RC25437 (WAT1401-009) January 2, 2014
Computer Science

IBM Research Report

An Ontology-Based Framework for Model-Driven Analysis of Situations in Data Centers

Yu Deng, Ronnie Sarkar, Harigovind Ramasamy,
Rafah Hosn, Ruchi Mahindru
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
USA

Research Division
Almaden – Austin – Beijing – Cambridge – Dublin - Haifa – India – Melbourne - T.J. Watson – Tokyo - Zurich

An Ontology-Based Framework for Model-Driven Analysis of Situations in Data Centers

Yu Deng, Ronnie Sarkar, Harigovind Ramasamy, Rafah Hosn, Ruchi Mahindru

IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598 USA
{dengy, sarkar, hvramasa, rhosn, rmahindr}@us.ibm.com

***Abstract*—The capability to analyze systems and applications is commonly needed in data centers to address diverse problems such as root cause analysis of performance problems and failures, investigation of security attack propagation, and problem determination for predictive maintenance. Such analysis is typically automated using a hodgepodge of procedural code and scripts representing heuristics to be applied, and configuration databases representing state. As entities in the data center and relationships among them change, it is a challenge to keep the analysis tools up-to-date. Typically, such changes are reflected by adhoc extensions to the code and state or not reflected at all. There is a strong need for a structured, knowledge-based approach to performing such analyses where updates to entities and their relationships in the data center are reflected easily, and preferably with some degree of automation. We describe a framework based primarily on the principle of interpreting declarative representations of knowledge rather than capturing such knowledge in procedural code, and a variety of techniques for facilitating the continuous update of knowledge and state. A metamodel representing data center-specific domain knowledge forms the foundation for the framework. A model of the data center topological elements is an instantiation of the metamodel. Both metamodel and model are created through a bootstrapping process, but continuously updated using semi-automated techniques. Using the framework, we present a methodology for conducting any data center analysis activity (e.g., root cause analysis) as a model-driven topology subtree traversal, governed by knowledge embedded in the corresponding metamodel nodes. We show how to apply this methodology successfully using two use cases drawn from diverse domains: performance problem determination of a 3-tier Web application running in a virtualized environment, and an InfoSphere Streams processing application.**

***Keywords*-Metamodel; Model; Model based analysis; Dynamic Metamodel and Model updates;**

I. Introduction

The capability to analyze systems and applications is commonly needed in data centers to address diverse problems such as root cause analysis of performance problems and failures, investigation of security attack propagation, and problem determination for predictive maintenance. With the ever-growing size and complexity of data centers, there is a clear need for automated, tool-oriented analysis methods. Despite the demand for more rigorous approaches, such analyses is often performed using ad-hoc custom-built tools. It is typical to find data center administrators using a hodgepodge of procedural code and scripts to implement their heuristics for analysis, using the state of various data center entities stored in diverse configuration databases.

A consequence of increased virtualization at all levels in data centers is that entities in the data center and relationships among them change frequently. For example, in cloud computing, as service providers try to optimize their resource usage on an ongoing basis, the set of VMs hosted on a given hypervisor or the network elements in between two VMs communicating with each other may change in a dynamic fashion. To account for these ongoing changes in today's data centers, administrators typically use a patchwork of extensions to their code and state. Such adhoc methods may render the analysis results ineffective.

Clearly, there is a strong need for a structured, knowledge-based approach to performing various analyses needed in data centers where updates to entities and their relationships in the data center are reflected easily and preferably, with a degree of automation. For facilitating such analyses, we describe a foundational framework that employs declarative representations of knowledge – in the form of metamodels and models – as first-class citizens. Procedural code capturing the knowledge may be used, but only within the context of those metamodels and models. The framework also includes techniques for facilitating the continuous update of knowledge.

A primarily declarative *metamodel* representing data center-specific domain knowledge forms the foundation for the framework. The metamodel captures data center entities and relationships between them. Additionally, the metamodel encodes entity- and relationship-specific knowledge useful for analyses, e.g., the probable cause of a performance problem in an application server may include memory contention. The metamodel is statically created using domain expert knowledge, but is updated continuously in a semi-automated fashion under the guidance of knowledge engineers. We describe techniques that can be used to generate metamodel update candidates.

