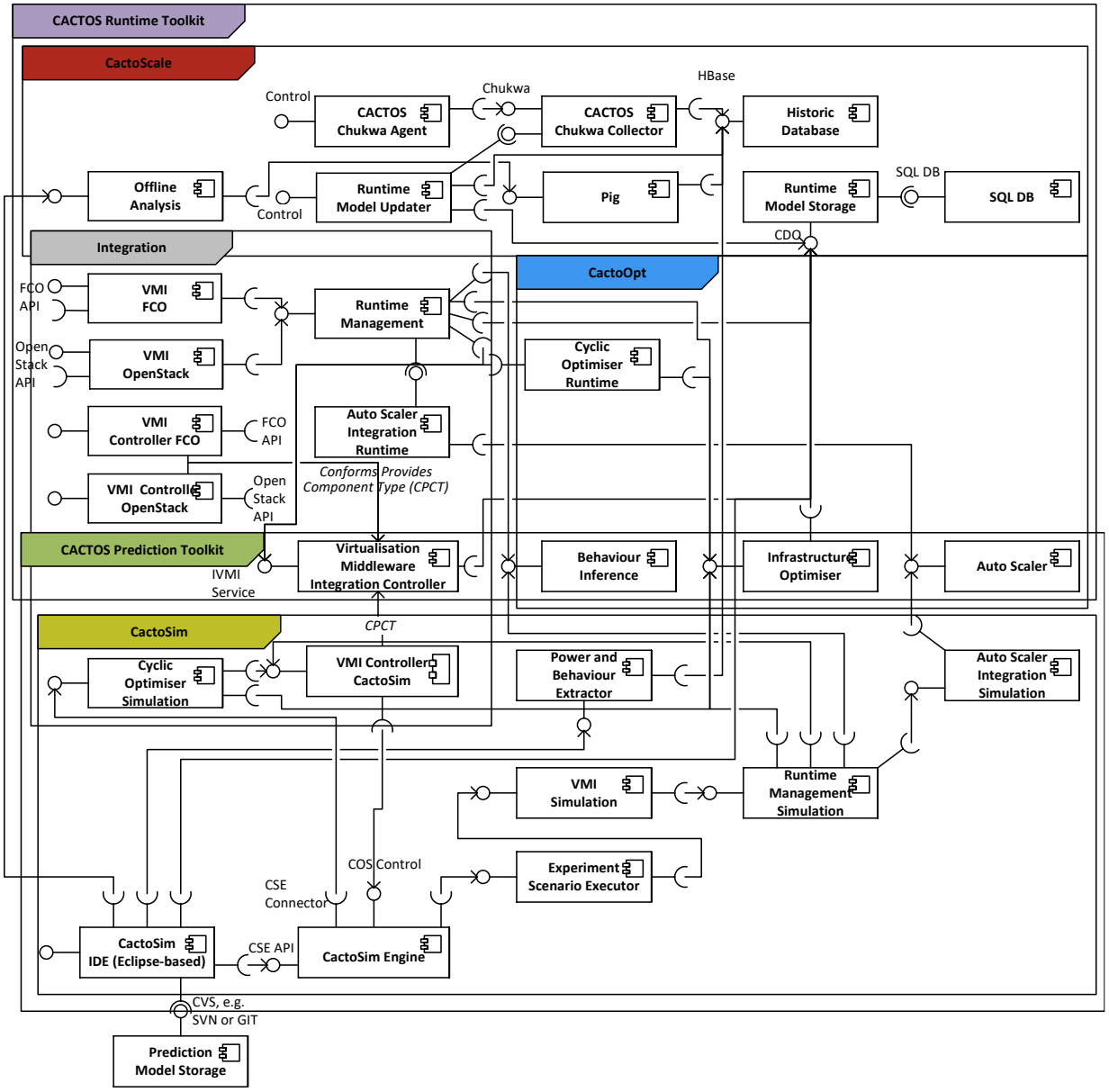

Figure 1: CACTOS Toolkits Architecture

Figure 1 shows the architecture of both CACTOS toolkits. It depicts the contents of the tool areas CACTOS Runtime Toolkit, the CACTOS Prediction Toolkit, and the tools CactoScale, CactoOpt, CactoSim, and Integration functionality.

The tool area *CactoScale* consists of the components CACTOS Chukwa Agent, CACTOS Chukwa Collector, a Historic Database, Offline Analysis, Pig, Runtime Model Updater, the Runtime Model Storage, and a SQL DB. Chukwa Agents gather data in the virtual and physical infrastructure and send it to the Chukwa Collector. The collector stores the information in the Historic Database to enable further analysis. It also provides the data to the Runtime Model Updated that stores the information on the data centre in the Runtime Model Storage. The Runtime Model Storage uses an

SQL DB for persisting the information. Offline Analysis uses Pig capabilities to analyse the Historic Database.

The tool area *CactoOpt* consists of the components Cyclic Optimiser Runtime, Behaviour Inference, Infrastructure Optimiser, and Auto Scaler. The Cyclic Optimiser Runtime is responsible for optimizing a data centre in regular intervals and therefore calls the Infrastructure Optimiser. The Infrastructure Optimiser suggests improved deployments and node states changes, e.g. switching unused nodes off. It is also responsible to suggest placements for incoming virtual machines. The Behaviour Inference allows to reason on the expected behaviour of incoming virtual machines, e.g. for supporting the best placement. The Auto Scaler supports decisions on the scaling factor. The scaling factor is the best number of an application's connected virtual machines for a horizontally scaled application instance.

The tool area *Integration* consists of the components Virtualization Middleware Infrastructure (VMI) components, VMI Controller components, Runtime Management, Auto Scaler Integration Runtime, and Auto Scaler Integration Simulation. The VMI components are proxy interfaces for supported VMI platforms, e.g. FCO or Open Stack. Users issue their calls to those components and cannot distinguish between CACTOS-enabled and not enabled data centres. They provide additional functions to manage applications running in CACTOS-enabled VMIs. The VMI components translate to information in CACTOS and forward relevant calls to the Runtime Management. The Runtime Management is responsible to manage applications and VMs in CACTOS. It updates information in the Runtime Model Storage, infers behaviour for incoming VMs using Behaviour Inference, requesting placement suggestion for the Infrastructure Optimiser, ensuring that scalable connectors are automatically scaled for white-box applications using Auto Scaler Integration Runtime, and finally ensures that incoming user requests for applications or VMs are executed using the VMI Controller. The VMI Controller are VMI specific and translate from CACTOS to the used VMI. Auto Scaler Integration Runtime requests scaling improvement suggestions from the Auto Scaler using the appropriate intervals specified in the application models for each connector.

The tool area *CactoSim* consists of the components Cyclic Optimiser Simulation, VMI Controller CactoSim, Power and Behaviour Extractor, Auto Scaler Integration Simulation, CactoSim IDE, CactoSim Engine, Experiment Scenario Executor, VMI Simulation, and Runtime Management Simulation. The CactoSim IDE allows querying data centre models from a Runtime Model Storage. The resulting models can be extended or modified according to the targeted simulation scenario and required metrics. Power models and the Behaviour of Black-Box VMs can be retrieved from a Runtime Model Storage as well using wizards of the Power and Behaviour Extractor. Models are stored locally within the IDE and can be versioned and shared (including simulation results) using the Prediction Model Storage. The Offline Analysis supports to include analyses of CactoScale in a model. A data centre model can be simulated using the CactoSim Engine. The CactoSim Engine supports dynamic incoming and leaving VMs and applications according to the specification in the Experiment Scenario part of the data centre model. The Experiment Scenario Executor carries out the necessary adaptations during the specified point in simulation time. It issues changes to the logical model part using the VMI Simulation, analogously to the VMIs of the Runtime Toolkit. The physical model part is modified directly. The VMI Simulation forwards requests to the Runtime Management Simulation. It differs from the Runtime Management counterpart in two aspects. It uses simulation time instead of real time and it has direct access to in-memory data centre models instead of using a Runtime

Model Storage. The latter applies for the Cyclic Optimiser Simulation, VMI controller CactoSim, and Auto Scaler Integration Simulation. Otherwise, the components of the Runtime Toolkit are re-used.

The Prediction Model Storage can be any file-based versioning system, such as SVN or GIT. The CACTOS Prediction Toolkit works without this component. Therefore it is depicted outside of the tool areas. It allows storing different version of data centre models including prediction results.

1. DEPLOYMENT ARTEFACTS

This section points out the available deployment artefacts. Refer to section III for information on provisioning CACTOS in your virtualisation infrastructure.

a) AVAILABLE ARTEFACTS

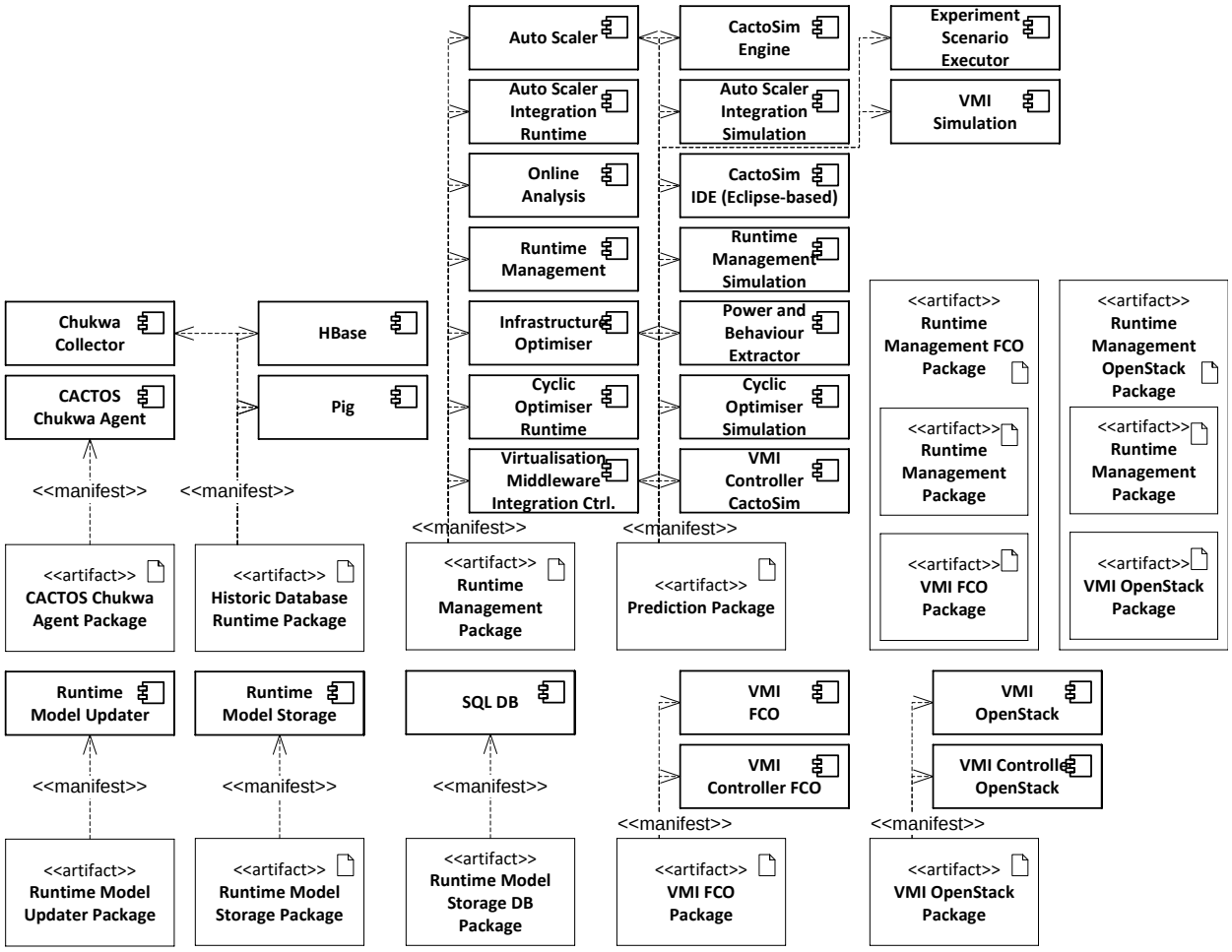

Figure 2: Available Artefacts for the CACTOS Toolkits

Figure 2 shows the deployment artefacts for both toolkits and their contents with respect to the components of the architecture. This information is relevant for developers. End users and data centre operators should follow the guidelines presented in section III to provision the corresponding artefacts. The Runtime Toolkit consists of the following artefacts: CACTOS Chuckwa Agent Package, Historic Database Runtime Package, a Runtime Management Package for the used VMI, Runtime

Model Update Package, Runtime Model Storage Package, and Runtime Model Storage DB Package. The Prediction Toolkit consists of the Prediction Package artefact only.

b) EXAMPLE

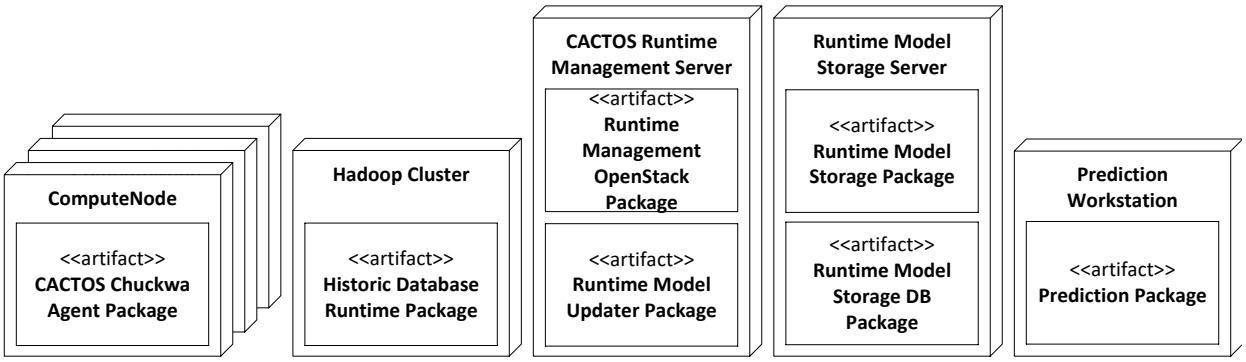

Figure 3: Artefacts deployed in the UULM testbed on OpenStack

Figure 3 provides an example how the artefacts for a full CACTOS-enabled virtualisation infrastructure with both toolkits can look like. The depicted Compute Nodes are the nodes managed by CACTOS and form the actual data centre.