A *data center model* (or, simply model) is an instantiation of the metamodel used to represent the data center state, where every object and relationship in the data center (physical and logical) topology corresponds to an instantiation of an entity and relationship (type) in the metamodel. We describe how discovery techniques (both static and

dynamic) can be used to build the model and keep it up to date. Using the framework, we present a methodology for conducting any datacenter analysis activity (e.g., root cause analysis) as a model-driven topology subtree traversal governed by knowledge embedded in the corresponding metamodel nodes. We show how to apply this methodology successfully using two use cases drawn from diverse domains: performance problem determination in a 3-tier Web application, and in an InfoSphere Streams application.

In the rest of the paper, we first describe the model-based analysis framework in Section II. Then we introduce the metamodel in Section III. In Section IV and Section V, we discuss how to statically and dynamically discover new model and metamodel elements. In Section VI, we illustrate the application of our approach in a virtualized environment and in IBM Infosphere Streams[1]. We conclude the paper in Section VII.

II. Model-based Analysis Using Data Center Topology and Domain Knowledge

Figure 1 is a block diagram illustrating the overall model-based analysis framework. The figure shows a metamodel representing data center domain knowledge, and a data center model representing its current state. Static and dynamic topology discovery techniques described in subsequent sections are used to build the data center model and keep it up to date. Furthermore, the metamodel is continuously updated by knowledge engineers, who are periodically provided with metamodel update candidates by tools described in subsequent sections. The figure also shows knowledge-based analysis being performed by processing monitoring metrics data that is continuously collected from data center elements, using the model and metamodel to control the sequence of analysis steps as a guided walk of a subset of the data center topology.

The model-based framework described here is targeted at systems specialized for analysis of customer workloads running on data centers to address problems such as detection and root cause analysis of performance bottlenecks and detection and analysis of security attacks, and also identification of remediation actions. Such analysis is heavily topology-driven, where a topology is a detailed representation of objects in the data center spanning infrastructure elements (e.g., server, storage and network entities), core software entities (e.g., operating systems, hypervisors, file systems), virtualized entities (e.g., virtual machines/servers, virtual switches) as well as software/middleware elements (e.g., Web application servers, database servers). The relationship between the objects is also a core aspect of the topology, representing state such as file system X uses storage volume Y, virtual server A runs on hypervisor B, etc.

[1]http://www.ibm.com/software/data/infosphere/streams/

Topology-based analysis of data centers is not a new topic [5] [2]. However, past work has focused primarily on using the topology, rooted at some data center element where problem symptoms are initially observed, to drive the problem determination process via some traversal algorithm. The analysis performed at each node of the topology is script-driven. By introducing a metamodel in the framework where all domain knowledge associated with an entity or a relationship can be explicitly modeled, and treating each node and link in the topology as an instance of an entity or a relationship type in the metamodel, the knowledge used to drive the analysis is easier to represent and manage translating into ease of evolution of the overall system. This is important as new diagnostic knowledge about an existing entity (e.g., a solid state disk drive) is found on a Web site, or as new types of entities and relationships are introduced into the data center and the domain of analysis has to be expanded to cover them.

Consider, for example, a performance problem analysis process. Applications of interest are monitored for response time or throughput. The definition of a service level agreement (SLA) from a customer, for an application running on the data center, might state the following: the average response time of the application over N consecutive user requests must be less than or equal to T seconds. An SLA violation, determined by an analysis of response time metrics over a sliding time window, triggers a problem determination (PD) process. Note that the exact analysis techniques used and the exact metrics collected to perform the analysis is not the focus. It is the diagnosis knowledge, captured in the metamodel and leveraged by the analysis, that is being highlighted.

One approach to tackling the root cause analysis task above may involve the following topology-based approach. Starting with the application node in the topology, examine its immediate dependent resources. The procedure to be executed on any topology node consists of the following steps:

1) Examine data relevant to the node. This could be metrics collected by a monitoring system - e.g., CPU metrics, or information extracted from a log file.
2) Apply a test on the data (e.g., check if the disk I/O reate is correlated with the increase in response time). The test to perform should ideally be represented as a unit of knowledge in the metamodel node.
3) If the test succeeds in a definitive manner, then the analysis can be considered to be complete, and the desired conclusion (e.g., of the root cause of a performance problem) is available.
4) If not, the test results should indicate which node to examine next (e.g., DB2 server node), which is also encoded in the domain knowledge associated with the metamodel node.
5) Continue the traversal until a conclusion is reached, or