2. INFRASTRUCTURE MODELS (UPDATE)

This section describes the infrastructure models modified since D5.2.1. Exemplary model instance are also linked in section IV.

a) INFRASTRUCTURE MODEL

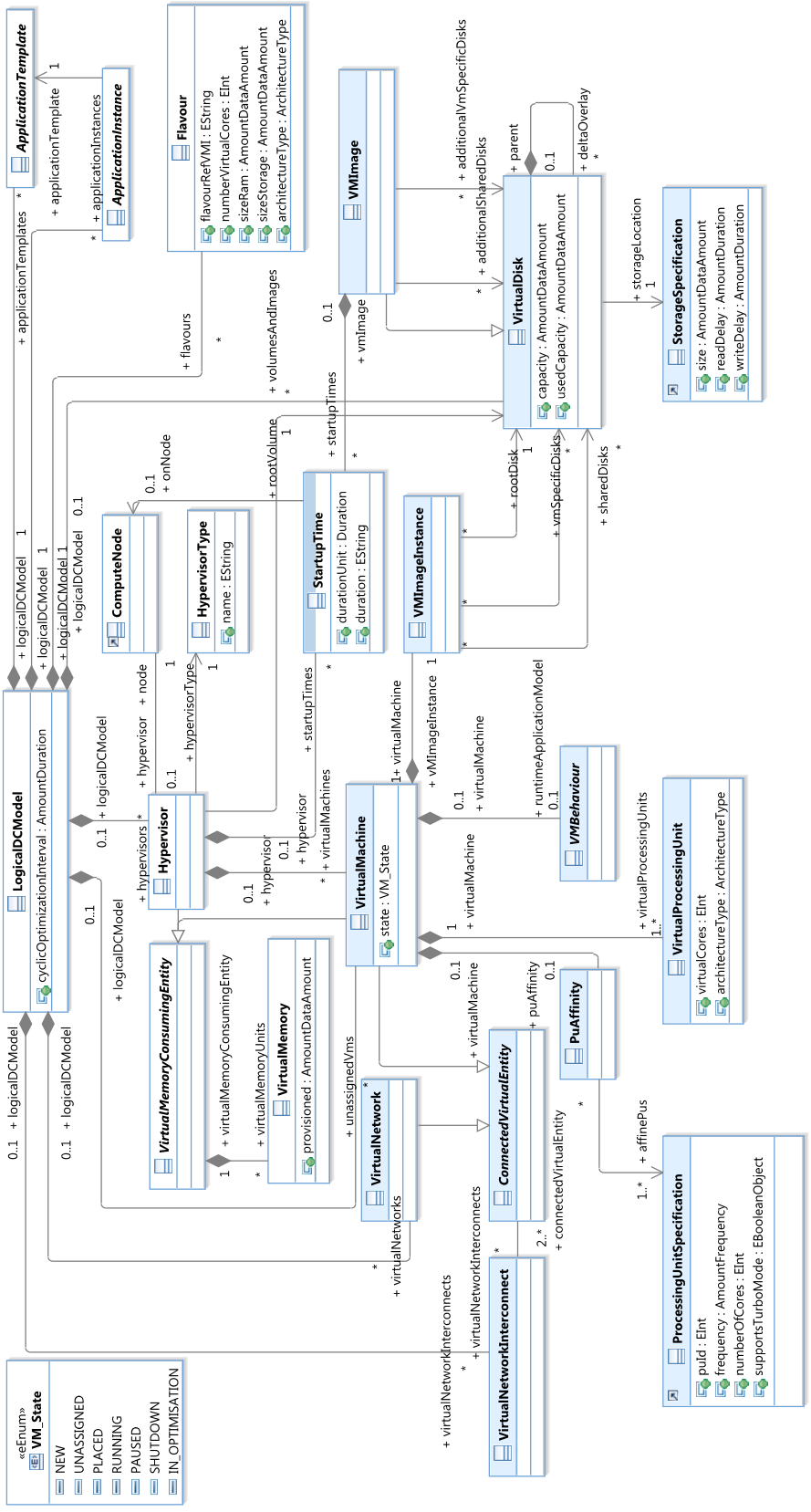

Figure 4: Logical Data Centre Model

Figure 4 shows the final result of all changes on the logical data centre model for completeness and to provide context. The individual changes are described in the following. The older elements are described in (D5.1 Model Integration Method and Supporting Tooling) section IV.

MEMORY CONSUMPTION OF HYPERVISORS. In the previous version the memory consumption of hypervisors was not stated explicitly. The consumption can now be specified for Hypervisor in the same way as for VirtualMachines. The new abstract class VirtualMemoryConsumingEntity was introduced to allow this consistent handling. Placement algorithm can now support Hypervisor with different memory consumption.

CONFIGURATION OF THE CYCLIC OPTIMISER INTERVAL. There was no explicit statement in the model. Now, the interval is stored in the models in the attribute cyclicOptimizationInterval of the element LogicalDCModel. This attribute can be changed at runtime, e.g. by a data centre operator.

FLAVOUR SUPPORT. In the first version of the models, flavours (see Abbreviations and Terms) were resolved in a VMI-specific way based on tags used in instantiation calls. Support for auto scaling requires to automatically determine the right flavour for a new VM. Flavours are stored directly at the LogicalDCModel (right-hand side, third form top). flavourRefVMI is a UUID and references the Flavour id used in the virtualisation infrastructure. The other attributes numberVirtualCores, sizeRam, sizeStorage are the typical properties of resource requirement specifications in virtualisation infrastructures.

STORAGE MODELING. The older version only knew about volumes and didn't take into account the representation of VMImages and (Virtual) Disks, that root file systems of VMs could be remote (see Figure 5 for a typical situation), and disk performance measurements should be easy to map regardless if local or remote.

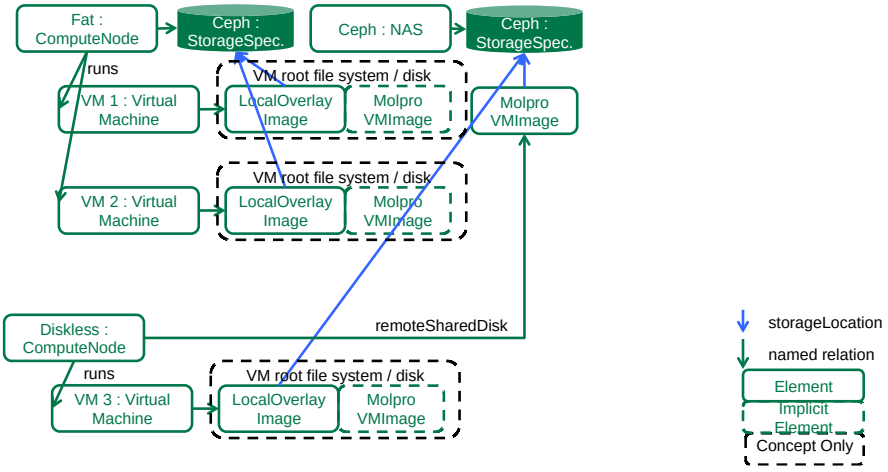

Figure 5: Updated Storage Modelling - Live Data Centre Example

Figure 5 shows a situation with two physical nodes: A Fat node with local disc storage using CEPH storage distribution technology and a Diskless node without a local disk. The diskless node uses a Network Attached Storage (NAS) that operates uses CEPH technology. VM 1 runs a Molpro VMImage and the changed data of this specific virtual machine is stored in a LocalOverlay image. The VMImage and the LocalOverlay are stored on the CEPH storage of the Fat node. The same applies for

VM 2. VM 3 runs on the Diskless node. It uses the Molpro VMImage stored on the CEPH storage of the NAS. It adds a LocalOverlay as above to store local data and let the VMImage remain unchanged.

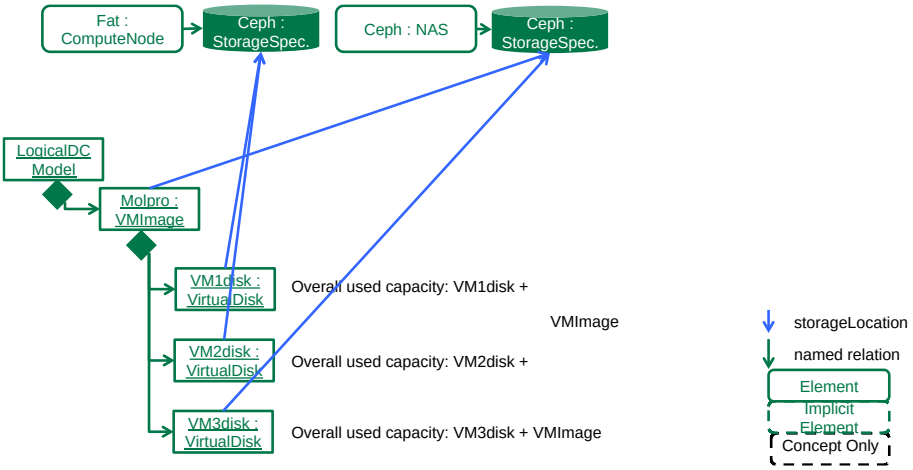

Figure 6: Updated Storage Modelling - Disk Model Example

Figure 6 shows a situation and focuses on how different overlays are represented in the infrastructure model of the data centre. The LogicalDCModel is the top model element. The Molpro VMImage is stored on the CEPH storage on the NAS node. VM1disk and VM2disk store their local overlay information on the CEPH storage of the Fat node and are technically contained in the VMImage model element. VM3disk is stored on the CEPH storage of NAS. It is still shown as contained in the Molpro VMImage element because it's based on it. The overall used capacity is the capacity of VM1disk + VM2disk + VMImage on CEPH storage at the Fat node and the capacity of VM3disk + VMImage on the CEPH sorage of NAS.

The update addresses all of these limitations. In the bottom right section of Figure 7, there is the new VirtualDisk. A VirtualDisk has a capacity and usedCapacity. It contains 'child' disks that contain the difference or delta when a virtual disk is mounted but modifications are written to a virtual disk on top of that. This is modelled with the deltaOverlay reference. See also Figure 6 for a visualization. An example model is also referenced in section IV. A VirtualDisk references the storageLocation where it resides physically. A VMImage is a VirtualDisk but can also point to additionalSharedDisks that will be made available when the VMImage itself is used. A VirtualMachine has exactly one VMImageInstance. Modelling this as two separate classes is due to a separation of concerns. A VMImageInstance must reference a root disk, which can be local or remote. It can additionally specify vmSpecificDisks and sharedDisks. Write requests to the former one will end up in deltaOverlay virtual disks. The second ones would be affected directly.

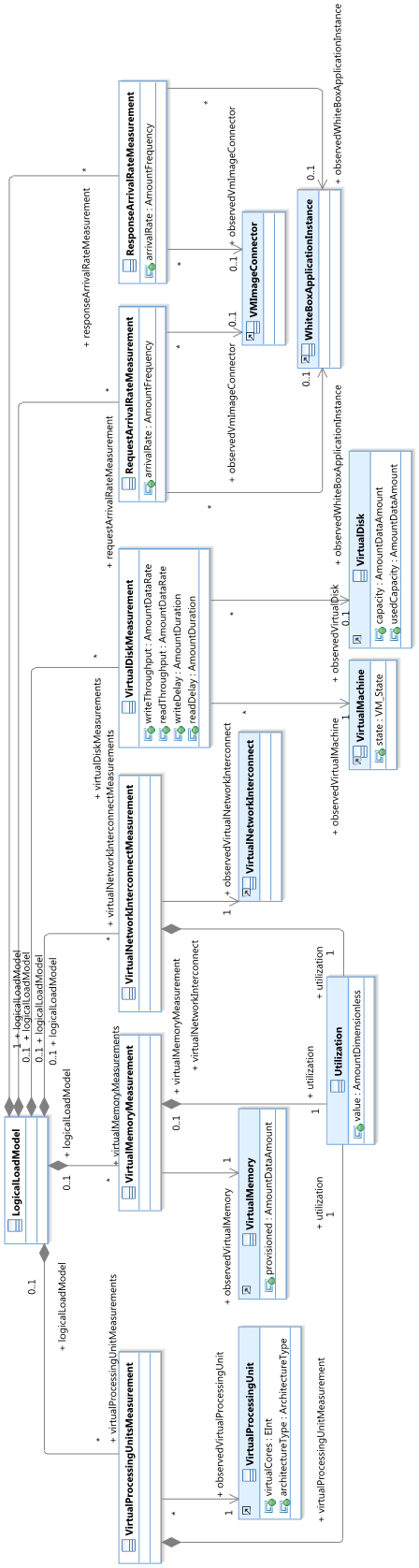

Figure 7: Logical Load Model

Figure 7 shows the final result of all changes on the logical data centre model for completeness and to provide context. The individual changes are described in the following. The older elements are described in (D5.1 Model Integration Method and Supporting Tooling) section IV.

UPDATED STORAGE MODELLING. The update lead to a removal of the VolumeMeasurement and introduction of the VirtualDiskMeasurement (fourth from left) as replacement. It can now additionally store delay information on read and write access.

ARRIVAL RATE MEASUREMENTS. Auto scaling support requires to know about arrival rates in order to reason on the optimal amount of VMs. The elements RequestArrivalRateMeasurement and ResponseArrivalRateMeasurement were introduced (right and second from right). The measurements reference the connection for which they are measured and the application instance.

STARTUP TIME MONITORING. The boot times of physical machines and virtual machine images could not be stated in the models. The following changes have been made: The update introduced the StartupTime element to state the duration as a Stochastic Expression and the durationUnit. This allows using distribution functions and proper units, e.g. seconds or minutes. StartupTime instance can be contained in Hypervisor elements in order to state how long it took to boot a physical machine including the hypervisor until it is ready. StartupTime instance can also be contained in VMImage elements in order to state how long the image took to be started on a node.

b) Optimization Plan Model

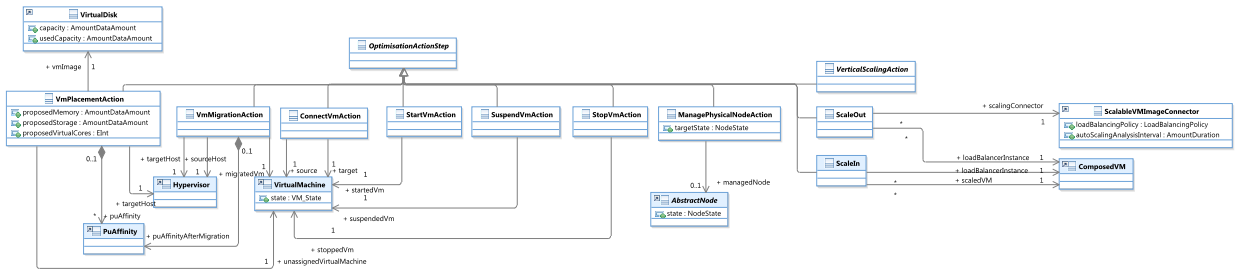

Figure 8: Optimization Plan Action Steps

Figure 8 shows the final result of all changes on the optimization plan model for completeness and to provide context. The individual changes are described in the following. The older elements are described in (D3.1 Prototype Optimization Model) section IV.3.

Scaling actions for white-box applications were introduces (bottom right of Figure 8). ScaleIn allows removing a VM from a horizontally scaled application and disconnecting it from its load balancer. ScaleOut allows creating a VM according to the application specification and wiring it in the end, e.g. with the load balancer, using the ConnectVmAction (third from left).

3. Application Modelling

This section describes the extensions that support VM-spanning applications. The modelling of single-VM-based applications is extended in a consistent way with single-VM applications. An exemplary VM-spanning white-box model instance is presented in section IV.

a) Application Structure

	Black-Box	Grey-Box	White-Box
Type	Any	Batch	Interactive
Source	Historic traces	Manual analysis	Manual analysis
Effort	Low / minutes	High / days to weeks	High / days to weeks
Tasks for Creation	▪ Load trace from Historic Database	▪ Analyse phases and resource consumption of application	▪ Specify architecture ▪ Analyse resource consumption and control flow ▪ Analyse usage scenarios
Metrics	+ Resource utilization + Power consumption	+ (All of Black-Box) + Response Time	+ (All of Grey-Box) + Number of scaled VMs
Advantage Limitation	+ Optimize resource usage − Single VM only − Effect of delay and user experience limited	+ (All of Black-Box) + Estimate time to completion − Single VM only	+ (All of Black-Box) + Detailed user experience metrics + Multiple and distributed VMs

Figure 9: Application Types Overview

Figure 9 shows the different application types that CACTOS supports. The White-Box application type has been newly introduces since (D5.2.1 CACTOS Toolkit Version 1). The figure provides information on how to get the required information to build a model, a rough effort estimate, typical creation tasks, relevant application metrics, and advantages as well as limitations. The source 'Historic trace' means that a model can be automatically inferred based on information in the Historic Database. The source 'Manual analysis' means that experts analyse an application including information from the Historic Database but also additional analysis. This allows insights into the application and specific measurement points depending on the application logic itself.