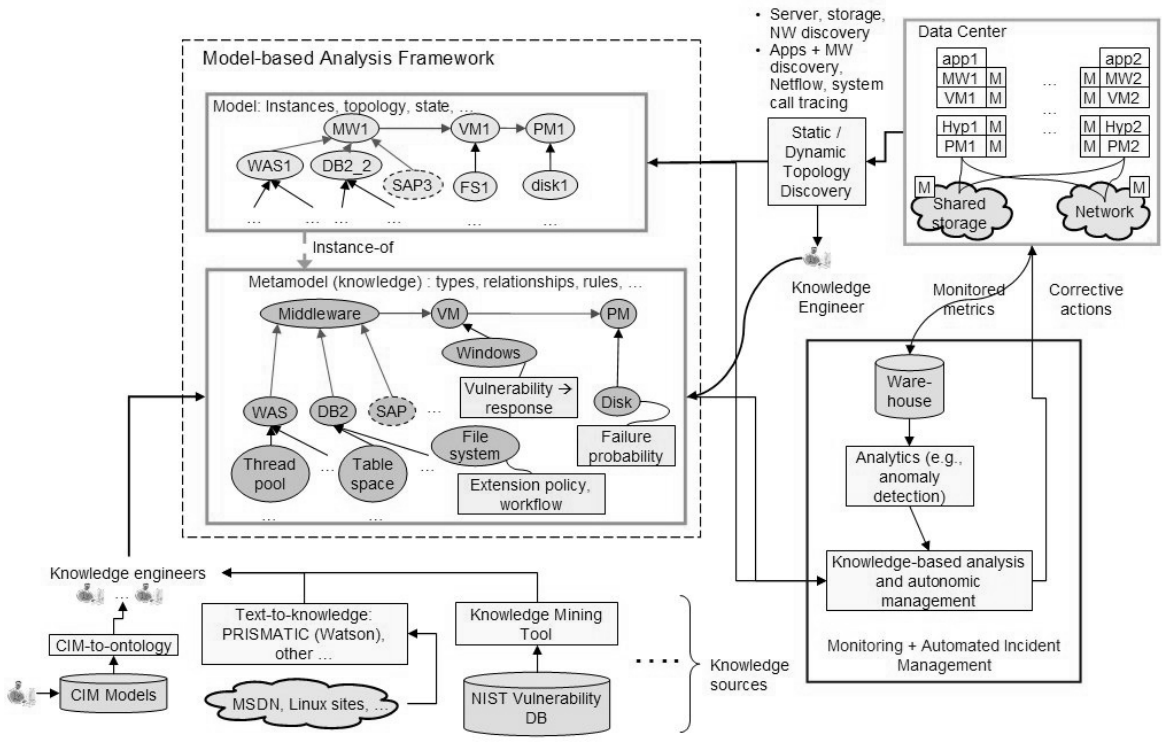

Figure 1. Model-Based Analysis and Autonomic Management of Data Centers

using heuristics, it can be concluded that no root cause can be determined using the current model and available (e.g., metrics) data. Note that the latter condition can trigger more detailed topology discovery.

The node-level test referenced above can be one or more of several alternatives such as - comparison with a threshold value (e.g., average CPU utilization over the last 30 minutes is over 95 percent), a statistical computation (e.g., correlation between a server's disk I/O rate and the increase in response time of the main application over the last 30 minutes is 0.9 or higher), or a more complex computation such as performing time series analyses of the history of a monitored metric associated with the topology node to detect anomalies [4].

An alternate approach to model-based analysis using a data center topology may involve the following steps, involving a different mode of topology traversal, but still leveraging diagnostic knowledge stored in the metamodel:

1) Consider a subset of the data center model that represents a "useful" unit of analysis, e.g., a hypervisor running on a physical machine, and all the virtual machine (VM) and other virtual resources provisioned on that hypervisor. In this scenario, the hypervisor of interest is the one that has been used to provision one or more VMs that are running components of the customer application exhibiting performance issues.
2) Construct a Bayesian Network that is dericed from the data center model of the hypervisor and its virtual resources. Nodes of the model form the nodes of the Bayesian network, and the links between the nodes are identical to instances of certain relationships in the topology, such as UsesResourceOf in Figure 5.
3) Initialize the Bayesian Network model with joint probability values that are captured in the metamodel nodes and relationships corresponding to the model subset. These probability values are assumed to have been learnt using standard Bayesian Network learning techniques. If the probability values are recomputed periodically using diagnostics history data, then they are reflected back in the metamodel.
4) Run the Bayesian Network using the metrics or other data collected in each node, as in the earlier approach described above.
5) If the Bayesian Network identifies the root cause resource that can explain the application performance problem with sufficiently high confidence, then the analysis task is complete. Otherwise, the focus of the analysis has to move to another part of the data center model.