The following outlines more details on the typical tasks required for constructing application models (*Tasks for Creation* in Figure 9):

- Load trace from Historic Database
  - Use Wizard in CactoSim
  - Alternative: Use Hbase API for direct access to historic data and analytics
- Analyse phases and resource consumption of application
  - Reason on number of phases and resource consumption per phase based on measurements in the Historic Database, application log files, and additional analyses of the grey-box application, e.g. with code review and tracing tools
- Specify architecture
  - State the interfaces in the control-flow of user requests across VMs
  - State scaled VMs
  - State default Flavours for each VM
- Analyze resource consumption and control flow
  - State the resource consumption from a resource load perspective during user request processing in each VM, e.g. based on the Historic Database, application log files, or additional analyses of the application
- Analyse usage scenarios

- State user types and how they typically use the application from a resource load perspective, e.g. based on the Historic Database, use case definitions, additional analyses, or application log files
- State how often users of each type arrive, e.g. based on the Historic Database, application log files or additional analyses

Supporting White-Box applications, which span several VMs, requires defining the structure of the application and which VMs are required to run an instance. This specification is done via Application Templates and is similar to a blueprint or deployment model if you are more familiar with those non-CACTOS terms. Auto scaling further requires knowing about the interfaces of applications in order to reason on throughput and optimal number of VMs.

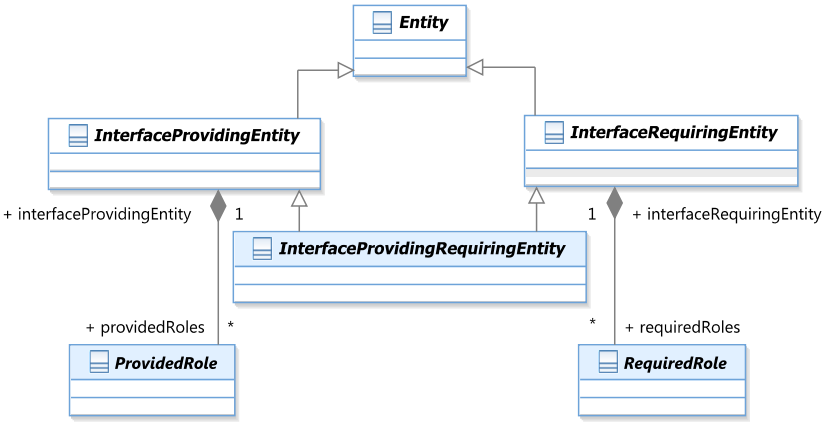

Figure 10: Modelling Infrastructure for Providing and Requiring Interfaces

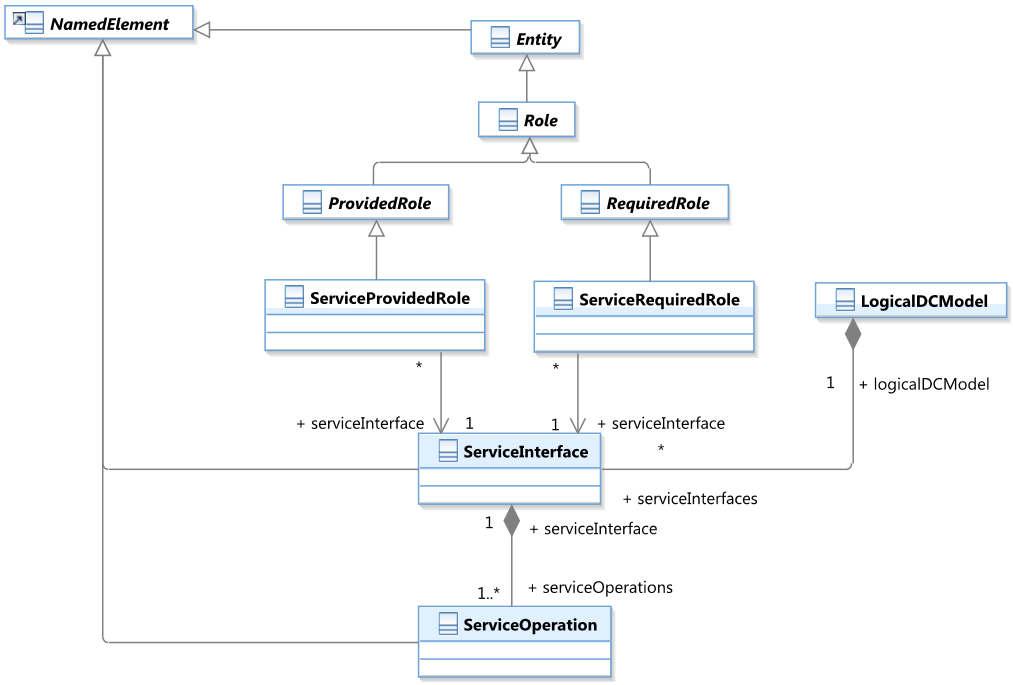

Figure 11: Modelling Infrastructure Application Interfaces

Figure 10 and Figure 11 show the modelling basics required to state interfaces for VMs. Interfaces can be provided or required by a VM. A single VM can provide or require more than an interface. The same interface can be used in different roles, e.g. accessing split user and media database both use a SQL interface but need to be kept separate. CACTOS groups different sets of operations on the same interface in ServiceInterfaces. A ServiceInterface consists of ServiceOperations, e.g. the http interface of a get, post, put, and delete operation.

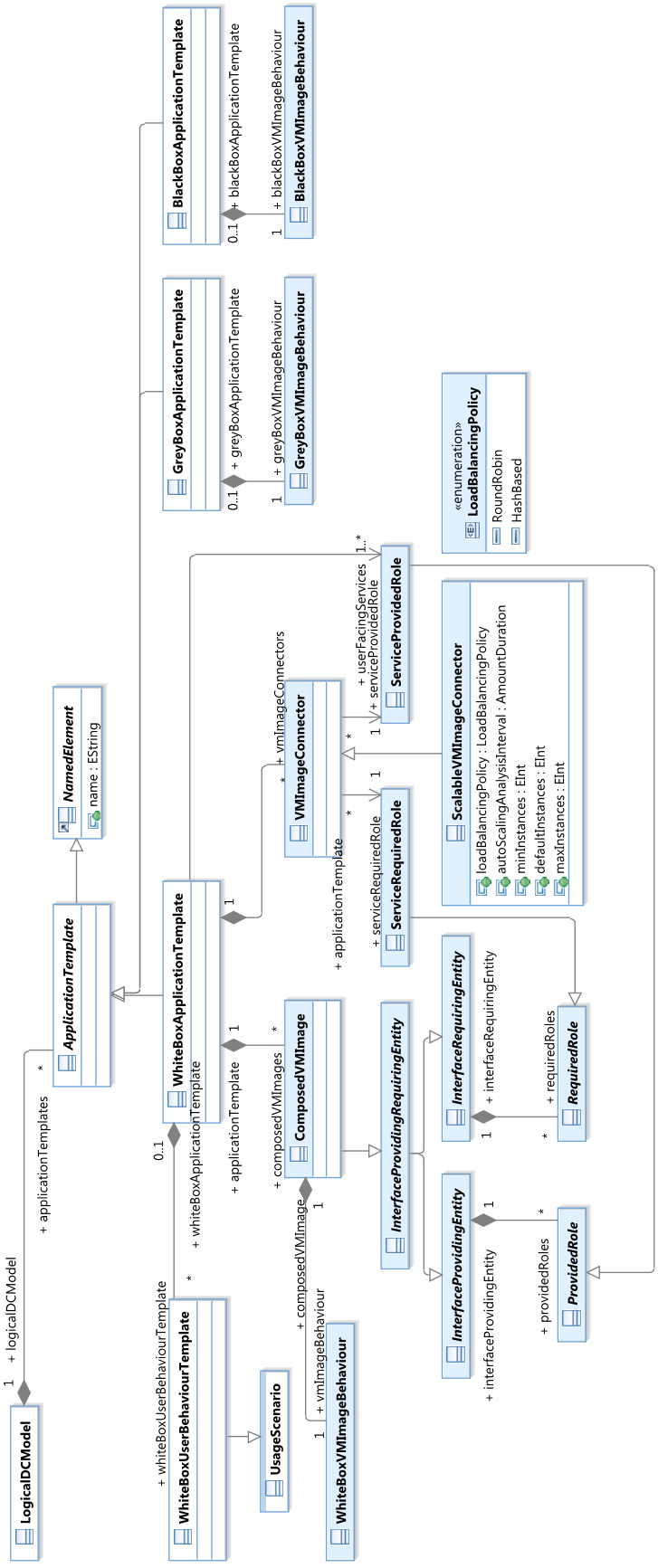

Figure 12: Application Description Templates (Assembly / Configuration Time)

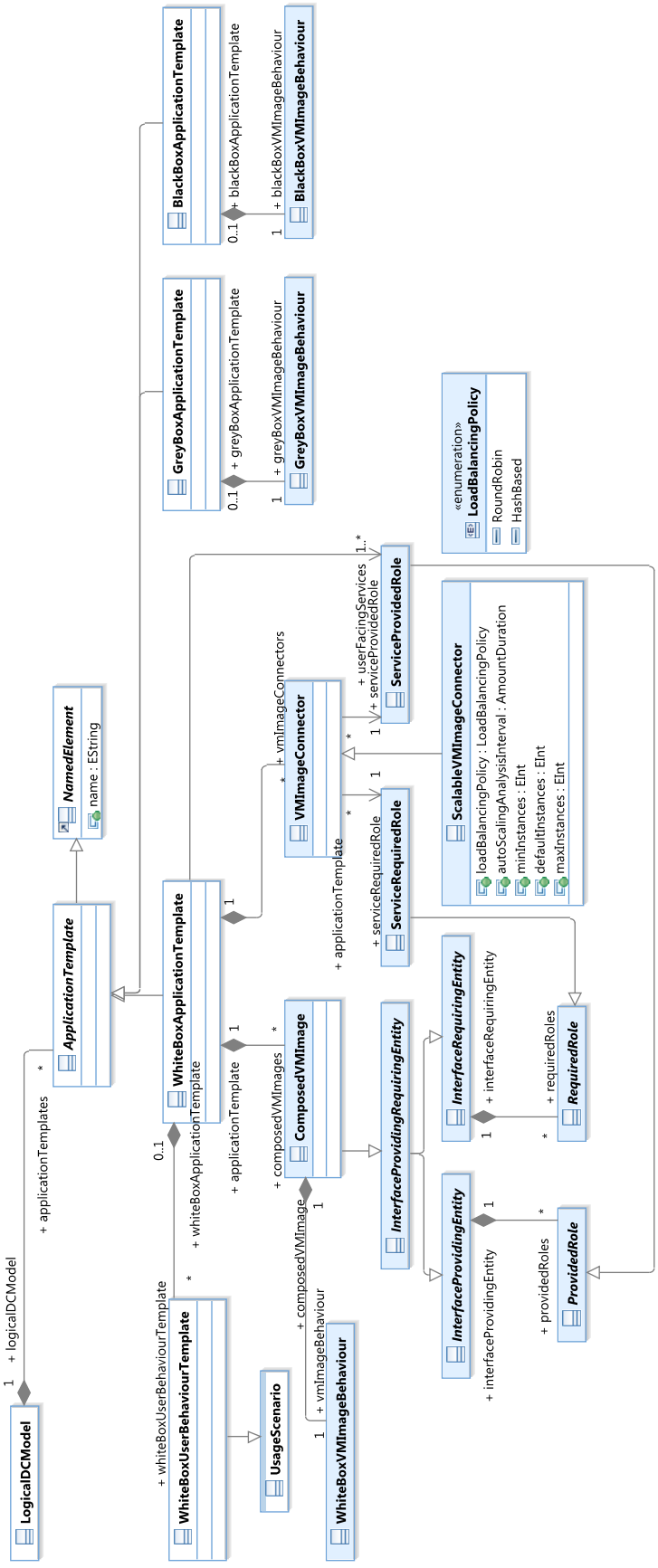
LogicalDCModel
+ logicalDCModel
+ applicationTemplates
ApplicationTemplate
NamedElement
name : EString
WhiteBoxApplicationTemplate
+ whiteBoxApplicationTemplate
+ applicationTemplate
GreyBoxApplicationTemplate
+ greyBoxApplicationTemplate
GreyBoxVMImageBehaviour
+ greyBoxVMImageBehaviour
BlackBoxApplicationTemplate
+ blackBoxApplicationTemplate
BlackBoxVMImageBehaviour
+ blackBoxVMImageBehaviour
«enumeration»
LoadBalancingPolicy
RoundRobin
HashBased
ScalableVMImageConnector
loadBalancingPolicy : LoadBalancingPolicy
autoScalingAnalysisInterval : AmountDuration
minInstances : EInt
defaultInstances : EInt
maxInstances : EInt
VMImageConnector
+ vmImageConnectors
+ applicationTemplate
+ userFacingServices
+ serviceProvidedRole
ServiceProvidedRole
ServiceRequiredRole
+ serviceRequiredRole
ComposedVMImage
+ composedVMImages
+ composedVMImage
InterfaceProvidingRequiringEntity
InterfaceRequiringEntity
+ interfaceRequiringEntity
+ requiredRoles
RequiredRole
InterfaceProvidingEntity
+ interfaceProvidingEntity
+ providedRoles
ProvidedRole
WhiteBoxUserBehaviourTemplate
+ whiteBoxUserBehaviourTemplate
UsageScenario
WhiteBoxVMImageBehaviour
+ vmImageBehaviour

Figure 12 shows all available templates for modelling applications in CACTOS – Black-Box (right), Grey-Box (second from right), and White-Box Application Templates (third from right). Figure 4 shows on the top right the location of ApplicationTemplate and ApplicationInstance within the LogicalDCModel. Black- and Grey-Box Application Templates consist only of a single VM and are therefore very compact. You can find the description of the corresponding VMImageBehaviour specifications below in the following Section Application Behaviour. White-Box Application Templates specify the structure – the ComposedVMImages that are the parts and are required to run the application and the VMImageConnectors to wire those composed images. White-Box Application Templates furthermore point to the userFacingServices that are not used internally but are accessible from the outside. White-Box Application Templates can have a (White-Box) Application User Behaviour. This is relevant for Prediction Toolkit only and described below in the Application User Behaviour section. ComposedVMImage can provide and require services, which is visible by the generalization relation to InterfaceProvidingRequiringEntity. VMImageConnector reference the provided and required service roles that are connected. VMImageConnector cannot be scaled but ScalableVMImageConnector can, e.g. to represent a load-balanced connection. ScalableVMImageConnector specify the minimal number of instances that should be connected, the number of instances that should be connected on instantiation as a default, and the maximum number of instances allowed. The autoScalingAnalysisInterval determines the interval for optimising the number of connected VMs. Two LoadBalancingPolicy are supported: Round-Robin and Hash-based. The former one is a statistical distribution across the VMs, the latter one allows binding to specific VMs connected based on a hash tag in incoming requests. A typical use of hash-based routing is that the same user's requests are routed to the same VM.