III. METAMODELS

Domain knowledge represented in the data center metamodel forms the core of the framework. The basic meta-

model is an OWL[2] ontology representing types of entities and relationships covering all objects of interest in a virtualized data center that may need to be inspected and analyzed.

The domain metamodel is primarily declarative in nature and represents data center entities (e.g., Server, StorageElement, MiddlewareSystem) and relationships (e.g., UsesResourceOf, RunsOn). Additional pieces of knowledge can be associated with the core domain entities and relationships. For example, a constraint can be attached to a VirtualMemory entity that participates in a PartOf relationship with a WindowsOS entity, representing a guideline for setting the page file size as a multiple of virtual memory size for attaining reasonable application performance. Such a guideline could have been discovered from a Web crawl of the Microsoft Developer Network (MSDN) Web site for structured knowledge extraction, as described in Section V. Another example of domain knowledge could be a published vulnerability of a given IT component and a response to a security attack exploiting that vulnerability, representing knowledge extracted from the National Vulnerability Database[3]. At its core, domain knowledge captures resource dependencies in running customer workloads, and information required to diagnose resource bottlenecks or the propagation of information (e.g., malware) along the dependency paths.

Domain metamodels need to be kept up to date to facilitate accurate model-based analysis. This is a complex task, and both established methods such as leveraging CIM[4] models and more state-of-the-art techniques for extracting structured knowledge from unstructured text [3] [9] [6] can be applied within the framework. Metamodel updates have to carefully supervised, and automated knowledge extraction techniques should only be considered as an aid, with the final inclusion of a piece of knowledge occurring under the supervision of an experienced knowledge engineer.

The paper also proposes some new approaches for extending the metamodel, which can be performed as a side effect of both static as well as dynamic topology discovery. Topology discovery is described in Section IV, and metamodel discovery leveraging topology discovery is described in Section V.

IV. Topology Discovery

Topology discovery is an important component of the framework, since functionally rich discovery mechanisms are required to construct accurate data center models for performing analysis. Static topology discovery as described in Section IV-A uses well-known techniques that the framework takes advantage of. Approaches for efficient dynamic topology discovery are outlined in Section IV-B.

[2]http://www.w3.org/TR/owl-ref/

[3]http://nvd.nist.gov/

[4]http://dmtf.org/standards/cim

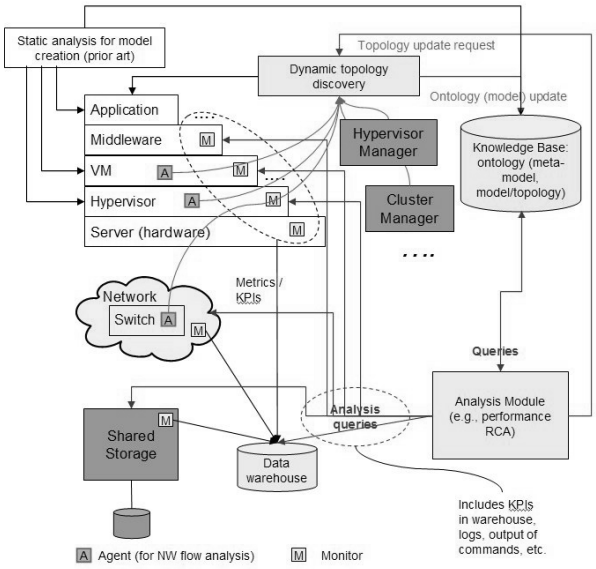

Figure 2. Dynamic Topology Discovery Architecture

A. Static Topology Discovery

Static topology discovery via configuration and code analysis is well known and implemented in commercial and research systems [1] [8] [7]. Static discovery tools leverage knowledge of configuration databases (e.g., the Windows registry) as well as the format of configuration files (e.g., EAR files for J2EE applications) to enumerate software components installed on a server, and the dependency of each component on other components which may reside on the same or different servers. File system configuration can be used to understand dependencies on storage systems that are local disks or remote file or block storage systems. Network topology constructed by network management tools can be queried to find network components - switches and routers - that a distributed application depends on.