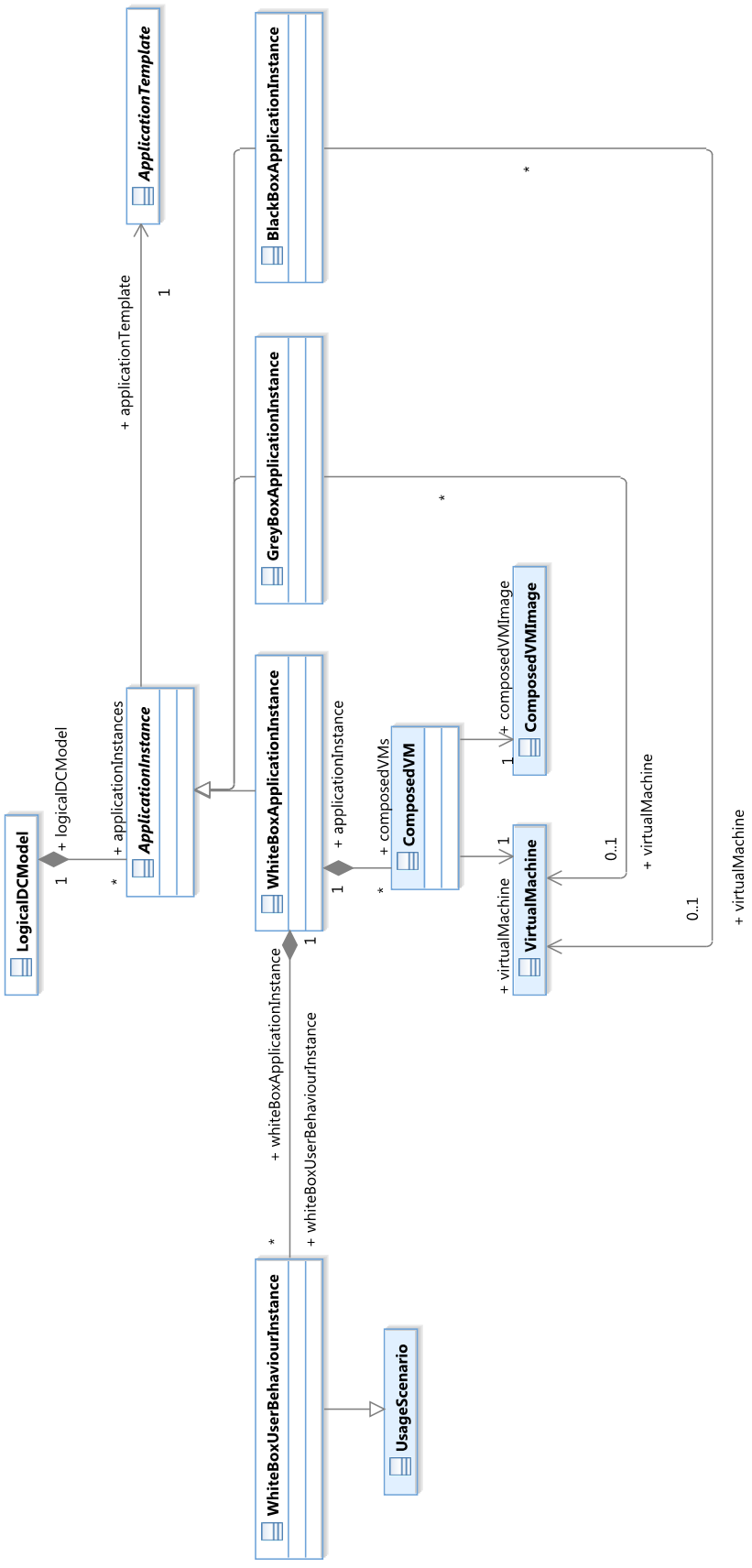

Figure 13: Application Instances (Instances of existing Templates)

Figure 13 shows application instances. Instances always refer to their corresponding application template. Black-Box (right) and Grey-Box Application Instance (second from right) reference the single Virtual Machine that matches the instance. There is exactly one VM per instance. White-Box Application Instance (third from right) can have a (White-Box) Application User Behaviour. This is relevant for Prediction Toolkit only and described below in the Application User Behaviour section below. The application template determines the possible wiring. The instance itself only tracks the ComposedVMs that belong to the instance. They can change due to scaling of VMs. Each ComposedVM references the VM running a composedVMImage, which is described in the template.

b) Application Behaviour

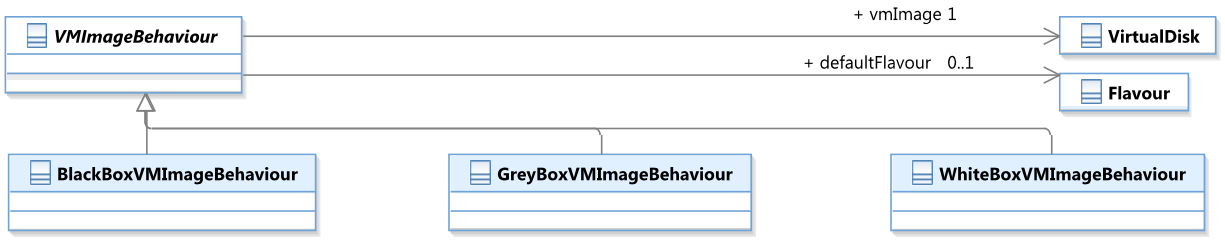

Figure 14: Modelling Infrastructure for Application Behaviour on the Template Level

Figure 14 shows the modelling infrastructure for application behaviour on the level of application templates and VMImages. A VMImageBehaviour must reference the vmImage that shows the specified behaviour. It can reference a defaultFlavour that is used when instantiating the VMImage with this behaviour. There is one subclass for each application behaviour type. The application behaviour types are described in the following paragraphs after the VMBehaviour descriptions.

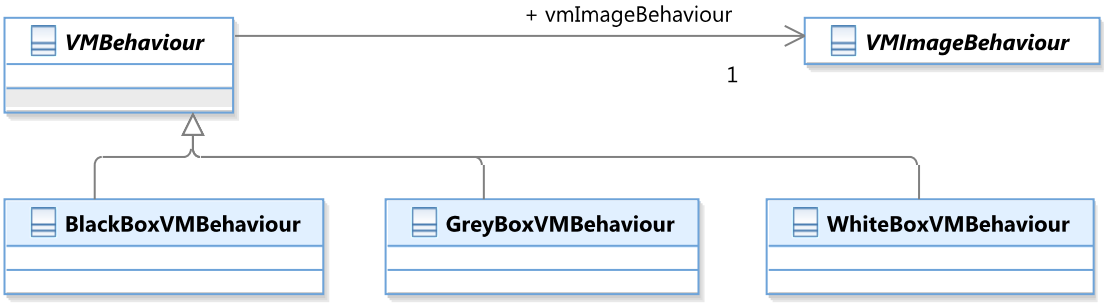

Figure 15: Modelling Infrastructure for Application Behaviour on the Instance Level

Figure 15 show the modelling infrastructure for application behaviour on the level of instances and VirtualMachines. A VMBehaviour must reference its template and therefor the VirtualDisk and Flavour as well. This ensures navigation from instance to template and that all descriptions for an application are available. There is one subclass for each application type. The application behaviour types are described in the following paragraphs.

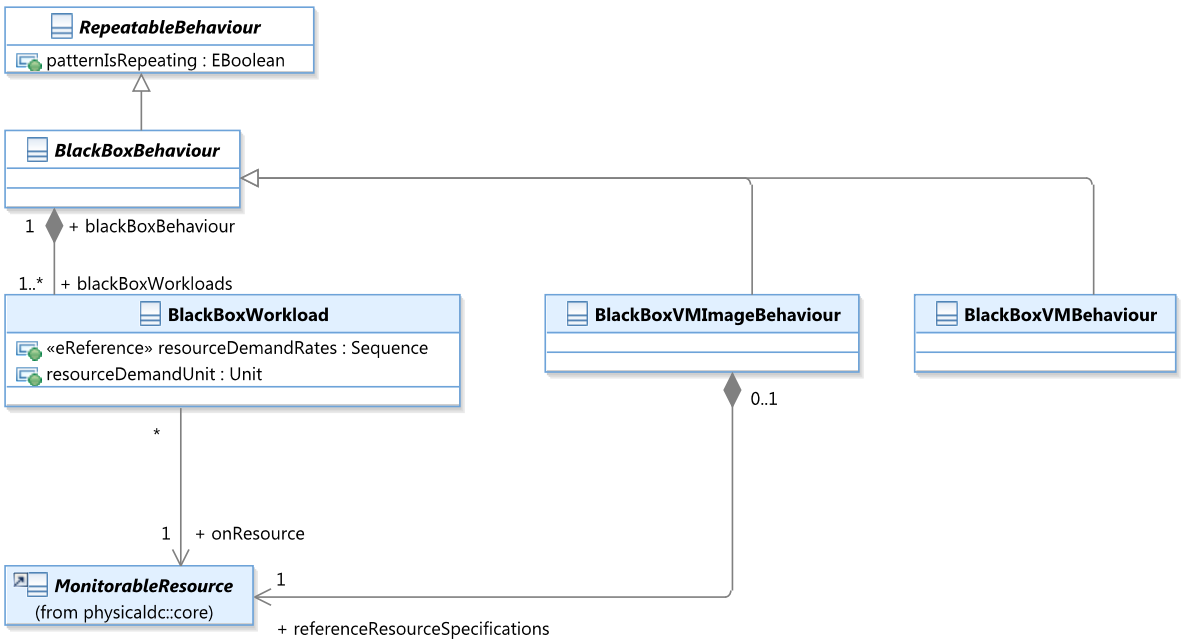

Figure 16: Behaviour specification for Black-Box Applications

Figure 16 shows behaviour specifications for Black-Box applications. Black-Box Behaviour can be repeatable. This means that the behaviour pattern defined runs in a loop and restarts from the beginning once it's finished. A Black-Box Behaviour consists of Black-Box Workload specifications. These specifications express the resourceDemandRates over time. The specification of the time unit, e.g. seconds, and the rate is kept separate for technical reasons. A Black-Box Workload is specified for a single resource, e.g. a processing unit or storage. The onResource reference points to that resource. All blackBoxWorkloads of a Black-Box Behaviour run in parallel and cause load on the resources. The Black-Box VM Behaviour (right) on the instance level is a Black-Box Behaviour. The Black-Box VM Image Behaviour is a Black-Box Behaviour as well and must reference the resource specifications used in the behaviour description. This is required as it cannot be guaranteed that the resource specifications are available in a data centre description when a template is described. They are available on the instance level as the instances actually run on the resource in the data centre.

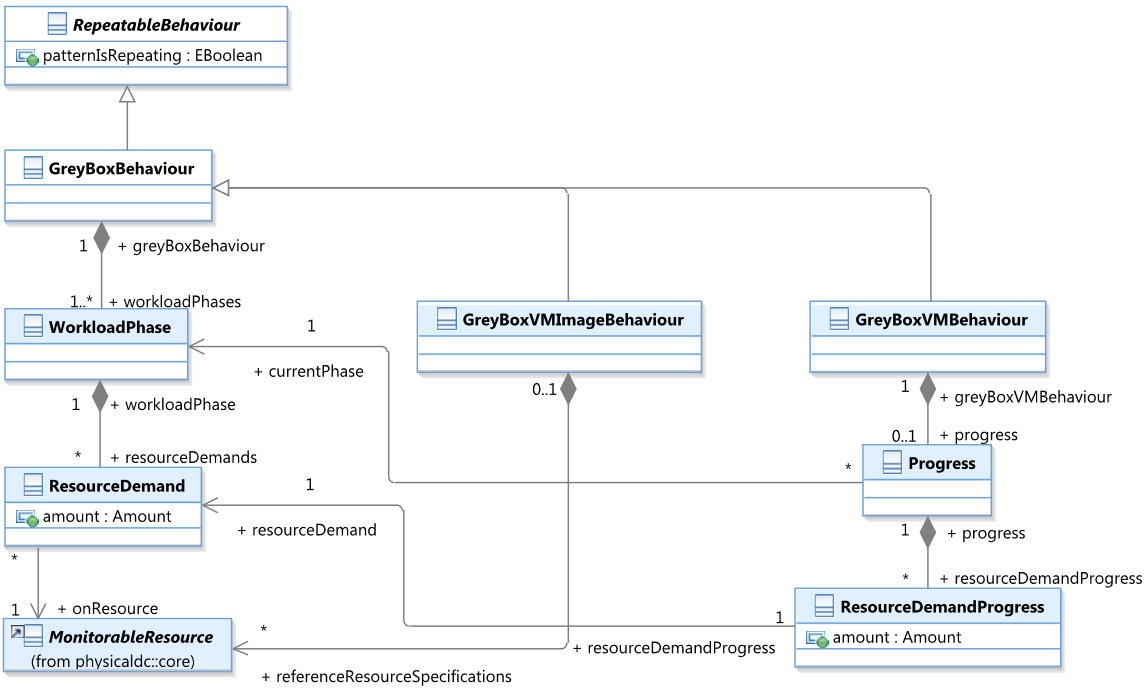

Figure 17: Behaviour Specification for Grey-Box Applications

Figure 17 shows behaviour specifications for Grey-Box applications. As with Black-Box Behaviour, Grey-Box Behaviour can be repeatable. This means that the behaviour pattern defined runs in a loop and restarts from the beginning once it's finished. A Grey-Box Behaviour consists of consecutive Workload Phases. In each Phase, the resourceDemands and amount on the different Resources (onResource) is expressed. A Workload Phase is finished if all resource demand specified for it has been processed. The execution then continues with the next phase. It does not cause load if it is not a repeatable behaviour and the last phase has finished. On the template level, Grey-Box VMImageBehaviour (middle) are Grey-Box Behaviour and must reference the resource specifications used in the behaviour description. This is required as it cannot be guaranteed that the resource specifications are available in a data centre description when a template is described. They are available on the instance level as the instances actually run on the resource in the data centre. On the instance level, Grey-Box VMBehaviour are Grey-Box Behaviour. They can have information on the progress made in processing. The Progress references the current WorkloadPhase as well as the individual progress for specified resource demands within that phase using the resourceDemandProgress reference. The Resource Demand Progress states the amount of processing that has been done so far of the corresponding resourceDemand.

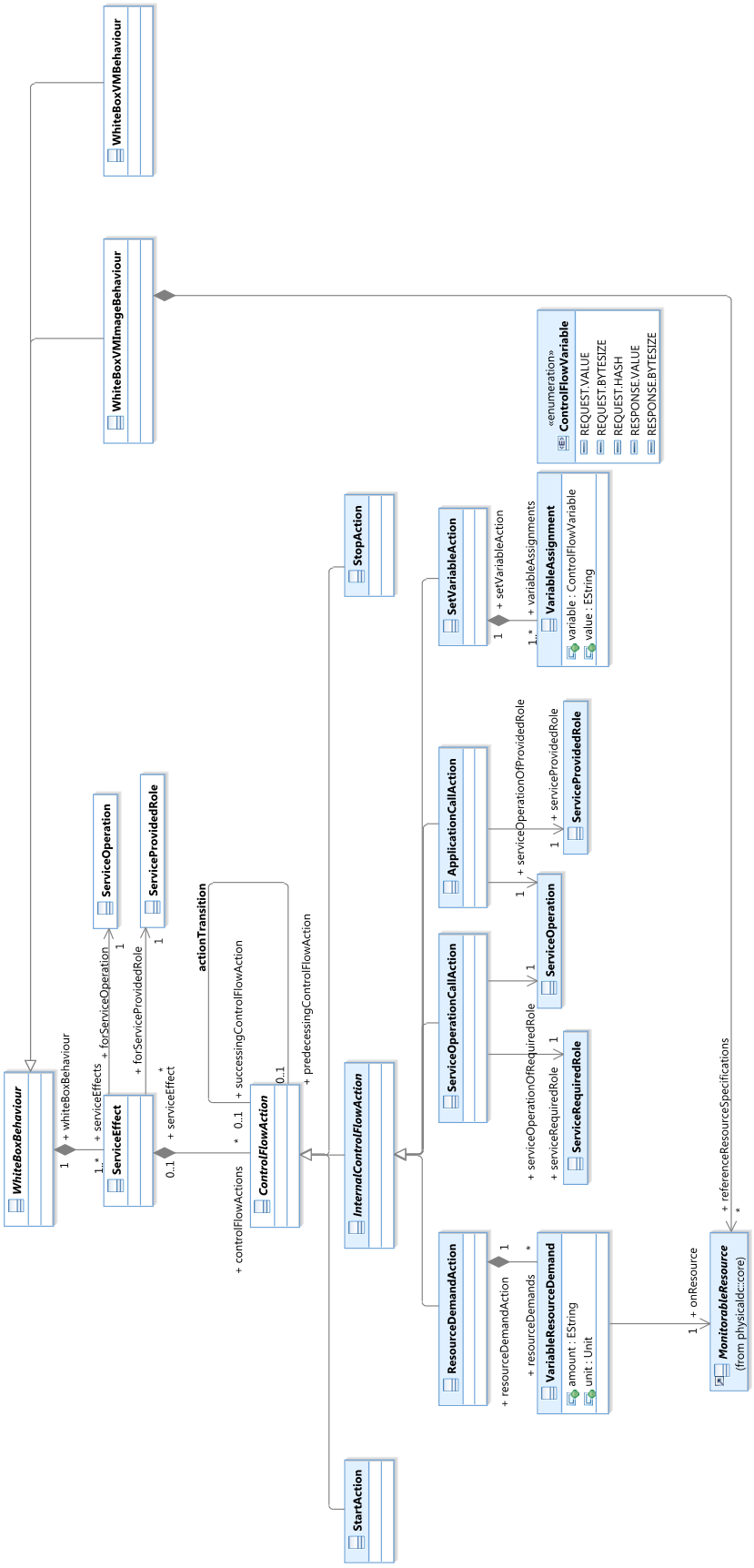

Figure 18: Behaviour Specification for White-Box Applications

Figure 18 shows behaviour specifications for White-Box applications, which consist for several VMs. A White-Box Behaviour always states the Behaviour of a single VM (or VMImage when on the template level). A White-Box Behaviour specifies the behaviour for each service it provides using the serviceEffects reference. A Service Effect references the ServiceProvidedRole and ServiceOperation. This allows that the same interface is provided with separate consumers and to specify the behaviour for each service operation. The processing of an incoming service request is described using the controlFlowActions reference. There are three types of Control Flow Actions: Start Action marking the entry point, Internal Control Flow Action for processing the request, and Stop Action marking that processing ended. There are four Internal Control Flow Actions: Resource Demand Actions, Service Operation Call Actions, Application Call Actions, and SetVariableActions. Resource Demand Actions specify the amount and unit request to be processed from a resource (onResource reference). One Variable Resource Demand per resource. All demands are requested in parallel. Service Operation Call Action allows calling required services, which are specified in the White-Box application template. It references the outgoing role and operation of the interface within that role. Application Call Action allows calling whole application based on the specified user-facing services in the template. This action is only required if Application User Behaviour is specified (see corresponding section below). Set Variable Action allows setting request and response parameters. Available parameters are specified in Variable Assignment using the variable attribute. Options are VALUE, BYTESIZE, and HASH for requests. VALUE can be an arbitrary value used in further control-flow processing. BYTESIZE allows specifying the size and therefore load on the network. The HASH is used for load-balancing (see also ScalableVMConnector above). Options for responses are VALUE and BYTESIZE. After processing has completed, Response will be sent back to the User of VM that made the call to the service (and control-flow processing continues at that point). On the template level, White-Box VMImageBehaviour (middle) are White-Box Behaviour and must reference the resource specifications used in the behaviour description. This is required as it cannot be guaranteed that the resource specifications are available in a data centre description when a template is described. On the instance level, White-Box VM Behaviour is simply a White-Box Behaviour (note the missing VM in the term).

c) Application User Behaviour (Prediction Toolkit only)

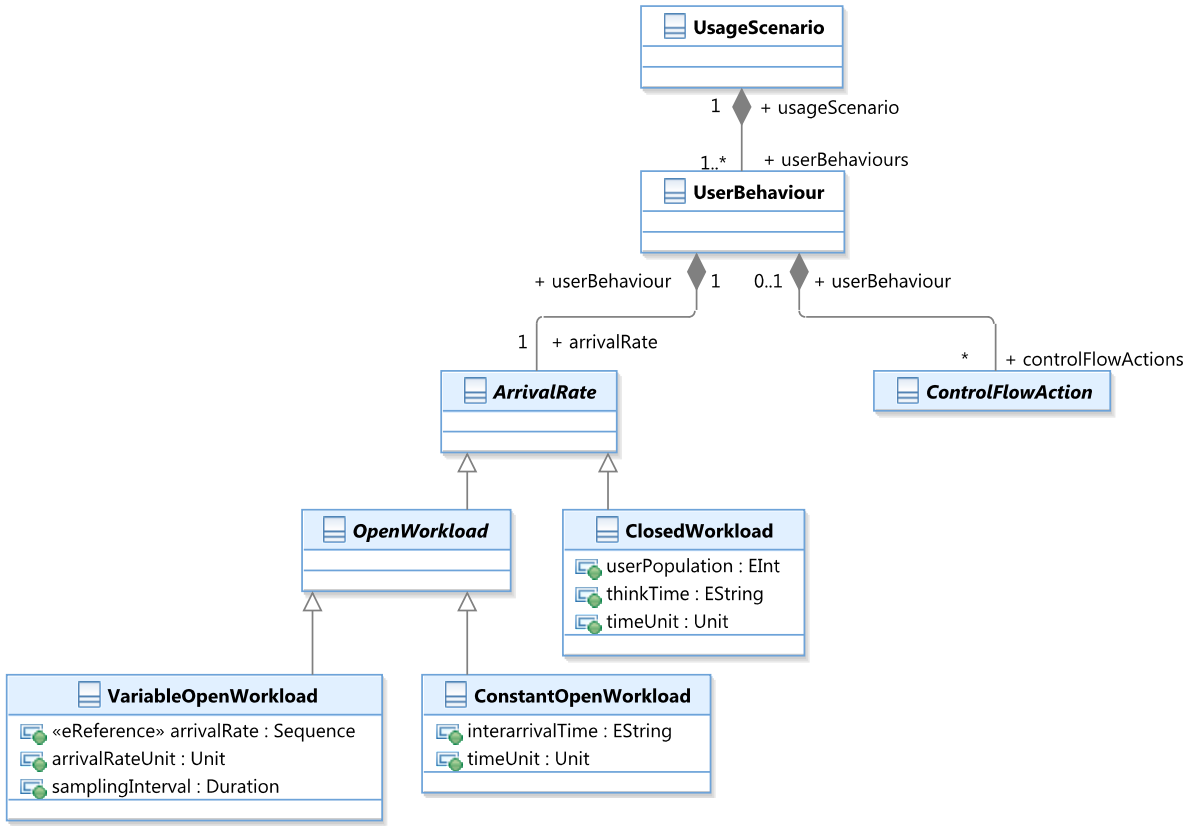

Figure 19: Application User Behaviour specification

Figure 19 shows the model for application user behaviour specifications. User behaviour specifications are used within the Prediction Toolkit in order to specify how users are using White-Box applications. It is not required in the real data centre as

user use the applications directly and do not need to be specified.

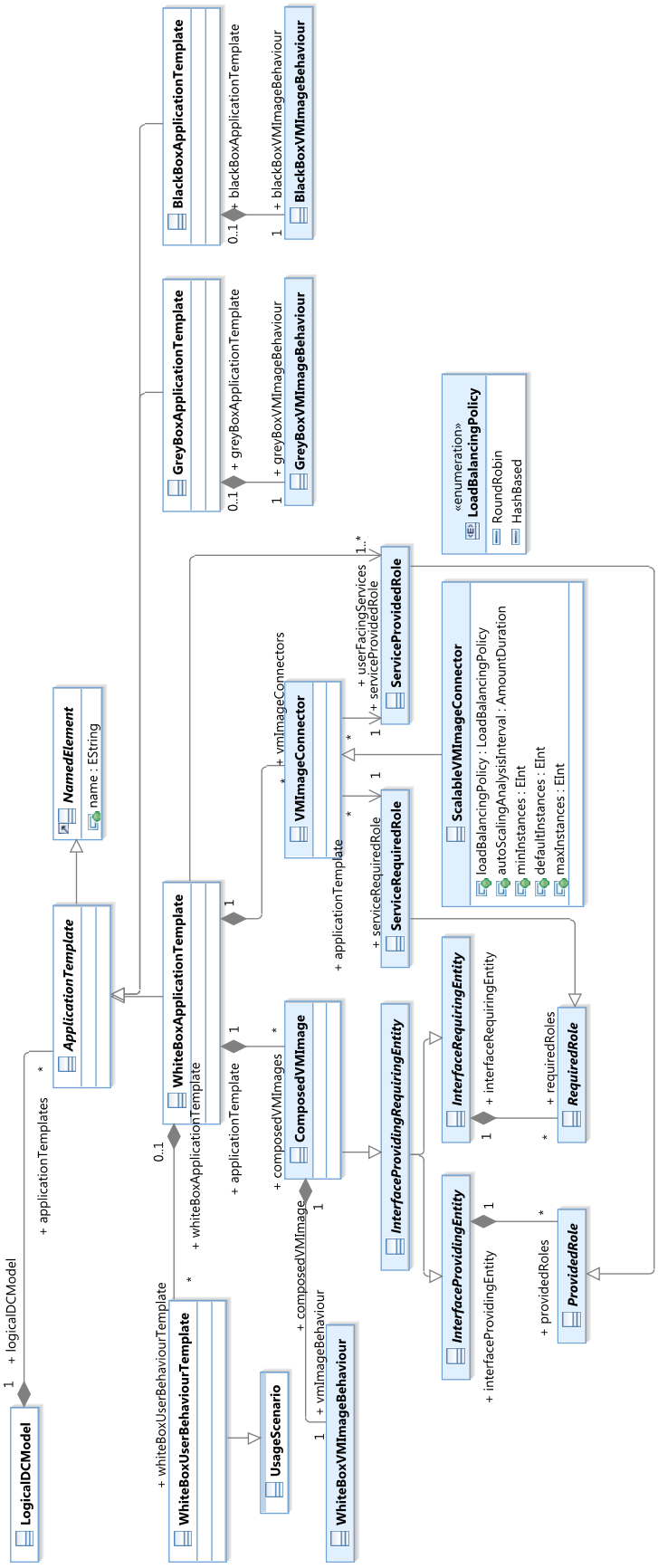

Figure 12 and Figure 13 show how these specifications are connected to application templates and instances using the Usage Scenario element. A Usage Scenario specifies the set of behaviours experienced for different user types. A User Behaviour is specified for each user type. It shows the behaviour itself (Figure 19, right hand side) using the controlFlowActions elements. Figure 18 shows the differentControlFlowAction subtypes. The User Behaviour follows the same structure as the Service Effect. Hence, replacing Service Effect with User Behaviour gives one a good idea how the final User Behaviour model looks like. The Arrival Rate (Figure 19) specifies how often new users of a type arrive. It can be a Close Workload setting. This means there are a predefined constant number of users – the userPopulation. A user behaves as specified in the controlFlowActions and then pauses for thinkTime, which is specified as Stochstic Expression in the separately provided timeUnit. There are also Open Workload arrivals: Variable Open Workload and Constant Open Workload. Constant Open Workload specified the interarrivalTime when a new user arrives using a Stoachstic Expression. This can be a probability distribution, e.g. an exponential distribution. The timeUnit is specified separately. Variable Open Workload extends Constance Open Workload and allows changing the interarrival time over time. This arrivalRate is specified with a separate arrivalRateUnit and samplingInterval to determine how often the arrival Rate is updated.

4. INTEGRATION

This section highlights the integration of tools and toolkits. It is relevant for developers only.

CACTOS is already available for the OpenStack and FCO virtualisation infrastructure. The VMI is a proxy in front of the platform, e.g. OpenStack. It forwards selected calls to the cloud API to CACTOS' Runtime Management before handing them to the cloud platform for execution (boot and delete requests to OpenStack and FCO). Other calls are directly relayed to the actual cloud API platform for execution. The VMI additionally provides a service interface for managing White-Box Applications. Applications that consist of several interconnected VMs and complex control flow.

a) STARTING INDIVIDUAL VIRTUAL MACHINES

This section exemplifies how incoming or leaving VMs are handled by the VMI and the other components of the CACTOS infrastructure.

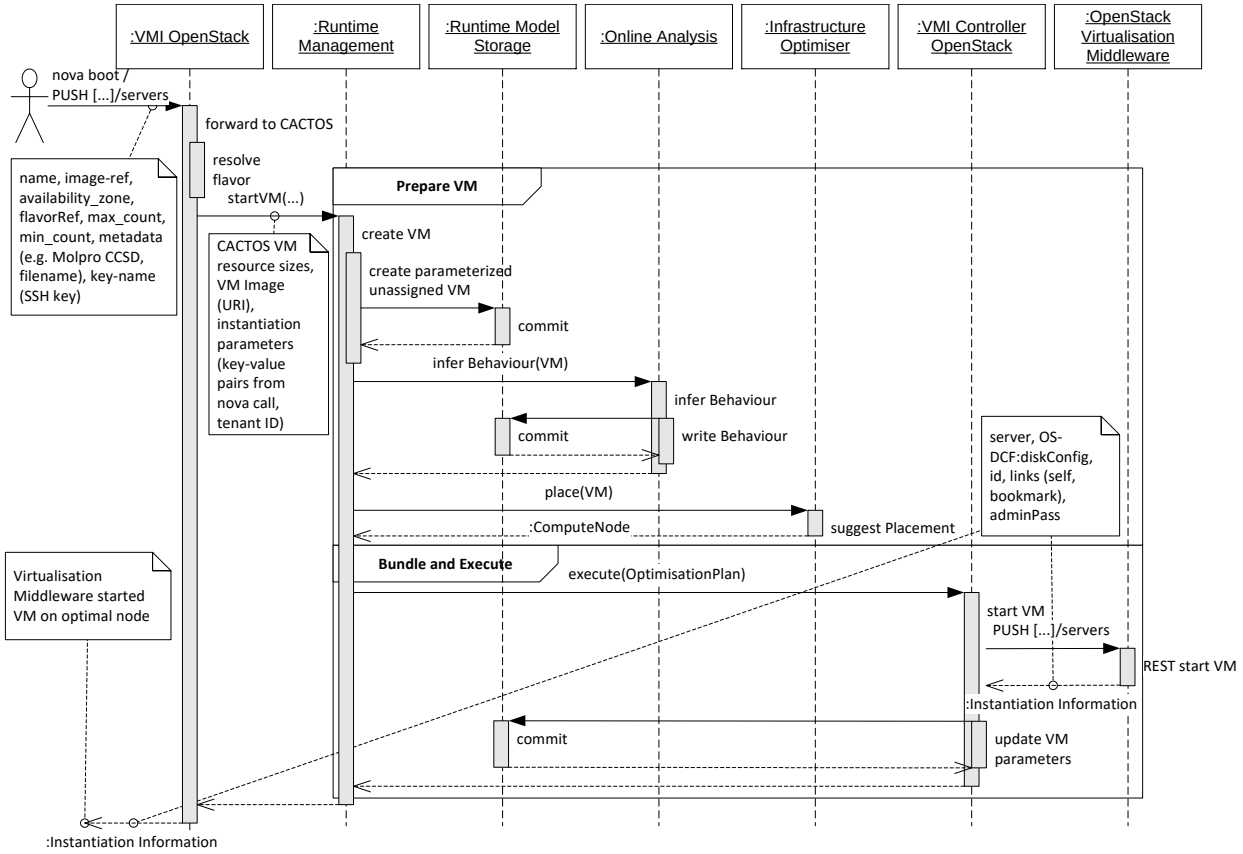

Figure 20: Exemplary sequence for processing incoming VM requests

Figure 20 illustrates the processing of a request for incoming VMs in the case of OpenStack (for Flexiant Cloud Orchestrator the approach is equivalent): The VMI OpenStack and VMI Controller OpenStack are the interfaces between CACTOS and the virtualisation infrastructure. The platform itself is shown on the right-hand side as OpenStack Virtualisation Middleware. Incoming boot requests are processed by VMI OpenStack and their parameters, e.g. flavours, translated to CACTOS terms. The resulting information is sent to the Runtime Management via the startVM method including all parameters. StartVM first creates the VM using the Runtime Model Storage without assigning a physical node to it. It infers the probable behaviour of the VM by using the Online Analysis. The Online Analysis allows plugging in different algorithms and replace them at runtime. This part is not visualized to reduce complexity. Third, a placement suggestion is requested from the Infrastructure Optimiser. This concludes the preparation steps and allows to continue with the bundle and execute steps. The suggested change of deploying the VM is stored in an Optimisation Model and executed by the VMI Controller OpenStack. This component translates from CACTOS to OpenStack and issues the boot command on the real OpenStack Virtualisation Middleware. The call returns instantiation information after completion, e.g. the VM ID in OpenStack. This information is stored at the VM description in the model. This concludes the processing on the CACTOS side. The instantiation information is passed back to the original boot call. This allows running a CACTOS-enabled optimised data centre with its users to know about it.