The model-based analysis framework of this paper depends on static discovery tools to populate a *configuration management database* (CMDB). A *mapping table* is subsequently used to convert each entry in the CMDB to instances of entity and relation types in the metamodel to construct the model. The mapping table has to be manually constructed[5], since the CMDB schema is a legacy data center artifact and the metamodel is an add-on component of our architecture.

B. Dynamic Topology Discovery

Demand-based dynamic topology discovery starts a statically discovered topology. The statically discovered model may be incomplete since application components may communicate with remotely located components in the data

[5]The size of the mapping table is linear in the size of the metamodel, which is normally not big.

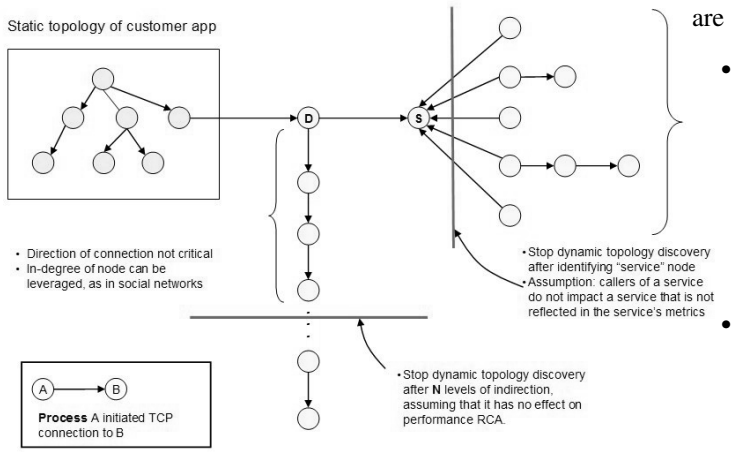

Figure 3. Network Flow-Based Dynamic Topology Discovery

center whose identities are hardcoded, or such dependencies might be defined in configuration files that are not well-documented, and thus are not processed by all topology discovery tools. Such remote components can be discovered by applying well-known techniques that involve either analysis of system calls or of network flows initiated by the *known* components of the model. Another requirement for dynamic topology updates may occur due to well-understood dynamics of a data center, e.g., caused by virtual machines (VMs) being migrated from one hypervisor to another, or scale-up and scale-down operations being performed on multi-node clusters, in order to deal with load balancing requirements. We refer to this as planned dynamic topology discovery.

Figure 2 describes the various components of the dynamic topology discovery architecture. For discovering topology changes of the "planned" flavor, the discovery component depends on cluster or hyperisor managers for information about topology changes that they initiate. If such systems do not offer notification mechanisms, then some type of polling mechanism can be implemented to detect dynamic changes.

The discovery of resource-usage relationships not covered by static topology discovery tools is initiated only when attempts at model-based analysis of data center problems fail repeatedly, indicating that the model may be incomplete. This approach is more efficient than continously performing such analyses to detect topology changes. In our framework, such dynamic topology discovery leverages network flow analysis techniques - either by examining network packet traces in individual servers using 'tcpdump' type of utilities, or by examining Netflow traces in switches. Agents depicted in Figure 2 enable the capture of network data.

Our approach to the discovery of new resource-usage relationships in a data center involves a 3-step process that is repeated until deterministic or heuristic stopping conditions are detected, as described below.

- Step 1: for each known topology node representing a software component, the operating system (OS) running on the server where the software component is running is identified. If the node is mappable to a process, the topology discovery system needs to determine whether the process is participating in network connections - for example, has an open TCP (Transmission Control Protocol) socket in LISTENING or CONNECTED mode.
- Step 2: a 'tcpdump'-like utility and operating system facilities are used for each such process to analyze network connections it is participating in, and to discover the identity of partner end points (IP-address, port-number) of each such connection. The use of Netflow tracing (a standard feature in modern network switches) to enable this type of analysis is also an option. Note that in the latter case, Netflow tracing is typically set in switches on a permanent basis and such trace data stored in a warehouse, where it may be accessed later as required for various analyses tasks. With Netflow data from the past, the "direction" of a TCP connection identifying the initiator and the listener can be determined, which is important for identifying *services* in the data center. For each such TCP connection, if the connection partner is not a known topology node, it is identified as a new node reflecting a new dependency and it is eventually added to the topology/model.
- Step 3: If the "listening" partner identified in step 2 is not an existing topology node, but the listening port is well-known, then this new node can be identified as an instance of a known metamodel node (for example, a Web server listens on port 80, IBM's DB2 database server listens on port 50000, etc.). Note that in some cases, more analysis work will be required to complete this step, e.g., to determine the exact identity of the Web server listening on port 80, for which well known discovery tools such as TADDM can be employed.
  If the TCP end point is not a well-known port, then potentially, a new metamodel node may have been identified if analysis of network flows to/from that node is used to determine that the node is a new type of service. The process of identifying a new service in the data center is described further below.