Stopping VMs based on user requests are handled similarly.

b) STARTING WHITE-BOX APPLICATIONS

This section explains the starting of white-box applications. This includes the orchestration of several components deployed on multiple virtual machines and wired with each other according to the associated WhiteBoxApplicationTemplate (cf. Section II.3.c).

In order to support the feature of multi-VM application deployment, we exploit products and artefacts produced by other projects: (i) The Cloudiator toolkit[1] is an open-source cloud orchestration tool developed as a key component in the PaaSage project (Domaschka, 2015). (ii) The CAMEL language (Rossini, 2015) is a powerful and comprehensive modelling language to describe the deployment of distributed applications over cloud infrastructures. (iii) The CAMEL adapter (Jähnert & Liang, 2015) is a simple parser that reads CAMEL as input and outputs invocations to Cloudiator.

As a general approach, whenever multi-VM applications shall be deployed, CACTOS creates a CAMEL application model based on the WhiteBoxApplicationTemplate, passes this model to the CAMEL adapter that applies the necessary invocations to Cloudiator. All of these steps have been integrated in the CACTOS Runtime Management and are bundled with it.

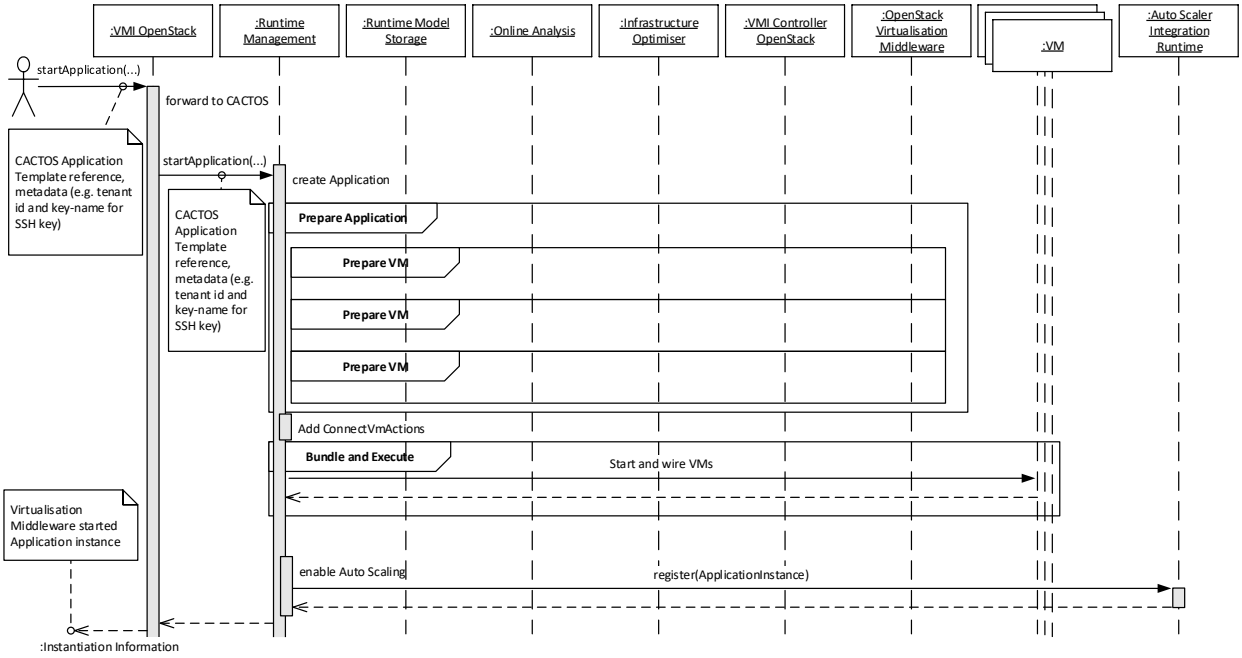

Figure 21: Exemplary sequence for processing incoming Application requests

Figure 21 illustrates the processing of a request for an incoming White-Box application consisting of several interconnected VMs. Again, VMI OpenStack and VMI Controller OpenStack are the platform-specific parts with CACTOS (and above tool chain) in between. This provides a coherent view and entry point to users of the data centre. An incoming service request to start an application is resolved to retrieve the CACTOS model elements and forwarded to the Runtime Management.

Here, the CAMEL model is loaded, the adapter and Cloudiator get invoked which eventually trigger the deployment of individual virtual machines as shown in Figure 20. Cloudiator also takes care of

[1] https://github.com/cloudiator/

the actual wiring of application components. Once this step has been completed, the respective models are generated in the model repository.

After this step, for those applications that support it, auto scaling is enabled for the application instance. The application instance is registered at the Auto Scaler Integration Runtime (cf. Section II.5.b). This component is responsible to ensure the optimal amount of VMs serving each load balancer given the intervals and settings in White-Box application descriptions. The detailed process is not depicted to reduce complexity. The application is now ready and the control flow returns to the initial request.

Stopping applications requests are handled similarly.

c) Runtime Management

```java
/**Entry point for managing Virtual Machines and Applications using CACTOS.
 * The semantic and content is already based on the CACTOS Infrastructure model.
 * This class allows (de)coupling Virtualization Middleware, e.g. Open Stack or FCO, and connecting them to CACTOS.
 *
 * @author hgroenda
 *
 */
public interface IRuntimeManagement {

    /**Starts a new virtual machine using a Flavour.
     * @param flavourRef UUID of the CACTOS Flavour to boot.
     * @param vmImageRef UUID of the CACTOS VMImage to boot.
     * @param inputParameters Input parameters, e.g. tenant id.
     * @return UUID of the instantiated Virtual machine. <code>null</code> if not successful.
     */
    public String startVM(String flavourRef, String vmImageRef, Map<String, String> inputParameters);

    /**Starts a new application.
     * @param appRef UUID of the application.
     * @param inputParameters Input parameters, e.g. tenant id.
     * @return UUID of the Application Instance. <code>null</code> if not successful.
     */
    public String startApplication(String appRef, Map<String, String> inputParameters);

    /**
     * Stops a running virtual machine.
     * @param vmRef UUID of the CACTOS Virtual Machine to remove.
     * @param inputParameters Input parameters, e.g. tenant id.
     * @return Success state. <code>true</code> if successful, <code>false</code> otherwise.
     */
    public boolean stopVM(String vmRef, Map<String, String> inputParameters);

    /**Stops a running application.
     * @param appInstanceRef UUID of the application instance.
     * @param inputParameters Input parameters, e.g. tenant id.
     * @return Success state. <code>true</code> if successful, <code>false</code> otherwise.
     */
    public boolean stopApplication(String appInstanceRef, Map<String, String> inputParameters);

}
```

Figure 22: Runtime Management Interface

Figure 22 shows the interface of Runtime Management. It allows starting and stopping VMs and applications. The information is passed via the models. The method parameters only contain the compact serializable identifiers and critical information.

5. Extensible Services Infrastructure

The consortium identified the following requirements for Optimization, VM placement, and Behaviour Inference algorithms:

- Lightweight extension

- 3rd party extension support, e.g. for advanced or customized algorithms
- Hot (re-)deployable during data centre operation
- Identical packages and behaviour for both CACTOS toolkits
- Support polyglot programming for individual services

The technical infrastructure behind the solution is based on OSGi services. The interfaces and a typical interaction are described in the following for both algorithm types.

OPTIMISATION SERVICE

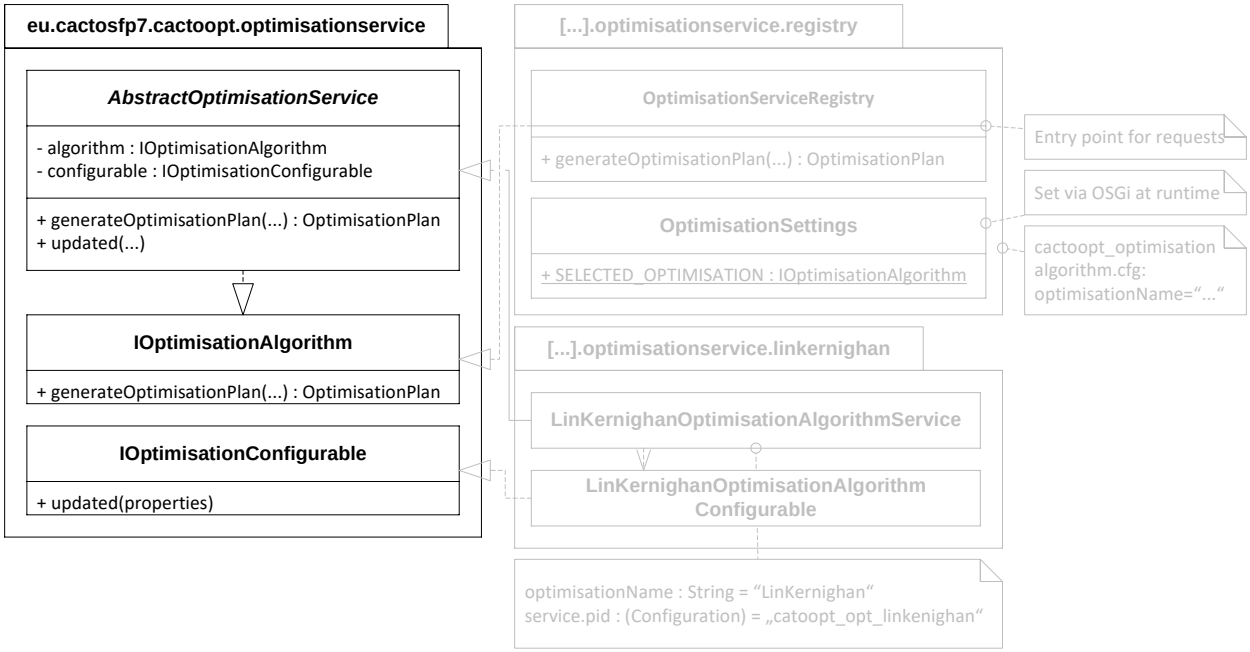

Figure 23: Optimisation Service Template

Figure 23 describes the template for optimisation services. Each service must be configurable and implement the IOptimisationConfigurable interface. Each service must provide the IOptimisationAlgorithm interface in order to run the algorithm on CACTOS infrastructure models. A default implementation is provided by AbstractOptimisationService in order to ease the development for new services.

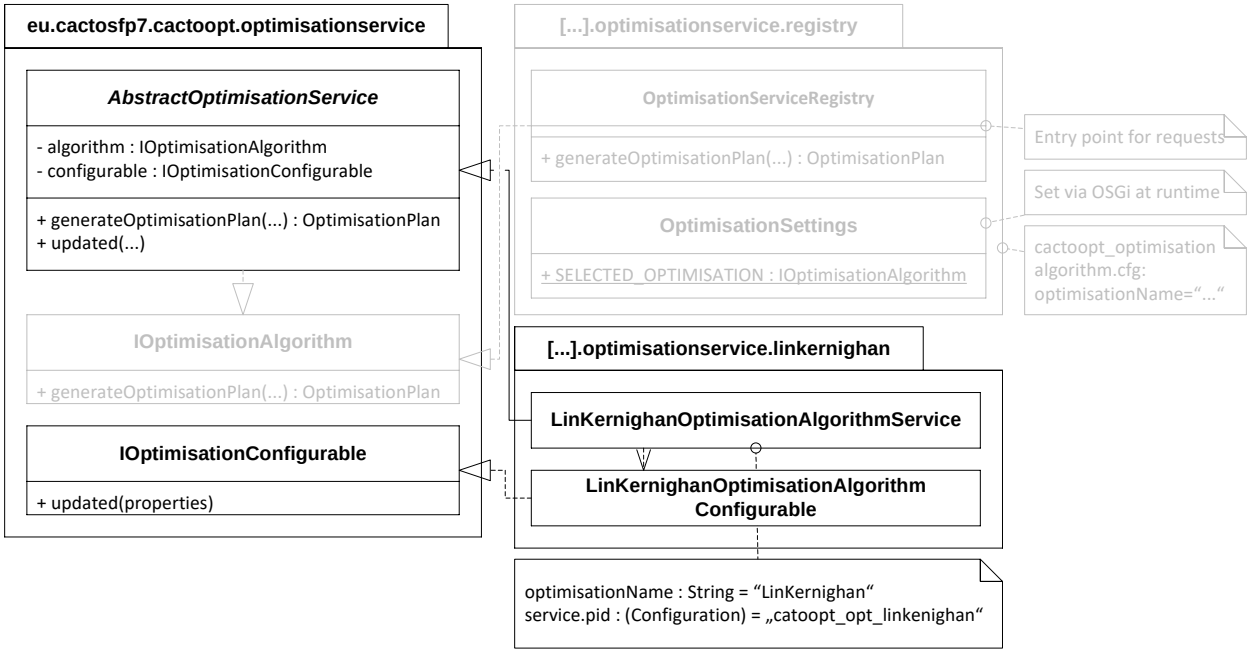

Figure 24: Optimisation Service Implementation Example

Figure 24 illustrates how a new service implementation uses the template. The example depicts the LinKernighanOptimisation service pre-shipped with CACTOS. The LinKernighanOptimisationAlgorithmService uses the default implementation and registers itself as OSGi service with the optimisationName LinKernighan and the service.pid cactoopt_opt_linkernighan. It reacts on its own configuration via the LinKernighanOptimisationAlgorithmConfigurable class. The configuration options should be set in the file [service.pid].cfg, e.g. cactoopt_opt_linkernighan.cfg. Felix file install technology ensures that the settings are updated and taken into account.

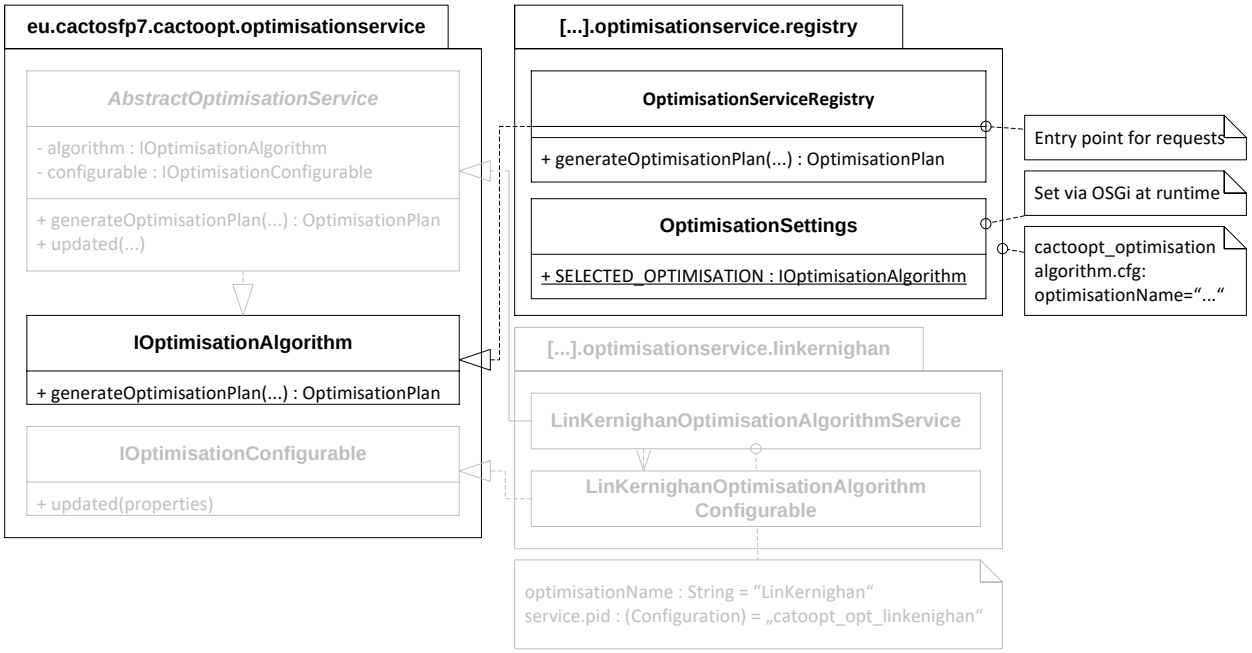

Figure 25: Optimisation Service Registry

Figure 25 shows how the single entry point for requesting an optimisation within a data centre. This point is represented as singleton by the OptimisationServiceRegistry. It provides the IOptimisationAlgorithm interface in order to allow seamless execution of the selected algorithm – transparent to the caller. The algorithm can be selected via settings in the file algorithm.cfg. The optimisationName must match the one of an available algorithm. This algorithm is stored internally in the OptimisationSettings.