Steps 1 through 3 above are repeated as long as new topology nodes are discovered, though an iteration may be terminated after a preconfigured threshold of N iterations have been performed. Additionally, discovery of a new service can be a stopping criterion for terminating dynamic topology discovery. The rationale for the latter is the observation that the performance of a topology node that provides a service is primarily determined by the load imposed on it by its clients, which can be measured by node-local resource usage metrics alone without analyzing client metrics.

Figure 3 illustrates the 3-step discovery process. As steps 1 through 3 are iteratively applied, new topology nodes are discovered as described by the algorithm. Node S has been identified as a (known or unknown) service, because a majority of connections to that node are found to have been initiated by other nodes - as indicated by the direction of the TCP connection setup found in switch Netflow records. Therefore, further discovery based on node S is not continued. However, network flow analysis of node D leads to the discovery that it uses another node, which uses another node, and so on, and as per the 3-step process, the transitive discovery step is terminated after N-steps where N could be statically configured but modified based on experience over time.

V. Metamodel Discovery

In a model-based framework where domain knowledge is a core component, the ability to create an accurate metamodel and keep it up to date is of critical importance. This section identifies several approaches for identifying new candidate pieces of knowledge which can be integrated into the existing metamodel. Figure 1 outlines some knowledge sources and extraction techniques.

The initial metamodel can be manually constructed by domain experts and knowledge engineers. An alternate approach for bootstrapping a starter metamodel could be to convert existing IT domain knowledge encoded in (machine readable) CIM models for IT management, where profiles are available for server, storage, network and even some middleware components, into OWL ontologies in a semi-automated manner. Such a process should be executed periodically since CIM models are regularly updated by different expert groups. For example, the CIM block storage device modeling subgroup published a new model to represent solid state drives when they were first introduced. Ingesting such a new CIM model into the ontology at the right location (subclass of DiskDrive entity) can automatically enrich the metamodel beyond just the mere addition of entity and relationship types, e.g., by inheriting from the already-defined diagnostic knowledge that a DiskDrive (and any subclass therof) can impact the performance of a file system that uses it.

Recently refined artificial intelligence (AI)-based methods to extract structured knowledge (relationship triples) from analysis of unstructured text can also be used to update the metamodel [3] [9] [6]. However, such automated approaches typically extract a huge number of relationships, and the filtering of useful intrinsic relationships for enhancing the metamodel can be a laborious task.

In addition to the above approaches that are well-documented, the authors propose two other methods to detect new candidate metamodel nodes. Both approaches are side effects of topology discovery.

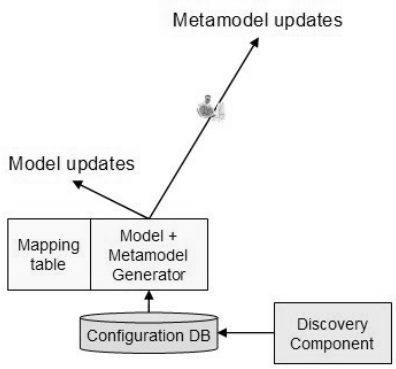

Figure 4. New Metamodel Node Learning

A new metamodel node may be discovered as a side effect of the dynamic topology discovery process described in Section IV-B. In Step 3, if the partner of the known topology node sharing a TCP connection is using a non-well known TCP port, and it is discovered that the number of TCP connections initiated to that node by other TCP end points greatly exceeds the number of TCP connections the node itself has initiated, then the new node is identified as a potentially new type of service that may play an important role in the data center. If true, then the new type of service would warrant modeling as a new entity type in the domain metamodel. The Dynamic Topology Discovery component in Figure 2 alerts the knowledge engineers of the finding, who can initiate further manual analysis steps to confirm or reject the recommendation. In knowledge engineering terms, extrinsic domain knowledge about a new node discovered in the data center, coupled with heuristic analysis of the number of incoming versus outgoing TCP connections for which that node is an end point, is used to infer intrinsic domain knowledge about a new type of entity in the data center.

New metamodel nodes may also be discovered as a side effect of static topology discovery. For example, a static discovery tool such as IBM's Tivoli TADDM product will record in a CMDB a new descriptor of a disk drive it found by querying the operating system's device configuration, which the operating system itself extracted from the disk drive during boot. That configuration information might describe the new disk drive as a "2 TB SAS drive" - for which no metamodel node currently exists because the knowledge engineers may be oblivious to the fact that such disk drives have been introduced in the data center and need to be modeled.

To analyze the output of discovery tools to build the data center topology/model and associate it with known metamodel nodes, mapping tables (such as depicted in Figure 4) are employed. Mapping tables enable parsing of configuration database fields to identify which database

items correspond to which types of metamodel nodes (e.g., disk drives, CPUs, virtual servers). When a new configuration item described as "2 TB SAS drive" is first encountered by configuration-to-model mapper component, no mapping table entry will exist. Based on the location of the item's entry in the CMDB, namely where information about other types of disk drives are stored, the metamodel discovery component can infer that the configuration item "2 TB SAS drive" represents a new type of disk drive, and the knowledge engineer, who can confirm or reject the system-generated hypothesis, is alerted.

VI. Case Study

We have applied a subset of the model-based approach to perform root cause analysis of performance problems in two application environments, one a traditional 3-tier Web application running in a virtualized environment, the other an InfoSphere Streams application. These two cases are described below.

A. Applications in a virtualized environment

The Apache DayTrader application was used for our first experiment. The 3-tier Web application and its resources were deployed in two virtual machines (VMs) provisioned on a common hypervisor, and a separate physical machine. One VM was used to run a Web server, Websphere Application Server and the DayTrader application. The second VM ran IBM DB/2. The second physical server ran the Linux NFS server to provide storage to the hypervisor. Additional VMs were provisioned to run JMeter to generate a trading-request load to DayTrader, and to also generate an external I/O request load to the NFS server.

In the setup, trading requests submitted from JMeter were processed by Websphere Application Server and backend database requests were forwarded to IBM DB/2, the table spaces of which are allocated on a virtual disk in the VM, mapped to a file on the hypervisor's NFS mounted file system. Figure 5 shows the data center model representing this topology, and also the metamodel nodes that represent the underlying entity and relationship types. In the model, the nodes WIN2K8VM1 represents the VM running DayTrader and the Web components, WIN2K8VM2 represents the VM running DB/2, and dushesh66 models the physical machine running the NFS server.

In the experimental testbed, operating system (OS)-specific monitoring agents were installed in each virtual and physical server to collect metrics covering CPU, memory and disk usage. Additionally, a special monitoring agent to measure Web application (HTTP request-response based) response time was also installed on a separate node. The response time agent monitored all network flows in promiscuous mode. All agents output their metrics periodically (and frequently) into a data warehouse. Constant ingestion of warehouse data was used by the diagnostic system to

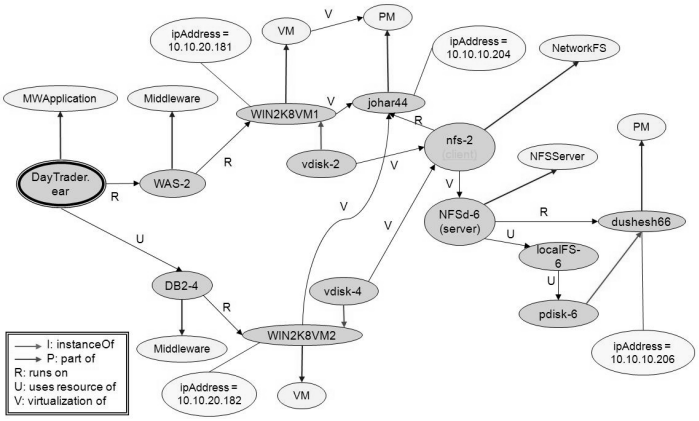

Figure 5. Topology of the DayTrader Application

drive the model-based root cause analysis of performance degradation.