[..].optimisationservice.linkernighan

- Plug-in project code
  - 13 LOC (Manifest)
- OSGi service configuration
  - 10 LOC (XML)
- Implementation
  - 3 Classes
  - 55 LOC (non-empty, including boilerplate, comments, ...)
- Algorithm Integration
  - 1 Class
  - 68 LOC (non-empty, including boilerplate, comments, ...)

Figure 26: Example of Complexity for Optimisation Service

VM PLACEMENT SERVICE

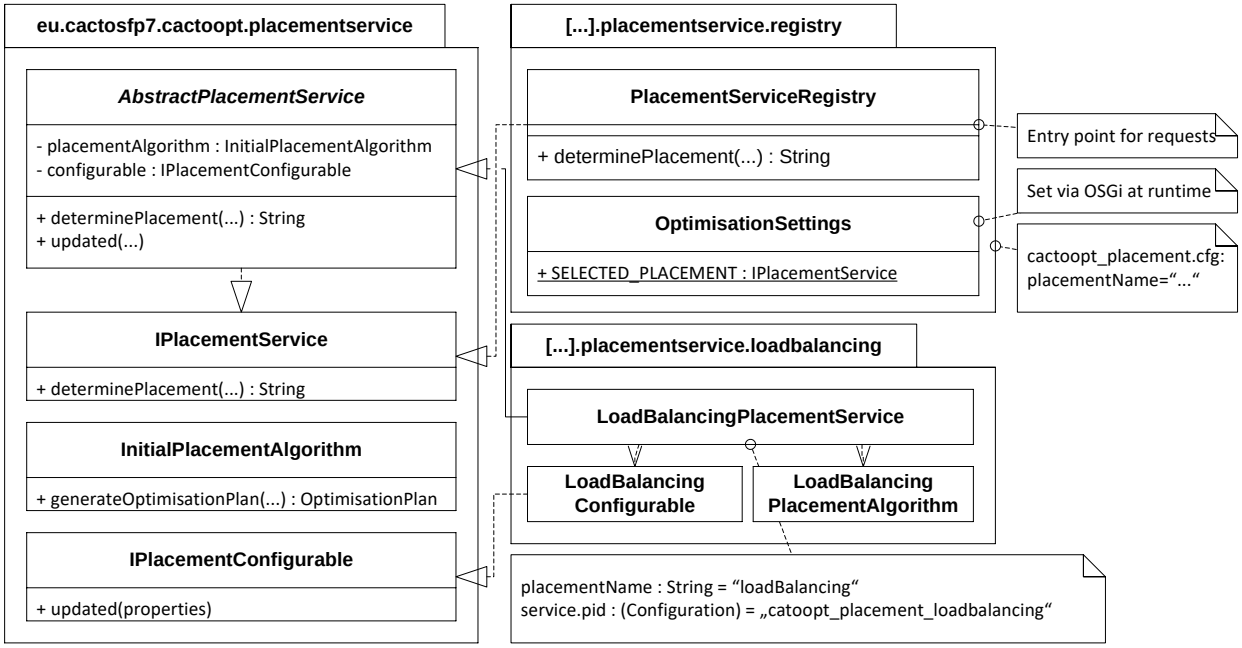

Figure 27: Placement Service Architecture

Placement is realized similarly to the Optimisation Service. Figure 27 provides an all-in-one view on the service template, a service example and the service registry. Because of the identical concept and shown interfaces and methods in the figure detailed comments are left out due to brevity.

[..].placementservice.loadbalancing

- Plug-in project code
  - 14 LOC (Manifest)
- OSGi service configuration
  - 10 LOC (XML)
- Implementation
  - 2 Classes
  - 23 LOC
    (non-empty, including boilerplate, comments, ...)
- Algorithm Integration
  - 1 Class
  - 138 LOC
    (non-empty, including boilerplate, comments, ...)

Figure 28: Example of Complexity for Placement Service

BEHAVIOUR INFERENCE SERVICE

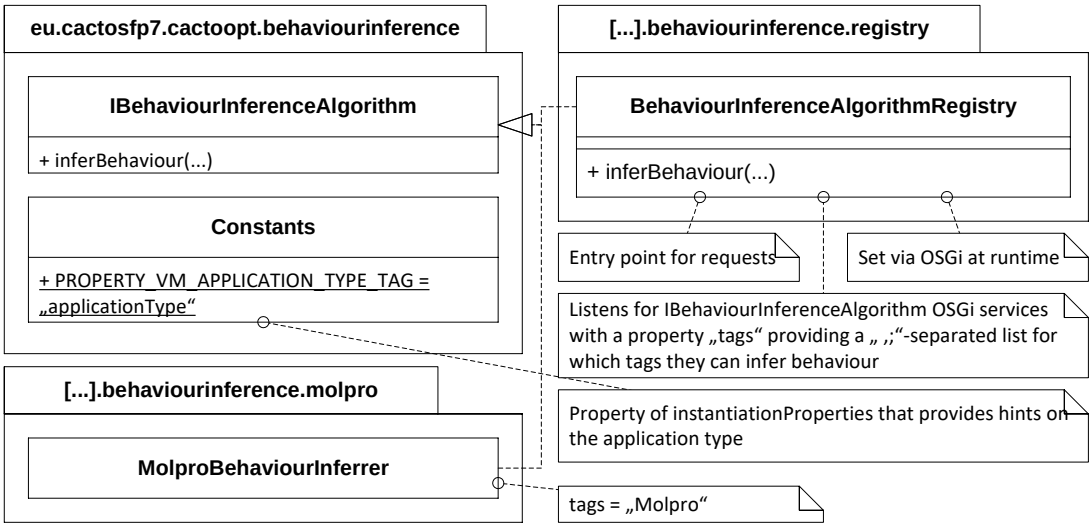

Figure 29: Behaviour Inference Service Architecture

Behaviour inference is realized similarly to the other two services. Figure 29 provides an all-in-one view on the service template, a service example and the service registry. Because of the identical concept and shown interfaces and methods in the figure detailed comments are left out due to brevity. The only difference is that algorithms do not need an individual configuration but are selected based on tags assigned to incoming VMs.

[..].behaviourinference.molpro

- Plug-in project code
  - 13 LOC (Manifest)
- OSGi service configuration
  - 8 LOC (XML)
- Implementation
  - (None)
- Algorithm Integration
  - 1 Class
  - 108 LOC
    (non-empty, including
    boilerplate, comments, ...)

Figure 30: Example of Complexity of Behaviour Inference Service

b) AUTO SCALING

Auto Scaling ensures that the optimal amount of VMs is present for a scalable connection. Such a connection is typically used for load balancers within White-Box applications.

```
1  /**
4  package eu.cactosfp7.autoscaler;
5
6  import eu.cactosfp7.infrastructuremodels.logicaldc.application.WhiteBoxApplicationInstance;
7
8  /**
9   * Interface for AutoScalerIntegration implementations allowing to manage
10  * scalable connectors of applications. AutoScalerIntegration implementations
11  * are responsible to manage each individual connector of an application
12  * instance at the points and with the settings specified in the description.
13  * Only White-Box applications need to be taken into account as they are the only
14  * type of application with more than one VM.
15  *
16  * @author hgroenda
17  *
18  */
19 public interface IAutoScalerIntegration {
20     /**
21      * Registers an application to be scaled automatically.
22      *
23      * @param appInstance
24      */
25     void register(WhiteBoxApplicationInstance appInstance);
26
27     /**De-registers an application to be scaled automatically.
28      * @param appInstance
29      */
30     void deregister(WhiteBoxApplicationInstance appInstance);
31 }
```

Figure 31: Interface for Auto Scaler Integration implementations

The AutoScaler Integration Runtime and AutoScaler Integration Simulation are responsible to keep track of all running application instances and issue calls to the AutoScaler for each scalable connector and application instance according to their description. They have to implement to IAutoScalerIntegration interface depicted in Figure 31.

```java
/**
 * Interface for an Auto Scaler that determines the optimal number of connected
 * VMs for a scalable connection.
 *
 */
public interface IAutoScaler {
    /**
     * Request a calculation of the optimal number of connected VMs for a
     * scalable connection.
     *
     * @param appInstance
     *            The application instance.
     * @param connector
     *            The scalable connector.
     * @param llm
     *            Load and request information.
     * @return An optimisation plan with suggested changes to get to the optimal
     *         number. Empty plan if nothing should be changed.
     */
    public OptimisationPlan optimiseScaling(ApplicationInstance appInstance,
            ScalableVMImageConnector connector, LogicalLoadModel llm);
}
```

Figure 32: Interface of the Auto Scaler

The Auto Scaler itself is responsible to determine the optimal number of connected VMs for an individual scalable connection instance. The application instance, connector and load model with request and response arrival rate measurements are used to suggest changes. An implementation of the interface, see Figure 32, can use polyglot programming and distribute the processing internally. There is only one implementation required (and shipped) as part of CactoOpt. This is used in both toolkits.

III. PROVISIONING OF THE CACTOS TOOLKIT

Due to brevity, this document references other existing documents that provide provisioning instructions. You can either start from the individual tools or the toolkits and artefacts.

Download the latest version of the CactoScale Guide from the bottom of the page http://www.cactosfp7.eu/cactoscale/, CactoOpt Guide from the bottom of the page http://www.cactosfp7.eu/cactoopt/, and CactoSim Guide from the bottom of the page http://www.cactosfp7.eu/cactosim/.

The different artefacts listed in section II.1.a) are available at http://www.cactosfp7.eu/code/ including documentation where required, e.g. the CactoSim Guide on how to provision the Prediction Package.

IV. EXAMPLE USE CASE

The example use case presented in (D5.2.1 CACTOS Toolkit Version 1) is still valid for this iteration. Please refer to the corresponding section in that document for further details and an explanation on how infrastructure information is extracted, infrastructure models created, and optimisations triggered at runtime and how the prediction complements this.

Application models for taking the prediction capabilities on a test drive without requiring the Runtime Toolkit in your own data centre are available at:

- Black-Box Behaviour: https://svn.fzi.de/svn/cactos/code/sim/trunk/eu.cactosfp7.cactosim.demo.bba.2.0
- Grey-Box Behaviour: https://svn.fzi.de/svn/cactos/code/sim/trunk/eu.cactosfp7.cactosim.demo.gba.2.0
- White-Box Behaviour: https://svn.fzi.de/svn/cactos/code/sim/trunk/eu.cactosfp7.cactosim.demo.wba.2.0

The White-Box application is oriented at the DataPlay application[2] but does not contain the final resource demands and the full control-flow. It is presented for demonstration purpose in the following.

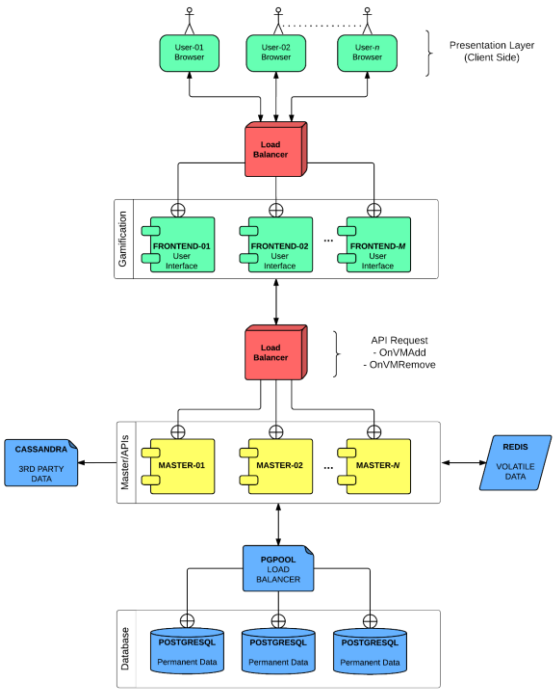

Figure 33: DataPlay Modelling Reference

[2] Available on https://github.com/playgenhub/DataPlay . Last retrieved: 22.03.2016.

Type	Name	#vCPU	Memory (GB)	Scaling per Application (Min / Default / Max)		
Load Balancer Frontend	loadbalancer	2	2	1	1	1
Frontend	frontend	2	1	1	1	3
Load Balancer Master	loadbalancer	2	2	1	1	1
Master	master	2	2	2	2	7
PgPool	pgpool	2	4	1	1	1
Postgresql	postgresql	2	2	2	2	5
Redis	redis	1	1	1	1	1
Cassandra	cassandra	4	8	1	1	1

Figure 34: DataPlay Resource Requirements (all have 20 GB storage and are based on Ubuntu 15.10 images

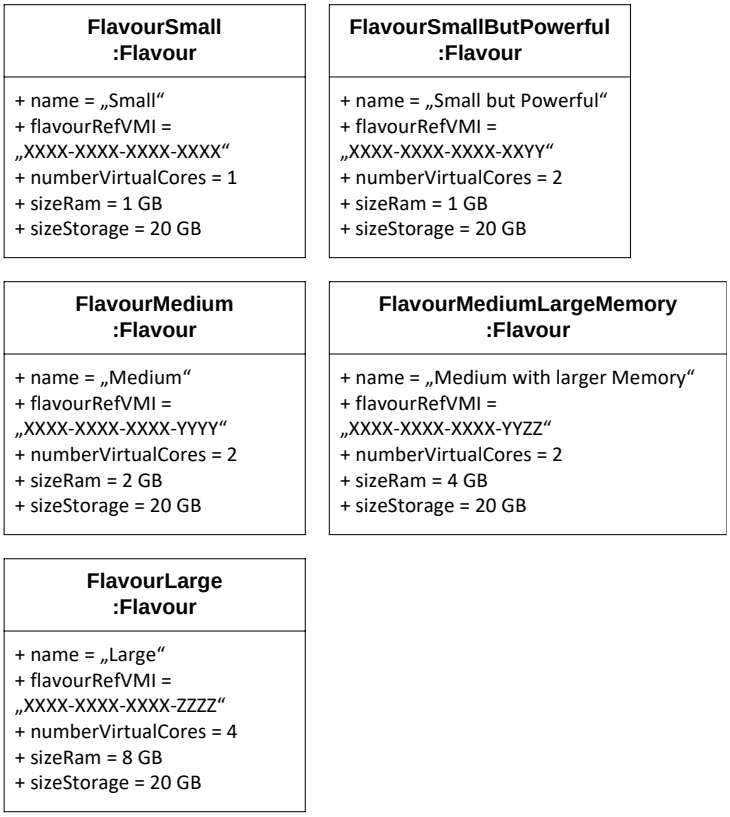

Figure 35: DataPlay Flavours in the model

The Flavours show the representation of resources in the virtualization infrastructure and in CACTOS. The mapping is done via the flavourRefVMI, which is a Cloud-specific UUID.

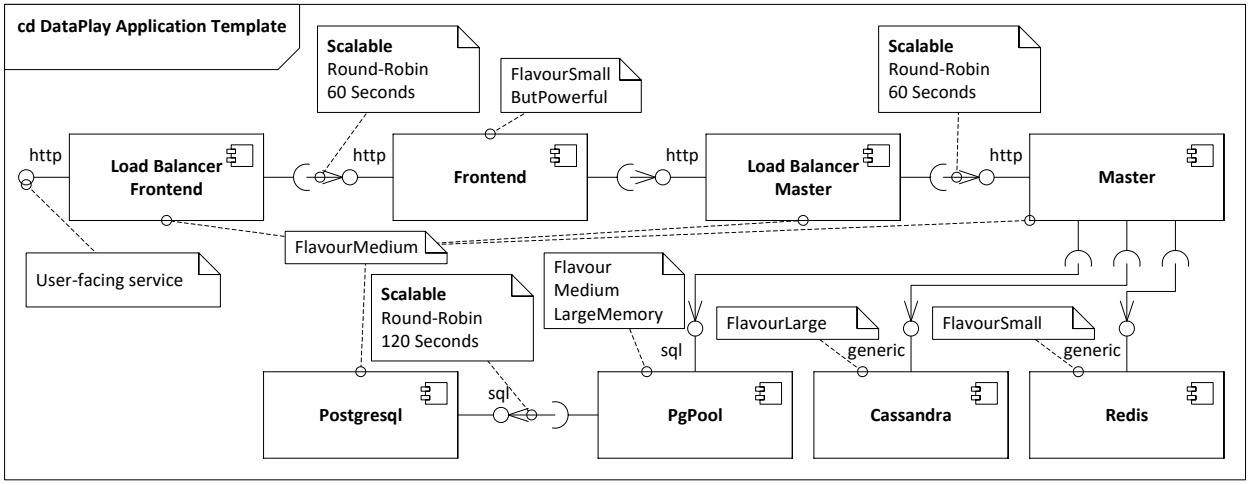

Figure 36: DataPlay Structural Architecture View

The structural architecture view specifies the VMs, their interfaces and connections used in the control-flow when processing user requests. The response time of the operation of interfaces is stored in the load model and supports auto scaling.

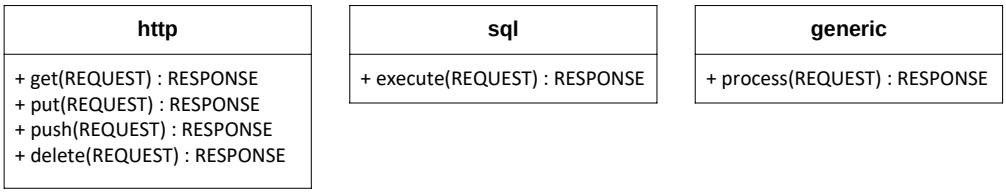

Figure 37: DataPlay Interfaces

The following sketches the structure of the White-Box Behaviour Models of DataPlay. The presented models are based on DataPlay's architecture and architecture but are still missing resource demand estimations and a finer-grained control flow modelling. The purpose of White-Box Behaviour Models is to simulate the demand and response times of individual user requests. The DataPlay interfaces show the interfaces and the relevant operations for tracking request and response arrival rates. The rates are stored in the load model and allow auto scaling. All have on default input and output parameter by convention when modelled in CACTOS.

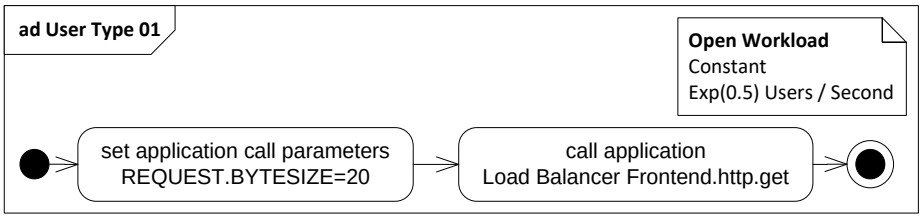

Figure 38: DataPlay Application Usage Scenario

The DataPlay application usage scenario is only required when running in the Prediction Toolkit. It states the number of users and how they interact with user-facing services. This causes load across the VMs and allows user experience metrics.

In the example, new users arrive with an inter-arrival time using an exponential distribution with λ=0.5. This value could be calculated based on Historic Database and Load information for Requests at Load Balancer Frontend. Within usage scenarios, application call parameters can be set. In this case, the bytesize (causing load on the network) is set to 20 bytes. After that, the actual operation of a user-facing interface is called, e.g. ‚get /status.html'

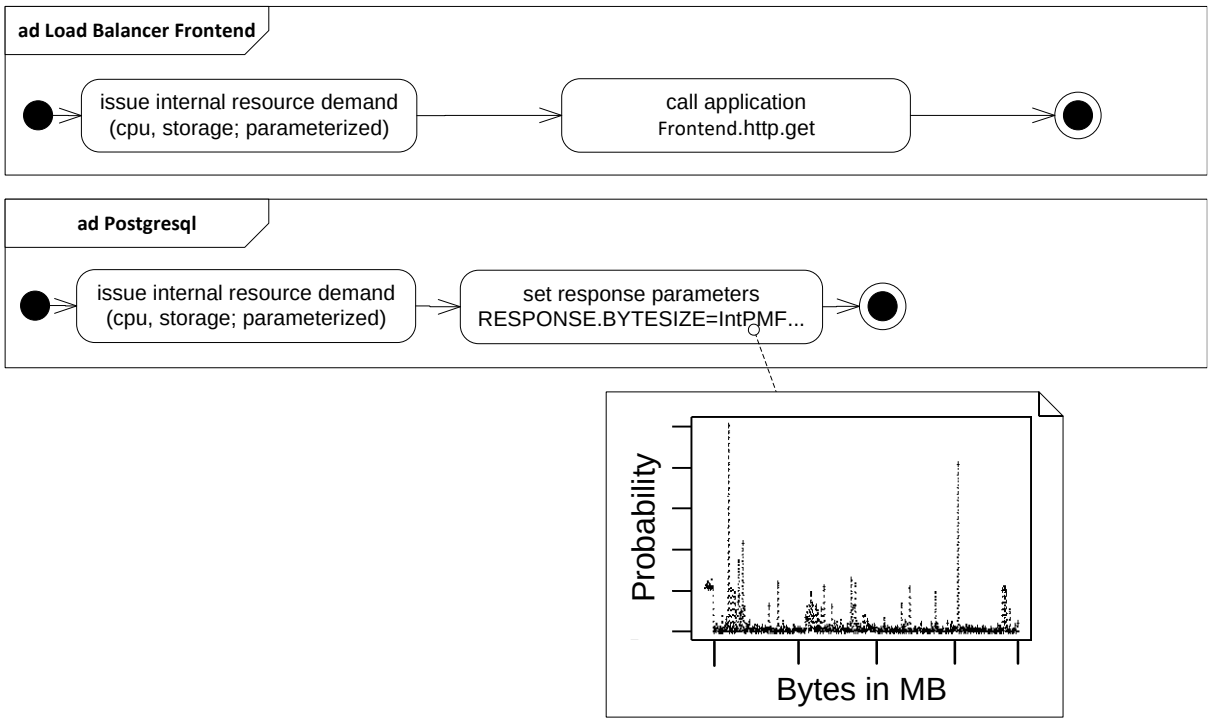

Figure 39: DataPlay Behaviour Template

The behaviour template shows the resource demand when processing a user request. This can be specified using parameters, e.g. ‚REQUEST.BYTESIZE * 2'. Probability distributions available in the Stochastic Expression language are supported.

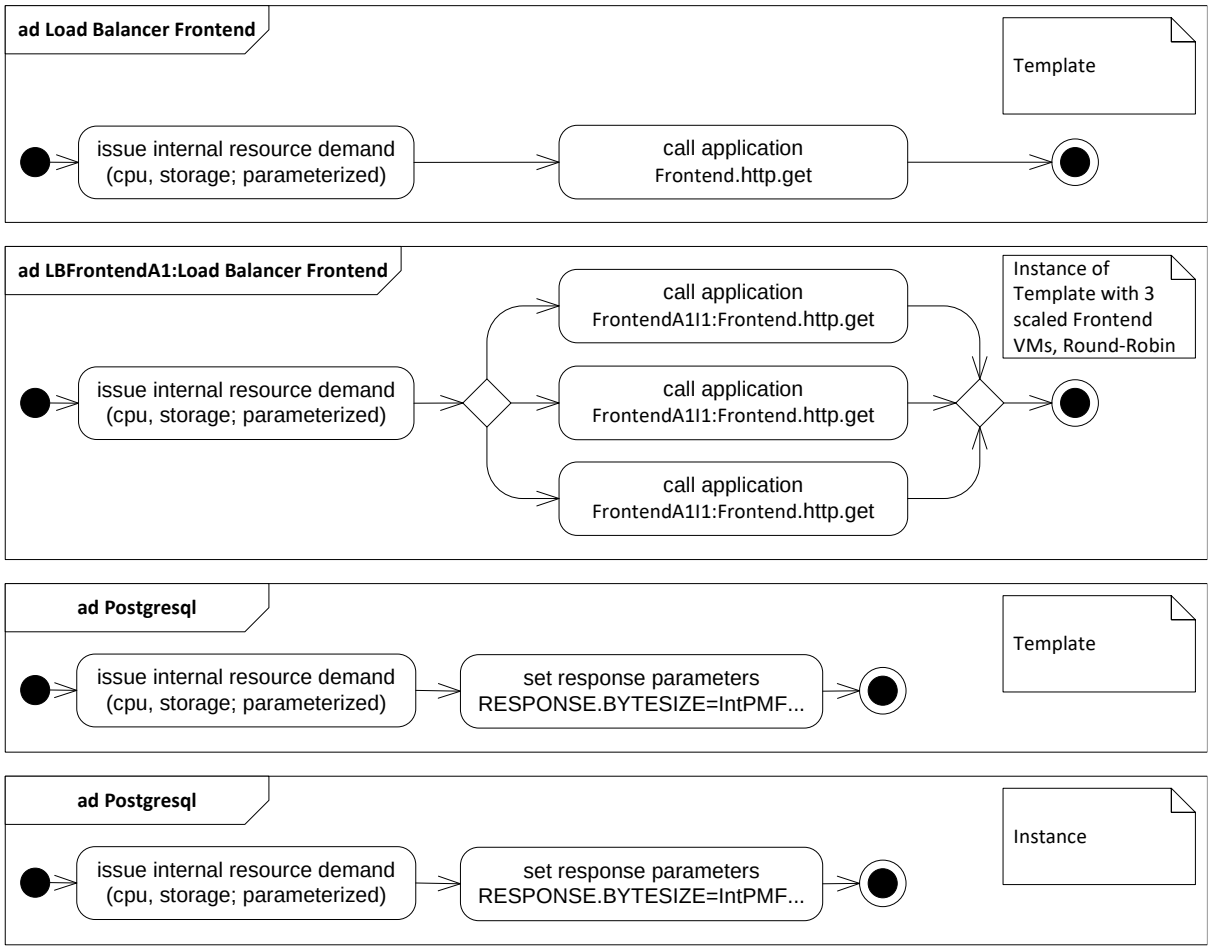

Figure 40: DataPlay Behaviour Instance and Template Level

The instantiation of a behaviour template is depicted in Figure 40. This shows how instances and the template differ. The difference is only important for the Prediction Toolkit, which needs to track the individual instances settings during simulation.

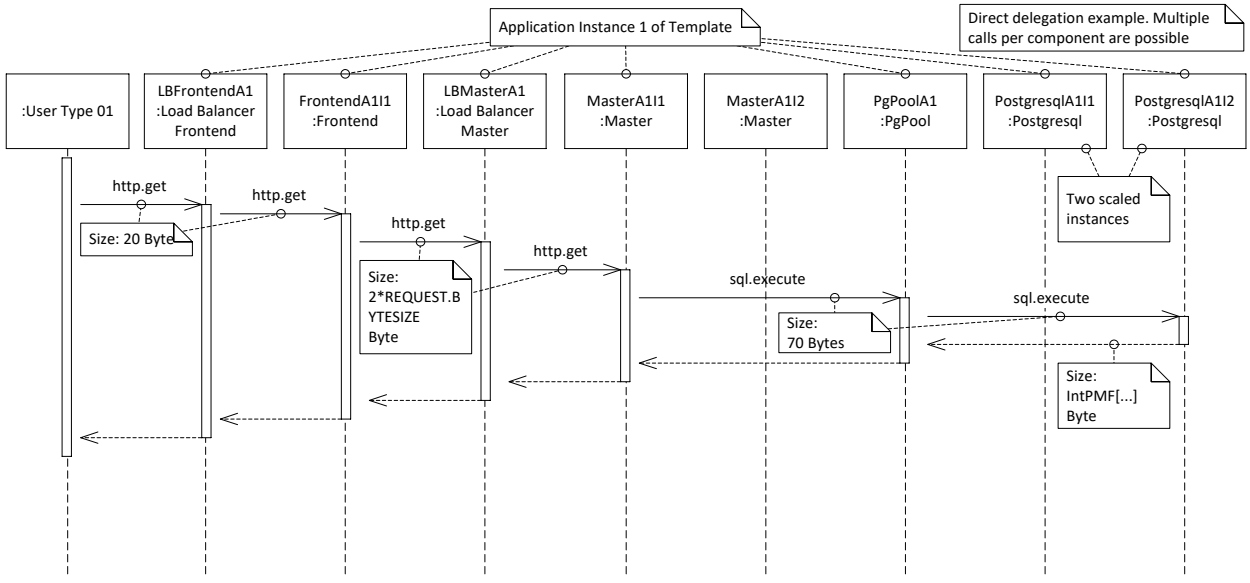

Figure 41: DataPlay Sequence Example

The sequence example shows how one instance of the DataPlay template processes a single request from a single user. The real or simulated application takes care about contention and parallel execution but it is not shown for brevity. It's shown how the network load and transferred bytes can change along the control flow. This should give an impression of a typical processing flow.

REFERENCES

Apache Hadoop. (2014, August 15). Retrieved August 29, 2014, from http://hadoop.apache.org/

Becker, M., Luckey, M., & Becker, S. (2013). Performance analysis of self-adaptive systems for requirements validation at design-time. *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures (QoSA '13)*. Retrieved from dl.acm.org/citation.cfm?id=2465489

Becker, S., Koziolek, H., & Reussner, R. (2009). The Palladio component model for model-driven performance prediction. *Journal of Systems and Software, 82*(1), 3-22.

CACTOS Consortium. (2014). *D3.1 Prototype Optimization Model.*

CACTOS Consortium. (2014). *D4.1 Data Collection Framework.*

CACTOS Consortium. (2014). *D4.2 Preliminary offline trace analysis.*

CACTOS Consortium. (2014). *D5.1 Model Integration Method and Supporting Tooling.*

CACTOS Consortium. (2014). *D5.2.1 CACTOS Toolkit Version 1.*

CACTOS Consortium. (2014). *D5.3 Operational Small Scale Cloud Testbed Managed by the CACTOS Toolkit.*

CACTOS Consortium. (2014). *D5.3 Operational Small Scale Cloud Testbed Managed by the CACTOS Toolkit.*

CACTOS Consortium. (2014). *D6.1 CactoSim Simulation Framework Initial Prototype.*

CACTOS Consortium. (2014). *D7.2.1 Physical Testbed.*

CACTOS Consortium. (2015). *D5.4 Evaluation Methodology for the CACTOS Runtime and Prediction Toolkit.*

CACTOS Consortium. (2016). *D6.4 CactoSim Simulation Framework Final Prototype.*

Domaschka, J. a. (2015). *Product Executionware, Deliverable D5.1.2.*

Flexiant. (2014, August 28). *Flexiant Cloud Orchestrator Documentation - FDL Server.* Retrieved August 28, 2014, from http://docs.flexiant.com/display/DOCS/FDL+Server

Flexiant. (2014, August 28). *Flexiant Cloud Orchestrator Documentation - Triggers.* Retrieved August 28, 2014, from http://docs.flexiant.com/display/DOCS/Triggers

Flexiant. (2014). *Flexiant Developer Language System API Documentation.* Retrieved August 28, 20014, from http://docs.flexiant.com/display/DOCS/System+API

Godard, S. (2014, August 30). *Sysstat*. Retrieved September 12, 2014, from http://sebastien.godard.pagesperso-orange.fr/

Jähnert, J., & Liang, Y. (2015). *Initial Setup Report, Deliverable D5.1.*

Rossini, A. a. (2015). *CAMEL Documentation, Deliverable D2.1.3.*

The Apache Software Foundation. (2014, June 21). *Hadoop MapReduce Next Generation - Cluster Setup*. Retrieved August 29, 2014, from http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/ClusterSetup.html