In the scenario, if the application SLA metric (e.g., ApplictionResponseTime exceeds a predefined threshold (as defined in the customer SLA), the model based diagnosis process is triggered. The diagnostic logic traverses the portion of the data center model starting at the node labelled "DayTrader.ear" in Figure 5 and starts the topology traversal (drill-down) process. The diagnostic process only leverages OS metrics applied to the virtual and physical servers hosting dependent resources. Network resource dependencies were ignored.

In each node analyzed during topology traversal (WIN2K8VM1, WIN2K8VM2, dushesh66), the system examines four different OS metrics: CPU-utilization, memory-utilization, disk-utilization, and disk-latency. The history of each metric is analyzed using four separate algorithms, two sets of time series analysis based anomaly detection algorithms (see [4]), computation of pair wise correlation of each OS metric and the application response time, and also comparison of the average value of each OS metric in the last T minutes compared with a threshold. By using a voting mechanism, the system determines if there is an anomaly in the "recent" values of the recorded metrics. If the degree of anomaly is definitive, then the traversal is stopped. Otherwise, the traversal continues to the next node.

The first server node to be examined is WIN2K8VM1, which is the computer system instance that DayTrader is running on. In the case of WIN2K8VM1, it turns out that none of the base metrics values exhibit any anomaly. Next, the system traverses next node hosting a resource that the application is dependent on, namely WIN2K8VM2 which hosts the DB2 instance that DayTrader uses. Analysis of the base metrics on that VM indicates that the disk latency is anomalous. That leads the system to focus next on the third node (dushesh66) hosting the NFS server, because the (virtual) disk of WIN2K8VM2 is mapped to a NFS-mounted

file system exported by that server. A resource constraint on the NFS server can result in delays in disk I/O request processing times in WIN2K8VM2, thereby affecting DB/2 performance. Analysis of the base metrics of dushesh66 indicates that the disk-utilization metric exhibits significant anomalies. At that point, the drill-down stops, and the load on the NFS server is declared as the root cause of the SLA violation detected in DayTrader.

VII. Conclusions and Future Work

We have presented a model based framework for performing automated analysis of complex applications using resource dependencies and diagnostic knowledge. In this framework, the metamodel is a critical component for capturing IT domain knowledge, and the model that is an instantiation of the metamodel is a critical component for capturing data center state that is leveraged in the analysis process. We have provided methodologies for the creation of both the metamodel and the model and to keep them current. We have also illustrated how to apply this framework to analyze application performance problems in both virtualized and Streams environments.

For our future work, we plan to enhance two aspects of our framework : (1) enhance the anomaly detection methods to enable adjustments based on application workloads; (2) enhance the analysis techniques for problem prediction.

References

[1] Tivoli application dependency discovery manager. www.ibm.com/software/tivoli/products/taddm.

[2] M. K. Agarwal, K. Appleby, M. Gupta, G. Kar, A. Neogi, and A. Sailer. Problem determination using dependency graphs and run-time behavior models. In *DSOM*, pages 171–182, 2004.

[3] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *Proceedings of IJCAI*, pages 2670–2676, 2007.

[4] A. Biem, H. Feng, A. V. Riabov, and D. S. Turaga. Real-time analysis and management of big time-series data. *IBM J. Research and Development*, 57(3/4):8:1–8:12, 2013.

[5] A. B. Brown, G. Kar, and A. Keller. An active approach to characterizing dynamic dependencies for problem determination in a distributed environment. In *Integrated Network Management*, pages 377–390, 2001.

[6] J. Fan, A. Kalyanpur, D. C. Gondek, and D. A. Ferrucci. Automatic knowledge extraction from documents. *IBM J. Research and Development*, 56(3/4):5:1–5:10, 2012.

[7] N. Joukov, B. Pfitzmann, H. V. Ramasamy, N. G. V. andM V Devarakonda, and T. Ager. Itbvm: It business value modeler. In *Proceedings of IEEE International Conference on Services Computing*, pages 128–135, 2009.

[8] K. Magoutis, M. Devarakonda, N. Joukov, and N. Vogl. Galapagos: Model-driven discovery of end-to-end application-storage relationships in distributed systems. *IBM J. Research and Development*, 52:367–378, 2008.

[9] P. P. Talukdar, D. Wijaya, and T. Mitchell. Acquiring temporal constraints between relations. In *Proceedings of the Conference on Information and Knowledge Management (CIKM 2012)*, Hawaii, USA, October 2012. Association for Computing Machinery.